EE512A - Advanced Inference in Graphical Models — Fall Quarter, Lecture 3 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 6th, 2014



Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F1/61 (pg.1/117)

Logistics

- 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.

Comparisons of families

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

Lemma 3.2.1

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

Lemma 3.2.2

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

• Therefore

Corollary 3.2.3

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

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

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

Trees defined in many ways

Theorem 3.2.2 (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:

- $\bullet~G$ 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.

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F5/61 (pg.5/117)

Trees, inference, and distributive law

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with |S| > 2 induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(f)})$ 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

$$p(x) = \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1})$$
(3.8)

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

$$p(x_3, x_4) = \sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \dots, x_N)$$
(3.9)

Summing, Marginalization, and variable elimination

- Problem: Sum over x_2 in Eq. **??** 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:

$$g(a)h(c) = \sum_{b} f_1(a,b)f_2(b,c)$$
(3.17)

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

$$f(x_A, x_C) = \sum_{x_B} f_1(x_A, x_B) f_2(x_B, x_C)$$
(3.18)

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

Logistics

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 3.2.2

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 incident edges from the graph.

- \bullet 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.

Review: graph theoretic node elimination

- $\sigma = (\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_n) \quad n = |l|$
- Let $\delta_G(v)$ denote the neighbors of node $v \in V$ in graph G.
- Elimination of a node $v \in V(G)$ from G forms graph G_1
- $G_1 = (V_1, E_1)$ where $V_1 = V \setminus \{v\}$, and where $E_1 = (E \cap V_1 \times V_1) \cup \underbrace{\delta_G(v) \times \delta_G(v)}_{\mathcal{S}_G(v)}$.
- In G_1 , $\delta_G(v)$ forms a clique.
- Any edges in G_1 not in G are part of the fill-in edges F.
- We can say that $G_1 = G_{\sigma_1}$ is the result of eliminating σ_1 from G
- Also, that $G_2 = (G_{\sigma_1})_{\sigma_2}$ is the result of eliminating σ_2 starting from G_{σ_1} , that $G_3 = ((G_{\sigma_1})_{\sigma_2})_{\sigma_3}$ is the result of eliminating σ_3 starting from G_2 and so on.
- Therefore, the ordering σ defines a sequence of graphs $(G_0, G_1, G_2, \ldots, G_{N-1})$ where $G_0 = G$ and where G_{N-1} consists of only one node σ_N .
- Reconstituted graph is $G' = (V, E \cup F)$.

Example elimination on graphs



Review: Other facts about node elimination

- A tree always has ≥ 2 leaf nodes.
- Eliminating leaf node always yields sub-tree
- Can continue eliminating nodes in a tree graph.
- Eliminating leaf-nodes never produce fill-in edges.
- data-structure for leaf nodes is easy
- Computation corresponding to tree-elimination is fast $O(Nr^2)$.

Logistics

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.

Logistics

Logistics

Review

Computations for marginal "rooted" at edge (x_1, x_2)



$$\phi_{\frac{1}{2}4,8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{3.19a}$$

$$\phi_{\overline{\gamma},3}(x_3) = \sum_{x_7} \psi_{7,3}(x_7, x_3) \tag{3.19b}$$

$$\phi_{8, 44, 3}(x_3) = \sum_{x_8} \psi_{8, 3}(x_8, x_3) \phi_{44, 8}(x_8) \tag{3.19c}$$

$$\phi_{\mathcal{G},3}(x_3) = \sum_{x_9} \phi_{\mathcal{G},3}(x_9, x_3) \tag{3.19d}$$

$$\underbrace{\gamma, \psi_{4,8}, g, g, 1}_{x_{3}}(x_{1}) = \sum_{x_{3}} \psi_{1,3}(x_{1}, x_{3}) \phi_{\overline{\gamma},3}(x_{3}) \phi_{\overline{\beta}, \psi_{4,3}}(x_{3}) \phi_{\overline{\beta},3}(x_{3}) \quad (3.19e)$$

$$\phi_{\mathcal{V}5,10}(x_{10}) = \sum_{x_{15}} \psi_{10,15}(x_{10}, x_{15}) \tag{3.19f}$$

$$\phi_{\mathcal{Y}5,\mathcal{Y}0,4}(x_4) = \sum_{x_{10}} \psi_{4,10}(x_4,x_{10})\phi_{\mathcal{Y}5,10}(x_{10}) \tag{3.19g}$$

$$\phi_{\mathcal{Y}1,4}(x_4) = \sum_{x_{11}} \psi_{4,11}(x_4, x_{11}) \tag{3.19h}$$

$$\phi_{\mathcal{Y}0,\mathcal{Y}1,\mathcal{Y}5,\mathcal{Y},1}(x_1) = \sum_{x_4} \psi_{1,4}(x_1,x_4)\phi_{\mathcal{Y}5,\mathcal{Y}0,4}(x_4)\phi_{\mathcal{Y}1,4}(x_4)$$
(3.19i)

- 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.

Logistics

Message passing protocol

Definition 3.2.4

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 → j indicates a message from i to j, then the protocol may be written as i → j only when ∀k ∈ δ(i) \ {j}, k → i, where δ(i) are the neighbors of node i in G.
- 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.

Arbitrary Queries

Non-Tree Graphs

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.

Prof. Jeff Bilmes

Tree Inference	Arbitrary Queries	Non-Tree Graphs

Better notation

• Notation is unwieldy. Rather than keep track of entire history, as in $\phi_{y_5,y_0,4}(x_4)$, use notation that only indicates neighbors in a message

Refs

Better notation

- Notation is unwieldy. Rather than keep track of entire history, as in $\phi_{15,10,4}(x_4)$, use notation that only indicates neighbors in a message
- We use µ_{i→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).

Tree Inference

Arbitrary Queries

Non-Tree Graphs

Better notation

- Notation is unwieldy. Rather than keep track of entire history, as in φ_{μ5,μ0,4}(x₄), use notation that only indicates neighbors in a message
 We use μ_{i→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

$$\phi_{\mathcal{Y}4,8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{3.1}$$

After

$$\mu_{14\to8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{3.2}$$

Tree Inference

Arbitrary Queries

Non-Tree Graphs

Better notation

- Notation is unwieldy. Rather than keep track of entire history, as in φ_{μ5,μ0,4}(x₄), use notation that only indicates neighbors in a message
 We use μ_{i→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

$$\phi_{\mathcal{Y}4,8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{3.1}$$

After

$$\mu_{14\to8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{3.2}$$

• Before

$$\phi_{7, \gamma 4, \beta, 9, 3, 1}(x_1) = \sum_{x_3} \psi_{1,3}(x_1, x_3) \phi_{7,3}(x_3) \phi_{8, \gamma 4, 3}(x_3) \phi_{9,3}(x_3)$$
(3.3)

After

$$\mu_{3\to 1}(x_1) = \sum_{x_3} \psi_{1,3}(x_1, x_3) \mu_{7\to 3}(x_3) \mu_{8\to 3}(x_3) \mu_{9\to 3}(x_3)$$
(3.4)

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F17/61 (pg.20/117)

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
Generic form	of message		
$\mu_{i ightarrow i}(z)$	$(x_i) = \sum_{i=1}^{n} (\psi_{i,i}(x_i, x_i))$	$\prod_{\mu_{k \to i}(x_i)} \mu_{k \to i}(x_i)$	(3.5)

 $\mu_{i \to j \langle u_j \rangle}$ $= \sum_{x_i} \left(\frac{\psi_{i,j}(x_i, x_j)}{k \in \delta(i) \setminus \{j\}} \mu_{k \to i}(x_i) \right)$

Message is of form:

- 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_i , thus motivating the terminology of a "message", and

 \bigcirc then finally marginalizing away x_i thus yielding the desired message to be delivered at the destination node x_i .

Arbitrary Queries

Multiple Tree Queries

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

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		
Multiple Tree Overies			

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = S$ 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.

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		
\mathbf{M} high \mathbf{T} \mathbf{O} is			

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = S$ 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.

Arbitrary Queries

Refs

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = S$ 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 ${\cal O}(kNr^2)$ 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.

Tree Inference

Arbitrary Queries

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = S$ 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 ${\cal O}(kNr^2)$ 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.



Tree Inference

Arbitrary Queries

Refs

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



 Tree Inference
 Arbitrary Queries
 Non-Tree Graphs

 IIII
 IIIIII
 IIIIIII

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 $\mu_{3\to1}(x_1)$.



Tree Inference Arbitrary Queries Non-Tree Graphs Refs

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

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 $\mu_{2\to1}(x_1)$.

Tree Inference Arbitrary Queries Non-Tree Graphs

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 $\mu_{2\to1}(x_1)$.
- 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.

Prof. Jeff Bilmes

F20/61 (pg.31/117)

Arbitrary 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.

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	111111		
Multiple Tree Overies			

Multiple Tree Queries

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

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

Arbitrary Queries

Non-Tree Graphs

Refs

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:

 $\mu_{7\to3}(x_3), \ \mu_{9\to3}(x_3), \ \mu_{15\to10}(x_{10}), \ \mu_{10\to4}(x_4), \ \mu_{11\to4}(x_4), \ \mu_{4\to1}(x_1), \ \mu_{5\to2}(x_2), \ \text{and} \ \mu_{12\to6}(x_6) \ \text{leaving chain} \ x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}.$



Arbitrary Queries

Non-Tree Graphs

Refs

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:

 $\mu_{7\to3}(x_3), \ \mu_{9\to3}(x_3), \ \mu_{15\to10}(x_{10}), \ \mu_{10\to4}(x_4), \ \mu_{11\to4}(x_4), \ \mu_{4\to1}(x_1), \ \mu_{5\to2}(x_2), \ \text{and} \ \mu_{12\to6}(x_6) \ \text{leaving chain} \ x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}.$



• remaining messages, from x_{14} to x_{13} and from x_{13} back to x_{14} .
Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	111111		
\mathbf{M} high \mathbf{T} \mathbf{A}			

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:

 $\mu_{7\to3}(x_3), \ \mu_{9\to3}(x_3), \ \mu_{15\to10}(x_{10}), \ \mu_{10\to4}(x_4), \ \mu_{11\to4}(x_4), \ \mu_{4\to1}(x_1), \ \mu_{5\to2}(x_2), \ \text{and} \ \mu_{12\to6}(x_6) \ \text{leaving chain} \ x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}.$



• 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)

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Re

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:

 $\mu_{7\to3}(x_3), \ \mu_{9\to3}(x_3), \ \mu_{15\to10}(x_{10}), \ \mu_{10\to4}(x_4), \ \mu_{11\to4}(x_4), \ \mu_{4\to1}(x_1), \ \mu_{5\to2}(x_2), \ \text{and} \ \mu_{12\to6}(x_6) \ \text{leaving chain} \ x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}.$



- 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.

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F22/61 (pg.38/117)

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		

All edge Queries

- As number of queries increases, so does efficiency (queries/message)
- Consider computing $p(x_i, x_j)$ for all $(i, j) \in E(G)$.
- Naive case, N-1 edges $O(N^2r^2)$.
- 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



Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		

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

$$p(x_i, x_j) = \psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \to i}(x_i) \prod_{k \in \delta(j) \setminus \{i\}} \mu_{k \to j}(x_j)$$
(3.6)

• Overall computation $O(Nr^2)$.



EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
All edge Queries			

Theorem 3.3.1

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 $p(x_i, x_j)$.

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		
All edge queries - alg	orithm		



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

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

- 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.

- 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.

- 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

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
Collect Evidence			

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

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.

- 1 foreach $u \in child(c)$ do
- 2 call CollectEvidence $(u \rightarrow c)$
- 3 Compute

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

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
Distribute Evidence			

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

- **Input**: A rooted tree G = (V, E) with a parent node $p \in V$ and a child $c \in \text{child}(p)$.
- **Result**: A message propagated from p to c that obeys the message passing protocol.
- 1 Compute

3

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

- 2 foreach $u \in \text{child}(c)$ do
 - call DistributeEvidence $(c \rightarrow u)$

Arbitrary Queries

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 3.6.
- 1 Designate an arbitrary node $r \in V$ as the root.
- 2 foreach $c \in \operatorname{child}(r)$ do
- $\mathbf{3} \quad | \quad \mathsf{call} \ \mathsf{CollectEvidence}(c \to r)$
- 4 foreach $c \in child(r)$ do

5 call DistributeEvidence $(r \rightarrow c)$

Prof. Jeff Bilmes

Tree Inference

Arbitrary Queries

Non-Tree Graphs

Collect/Distribute Evidence and MPP

• All messages obey the message passing protocol.

- 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.

Tree Inference

Arbitrary Queries

Non-Tree Graphs

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 ...

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F31/61 (pg.54/117)

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		

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?

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		

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?



Arbitrary Queries

Non-Tree Graphs

Refs

Collect/Distribute Evidence

• Why called Collect/Distribute Evidence??

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F32/61 (pg.58/117)

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

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



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



- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$
- easy to obtain conditionals $p(x_i, x_j | \bar{x}_E)$
- Current framework is consistent with the fact that $p(x_i, \bar{x}_j) = p(x_i, x_j) \delta(x_j, \bar{x}_j)$, so the delta functions annihilate excess terms.

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$
- easy to obtain conditionals $p(x_i, x_j | \bar{x}_E)$
- Current framework is consistent with the fact that $p(x_i, \bar{x}_j) = p(x_i, x_j)\delta(x_j, \bar{x}_j)$, so the delta functions annihilate excess terms.
- Collect/Distribute Evidence allows many orders, different roots, different orders too/from the roots, and also parallel implementations.

Refs

Refs

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 $k \in \delta(i) \setminus \{j\}$.
- $O(|E|(2r+r^2))$ total storage.

Arbitrary Queries

Non-Tree Graphs

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

$$\psi_{i,j}'(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j) \mu_{k \to i}(x_i)$$
(3.7)

• Final factor, after incorporating all messages, has value $\psi_{i,j}'(x_i,x_j)$ where:

$$\psi'_{i,j}(x_i, x_j) = \psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \to i}(x_i)$$
 (3.8)

 $\bullet\,$ Outgoing message to j depends only on the edge function, and becomes

$$\mu_{i \to j}(x_j) = \sum_{x_i} \psi'_{i,j}(x_i, x_j).$$
(3.9)

• Never require storage at only node i

Prof. Jeff Bilmes



- 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 $k \in \delta(j) \setminus \{i\}$.

Tree Inference	Arbitrary Queries	Non-Tree Graphs	Refs
	11111		
Alternative propaga	ation 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 $\delta(i) \setminus \{j\}.$



- 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.

- 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).
- 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).
- ... need somehow to divide out first set of messages before sending back, but can't do that if lost that info.

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 $\mu_{i \rightarrow j}(x_j)$

- 1 $\mu_{i \to j}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \to i}(x_i)$; /* message as normal */
- 2 $\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j)/\mu_{i \to j}(x_j)$; /* table update divide outgoing message out */
- 3 if j is not the root then
- 4 Let $k \in \delta(j)$ be the neighbor of j towards the root ;
- 5 $\psi'_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \mu_{i \to j}(x_j)$; /* 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.

Non-Tree Graphs

Refs

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 $\phi_{7,3,1}(x_3)$.

- Use only two extra tables per separator (square) node $i \in V$, which store incoming messages at i
- The two tables $\phi_{ijk}^n(x_j)$ (new) and $\phi_{ijk}^p(x_j)$ (previous) at each separator node, which keeps track of incoming messages.
- At start, initialize both tables to unity $\phi_{ijk}^n(x_j) = 1$, $\phi_{ijk}^p(x_j) = 1$ $\forall x_j \in \mathsf{D}_{X_j}$.
- 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

Refs

Alternative propagation styles

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

- $\begin{array}{l} \mathbf{1} \ \phi_{i,j,k}^n(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j) \ ; \ /^* \ \text{message as normal stored in node} & */\\ \mathbf{2} \ \psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \frac{\phi_{i,j,k}^n(x_j)}{\phi_{i,j,k}^p(x_j)} \ ; \ /^* \ \text{update} \ (j,k) \ \text{edge potential.} & */ \end{array}$
 - At this point, step 2 same as $\psi_{j,k}(x_j,x_k) \leftarrow \psi_{j,k}(x_j,x_k) \phi_{i,j,k}^n(x_j)$
 - 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 $\mu_{i \rightarrow j}(x_j)$

1 $\phi_{i,j,k}^n(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */ 2 $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \frac{\phi_{i,j,k}^n(x_j)}{\phi_{i,j,k}^p(x_j)}$; /* 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.
- \bullet uniform message style, and can once again send multiple messages along an edge if we want. \odot
- ullet If divide same cost as multiply, less compute than previous style. $\ensuremath{\textcircled{}}$
- On the other hand, once again more storage, even more than originally!! ③

Non-Tree Graphs

Refs

Alternative propagation styles

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

1 $\phi_{i,j,k}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */ 2 $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k)\phi_{i,j,k}(x_j)$; /* update (j,k) edge potential. */

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

1 foreach $x_j \in D_{X_j}$ do 2 $\phi_{i,j,k}(x_j) \leftarrow \frac{1}{\phi_{i,j,k}(x_j)} \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */ 3 $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \phi_{i,j,k}(x_j)$; /* update (j, k) edge potential. */

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

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

F44/61 (pg.79/117)

- 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.

	Arbitrary Queries	Non-Tree Graphs	Refs
Tree queries with arbitr	arv S		

• So far, we have said that $\mathcal{S} \subset E$ so each query $p(x_S)$, $S = (i, j) \in E$.



- So far, we have said that $\mathcal{S} \subset E$ so each query $p(x_S)$, $S = (i, j) \in E$.
- Ex: 4-node Markov chain: $G = x_1 x_2 x_3 x_4$ with $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, \mathcal{M}^{(f)})$, goal is $p(x_1, x_2, x_3)$



- So far, we have said that $\mathcal{S} \subset E$ so each query $p(x_S)$, $S = (i, j) \in E$.
- Ex: 4-node Markov chain: $G = x_1 x_2 x_3 x_4$ with $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, \mathcal{M}^{(f)})$, goal is $p(x_1, x_2, x_3)$
- We eliminate x_4 in

$$\sum_{x_4} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \psi_{3,4}(x_3, x_4)$$
(3.10)



- So far, we have said that $\mathcal{S} \subset E$ so each query $p(x_S)$, $S = (i, j) \in E$.
- Ex: 4-node Markov chain: $G = x_1 x_2 x_3 x_4$ with $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, \mathcal{M}^{(f)})$, goal is $p(x_1, x_2, x_3)$
- We eliminate x_4 in

$$\sum_{x_4} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \psi_{3,4}(x_3, x_4)$$
(3.10)

• $O(r^2)$ computation.



- So far, we have said that $S \subset E$ so each query $p(x_S)$, $S = (i, j) \in E$.
- Ex: 4-node Markov chain: $G = x_1 x_2 x_3 x_4$ with $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, \mathcal{M}^{(f)})$, goal is $p(x_1, x_2, x_3)$
- We eliminate x_4 in

$$\sum_{x_4} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \psi_{3,4}(x_3, x_4)$$
(3.10)

- $O(r^2)$ computation.
- General property: if S is a sub-tree in G = (V, E) then can do the trick above, resulting $p(x_S)$ can be obtained by "rooting" the tree at the subtree S, and still have $O(r^2)$ computation.

Tree queries with arbitrary S



- Above, $S = \{1, 2, 3, 4, 6\}$ which induces a sub-tree in G, so all messages sent towards nearest node inside of S.
- Once we have $p(x_S)$ we have efficient representation for it, using only r^2 tables.

Refs



Tree queries with arbitrary S

- what if S is not a sub-tree?
- Ex: 3-node Markov chain: $G = x_1 x_2 x_3$ with $p(x_1, x_2, x_3) \in \mathcal{F}(G, \mathcal{M}^{(f)})$, goal is $p(x_1, x_3)$
- Only choice is to eliminate x_2 in

$$\sum_{x_2} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \tag{3.11}$$

- $O(r^3)$ computation.
- fill-in edge has occurred (might as well have started with larger family with additional edge $x_1 x_3$.



Refs

Tree queries with arbitrary S

- We eliminate $x_{V \setminus S}$, which might introduce edges.
- Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$ be an ordering of the nodes. Also $\sigma^{-1}(v)$ for $v \in V(G)$ gives number that node v is eliminated by order σ . We have following theorem

Theorem 3.4.1

Let G = (V, E) be an undirected graph with a given elimination ordering σ that maps G to G' = (V, E') where $E' = E \cup F_{\sigma}$, and where F_{σ} are the fill-in edges added during elimination with order σ . Then $(v, w) \in E'$ is an edge in G' iff there is a path in G with endpoints v and w, and where any nodes on the path other than v and w are eliminated before v and w in order σ . I.e., if there is a path $(v = v_1, v_2, \ldots, v_{k+1} = w)$ in G such that

$$\sigma^{-1}(v_i) < \min(\sigma^{-1}(v), \sigma^{-1}(w)), \text{ for } 2 \le i \le k$$
 (3.12)





•

Tree queries with arbitrary S

Proof.

First part: Induction on $\ell = \min(\sigma^{-1}(v), \sigma^{-1}(w))$ that given any $(v, w) \in G'$ the equation holds. If $\ell = 1$ then $(v, w) \in E$ and $(v, w) \in E'$. Suppose holds for $\ell \leq \ell_0$ and consider $\ell = \ell_0 + 1$. If $(v, w) \in E$ then the equation holds. Otherwise, $(v, w) \in F_{\sigma}$, and by definition, we have an $x \in V$ with $\sigma^{-1}(x) \leq \min(\sigma^{-1}(v), \sigma^{-1}(w))$ and x - v, x - w in G'. Induction hypothesis implies existence of x, v and x, w paths in G satisfying the equation, combining these chains gives the required v, w path.

Converse: Induction on k, length of path. If k = 1 clearly $(v, w) \in E'$. Suppose holds for $k \le k_0$ and consider $k = k_0 + 1$. From path $(v = v_1, v_2, \ldots, v_{k+1} = w)$, choose $x = v_i$ where $\sigma^{-1}(v_i) = \max \{\sigma^{-1}(v_j) | 2 \le j \le k\}$. Induction hypothesis implies v - x and x - w in G'. Therefore v - w in G'.



• So if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then v, w will be connected once elimination has run.



- So if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then v, w will be connected once elimination has run.
- Worst case, S can become a clique, and computation will be exponential in |S|.





- So if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then v, w will be connected once elimination has run.
- Worst case, S can become a clique, and computation will be exponential in |S|.
- Best case, for any $v, w \in S$ there is no path between them outside of S this is the case where S induces a tree.



- So if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then v, w will be connected once elimination has run.
- Worst case, S can become a clique, and computation will be exponential in |S|.
- Best case, for any $v, w \in S$ there is no path between them outside of S this is the case where S induces a tree.
- typical case: somewhere in between, depends on the query.

- So if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then v, w will be connected once elimination has run.
- Worst case, S can become a clique, and computation will be exponential in |S|.
- Best case, for any $v, w \in S$ there is no path between them outside of S this is the case where S induces a tree.
- typical case: somewhere in between, depends on the query.
- ullet Bad news for scientists who want to do exact inference! $\ensuremath{\textcircled{\sc b}}$

Arbitrary Queries

Perfect elimination orders

Definition 3.5.1 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ , there are zero fill edges in the resulting reconstituted graph.

Arbitrary Queries

Definition 3.5.1 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ , there are zero fill edges in the resulting reconstituted graph.

• For a tree, there is always a perfect elimination order.

Arbitrary Queries

Refs

Perfect elimination orders

Definition 3.5.1 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ , there are zero fill edges in the resulting reconstituted graph.

• For a tree, there is always a perfect elimination order. Why?

Arbitrary Queries

Definition 3.5.1 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ , there are zero fill edges in the resulting reconstituted graph.

• For a tree, there is always a perfect elimination order. Why? Because there are always leaf nodes available.

Arbitrary Queries

Perfect elimination orders

Definition 3.5.1 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ , there are zero fill edges in the resulting reconstituted graph.

- For a tree, there is always a perfect elimination order. Why? Because there are always leaf nodes available.
- For arbitrary graphs, must there be a perfect elimination order?



- Any node will produce a fill in. $O(r^3)$ query seems unavoidable.
- $\bullet\,$ There are no leaf nodes, and no node v such that $\delta(v)$ induces a clique in G.
- Might as well have started with graph on the right, no penalty for eliminating x_1 first (but there is for x_2).
- Consider message passing on this graph, could oscillate.

	Arbitrary Queries	Non-Tree Graphs	Refs
Non trop graphs			

- 4-cycle states both $X_1 \perp \!\!\!\perp X_4 | \{X_2, X_3\}$ and $X_2 \perp \!\!\!\perp X_3 | \{X_1, X_4\}$, while right graph only requires first property.
- extra independence properties of the 4-cycle does not help us computationally.

Iree Inference	Arbitrary Quei	ries Non-Tree Graphs	
Non-tree grap	ohs		
circh	x_1 x_2 x_3 x_4 x_5 x_6 x_7	x_1 x_2 x_3 x_4 x_5 x_5	

• Left: Eliminating x_4 is bad, but other nodes are better.

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			



- Left: Eliminating x_4 is bad, but other nodes are better.
- Left: No node results in zero fill in! \hfill

 \mathbf{O}

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			



- Left: Eliminating x_4 is bad, but other nodes are better.
- Left: No node results in zero fill in! ③
- Right: Is there a perfect elimination order?

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			



- Left: Eliminating x_4 is bad, but other nodes are better.
- Left: No node results in zero fill in! \hfill
- Right: Is there a perfect elimination order?
- For exact inference and some queries, inevitable that we work with a larger family since $\mathcal{F}((V, E), \mathcal{M}^{(f)}) \subset \mathcal{F}((V, E \cup F), \mathcal{M}^{(f)})$.

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			



- Left: Eliminating x_4 is bad, but other nodes are better.
- Left: No node results in zero fill in! \hfill
- Right: Is there a perfect elimination order?
- For exact inference and some queries, inevitable that we work with a larger family since $\mathcal{F}((V, E), \mathcal{M}^{(f)}) \subset \mathcal{F}((V, E \cup F), \mathcal{M}^{(f)})$.
- Appears to be computational equivalence classes of families of models.
Lemma 3.5.2

The reconstituted graph on which elimination has been run is the family on which we are running inference. If fill-in is caused by elimination, inference is solved on a family larger than that specified by the original graph, and we might as well have started with that family to begin with. If an elimination order produces no fill-in, we are solving the inference query optimally.

Refs

Lemma 3.5.2

The reconstituted graph on which elimination has been run is the family on which we are running inference. If fill-in is caused by elimination, inference is solved on a family larger than that specified by the original graph, and we might as well have started with that family to begin with. If an elimination order produces no fill-in, we are solving the inference query optimally.

• Also, ordering σ matters. Using σ a second time results in a perfect elimination order.

Refs

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			

Lemma 3.5.3

When elimination is run for a second time on the reconstituted graph with the same order, the set of neighbors v at the time v is eliminated is the same in both the original and in the reconstituted graph.

Proof.

Any neighbor of v in the reconstituted graph must be either an original-graph edge, or it must be due to a fill-in edge between v and some other node that is not an original graph neighbor. All of the fill-in neighbors must be due to elimination of nodes before v since after v is eliminated no new neighbors can be added to v. But the point at which vis eliminated at the original graph and the point at which it v is eliminated in the reconstituted graph, the same previous set of nodes have been eliminated, so any neighbors of v in the reconstituted graph will have been already added to the original graph when v is eliminated in the original graph.

Prof. Jeff Bilmes

EE512a/Fall 2014/Graphical Models - Lecture 3 - Oct 6th, 2014

	Arbitrary Queries	Non-Tree Graphs	Refs
New twee was dee			

Non-tree graphs

Lemma 3.5.4

Given an elimination order, the computational complexity of the elimination process is $O(r^{k+1})$ where k is the largest set of neighbors encountered during elimination. This is the size of the largest clique in the reconstituted graph.

Proof.

First, when we eliminate σ_i in G_{i-1} , eliminating variable v when it is in the context of its current neighbors will cost $O(r^{\ell})$ where $\ell = |\delta(v) + 1|$ — thus, the overall cost will be $O(r^{k+1})$.

Next, we show that largest clique in the reconstituted graph is equal to the complexity. Consider the reconstituted graph, and assume its largest clique is of size k + 1. When we re-run elimination on this graph, there will be no fill in.

	Arbitrary Queries	Non-Tree Graphs	Refs
Proof cont			

. . . continued.

However, the cost of the elimination step upon reaching the first vertex v of the clique of size k + 1 will be $O(r^{k+1})$ since k of the variables of the clique will be neighbors of v, but no other nodes will be neighbors since it is a perfect elimination order in the reconstituted graph. This will be the same cost as what was incurred during the initial elimination procedure since v has the same set of neighbors. Therefore, the largest clique in the reconstituted graph is the complexity of doing elimination.

• This means that any perfect elimination ordering on a perfect-elimination graph will have complexity exponential in the size of the largest clique in that graph.

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			

 \mathbf{O}

• $G' = (V, E \cup F_{\sigma})$ always has at least one perfect elimination order

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			

- $G' = (V, E \cup F_\sigma)$ always has at least one perfect elimination order
- When we run elimination algorithm, we will always end up with such a graph inevitable

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			

- $G' = (V, E \cup F_\sigma)$ always has at least one perfect elimination order
- When we run elimination algorithm, we will always end up with such a graph inevitable
- Perhaps we should deal only with such graphs?

	Arbitrary Queries	Non-Tree Graphs	Refs
Non-tree graphs			

- $G' = (V, E \cup F_\sigma)$ always has at least one perfect elimination order
- When we run elimination algorithm, we will always end up with such a graph inevitable
- Perhaps we should deal only with such graphs?
- Is finding the order that minimizes fill-in optimal? (HW problem)

	Arbitrary Queries	Non-Tree Graphs	Refs
		1111111	
Non-tree graphs			

- $G' = (V, E \cup F_\sigma)$ always has at least one perfect elimination order
- When we run elimination algorithm, we will always end up with such a graph inevitable
- Perhaps we should deal only with such graphs?
- Is finding the order that minimizes fill-in optimal? (HW problem)
- We can characterize the complexity of a given elimination order.

Tree Inference

Arbitrary Queries

Non-Tree Graphs

Refs

Sources for Today's Lecture

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