

EE512 – Graphical Models – Fall 2009

Prof. Jeff Bilmes

University of Washington, Seattle
Department of Electrical Engineering
Fall Quarter, 2009

<http://ssli.ee.washington.edu/~bilmes/ee512fa09>

Lecture 9 - October 29nd, 2009

Last updated \$Id: lec9.tex,v 1.1 2009/11/03 09:29:13 bilmes Exp \$

Class Road Map

- L1 (10/1): intro, cond. indep., ex. GMs
- L2 (10/6): GMs and MRFs
- L3 (10/8): MRFs, mobius, FGs
- L4 (10/13): Sem BNs
- L5 (10/15): Sem BNs 2.
- L6 (10/20): Evidence
- L7 (10/22): Inf. Trees 1
- L8 (10/27): Inf. Trees 2
- L9 (10/29): Inf. Trees 3
- L10 (11/03):
- L11 (11/05):
- L12 (11/10):
- L13 (11/12):
- L14 (11/17):
- L15 (11/19):
- L16 (11/24):
- L17 (12/01):
- L18 (12/03):
- L19 (12/08): (video lecture)
- L20 (12/10): (video lecture)
- L21 (12/18): Friday, 2:30-4:20pm, final presentations

Readings

- Read “tree_inference.pdf”
- Read “evidence.pdf”
- Read “dgms.pdf”
- Read “ugms.pdf” .
- Read “intro.pdf” .
- Optionally (but encouraged): read chapters 1 through 5 in Jordan text.

Homework

- Reminder: Do “hw1.pdf” on the web, due Friday, 10/30 5:00pm by PDF email.
- Note, problem 5 had a bug. This was fixed last Monday night (around 8:00pm), please re-upload recent version if you have not done so.
- Note other bug, problem 11 and 12 are identical, do only problem 11.

Review

- On trees, equation elimination same as graph elimination.
- Elim leaf nodes leaves a tree ensures we have sub-tree
- elimination of leaf nodes seen as messages on graphs
- Message passing protocol: ensures computation is correct and optimal

Generic form of message

$$\mu_{i \rightarrow j}(x_j) = \sum_{x_i} \left(\psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \rightarrow i}(x_i) \right) \quad (1)$$

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 .

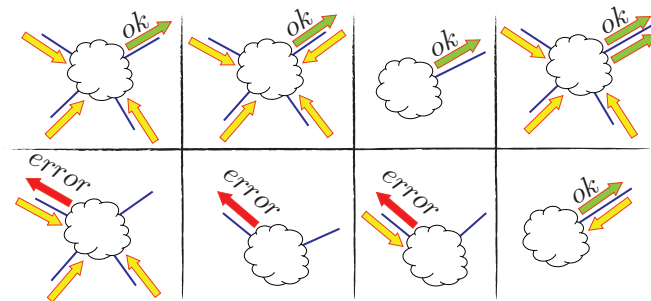
Message passing protocol

Definition

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 $i \rightarrow j$ indicates a message from i to j , then the protocol may be written as $i \rightarrow j$ only when $\forall k \in \delta(i) \setminus \{j\}, k \rightarrow i$.
- 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.
- we're also guaranteed that computation is optimal.

Message passing protocol examples

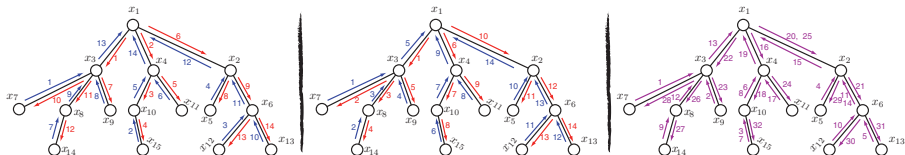


- Examples of valid and invalid messages. Yellow arrows correspond to incoming messages. Green outgoing arrows correspond to messages that obey MPP, and red outgoing arrows are messages that disobey MPP.

Further Review

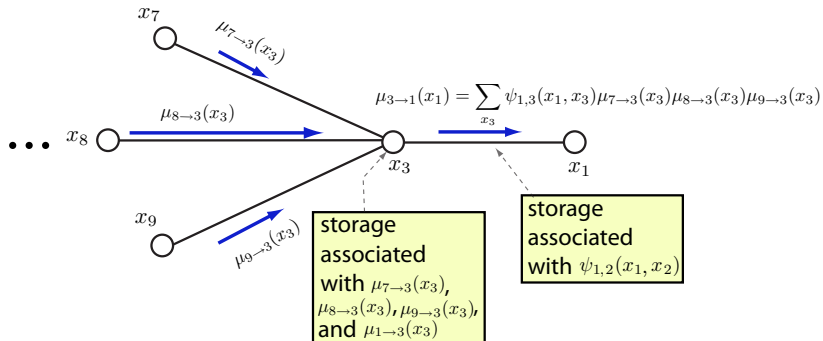
- Multiple queries and redundancy: the more queries the better
- Collect/Distribute evidence: an algorithm to compute all edge queries

Collect/Distribute Evidence



- Examples of the messages to compute all edge queries. The blue arrows indicate messages towards the root (node 1) and the red arrow indicate messages away from the root. The numbers next to each arrow indicate the order of the message. Do both the left and middle figures abide by MPP? Do they correspond to collect/distribute evidence? What about the right figure?

Associated storage with message propagation



$$\mu_{i \rightarrow j}(x_j) = \sum_{x_i} \left(\psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \rightarrow i}(x_i) \right) \quad (2)$$

- for each edge (i, j) , storage associated with edge itself, $\psi_{i,j}(x_i, x_j)$, and all incoming messages $\mu_{k \rightarrow i}(x_i)$ for all $k \in \delta(i) \setminus \{j\}$. 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:

$$\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j) \mu_{k \rightarrow i}(x_i) \quad (3)$$

- Final factor, after incorporating all messages, would be

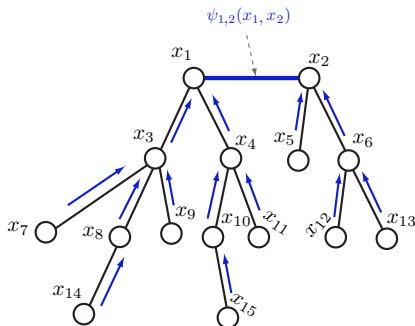
$$\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \rightarrow i}(x_i) \quad (4)$$

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

$$\mu_{i \rightarrow j}(x_j) = \sum_{x_i} \psi'_{i,j}(x_i, x_j). \quad (5)$$

Alternative propagation styles

- Never require storage at only node i
- 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

- but problem when computing all edge marginals: Ultimately, messages start arriving at x_j via nodes $k \in \delta(j) \setminus \{i\}$.
- Updated edge table is no longer valid for sending message back to i and $\delta(i) \setminus \{j\}$.
- 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 1: First phase message update $\mu_{i \rightarrow j}(x_j)$

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

$\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j) / \mu_{i \rightarrow j}(x_j) ;$ /* 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 ;

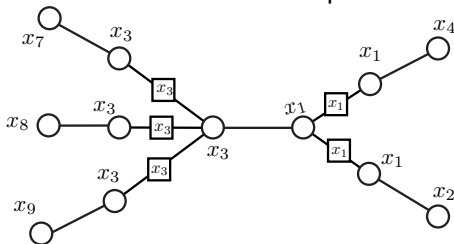
$\psi'_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \mu_{i \rightarrow j}(x_j) ;$ /* table update - multiply in incoming message */

Alternative propagation styles

- By dividing out $\mu_{i \rightarrow j}(x_j)$ from $\psi_{i,j}(x_i, x_j)$, we are sure that the $\mu_{i \rightarrow j}(x_j)$ will not be double counted once it is multiplied back in from the message coming back from k in $\mu_{k \rightarrow j}(x_j)$.
- 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 $\phi_i^n(x_i)$ and $\phi_i^d(x_i)$ at each separator node, which keeps track of incoming messages.
- At start, initialize both tables to unity $\phi_i^n(x_i) = 1$, $\phi_i^d(x_i) = 1 \forall x_i \in D_{X_i}$.
- we follow the collect/distribute evidence schedule for sending messages

Alternative propagation styles

Algorithm 2: collect evidence message update $\mu_{i \rightarrow j}(x_j)$

$\phi_{i,j,k}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* 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. */

- 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 3: distribute evidence message update $\mu_{i \rightarrow j}(x_j)$

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

$\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \frac{\phi'_{i,j,k}(x_j)}{\phi_{i,j,k}(x_j)}$; /* 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.
- uniform message style, but once again more storage, even more than originally

Alternative propagation styles

Algorithm 4: asymmetric distribute evidence message update $\mu_{i \rightarrow j}(x_j)$

foreach $x_j \in D_{X_j}$ **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. */

- Alternate distribute evidence phase message
- lost symmetry but recovered some storage

Three propagation styles

- The three different message styles we have described are called, respectively, the Shenoy-Shafer, the Lauritzen-Spiegelhalter, 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.

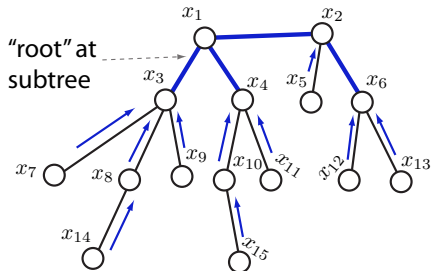
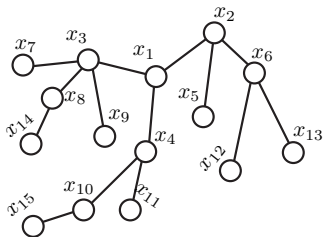
Tree queries with arbitrary S

- 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 $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, R^{(f)})$, goal is $p(x_1, x_2, x_3)$
- We eliminate x_4 in

$$\sum_{x_4} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \psi_{3,4}(x_3, x_4) \quad (6)$$

- $O(r^2)$ computation.
- General issue, 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.

Tree queries with arbitrary S

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

$$\sum_{x_2} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \quad (7)$$

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

Tree queries with arbitrary S

- We eliminate $x_{V \setminus S}$, which might introduce edges.
- Recall $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$ is 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

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 $(v = v_1, v_2, \dots, v_{k+1} = w)$ in G such that

$$\sigma^{-1}(v_i) < \min(\sigma^{-1}(v), \sigma^{-1}(w)), \text{ for } 2 \leq i \leq k \quad (8)$$

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 $k \leq k_0$ and consider $k = k_0 + 1$. From path $(v = v_1, v_2, \dots, v_{k+1} = w)$, choose $x = v_i$ where $\sigma^{-1}(v_i) = \max\{\sigma^{-1}(v_j) \mid 2 \leq j \leq k\}$. Induction hypothesis implies $v-x$ and $x-w$ in G' . Therefore $v-w$ in G' . \square

Tree queries with arbitrary 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.
- worse 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.
- bad news for scientists who want to do exact inference!

Bayesian network specific queries

- if p really comes from a moralized BN, certain independence properties true of p are not available when in $\mathcal{F}(m(G), R^{(f)})$.
- Suppose S consists only of unmarried parents so that for all $u, w \in S$ $X_u \perp\!\!\!\perp X_w$. In such case:

$$p(x_S) = \prod_{v \in S} p(x_v) \quad (9)$$

and the computation of the marginal is trivial (no computation required).

- This potential benefit is lost in the MRF world.
- Again, the scientists loose.
- Note: for machine learning, we generally want $p(x_C)$ for all $C \in \mathcal{C}(G)$, and in the tree case $\mathcal{C}(G)$ consists of all edges.

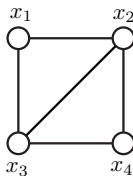
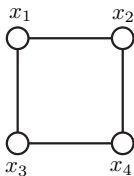
Perfect elimination orders

Definition (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.

Non-tree graphs

- If not tree, might not be a fill-in-free elimination order.
Example:

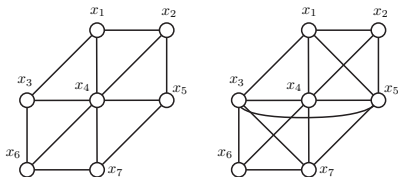


- Any node will produce a fill in. $O(r^3)$ query seems unavoidable.
- There are no leaf nodes, and no node v such that $\delta(v)$ induces a clique in G .
- Might as well have started with graph on the right, no penalty for eliminating x_1 first (but there is for x_2).
- Consider message passing on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

Non-tree graphs

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

Non-tree graphs



- Eliminating x_4 is bad. why?
- Eliminating other nodes are better.
- Exists no node with zero fill in.
- Inevitable that we work with a larger family since $\mathcal{F}((V, E), R^{(f)}) \subset \mathcal{F}((V, E \cup F), R^{(f)})$.
- there seems to be computational equivalence classes of families of models.
- Obvious limits: we don't want to move up to a grand clique over ground set.

Non-tree graphs

Lemma

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.

Non-tree graphs

Lemma

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 v is 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

Non-tree graphs

Lemma

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. □

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 recon graph is the complexity of doing elimination. \square

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