EE512A – Advanced Inference in Graphical Models

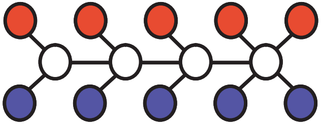
— Fall Quarter, Lecture 9 —

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 27th, 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.
- Wednesday, no in person lecture. Will be posted on youtube during a makeup class sometime later this quarter (i.e., next time we meet is one week from today).

Class Road Map - EE512a

- L1 (9/29): Introduction, Families, Semantics
- L2 (10/1): MRFs, elimination, Inference on Trees
- L3 (10/6): Tree inference, message passing, more general queries, non-tree)
- L4 (10/8): Non-trees, perfect elimination, triangulated graphs
- L5 (10/13): triangulated graphs, k-trees, the triangulation process/heuristics
- L6 (10/15): multiple queries, decomposable models, junction trees
- L7 (10/20): junction trees, begin intersection graphs
- L8 (10/22): intersection graphs, inference on junction trees
- L9 (10/27): inference on junction trees, semirings, conditioning, hardness
- 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.

Review

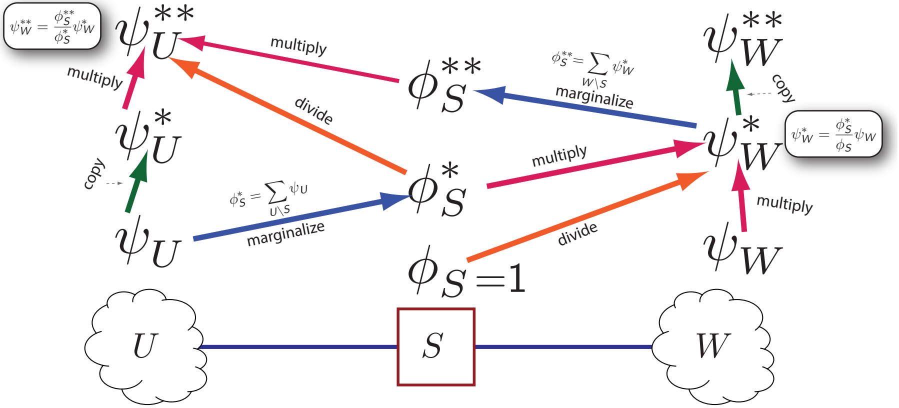
- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.

Review

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.

Forward/Backward Messages Along Cluster Tree Edge

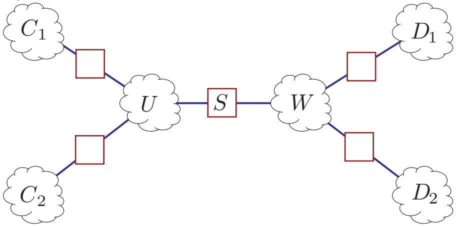
Summarizing, forward and backwards messages proceed as follows:



Recall: $S = U \cap W$, and we initialize ψ_U and ψ_W with factors that are contained in U or W.

Less simple example: general tree

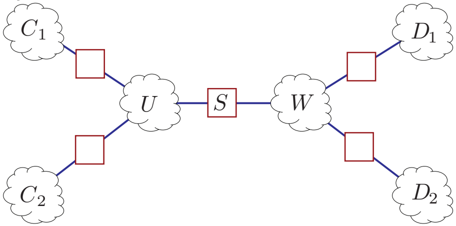
How to ensure any local consistency we achieved not ruined by later message passing steps?



E.g. once we send message $U \to W$ and then $W \to U$, we know W and U are consistent. If we next send messages $W \to D_1$ and $D_1 \to W$, then W & D_1 are consistent, but U & W might no longer be consistent.

Less simple example: general tree

How to ensure any local consistency we achieved not ruined by later message passing steps?



E.g. once we send message $U \to W$ and then $W \to U$, we know W and U are consistent. If we next send messages $W \to D_1$ and $D_1 \to W$, then W & D_1 are consistent, but U & W might no longer be consistent.

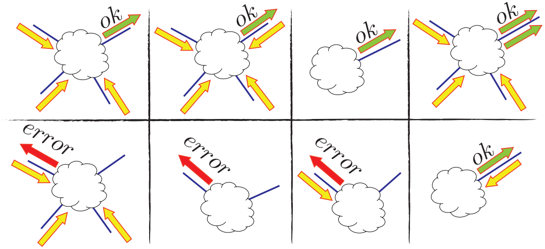
Basic problem, future messages might "mess up" achieved local marginal consistency.

Ensuring consistency over all marginals

We use same scheme we saw for 1-trees. I.e., recall from earlier lectures:

Definition 9.3.1 (Message passing protocol)

A clique can send a message to a neighboring cluster in a JT **only** after it has received messages from all of its *other* neighbors.



We already know collect/distribute evidence is a simple algorithm that obeys MPP (designate root, and do bottom up messages and then top-down messages). Does this achieve consistency?

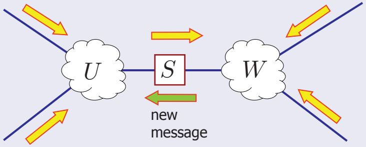
MPP renders cliques locally consistent

Theorem 9.3.2

The message passing protocol renders the cliques locally consistent between all pairs of connected cliques in the tree.

Proof.

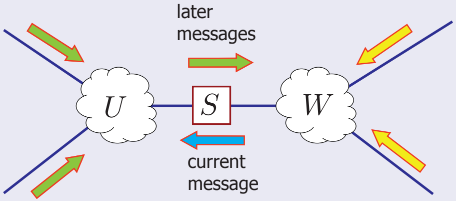
Suppose W has received a message from all other neighbors, and is sending a message to U. There are two possible cases: Case A: U already sent a message to W before, so U already received message from all other neighbors, & message renders the two consistent since neither receives any more messages.



MPP renders cliques locally consistent

proof continued.

Case B: U has not yet sent a message to W, so W sends to U & waits. Later, U will have received message from all other neighbors & will send message back to W, but this will contain appropriate update from W.



□

Another way we can see it: If we abide by MPP, the potential functions will just be scaled, and thanks to commutativity of multiplication, we'll be back at the same case that we were before with two cliques.

MPP renders cliques locally consistent

- For a general Tree, when we send messages abiding by MPP, we get:

MPP renders cliques locally consistent

- For a general Tree, when we send messages abiding by MPP, we get:

Theorem 9.3.3

Sending all messages along a cluster tree following message passing protocol renders the cliques locally consistent between all pairs of connected cliques in the tree of clusters.

MPP renders cliques locally consistent

- For a general Tree, when we send messages abiding by MPP, we get:

Theorem 9.3.3

Sending all messages along a cluster tree following message passing protocol renders the cliques locally consistent between all pairs of connected cliques in the tree of clusters.

- Note, we need only that it is a cluster tree. Result holds even if r.i.p. not satisfied!

MPP renders cliques locally consistent

- For a general Tree, when we send messages abiding by MPP, we get:

Theorem 9.3.3

Sending all messages along a cluster tree following message passing protocol renders the cliques locally consistent between all pairs of connected cliques in the tree of clusters.

- Note, we need only that it is a cluster tree. Result holds even if r.i.p. not satisfied!
- But we want more than this, we want to ensure that potentials over any two clusters, with common variables, even if not directly connected, agree on their common variables.

Local implies global consistency

Theorem 9.3.4

In any junction tree of clusters, any configuration of cluster functions that are locally (neighbor) consistent will be globally consistent. I.e., for any clusters pair C_1, C_2 with $C_1 \cap C_2 \neq \emptyset$ we have:

$$\sum_{x_{C_1 \setminus C_2}} \psi_{C_1}(x_{C_1}) = \psi_{C_1}(x_{C_1 \cap C_2}) = \psi_{C_2}(x_{C_1 \cap C_2}) = \sum_{x_{C_2 \setminus C_1}} \psi_{C_2}(x_{C_2}) \tag{9.1}$$

for all values $x_{C_1 \cap C_2}$.

Local implies global consistency

Theorem 9.3.4

In any junction tree of clusters, any configuration of cluster functions that are locally (neighbor) consistent will be globally consistent. I.e., for any clusters pair C_1, C_2 with $C_1 \cap C_2 \neq \emptyset$ we have:

$$\sum_{x_{C_1 \setminus C_2}} \psi_{C_1}(x_{C_1}) = \psi_{C_1}(x_{C_1 \cap C_2}) = \psi_{C_2}(x_{C_1 \cap C_2}) = \sum_{x_{C_2 \setminus C_1}} \psi_{C_2}(x_{C_2}) \tag{9.1}$$

for all values $x_{C_1 \cap C_2}$.

Proof.

Local consistency implies that for neighboring C_1, C_2, the above equality holds. For non-neighboring C_1, C_2, cluster intersection property (r.i.p.) ensures that intersection $C_1 \cap C_2$ exists along unique path between C_1 and C_2. Each edge along that path is locally consistent. By transitivity, each distance-2 pair is consistent. Repeating this argument for any path length gives the result. □

Consistency gives Marginals

Theorem 9.3.5

Given junction tree of clusters $\mathcal{C}$ and separators $\mathcal{S}$, and given above initialization, after all messages are sent and obey MPP (what we call "message passing", or just MP), cluster and separator potentials will reach the marginal state, i.e.,:

$$\psi_C(x_C) = p(x_C) \text{ and } \phi_S(x_S) = p(x_S) \tag{9.2}$$

Consistency gives Marginals

Theorem 9.3.5

Given junction tree of clusters $\mathcal{C}$ and separators $\mathcal{S}$, and given above initialization, after all messages are sent and obey MPP (what we call "message passing", or just MP), cluster and separator potentials will reach the marginal state, i.e.,:

$$\psi_C(x_C) = p(x_C) \text{ and } \phi_S(x_S) = p(x_S) \tag{9.2}$$

Proof.

Separators are marginalizations of clusters, so ensuring clusters are marginals is sufficient for separators as marginals.

Consistency gives Marginals

Theorem 9.3.5

Given junction tree of clusters $\mathcal{C}$ and separators $\mathcal{S}$, and given above initialization, after all messages are sent and obey MPP (what we call "message passing", or just MP), cluster and separator potentials will reach the marginal state, i.e.,:

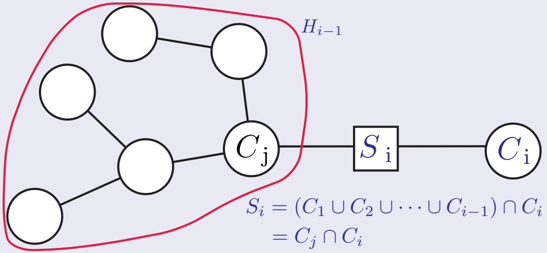
$$\psi_C(x_C) = p(x_C) \text{ and } \phi_S(x_S) = p(x_S) \tag{9.2}$$

Proof.

Separators are marginalizations of clusters, so ensuring clusters are marginals is sufficient for separators as marginals.
Induction: base case: One cluster is a marginal. Two clusters reach marginals (we verified above).
Assume true for $i-1$ clusters marginals, and show for i. Given JT with clusters $C_1, \ldots, C_{i-1}$ and add new cluster C_i connecting to C_j and obeying r.i.p. We have separator $S_i = C_i \cap C_j$.

Consistency gives Marginals

... proof continued.



In current case, we may assume $V = H_i$ and $p(x) = p(x_V)$, so that

$$p(x_V) = p(x_{C_i \setminus S_i}, x_{S_i}, x_{V \setminus C_i}) = p(x_{C_i \setminus S_i} | x_{S_i}) p(x_{S_i \cup (V \setminus C_i)}) \tag{9.3}$$

$$= p(x_{C_i \setminus S_i} | x_{S_i}) p(x_{H_{i-1}}) \tag{9.4}$$

due to conditional independence property of sepator S

$$X_{C_i \setminus S_i} \perp\!\!\!\perp X_{V \setminus C_i} | X_S \tag{9.5}$$

Consistency gives Marginals

... proof continued.

We have the representation of $p(x_V) = p(x_{H_i})$ as

$$p(x_V) = \frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d(S)-1}} \tag{9.6}$$

When we run message passsing (MP) on a junction tree with i nodes coresponding to the above, we have both local and global consistency. Hence, the separator S_i is a marginal of the form:

$$\phi_{S_i}(x_{S_i}) = \sum_{x_{C_j \setminus S_i}} \psi_{C_j}(x_{C_j}) \tag{9.7}$$

□

Consistency gives Marginals

... proof continued.

Assume MP has been run on a JT with i nodes. Then, we have

$$p(x_{S_i \cup (V \setminus C_i)}) = \sum_{x_{C_i \setminus S_i}} p(x_V) = \sum_{x_{C_i \setminus S_i}} \frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d(S)-1}} \tag{9.8}$$

$$= \sum_{x_{C_i \setminus S_i}} \frac{\psi_{C_i}(x_{C_i}) \prod_{C \neq C_i} \psi_C(x_C)}{\phi_{S_i}(x_{S_i}) \prod_{S \in \mathcal{S}} \phi_S(x_S)^{d'(S)-1}} \tag{9.9}$$

$$= \frac{\sum_{x_{C_i \setminus S_i}} \psi_{C_i}(x_{C_i})}{\phi_{S_i}(x_{S_i})} \frac{\prod_{C \neq C_i} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d'(S)-1}} \tag{9.10}$$

$$= \frac{\prod_{C \neq C_i} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d'(S)-1}} \tag{9.11}$$

since $\sum_{x_{C_i \setminus S_i}} \psi_{C_i}(x_{C_i}) = \phi_{S_i}(x_{S_i})$ and since the only cluster containing $C_i \setminus S_i$ is C_i. $d'(S) = d(S)$ except at S_i where one less.

Consistency gives Marginals

... proof continued.

MP on JT with i nodes is a valid MP on a JT with $i-1$ nodes. But with only $i-1$ cliques, after message passing is performed, JT will have cluster functions as marginals (by induction), which gives us marginals $\psi_{C_j}(x_{C_j}) = p(x_{C_j})$ for $j < i$. In other words, we have that:

$$p(x_{S_i \cup (V \setminus C_i)})) = \frac{\prod_{C \neq C_i} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d'(S)-1}} = \frac{\prod_{C \neq C_i} p_C(x_C)}{\prod_{S \in \mathcal{S}} p_S(x_S)^{d'(S)-1}} \tag{9.12}$$

We need to show that $\psi_{C_i}(x_{C_i})$ is also a valid marginal.

$$p(x_{C_i \setminus S_i} | x_{S_i}) = \frac{p(x_V)}{p(x_{S_i \cup (V \setminus C_i)})} = \frac{\frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d(S)-1}}}{\frac{\prod_{C \neq C_i} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)^{d'(S)-1}}}, \tag{9.13}$$

where the first equality follows from Equation (9.4).

Consistency gives Marginals

... proof continued.

which yields

$$p(x_{C_i \setminus S_i} | x_{S_i}) = \frac{\psi_{C_i}(x_{C_i})}{\phi_{S_i}(x_{S_i})} = \frac{\psi_{C_i}(x_{C_i})}{p(x_{S_i})} \tag{9.14}$$

this then gives that:

$$\psi_{C_i}(x_{C_i}) = p(x_{C_i \setminus S_i} | x_{S_i}) p(x_{S_i}) = p(x_{C_i}) \tag{9.15}$$

a marginal as desired.

□

Redundant Messages

- Once all messages have been sent according to MPP, what would happen if we send more messages?
- 1-tree formulation:

$$\mu_{i\to j}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \tag{9.16}$$

- Junction-tree formulation: marginalize and rescale

$$\phi_S^{\text{new}} = \sum_{U\setminus S} \psi_U \text{ and then } \psi_W^{\text{new}} = \frac{\phi_S^{\text{new}}}{\phi_S^{\text{old}}} \psi_W \tag{9.17}$$

Redundant Messages

- Once all messages have been sent according to MPP, what would happen if we send more messages?
- 1-tree formulation:

$$\mu_{i\to j}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \tag{9.16}$$

- Junction-tree formulation: marginalize and rescale

$$\phi_S^{\text{new}} = \sum_{U\setminus S} \psi_U \text{ and then } \psi_W^{\text{new}} = \frac{\phi_S^{\text{new}}}{\phi_S^{\text{old}}}\psi_W \tag{9.17}$$

- In either case, extra messages would not change functions - they're redundant, joint "state" has "converged" since $\phi_S^{\text{new}} = \phi_S^{\text{old}}$.
- all messages could run in parallel, convergence achieved once we've done D parallel steps where D is tree diameter.

Distributive Law and Other Objects

- Only one property needed for this algorithm to work, namely distributive law $ab + ac = a(b + c)$ along with factorization.
- Distributive law allows sending sums inside of factors.
- Other objects have distribute law, and in general any set of objects that is a commutative semiring will work as well

Commutative Semirings

Definition 9.4.1

A *commutative semiring* is a set K with two binary operators "$+$" and "$\cdot$" having three axioms, for all $a, b, c \in K$.
S1: "$+$" is commutative $(a + b) = (b + a)$ and associative $(a + b) + c = a + (b + c)$, and $\exists$ additive identity called "0" such that $k + 0 = k$ for all $k \in K$. I.e., $(K, +)$ is a commutative monoid.
S2: "$\cdot$" is also associative, commutative, and $\exists$ multiplicative identity called "1" s.t. $k \cdot 1 = k$ for all $k \in K$ ($(K, \cdot)$ is also a comm. monoid).
S3: distributive law holds: $(a \cdot b) + (a \cdot c) = a(b + c)$ for all $a, b, c \in K$.

This, and factorization w.r.t. a graph G is all that is necessary for the above message passing algorithms to work. There are many commutative semirings.

Commutative Semirings

- Additive inverse need not exist. If additive inverse exists, then we get a commutative ring ("semi-ring" since we need not have additive inverse). Note, in algebra texts, a ring often doesn't require multiplicative identity, but we assume it exists here.

Commutative Semirings

- Additive inverse need not exist. If additive inverse exists, then we get a commutative ring (”semi-ring” since we need not have additive inverse). Note, in algebra texts, a ring often doesn’t require multiplicative identity, but we assume it exists here.
- Above definition does not mention $0 \cdot k = 0$, but this follows from above properties since $k \cdot k = k(k + 0) = k \cdot k + k \cdot 0$ so that $k0$ must also be an additive identity, meaning that $k \cdot 0 = 0$. This is useful with evidence witih delta functions, where the delta functions multiplies by zero anything that does not obide by the evidence value.

Commutative Semirings

- Additive inverse need not exist. If additive inverse exists, then we get a commutative ring ("semi-ring" since we need not have additive inverse). Note, in algebra texts, a ring often doesn't require multiplicative identity, but we assume it exists here.
- Above definition does not mention $0 \cdot k = 0$, but this follows from above properties since $k \cdot k = k(k + 0) = k \cdot k + k \cdot 0$ so that $k0$ must also be an additive identity, meaning that $k \cdot 0 = 0$. This is useful with evidence witih delta functions, where the delta functions multiplies by zero anything that does not obide by the evidence value.
- Same message passing protocol and message passing scheme on a junction tree will work to ensure that all clusters reach a state where they are the appropriate "marginals"

Commutative Semirings

- Additive inverse need not exist. If additive inverse exists, then we get a commutative ring ("semi-ring" since we need not have additive inverse). Note, in algebra texts, a ring often doesn't require multiplicative identity, but we assume it exists here.
- Above definition does not mention $0 \cdot k = 0$, but this follows from above properties since $k \cdot k = k(k + 0) = k \cdot k + k \cdot 0$ so that $k0$ must also be an additive identity, meaning that $k \cdot 0 = 0$. This is useful with evidence witih delta functions, where the delta functions multiplies by zero anything that does not obide by the evidence value.
- Same message passing protocol and message passing scheme on a junction tree will work to ensure that all clusters reach a state where they are the appropriate "marginals"
- Marginals in this case dependent on ring.

Other Semi-Rings

Here, A denotes arbitrary commutative semiring, S is arbitrary finite set, Λ is arbitrary distributed lattice.

	K	"$(+,0)$"	"$(\cdot,1)$"	short name
1	A	$(+,0)$	$(\cdot,1)$	semiring
2	$A[x]$	$(+,0)$	$(\cdot,1)$	polynomial
3	$A[x,y,\dots]$	$(+,0)$	$(\cdot,1)$	polynomial
4	$[0,\infty)$	$(+,0)$	$(\cdot,1)$	sum-product
5	$(0,\infty]$	$(\min,\infty)$	$(\cdot,1)$	min-product
6	$[0,\infty)$	$(\max,0)$	$(\cdot,1)$	max-product
7	$[0,\infty)+$	$(\text{kmax},0)$	$(\cdot,1)$	k-max-product
8	$(-\infty,\infty]$	$(\min,\infty)$	$(+,0)$	min-sum
9	$[-\infty,\infty)$	$(\max,-\infty)$	$(+,0)$	max-sum
10	$\{0,1\}$	$(\text{OR},0)$	$(\text{AND},1)$	Boolean
11	2^S	$(\cup,\emptyset)$	$(\cap,S)$	Set
12	Λ	$(\vee,0)$	$(\wedge,1)$	Lattice
13	Λ	$(\wedge,1)$	$(\vee,0)$	Lattice

Example: Viterbi/MPE

- Most-probable explanation (e.g., Viterbi assignment) is just the max-product ring.
- Here, we wish to compute

$$\underset{x_{V \setminus E}}{\operatorname{argmax}}\, p(x_{V \setminus E}, \bar{x}_E) \tag{9.18}$$

- After message passing with the max-product ring on a junction tree, cluster functions will reach the "max-marginal" state, where we have:

$$\psi_C(x_C) = \max_{x_{V \setminus C}} p(x_C, x_{V \setminus C}) \tag{9.19}$$

- What about a "k-max" operation (i.e., finding the k highest scoring assignments to the variables?) How would we define the operators "$+$" and "$\cdot$"?

Recap

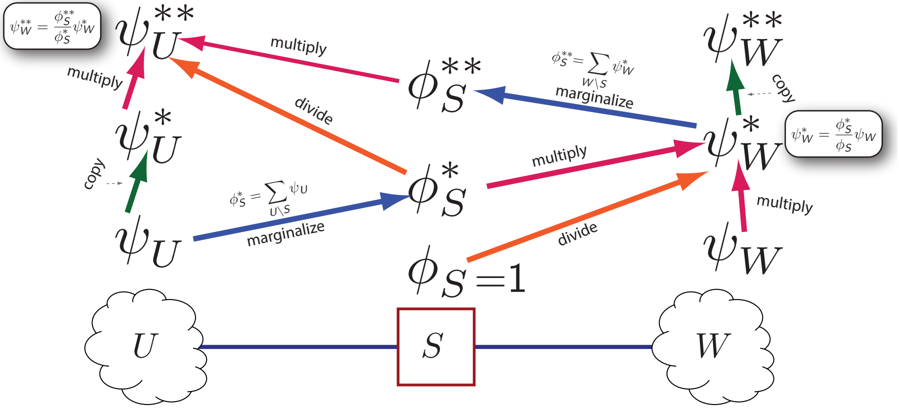
- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.

Forward/Backward Messages Along Cluster Tree Edge

Summarizing, forward and backwards messages proceed as follows:



Recall: $S = U \cap W$, and we initialize ψ_U and ψ_W with factors that are contained in U or W.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.
- Thm: MPP renders cliques locally consistent between pairs.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.
- Thm: MPP renders cliques locally consistent between pairs.
- In JT (r.i.p.) locally consistent ensures globally consistent.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.
- Thm: MPP renders cliques locally consistent between pairs.
- In JT (r.i.p.) locally consistent ensures globally consistent.
- In JT (r.i.p.), running MPP gives marginals.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.
- Thm: MPP renders cliques locally consistent between pairs.
- In JT (r.i.p.) locally consistent ensures globally consistent.
- In JT (r.i.p.), running MPP gives marginals.
- Commutative semiring - other algebraic objects can be used.

Recap

- Message passing on junction tree nodes, definition of messages, divide out old, multiply in new.
- Messages in both directions.
- For general tree, we have MPP like in 1-tree case.
- Suff condition: locally consistent.
- Thm: MPP renders cliques locally consistent between pairs.
- In JT (r.i.p.) locally consistent ensures globally consistent.
- In JT (r.i.p.), running MPP gives marginals.
- Commutative semiring - other algebraic objects can be used.
- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).
- Then cost is $O(Nr^{\omega+1})$, exponential in the tree-width.

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).
- Then cost is $O(Nr^{\omega+1})$, exponential in the tree-width.
- Finding the smallest tree-width JT cover is NP-complete as we have seen. Memory: Storing cluster tables, will also need $O(Nr^{\omega+1})$ memory to store a table with $\omega + 1$ variables.

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).
- Then cost is $O(Nr^{\omega+1})$, exponential in the tree-width.
- Finding the smallest tree-width JT cover is NP-complete as we have seen. Memory: Storing cluster tables, will also need $O(Nr^{\omega+1})$ memory to store a table with $\omega + 1$ variables.
- What if space complexity is most important ? We can compute $p(\bar{x}_E)$ in only $O(N)$ space as follows:

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).
- Then cost is $O(Nr^{\omega+1})$, exponential in the tree-width.
- Finding the smallest tree-width JT cover is NP-complete as we have seen. Memory: Storing cluster tables, will also need $O(Nr^{\omega+1})$ memory to store a table with $\omega + 1$ variables.
- What if space complexity is most important ? We can compute $p(\bar{x}_E)$ in only $O(N)$ space as follows:

```
1 $\alpha = 0$ ;                                   /* $\alpha$ is our accumulator */
2 forall the $x_{V \setminus E} \in \mathsf{D}_{X_{V \setminus E}}$ do
3     $\alpha \leftarrow \alpha + p(\bar{x}_E, x_{V \setminus E})$
```

Complexity of Inference

- Let ω be the tree-width of the junction tree (size of largest cluster minus 1).
- Then cost is $O(Nr^{\omega+1})$, exponential in the tree-width.
- Finding the smallest tree-width JT cover is NP-complete as we have seen. Memory: Storing cluster tables, will also need $O(Nr^{\omega+1})$ memory to store a table with $\omega+1$ variables.
- What if space complexity is most important ? We can compute $p(\bar{x}_E)$ in only $O(N)$ space as follows:

```
1 $\alpha = 0$ ;                                        /* $\alpha$ is our accumulator */
2 forall the $x_{V \setminus E} \in \mathsf{D}_{X_{V \setminus E}}$ do
3     $\alpha \leftarrow \alpha + p(\bar{x}_E, x_{V \setminus E})$
```

- But problem here is complexity is $O(r^N)$

Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.

Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

1. $O(Nr^{\omega+1})$ time and space (via JT),

Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

1. $O(Nr^{\omega+1})$ time and space (via JT),
2. $O(r^N)$ time and $O(N)$ space, or

Complexity of Inference

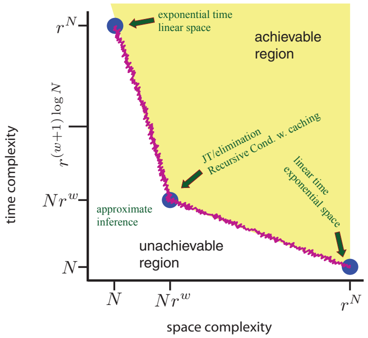
- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

1. $O(Nr^{\omega+1})$ time and space (via JT),
2. $O(r^N)$ time and $O(N)$ space, or
3. $O(N)$ time and $O(r^N)$ space,

Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

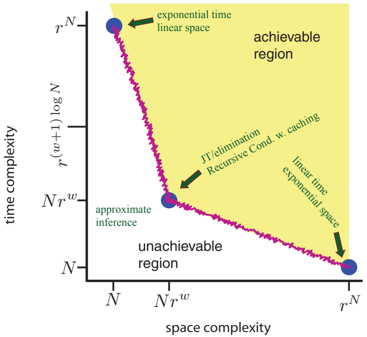
1. $O(Nr^{\omega+1})$ time and space (via JT),
2. $O(r^N)$ time and $O(N)$ space, or
3. $O(N)$ time and $O(r^N)$ space,



Complexity of Inference

- What if (online) time complexity was most precious, with ∞ space available? We can pre-compute $p(\bar{x}_E)$ for all possible values of $p(\bar{x}_E)$ and all sets E, and do a table lookup $O(N)$ time (not counting pre-compute time as that is amortized over many queries), and $O(r^N)$ space.
- So we can do inference either with:

1. $O(Nr^{\omega+1})$ time and space (via JT),
2. $O(r^N)$ time and $O(N)$ space, or
3. $O(N)$ time and $O(r^N)$ space,



- Are there any other useful/practical points in between?

Conditioning

- Other ways of doing inference in discrete networks, not based (as much) on graph theoretic properties — these methods are based on methods used in SAT solvers (DPLL) and CSP (constraint satisfaction problem) solvers (such as map-coloring). These are all *search* based methods, and are in one form or another, a form of conditioning.

Conditioning

- Other ways of doing inference in discrete networks, not based (as much) on graph theoretic properties — these methods are based on methods used in SAT solvers (DPLL) and CSP (constraint satisfaction problem) solvers (such as map-coloring). These are all *search* based methods, and are in one form or another, a form of conditioning.
- When we condition on a set of variables, we may treat them as observed.

Conditioning

- Other ways of doing inference in discrete networks, not based (as much) on graph theoretic properties — these methods are based on methods used in SAT solvers (DPLL) and CSP (constraint satisfaction problem) solvers (such as map-coloring). These are all *search* based methods, and are in one form or another, a form of conditioning.
- When we condition on a set of variables, we may treat them as observed.
- During a nested loop over variable values, in an inner loop, the relative outer loop variables are essentially "conditioned on" and can be treated as if they are observed at their current values from the perspective of the inner loops.

Conditioning

- Observed values can cut a network, and induce new independence and factorization properties that were not there in general (when the variables were hidden).

Conditioning

- Observed values can cut a network, and induce new independence and factorization properties that were not there in general (when the variables were hidden).
- Sometimes such factorizations exist only for certain (but not all) variable values. Therefore, different sets of inner loops can be performed (even at the same loop nest depth) based on the set of variable values (and consequent value-specific factorizations) that are currently active.

Conditioning

- Observed values can cut a network, and induce new independence and factorization properties that were not there in general (when the variables were hidden).
- Sometimes such factorizations exist only for certain (but not all) variable values. Therefore, different sets of inner loops can be performed (even at the same loop nest depth) based on the set of variable values (and consequent value-specific factorizations) that are currently active.
- Simplest example of this is Pearl's *cutset conditioning*.

Conditioning

- Recall, general problem is to compute $p(\bar{x}_E)$ as:

$$p(\bar{x}_E) = \sum_{x_{V \setminus E}} p(x_{V \setminus E}, \bar{x}_E) \tag{9.20}$$

Conditioning

- Recall, general problem is to compute $p(\bar{x}_E)$ as:

$$p(\bar{x}_E) = \sum_{x_{V \setminus E}} p(x_{V \setminus E}, \bar{x}_E) \tag{9.20}$$

- If graph is a 1-tree, we can do this in $O(Nr^2)$, on 1-tree $G = (V, E)$ with $N = |V|$. If graph is not a 1-tree, we could "condition" on a set of nodes such that G', the remainder non-conditioned-on set, is a 1-tree.

Conditioning

- Recall, general problem is to compute $p(\bar{x}_E)$ as:

$$p(\bar{x}_E) = \sum_{x_{V \setminus E}} p(x_{V \setminus E}, \bar{x}_E) \tag{9.20}$$

- If graph is a 1-tree, we can do this in $O(Nr^2)$, on 1-tree $G = (V, E)$ with $N = |V|$. If graph is not a 1-tree, we could “condition” on a set of nodes such that G', the remainder non-conditioned-on set, is a 1-tree.
- I.e., $G' = (V \setminus C, E')$ where $E' = E \cap (V \setminus C \times V \setminus C)$ is an induced subgraph, and C is chosen so that G' is a 1-tree.

Conditioning

- If $C \subset V$ (so that x_C) is observed, then G' is a 1-tree from a state space perspective, solvable in $O(Nr^2)$ again.

Conditioning

- If $C \subset V$ (so that x_C) is observed, then G' is a 1-tree from a state space perspective, solvable in $O(Nr^2)$ again.
- If x_C is not observed, we can consider all values $x_C \in \mathsf{D}_{X_C}$ in turn.

Conditioning

- If $C \subset V$ (so that x_C) is observed, then G' is a 1-tree from a state space perspective, solvable in $O(Nr^2)$ again.
- If x_C is not observed, we can consider all values $x_C \in \mathsf{D}_{X_C}$ in turn.

```
1 foreach $x_C \in \mathsf{D}_{X_C}$ do
2     consider $x_C$ to be observed, $\bar{x}_C$
3     compute $p(\bar{x}_{C \cup E})$ using message passing on tree $G' = (V \setminus C, E')$ ;
      /* doable in $O(Nr^2)$ */
4     Accumulate $\alpha = \alpha + p(\bar{x}_{C \cup E})$
```

Conditioning

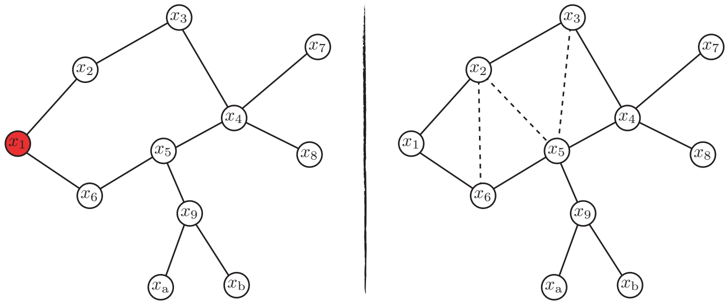
- If $C \subset V$ (so that x_C) is observed, then G' is a 1-tree from a state space perspective, solvable in $O(Nr^2)$ again.
- If x_C is not observed, we can consider all values $x_C \in \mathsf{D}_{X_C}$ in turn.

```
1 foreach $x_C \in \mathsf{D}_{X_C}$ do
2     consider $x_C$ to be observed, $\bar{x}_C$
3     compute $p(\bar{x}_{C \cup E})$ using message passing on tree $G' = (V \setminus C, E')$ ;
      /* doable in $O(Nr^2)$ */
4     Accumulate $\alpha = \alpha + p(\bar{x}_{C \cup E})$
```

- When done, result will be $p(\bar{x}_E)$ as desired.

Conditioning

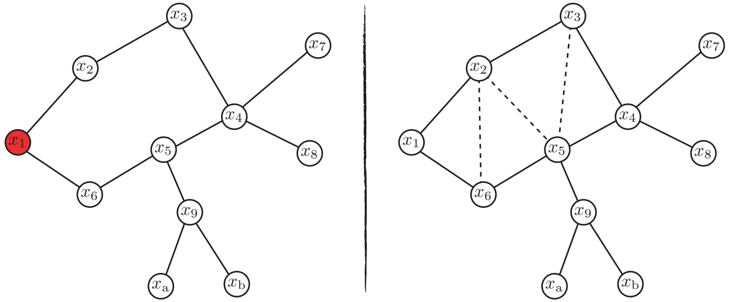
- Such a set C is called a *cycle-cutset*, since it is a cutset that cuts all cycles (so yields a forest or tree). Overall cost of this is $O(r^{|C|}Nr^2)$.



- Cutset condition cost $O(rNr^2)$ and elimination on optimal junction tree cost $O(Nr^3)$

Conditioning

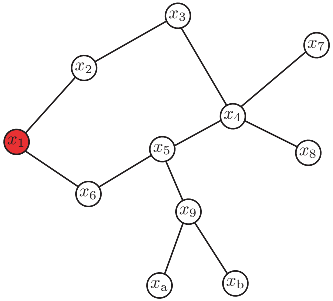
- Such a set C is called a *cycle-cutset*, since it is a cutset that cuts all cycles (so yields a forest or tree). Overall cost of this is $O(r^{|C|}Nr^2)$.

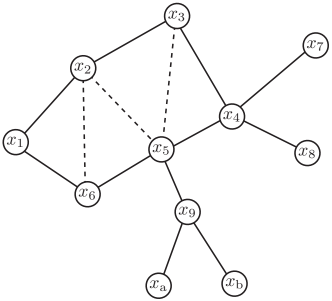


- Cutset condition cost $O(rNr^2)$ and elimination on optimal junction tree cost $O(Nr^3)$
- What have we gained here?

Conditioning

- Such a set C is called a *cycle-cutset*, since it is a cutset that cuts all cycles (so yields a forest or tree). Overall cost of this is $O(r^{|C|} N r^2)$.





- Cutset condition cost $O(rNr^2)$ and elimination on optimal junction tree cost $O(Nr^3)$
- What have we gained here? Memory: Cutset conditioning case (left) is now $O(r^2)$ memory, while original case (right) is $O(r^3)$.

Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).

Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).
- other algorithms exist as well, fall along the time-space tradeoff frontier. The two extremes we've seen perhaps are not useful.

Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).
- other algorithms exist as well, fall along the time-space tradeoff frontier. The two extremes we've seen perhaps are not useful.
- But other algorithms exist that trade-off between time and space complexity. The above cycle-cutset example exhibited a point along that trade-off: It was neither $O(N)$ time nor memory but did reduce memory with the same time cost.

Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).
- other algorithms exist as well, fall along the time-space tradeoff frontier. The two extremes we've seen perhaps are not useful.
- But other algorithms exist that trade-off between time and space complexity. The above cycle-cutset example exhibited a point along that trade-off: It was neither $O(N)$ time nor memory but did reduce memory with the same time cost.
- Boundary between what is possible along the time/space complexity tradeoff.

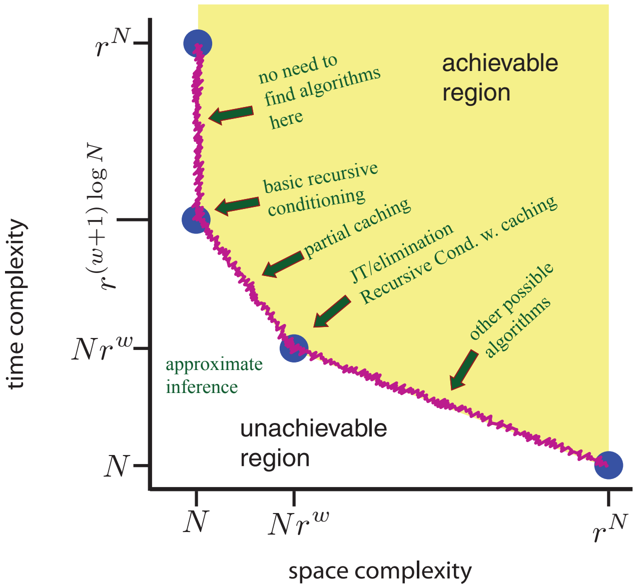
Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).
- other algorithms exist as well, fall along the time-space tradeoff frontier. The two extremes we've seen perhaps are not useful.
- But other algorithms exist that trade-off between time and space complexity. The above cycle-cutset example exhibited a point along that trade-off: It was neither $O(N)$ time nor memory but did reduce memory with the same time cost.
- Boundary between what is possible along the time/space complexity tradeoff.
- Achievable region: shows where it is possible to compute exact inference.

Time-Space Tradeoffs

- We already have $O(Nr^{w+1})$ time/space complexity solution with the junction-tree or elimination (assuming we can optimally triangulate).
- other algorithms exist as well, fall along the time-space tradeoff frontier. The two extremes we've seen perhaps are not useful.
- But other algorithms exist that trade-off between time and space complexity. The above cycle-cutset example exhibited a point along that trade-off: It was neither $O(N)$ time nor memory but did reduce memory with the same time cost.
- Boundary between what is possible along the time/space complexity tradeoff.
- Achievable region: shows where it is possible to compute exact inference.
- Unachievable region: where not possible to compute exact inference, where approximate inference lies.

Time-Space Tradeoffs



Recursive Conditioning

- recursive conditioning generalizes cutset conditioning in that it does decomposition like before but might not be a cycle cutset, and then recursively applies the same idea.

Recursive Conditioning

- recursive conditioning generalizes cutset conditioning in that it does decomposition like before but might not be a cycle cutset, and then recursively applies the same idea.
- It is possible with recursive conditioning to achieve a variety of points on the time-space tradeoff achievable frontier (as we will see). In each case, there is at some point an implicit triangulation.

Recursive Conditioning

- recursive conditioning generalizes cutset conditioning in that it does decomposition like before but might not be a cycle cutset, and then recursively applies the same idea.
- It is possible with recursive conditioning to achieve a variety of points on the time-space tradeoff achievable frontier (as we will see). In each case, there is at some point an implicit triangulation.
- Many ways to formulate it, here is a simple approach that uses notation similar to what we've been using. Consider nodes of $G = (V, E)$ a JT $C_1, C_2, \ldots, C_M$ with $C_i \in \mathcal{C}$, ordered arbitrarily.

Recursive Conditioning: naive approach

Input: Dist. $p \in \mathcal{F}(G, \mathcal{M}^{(\mathrm{f})})$, JT nodes (clusters) $C_{1:M}$, evidence
Output: Value of $p(\bar{x}_E)$

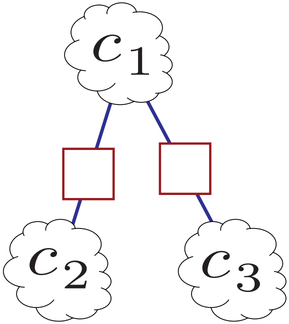
```
1 $\alpha \leftarrow 0$ ;
2 for $x_{C_1} \in \mathsf{D}_{X_{C_1}}$ do
3     for $x_{C_2 \setminus C_1} \in \mathsf{D}_{X_{C_2 \setminus C_1}}$ do
4         for $x_{C_3 \setminus (C_1 \cup C_2)} \in \mathsf{D}_{X_{C_3 \setminus (C_1 \cup C_2)}}$ do
5             for $\ldots$ do
6                 for $x_{C_N \setminus C_{1:N-1}} \in \mathsf{D}_{X_{C_N \setminus C_{1:N-1}}}$ do
7                     $\alpha += p(x)$
```

This is $O(N)$ space and $O(r^N)$ time (same as linear space idea we saw before), so again not useful since time complexity is exorbitant.

Recursive Conditioning with good order

Example: 3-cluster version

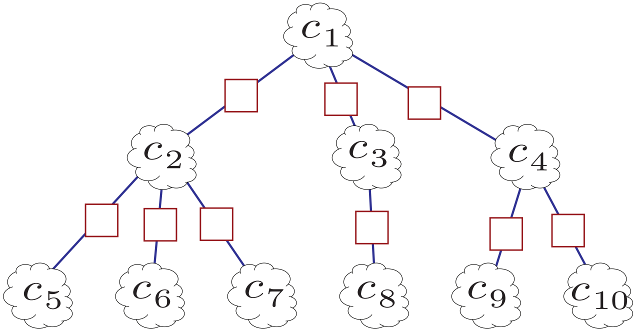
```
1 for $x_{C_1} \in \mathsf{D}_{X_{C_1}}$ do
2     for $x_{C_2 \setminus C_1} \in \mathsf{D}_{X_{C_2 \setminus C_1}}$ do
3         $\alpha_{2|1} \mathrel{+}= p(x_{C_1 \cup C_2})$
4     for $x_{C_3 \setminus C_1} \in \mathsf{D}_{X_{C_3 \setminus C_1}}$ do
5         $\alpha_{3|1} \mathrel{+}= p(x_{C_1 \cup C_3})$
6     $\alpha_1 \mathrel{+}= \alpha_{2|1}\alpha_{3|1}$
```



- Outer loop costs $O(|\mathsf{D}_{X_{C_1}}|)$. Inner loops each cost $O(|\mathsf{D}_{X_{C_2 \setminus C_1}}|)$ (assuming C_1 and C_2 are same size).
- Total cost is $O(|\mathsf{D}_{X_{C_1 \cup C_2}}|)$, better than $O(|\mathsf{D}_{X_{C_1 \cup C_2 \cup C_3}}|) = O(r^N)$
- Memory: still linear.

Recursive Conditioning with good order

- We can order the cliques in a different way though. Note that this is not necessarily a junction tree, although it could easily be. Rather, this is more akin to the decomposition trees we saw earlier in the course.
- Depth of tree is $d = O(\log N)$



Recursive Conditioning with good order

```
1  for $x_{C_1} \in \mathsf{D}_{X_{C_1}}$ do
2      for $x_{C_2 \setminus C_1} \in \mathsf{D}_{X_{C_2 \setminus C_1}}$ do
3          for $x_{C_5 \setminus C_{1,2}} \in \mathsf{D}_{X_{C_5 \setminus C_{1,2}}}$ do
4              $\alpha_{5|1,2} \mathrel{+}= p(x_{C_{1,2,5}})$
5          for $x_{C_6 \setminus C_{1,2}} \in \mathsf{D}_{X_{C_6 \setminus C_{1,2}}}$ do
6              $\alpha_{6|1,2} \mathrel{+}= p(x_{C_{1,2,6}})$
7          for $x_{C_7 \setminus C_{1,2}} \in \mathsf{D}_{X_{C_7 \setminus C_{1,2}}}$ do
8              $\alpha_{7|1,2} \mathrel{+}= p(x_{C_{1,2,7}})$
9          $\alpha_{2|1} \mathrel{+}= \alpha_{5|1,2}\alpha_{6|1,2}\alpha_{7|1,2}$
0      Include lines 1-12 here
```

```
1  Lines 1-12, include at line 10 above
2  for $x_{C_3 \setminus C_1} \in \mathsf{D}_{X_{C_3 \setminus C_1}}$ do
3      for $x_{C_8 \setminus C_{1,3}} \in \mathsf{D}_{X_{C_8 \setminus C_{1,3}}}$ do
4          $\alpha_{8|1,3} \mathrel{+}= p(x_{C_{1,3,8}})$
5      $\alpha_{3|1} \mathrel{+}= \alpha_{8|1,3}$
6  for $x_{C_4 \setminus C_1} \in \mathsf{D}_{X_{C_4 \setminus C_1}}$ do
7      for $x_{C_9 \setminus C_{1,4}} \in \mathsf{D}_{X_{C_9 \setminus C_{1,4}}}$ do
8          $\alpha_{9|1,4} \mathrel{+}= p(x_{C_{1,4,9}})$
9      for $x_{C_{10} \setminus C_{1,4}} \in \mathsf{D}_{X_{C_{10} \setminus C_{1,4}}}$ do
10         $\alpha_{10|1,4} \mathrel{+}= p(x_{C_{1,4,10}})$
11     $\alpha_{4|1} \mathrel{+}= \alpha_{9|1,4}\alpha_{10|1,4}$
12 $\alpha_1 \mathrel{+}= \alpha_{2|1}\alpha_{3|1}\alpha_{4|1}$
```

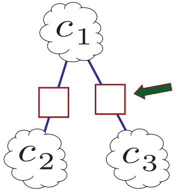
Recursive Conditioning with good order

- When we're all done, $\alpha_1 = p(\bar{x}_E)$ (again, assuming evidence is treated as multiplies by $\delta(x, \bar{x})$).
- How much space is needed? $O(N)$ still since in worst case, depth of the tree is number of maxcliques (which is $O(N)$).
- How much time? Depends on number of α-accumulates, or number of leaf-nodes in the tree. Depth is $d = \log N$. Each clique gets run about r^{w+1} times, and runs the nodes below it about that many times.
- We get a time complexity of:

$$\underbrace{r^{w+1} r^{w+1} \dots r^{w+1}}_{d \text{ times}} = r^{(w+1)\log N} \tag{9.21}$$

Recursive Conditioning with good order

- How to get other points on frontier?
- Note that in previous algorithm, for each set of variable values in intersection set (square boxes), we were solving the same sub-problem multiple times.
- We can cache the solutions for each value, at the cost of more memory. If everything is cached, space complexity will increase to $O(Nr^w)$ and time complexity will decrease to $O(Nr^w)$ (like the JT case).



we need not solve each entry in this intersection set multiple times. Instead, we can cache values. Total number of entrie is $O(Nr^W)$

Value-specific Caching

- Many algorithms use value specific caching. I.e., depending on the values of some variables currently conditioned on, we might actually get an entirely different set of maxcliques (or set of sets of maxcliques) below. Each should ideally be treated differently.
- We can construct and memoize the *dependency sets*, the set of variables and their values that induce particular sub-computations. Each sub-computation might be a computation of a sum, or it might even be a computation of zero (called a no-good, or a conflict). Each of these can be memoized and re-used whenever the dependency set becomes active again.
- the order of the cliques and the order of the variables in the cliques might dynamically change depending on previously instantiated values. We might not even use cliques at all, and do this at the granularity of variables and their values.

Value-Elimination

- This is the basis of the value elimination algorithm (Bacchus-2003), a general procedure for probabilistic inference. It gets much of its inspiration from the techniques used to produce fast SAT and constraint satisfaction problem (CSP) engines.
- This is especially useful if we have many zeros (sparsity) in the distribution and/or if there is much value specific independence.

Hardness

- Even with conditioning, search, etc. Complexity of exact inference is always exponential in at least the tree-width of any covering graph.
- Indeed, finding the best covering triangulated graph (with minimal tree-width) is an NP-complete optimization.
- Even worse, inference itself is NP-complete. There are some graphs that can't be solved in polynomial time unless P=NP.

Hardness of Inference

- Consider the 3-SAT problem (which is a canonical NP-complete problem). Given list of N variables, and a collection of M clauses (constraints), where each clause is a disjunction of 3 literals (a variable or its negation). Clauses are organized in a conjunction. *Question:* is there a satisfying truth assignment of the variables (assignment of variable values that makes the conjunction of disjunctions true).
- examples:

$$(x_1 \vee x_4 \vee \bar{x}_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)$$
$$\wedge (\bar{x}_1 \vee x_4 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$$

and also

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_7 \vee x_8 \vee x_9)$$
$$\wedge (\bar{x}_9 \vee x_{10} \vee x_{11}) \wedge (\bar{x}_{11} \vee \bar{x}_{12} \vee \bar{x}_3)$$

Hardness of Inference

- In the general case, we have N variables and M clauses, either of which might be very large. If we can solve this problem in polynomial time in N, then all NP-complete problems can be solved in polynomial time.

Hardness of Inference

- In the general case, we have N variables and M clauses, either of which might be very large. If we can solve this problem in polynomial time in N, then all NP-complete problems can be solved in polynomial time.
- To show that inference in Bayesian networks is NP-complete, all we need to do is find a BN or MRF that encodes this problem using the appropriate commutative semiring (which in our case, we'll take to be the max-product semiring).

Hardness of Inference

- In the general case, we have N variables and M clauses, either of which might be very large. If we can solve this problem in polynomial time in N, then all NP-complete problems can be solved in polynomial time.
- To show that inference in Bayesian networks is NP-complete, all we need to do is find a BN or MRF that encodes this problem using the appropriate commutative semiring (which in our case, we'll take to be the max-product semiring).
- Let $\{x_i\}_{i=1}^N$ be the set of variables, and let C_j be the index set of the variables for clause $0 \leq j \leq M$.

Hardness of Inference

- In the general case, we have N variables and M clauses, either of which might be very large. If we can solve this problem in polynomial time in N, then all NP-complete problems can be solved in polynomial time.
- To show that inference in Bayesian networks is NP-complete, all we need to do is find a BN or MRF that encodes this problem using the appropriate commutative semiring (which in our case, we'll take to be the max-product semiring).
- Let $\{x_i\}_{i=1}^N$ be the set of variables, and let C_j be the index set of the variables for clause $0 \leq j \leq M$.
- Define binary-valued functions $f_j(x_{C_j})$ such that $f_j = 1$ iff the clause is satisfied by the current values of the variables x_{C_j}, otherwise $f_j = 0$.

Hardness of Inference

- With this formulation, we get factorization as follows

$$\prod_j f_j(x_{C_j}) \tag{9.22}$$

  which is possible to evaluate to unity iff the logic formula is satisfiable.
- Next, consider BN with N binary variables $\{x_i\}_{i=1}^N$ and M additional variables $\{y_j\}_{j=1}^M$ with M CPTS of the form:

$$p(y_j = 1 | x_{C_j}) = \begin{cases} 1 & \text{if } f_j(x_{C_j}) = 1 \\ 0 & \text{else} \end{cases}, \text{ and for } x_i \; p(x_i = 1) = 0.5 \tag{9.23}$$

- This gives joint distribution that factorizes

$$p(x_{1:N}, y_{1:M}) = \prod_i p(x_i) \prod_j p(y_j | x_{C_j})$$

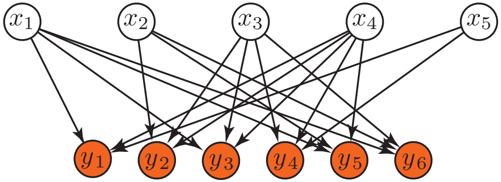
Hardness of Inference

- Create following BN, as evidence set use $y_j = 1$ for all $j \in 1 \dots M$
- Use max-sum semi-ring, so goal is to find the assignment to the x variables that maximize the joint probability.
- Resulting max evaluation is 1 iff original 3-SAT formula is satisfiable.

Hardness of Inference

- Example: $N = 5, M = 6$ in following 3-SAT formula and BN

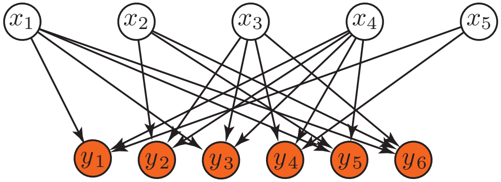
$(x_1 \vee x_4 \vee \bar{x}_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_4 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3$



Hardness of Inference

- Example: $N = 5, M = 6$ in following 3-SAT formula and BN

$(x_1 \vee x_4 \vee \bar{x}_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_4 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x$

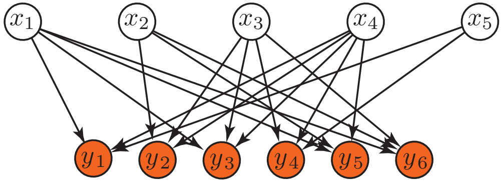


- MPE/Viterbi assignment to $x_{1:5}$ has non-zero probability iff original formula is SAT, BN inference (in general) NP-complete.

Hardness of Inference

- Example: $N = 5, M = 6$ in following 3-SAT formula and BN

$(x_1 \vee x_4 \vee \bar{x}_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_4 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x$

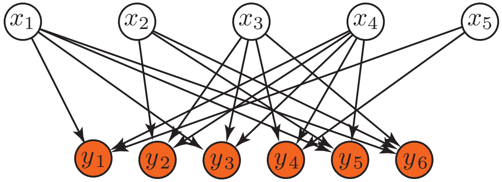


- MPE/Viterbi assignment to $x_{1:5}$ has non-zero probability iff original formula is SAT, BN inference (in general) NP-complete.
- **Doesn't** mean exact inference is always intractable, rather can't hope for a polynomial solution in all cases unless $P = NP$.

Hardness of Inference

- Example: $N = 5, M = 6$ in following 3-SAT formula and BN

$(x_1 \vee x_4 \vee \bar{x}_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_4 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x$



- MPE/Viterbi assignment to $x_{1:5}$ has non-zero probability iff original formula is SAT, BN inference (in general) NP-complete.
- **Doesn't** mean exact inference is always intractable, rather can't hope for a polynomial solution in all cases unless $P = NP$.
- Moreover, even low tree-width graphs can be computationally challenging (i.e., large state space or random variable domain size).

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.
- We can use conditioning (e.g., cutset conditioning) to get other points. E.g., condition on a set that renders the remainder of the set a tree. Same computation less memory.

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.
- We can use conditioning (e.g., cutset conditioning) to get other points. E.g., condition on a set that renders the remainder of the set a tree. Same computation less memory.
- Recursive conditioning allows is to get linear memory but a time complexity of $O(r^{(w+1)\log N})$.

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.
- We can use conditioning (e.g., cutset conditioning) to get other points. E.g., condition on a set that renders the remainder of the set a tree. Same computation less memory.
- Recursive conditioning allows is to get linear memory but a time complexity of $O(r^{(w+1)\log N})$.
- In general, many time-space tradeoffs for exact inference. Many algorithms along the achievable/unachievable frontier are SAT/CSP based, and use conditioning combined with various caching, and clause learning/deduction (e.g., nogood learning).

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.
- We can use conditioning (e.g., cutset conditioning) to get other points. E.g., condition on a set that renders the remainder of the set a tree. Same computation less memory.
- Recursive conditioning allows is to get linear memory but a time complexity of $O(r^{(w+1)\log N})$.
- In general, many time-space tradeoffs for exact inference. Many algorithms along the achievable/unachievable frontier are SAT/CSP based, and use conditioning combined with various caching, and clause learning/deduction (e.g., nogood learning).
- To get a better time/space profile, need to do approximation.

Recap

- Time and memory complexity is $O(Nr^{\omega+1})$ where ω is the tree-width.
- We can use conditioning (e.g., cutset conditioning) to get other points. E.g., condition on a set that renders the remainder of the set a tree. Same computation less memory.
- Recursive conditioning allows is to get linear memory but a time complexity of $O(r^{(w+1)\log N})$.
- In general, many time-space tradeoffs for exact inference. Many algorithms along the achievable/unachievable frontier are SAT/CSP based, and use conditioning combined with various caching, and clause learning/deduction (e.g., nogood learning).
- To get a better time/space profile, need to do approximation.
- For any given degree of distortion, there is a time/space tradeoff profile.

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
    1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
    2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference
  1. Message or other form of propagation, variational approaches, LP relaxations

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference
  1. Message or other form of propagation, variational approaches, LP relaxations
  2. sampling

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference
  1. Message or other form of propagation, variational approaches, LP relaxations
  2. sampling
  3. etc.

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference
  1. Message or other form of propagation, variational approaches, LP relaxations
  2. sampling
  3. etc.
- Both methods only guaranteed approximate quality solutions.

Approximation: Two general approaches

- exact solution to approximate problem - approximate problem
  1. learning with or using a model with a structural restriction, structure learning, using a k-tree for a lower k than one knows is true. Make sure k is small enough so that exact inference can be performed, and make sure that, in that low tree-width model, one has best possible graph
  2. Functional restrictions to the model (i.e., use factors or potential functions that obey certain properties). Then certain fast algorithms (e.g., graph-cut) can be performed.
- approximate solution to exact problem - approximate inference
  1. Message or other form of propagation, variational approaches, LP relaxations
  2. sampling
  3. etc.
- Both methods only guaranteed approximate quality solutions.
- No longer in the achievable region in time-space tradoff graph, new set of time/space tradeoffs to achieve a particular accuracy.

Sources for Today's Lecture

- Most of this material comes from a variety of sources. Best place to look is in our standard reading material.