Graphical Models

Lecture 12:

Belief Update Message Passing

Andrew McCallum
mccallum@cs.umass.edu

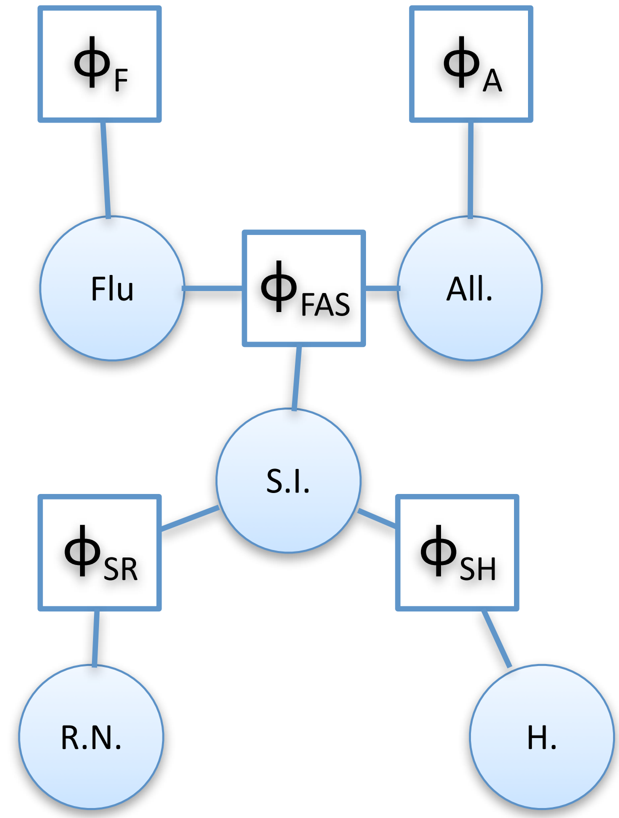
Thanks to Noah Smith and Carlos Guestrin for slide materials.

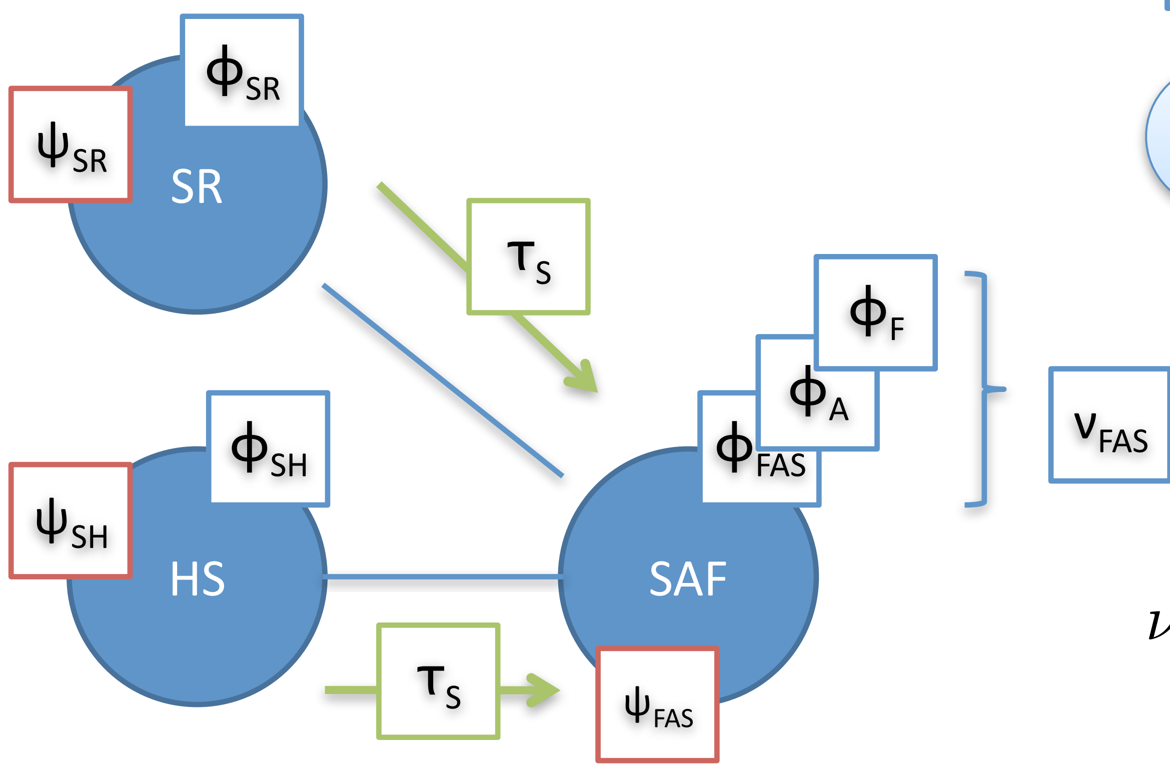
Today's Plan

- Quick Review: Sum Product Message Passing (also known as "Shafer-Shenoy")
- Today: Sum Product **Divide** Message Passing (also known as "belief update" message passing "Lauritzen-Spiegelhalter" and "belief propagation")
- Mathematically equivalent, but different intuitions.
- Moving toward approximate inference.

Quick Review

- {H, R, S, A, F}





$$\nu_j(\boldsymbol{C}_j) = \prod_{\phi \in \Phi_j} \phi$$

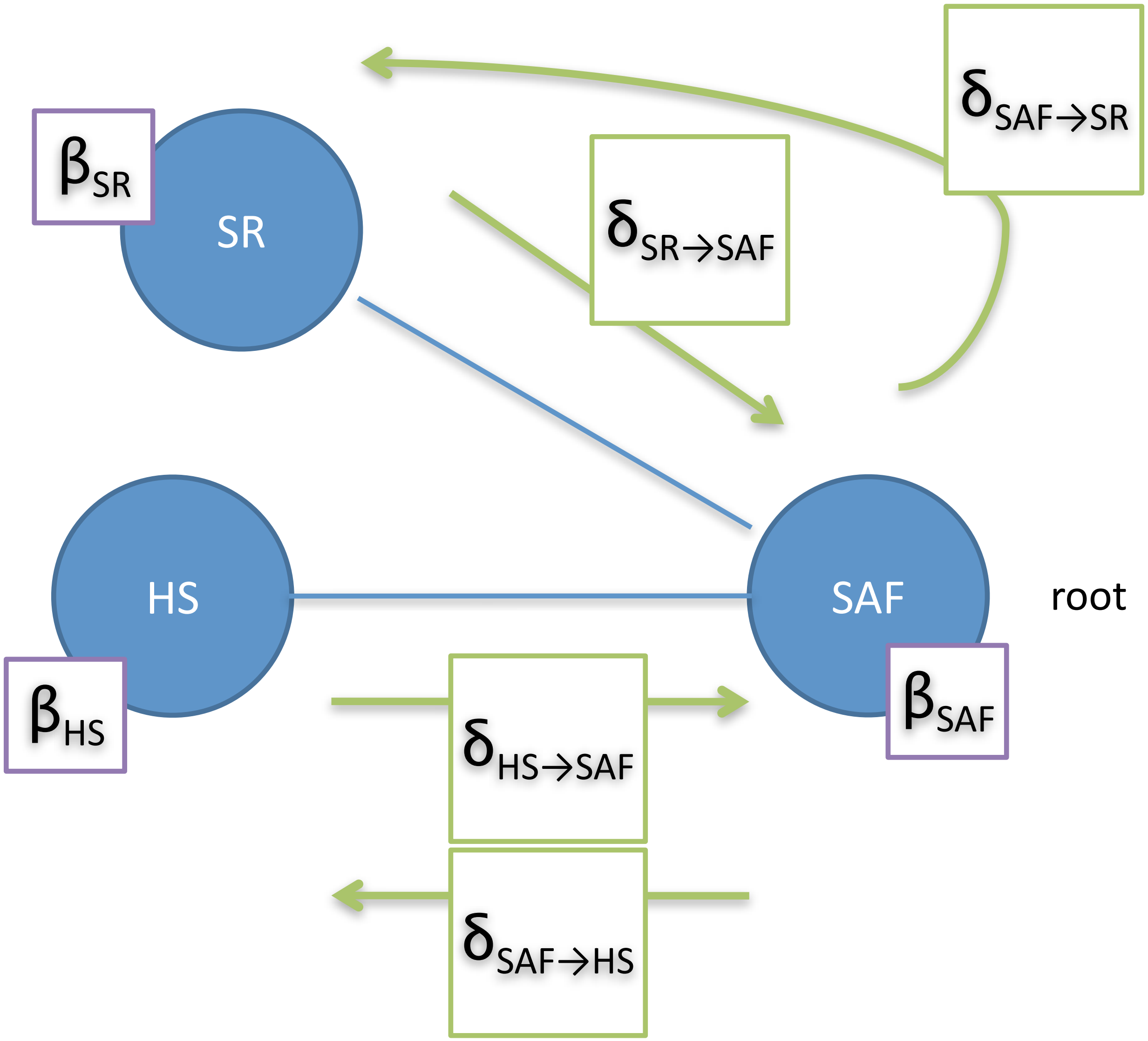
Message Passing (One Root)

- Input: clique tree $\mathcal{T}$, factors $\mathbf{\Phi}$, root $\mathbf{C}_r$
- For each clique $\mathbf{C}_i$, calculate ν_i
- While $\mathbf{C}_r$ is still waiting on incoming messages:
  - Choose a $\mathbf{C}_i$ that *has* received all of its incoming messages.
  - Calculate and send the message from $\mathbf{C}_i$ to $\mathbf{C}_{\text{upstream-neighbor(i)}}$

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \mathrm{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

$$\begin{aligned} \beta_r &= \nu_r \prod_{k \in \mathrm{Neighbors}_r} \delta_{k \to r} \\ &= \sum_{\boldsymbol{X} \setminus \boldsymbol{C}_i} \prod_{\phi \in \Phi} \phi \\ &= Z \cdot P(\boldsymbol{C}_r) \end{aligned}$$

Different Roots; Same Messages



Sum-Product Message Passing

- Each clique tree vertex $\mathbf{C}_i$ passes messages to each of its neighbors once it's ready to do so.
- At the end, for all $\mathbf{C}_i$:

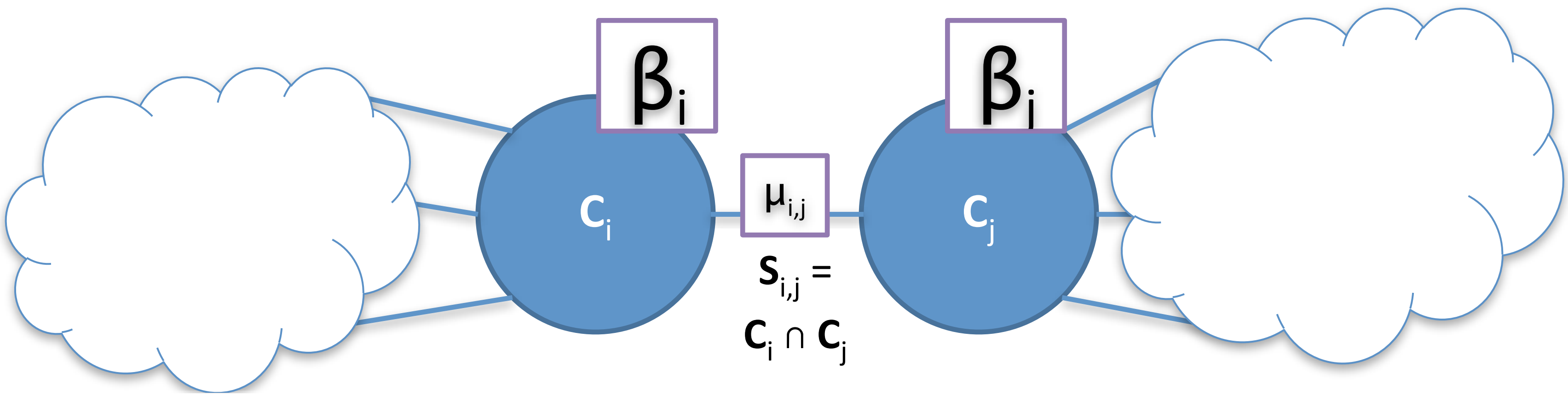
$$\beta_i = \nu_i \prod_{k \in \mathrm{Neighbors}_i} \delta_{k \to i}$$

  – This is the unnormalized marginal for $\mathbf{C}_i$.

Calibrated Clique Tree

- Two adjacent cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ are calibrated when:

$$\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i = \sum_{\boldsymbol{C}_j \setminus \boldsymbol{S}_{i,j}} \beta_j = \mu_{i,j}(\boldsymbol{S}_{i,j})$$



Calibrated Clique Tree as a

- Original (unnormalized) factor model and calibrated clique tree represent the same (unnormalized) measure:

clique **beliefs**

$$\prod_{\phi \in \Phi} \phi = \frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \beta_{\boldsymbol{C}}}{\prod_{\boldsymbol{S} \in \text{Edges}(\mathcal{T})} \mu_{\boldsymbol{S}}}$$

sepset **beliefs**

unnormalized Gibbs distribution from original factors **Φ**

Inventory of Factors

- original factors ϕ
- initial potentials ν
- messages δ
- intermediate factors ψ (no longer explicit)
- clique beliefs β
- sepset beliefs μ

Inventory of Factors

- original factors ϕ
- initial potentials ν
- messages δ
- intermediate factors ψ (no longer explicit)
- clique beliefs β
- sepset beliefs μ

New algorithm collapses everything into **beliefs**!

Another Operation: Factor Division

- 0 / 0 is defined to be 0
- a / 0 is undefined when a > 0

A	B	C	$\phi_1(A, B, C)$
0	0	0	7000
0	0	1	30
0	1	0	5
0	1	1	500
1	0	0	100
1	0	1	1
1	1	0	10
1	1	1	1000

/

B	C	$\phi_2(B, C)$
0	0	100
0	1	1
1	0	2
1	1	100

=

A	B	C	$\phi_3(A, B, C)$
0	0	0	70
0	0	1	30
0	1	0	2.5
0	1	1	5
1	0	0	1
1	0	1	1
1	1	0	5
1	1	1	10

Messages

- When computing the message $\delta_{i \to j}$ from i to j, we multiply together all incoming messages to i *except* the one from j to i, $\delta_{j \to i}$.
- Alternative: multiply *all* messages, and *divide* out the one from j to i.

$$\beta_i = \nu_i \prod_{k \in \mathrm{Neighbors}_i} \delta_{k \to i}$$

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \mathrm{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

$$\delta_{i \to j} = \frac{\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i}{\delta_{j \to i}}$$

Key Idea

- We can "forget" the initial potentials ν.
- We do not need to calculate the messages δ explicitly.

- Store a partially calculated β on each vertex and a partially calculated μ on each edge; *update* whenever new information comes in.

A Single Belief Update

- At any point in the algorithm:

$$
\begin{aligned}
\sigma_{i\to j} &\leftarrow \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i \\
\beta_j &\leftarrow \beta_j \times \frac{\sigma_{i\to j}}{\mu_{i,j}} \\
\mu_{i,j} &\leftarrow \sigma_{i\to j}
\end{aligned}
$$

Belief Update Message Passing

- Maintain beliefs at each vertex (β) and edge (μ).
- Initialize each β_i to ν_i.
- Initialize each $\mu_{i,j}$ to **1**.
- Pass belief update messages.

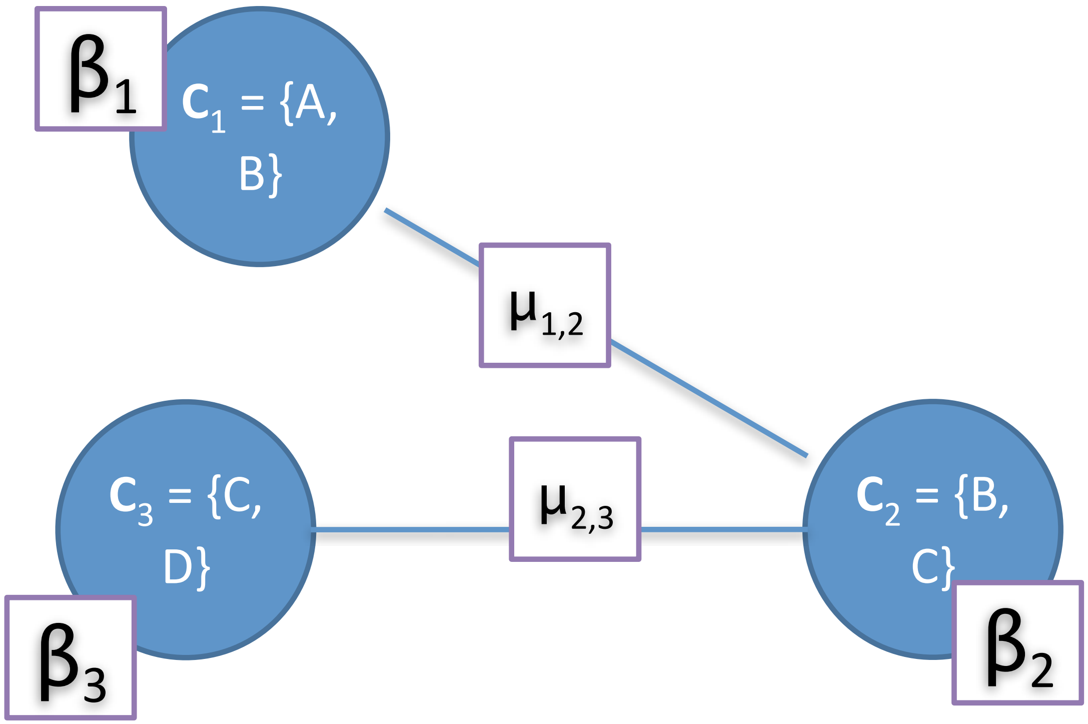
$$
\begin{aligned}
\sigma_{i \to j} &\leftarrow \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i \\
\beta_j &\leftarrow \beta_j \times \frac{\sigma_{i \to j}}{\mu_{i,j}} \\
\mu_{i,j} &\leftarrow \sigma_{i \to j}
\end{aligned}
$$

Three Clique Example

$$\sigma_{i\to j} \leftarrow \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i$$

$$\beta_j \leftarrow \beta_j \times \frac{\sigma_{i\to j}}{\mu_{i,j}}$$

$$\mu_{i,j} \leftarrow \sigma_{i\to j}$$



Worries

- Does the order of the messages matter?
- What if we pass the same message twice?
- What if we pass a message based on partial information?

Claims

- At convergence, we will have a calibrated clique tree.

$$\begin{aligned}\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i &= \sum_{\boldsymbol{C}_j \setminus \boldsymbol{S}_{i,j}} \beta_j \\ &= \mu_{i,j}(\boldsymbol{S}_{i,j})\end{aligned}$$

- Invariant: throughout the algorithm:

$$\prod_{\phi \in \Phi} \phi = \prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} = \frac{\prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \beta_{\boldsymbol{C}}}{\prod_{\boldsymbol{S} \in \mathrm{Edges}(\mathcal{T})} \mu_{\boldsymbol{S}}}$$

Equivalence

- Sum product message passing and
  sum product divide message passing
  lead to the same result: a calibrated clique tree.
  - *SumProduct* lets you calculate beliefs at the very end.
  - *BeliefUpdate* has beliefs from the start, and keeps them around the whole time.

Complexity

- Linear in number of cliques, total size of all factors.
- Can accomplish convergence in the same upward/downward passes we used for the earlier version.

Dealing with Evidence

Advantage: Incremental Updates

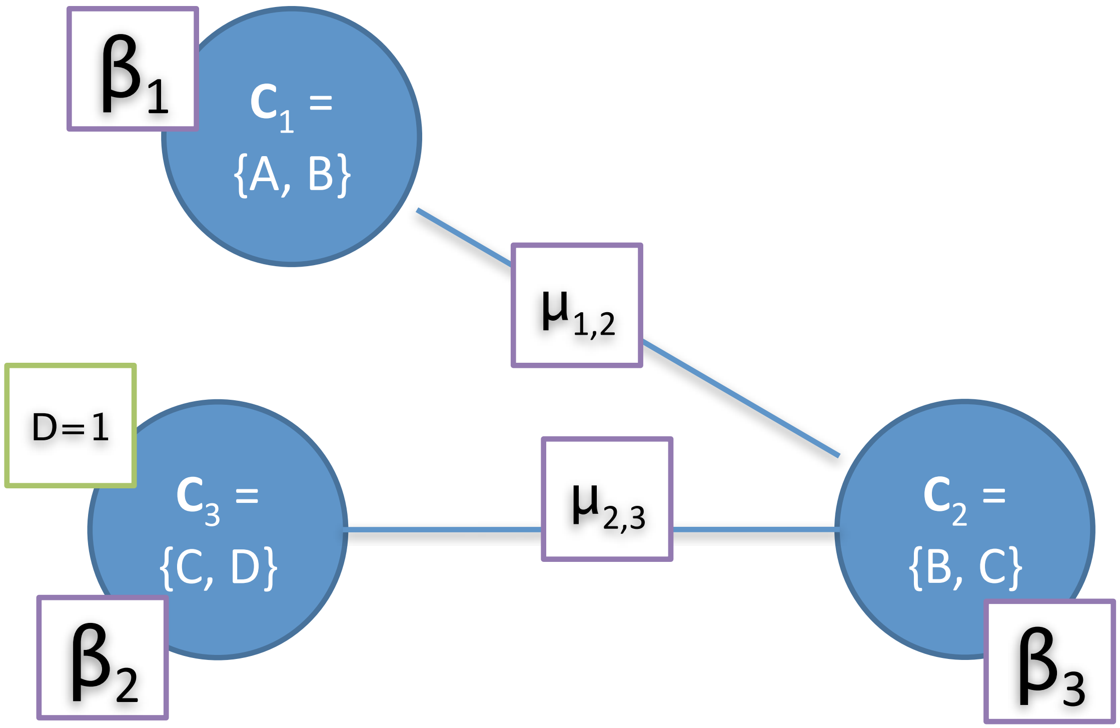
- Naively, if we have evidence, we can alter the initial potentials at the start, then calibrate using message passing.
- Better: think of evidence as a newly arrived factor into some clique tree node(s).

Example

$$\sigma_{i\to j} \leftarrow \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i$$

$$\beta_j \leftarrow \beta_j \times \frac{\sigma_{i\to j}}{\mu_{i,j}}$$

$$\mu_{i,j} \leftarrow \sigma_{i\to j}$$



Advantage: Incremental Updates

- Naively, if we have evidence, we can alter the initial potentials at the start, then calibrate using message passing.
- Better: think of evidence as a newly arrived factor into some clique tree node i.
- Recalibrate: pass messages out from node i. Single pass!
- Retraction: can't recover anything multiplied by zero.

Queries across cliques

Advantage: Queries across Cliques

- Naively: enforce that all query variables are in some clique.
  - Every query might need its own clique tree!

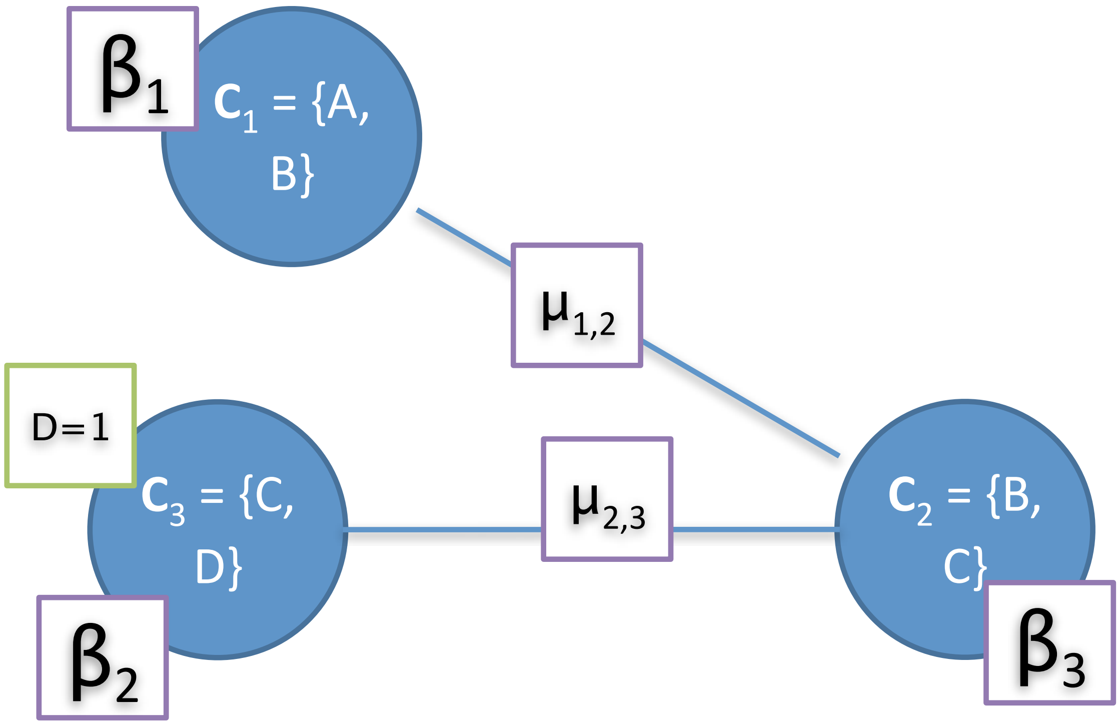
- Better: variable elimination in a calibrated clique tree.
  - Bonus: only have to use a subtree that includes all query variables.

Multi-Clique Queries

- Find a subtree of T that includes all query variables **Q**. Call it T' and its scope **S**.
- Pick a root node r in T'.
- Run variable elimination of **S \ Q** with factors (for all I in T'):

$$\phi_i = \frac{\beta_i}{\mu_{i,\mathrm{upstream}(i)}}$$

Example: $Z \cdot P(B, D)$



Advantage: Multiple Queries

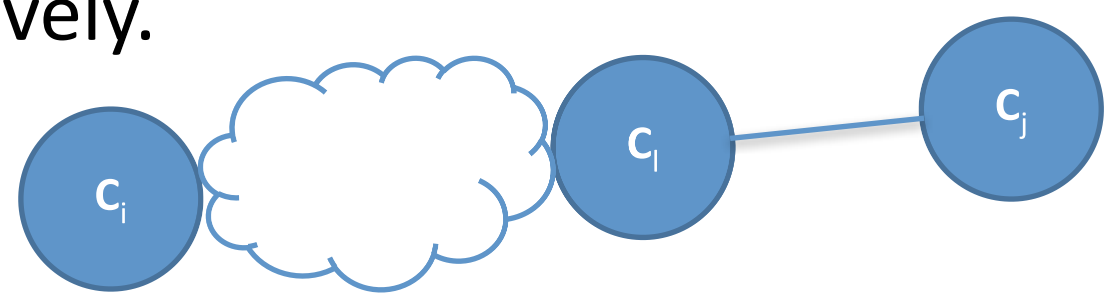
- Suppose we want the marginal for every *pair* of variables X, Y.
- Naïve: construct a clique tree so all nodes pair together. (Very bad.)
- Naïve: run VE n-choose-2 times.
- Better: dynamic programming.

Dynamic Programming for All Pairs

- Construct a table so that $A_{i,j}$ contains $U(\mathbf{C}_i, \mathbf{C}_j) = Z \cdot P(\mathbf{C}_i, \mathbf{C}_j)$.
- Base case: $\mathbf{C}_i$ and $\mathbf{C}_j$ are neighboring cliques.

$$\begin{aligned} A_{i,j} &= U(\boldsymbol{C}_i, \boldsymbol{C}_j) \\ &= U(\boldsymbol{C}_j \mid \boldsymbol{C}_i) U(\boldsymbol{C}_i) \\ &= \frac{\beta_j}{\mu_{i,j}} \beta_i \end{aligned}$$

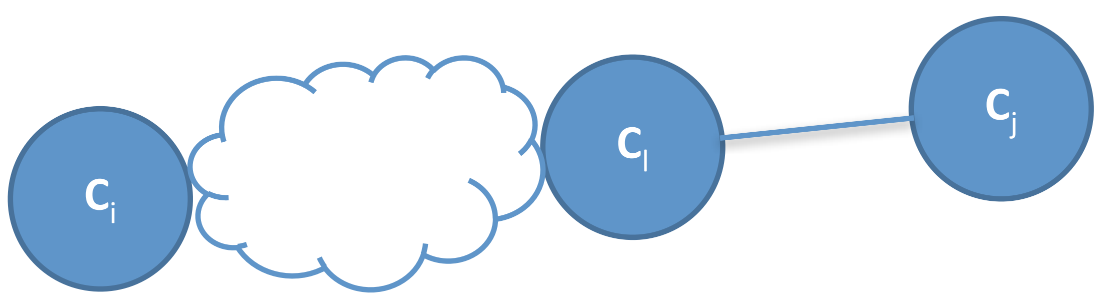
- Proceed to farther more distant pairs recursively.



Dynamic Programming for All Pairs

- $\mathbf{C}_i$ and $\mathbf{C}_j$ are independent given $\mathbf{C}_l$.
- We already have $U(\mathbf{C}_i, \mathbf{C}_l)$ and $U(\mathbf{C}_l, \mathbf{C}_j)$.

$$\begin{aligned} A_{i,j} &= U(\boldsymbol{C}_i, \boldsymbol{C}_j) \\ &= \sum_{\boldsymbol{C}_l \setminus \boldsymbol{C}_j} U(\boldsymbol{C}_i, \boldsymbol{C}_l) U(\boldsymbol{C}_j \mid \boldsymbol{C}_l) \\ &= \sum_{\boldsymbol{C}_l \setminus \boldsymbol{C}_j} A_{i,l} \frac{\beta_j}{\mu_{j,l}} \end{aligned}$$



Pros and Cons: Message Passing in Clique Trees

- Multiple queries
- Incremental updates
- Calibration operation has transparent complexity.

But:

- Complexity can be high (space!)
- Slower than VE for a single query
- Local factor structure is lost

Summary: Message Passing in Clique Trees

- How to construct a clique tree (from VE elimination order or triangulated chordal graph)
- Marginal queries for *all* variables solved in only twice the time of one query!
- Belief update version: clique potentials are reparameterized so that the clique tree invariant always holds.
- Runtime is linear in number of cliques, exponential in size of the largest clique (# variables; induced width).