Stat 521A
Lecture 8

Outline

- Forwards backwards on chains
- FB on trees
- FB on clique chains
- FB on clique trees
- Message passing on clique trees (10.2-10.3)
- Creating clique trees (10.4)

Forwards algorithm

1. predict: compute the the **one-step-ahead predictive density** $p(S_t|\mathbf{x}_{1:t-1})$ as follows:

$$p(S_t = j|\mathbf{x}_{1:t-1}) = \sum_i p(S_t = j, S_{t-1} = i|\mathbf{x}_{1:t-1}) \quad (1)$$

$$= \sum_i p(S_t = j|S_{t-1} = i)p(S_{t-1} = i|\mathbf{x}_{1:t-1}) \quad (2)$$

   In the second step we used the fact that $S_t \perp X_{1:t-1}|S_{t-1}$.

2. update: compute $p(S_t|\mathbf{x}_t, \mathbf{x}_{1:t-1})$ using Bayes rule, where we use $p(S_t|\mathbf{x}_{1:t-1})$ as the prior:

$$p(S_t = j|\mathbf{x}_{1:t}) = \frac{1}{c_t} p(\mathbf{x}_t|S_t = j)p(S_t = j|\mathbf{x}_{1:t-1}) \quad (3)$$

   where we used the fact that $X_t \perp X_{1:t-1}|S_t$. The normalizing constant c_t is given by

$$c_t = p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \sum_j p(\mathbf{x}_t|S_t = j)p(S_t = j|\mathbf{x}_{1:t-1}) \quad (4)$$

The base case is

$$p(S_1 = j|\mathbf{x}_1) \propto p(S_1 = j)p(\mathbf{x}_1|S_1 = j) = \pi_j p(\mathbf{x}_1|S_1 = j) \quad (5)$$

Matrix vector form

$$\alpha_t(j) = p(S_t = j|\mathbf{x}_{1:t}) \quad (1)$$

$$b_t(j) = p(\mathbf{x}_t|S_t = j) \quad (2)$$

$$A(i,j) = p(S_t = j|S_{t-1} = i) \quad (3)$$

Hence the recursion step is

$$\alpha_t(j) \propto b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \quad (4)$$

This can be rewritten in matrix-vector notation as

$$\boldsymbol{\alpha}_t \propto \text{diag}(\mathbf{b}_t)\mathbf{A}^T \boldsymbol{\alpha}_{t-1} \quad (5)$$

It is somewhat clearer if we use Matlab-style notation, and use $.*$ to denote elementwise multiplication by a vector:

$$\boldsymbol{\alpha}_t \propto \mathbf{b}_t . * (\mathbf{A}^T \boldsymbol{\alpha}_{t-1}) \quad (6)$$

The log-likelihood of the data sequence can be computed from the normalizing constants as follows:

$$\log p(\mathbf{x}_{1:T}) = \sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \sum_{c=1}^{T} \log c_t \quad (7)$$

Matlab

Listing 1: Listing of `hmmFilter`

```
function [alpha, loglik] = hmmFilter(initDist, transmat, obslik)
% initDist(i) = Pr(Q(1) = i)
% transmat(i,j) = Pr(Q(t) = j | Q(t-1)=i)
% obslik(i,t) = Pr(Y(t)| Q(t)=i)
[K T] = size(obslik);
alpha = zeros(K,T);
[alpha(:,1), scale(1)] = normalize(initDist(:) .* obslik(:,1));
for t=2:T
  [alpha(:,t), scale(t)] = normalize((transmat' * alpha(:,t-1)) .* obslik(:,t));
end
loglik = sum(log(scale+eps));
```

Listing 2: Listing of `makeLocalEvidence`

```
function localEvidence = makeLocalEvidence(model,obs)
% localEvidence(i,t) = p(Y(t) | Z(t)=i)
localEvidence = zeros(model.nstates,size(obs,2));
for i = 1:model.nstates
 localEvidence(i,:) = exp(logprob(model.emissionDist{i},obs'));
end
```

Offline estimation: goals

- Single slice marginals:

$$\gamma_t(j) \stackrel{\text{def}}{=} p(S_t = j|\mathbf{x}_{1:T}, \boldsymbol{\theta}) \quad (1)$$

  for all $1 \leq t \leq T$. This can be computed via the **forwards backwards** algorithm, as we discuss in Section **??**.

- Two-slice marginals

$$\xi_{t-1,t}(i,j) \stackrel{\text{def}}{=} p(S_{t-1} = i, S_t = j|\mathbf{x}_{1:T}, \boldsymbol{\theta}) \quad (2)$$

  These are needed for parameter estimation, as described in Section **??**. These quantities are easy to compute using forwards-backwards, as we describe in Section **??**.

- The posterior mode, or most probable path:

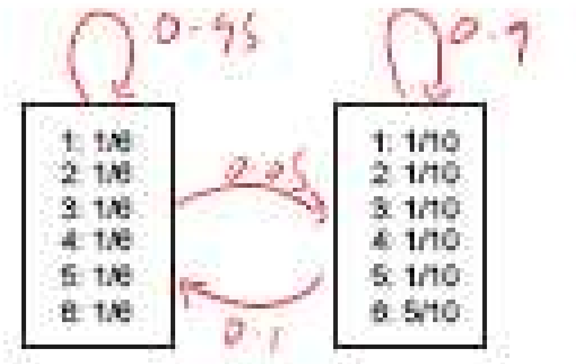
$$\mathbf{s}^*_{1:T} = \arg\max_{\mathbf{s}_{1:T}} p(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}, \boldsymbol{\theta}) \quad (3)$$

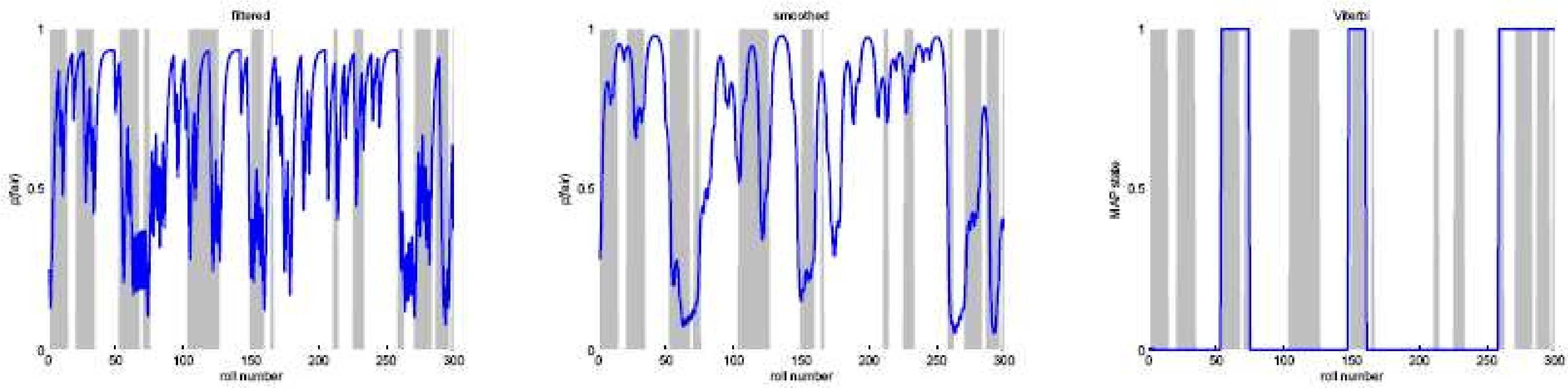
  This can be computed by the **Viterbi algorithm**, as we describe in Section **??**.

- Samples from the posterior

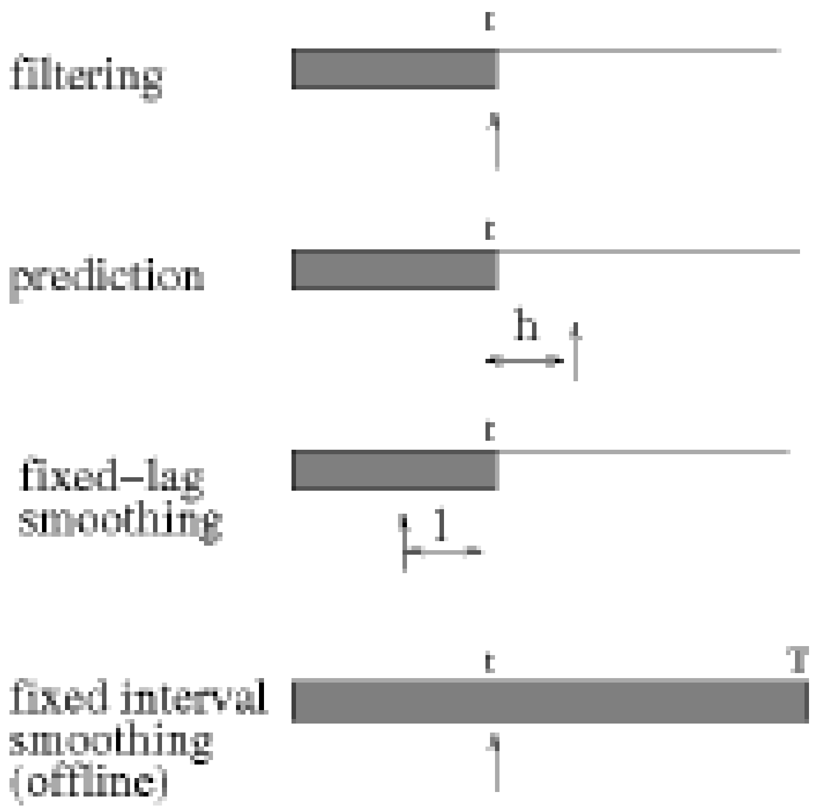
$$\mathbf{s}_{1:T} \sim p(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}, \boldsymbol{\theta}) \quad (4)$$

Filtering vs smoothing vs Viterbi



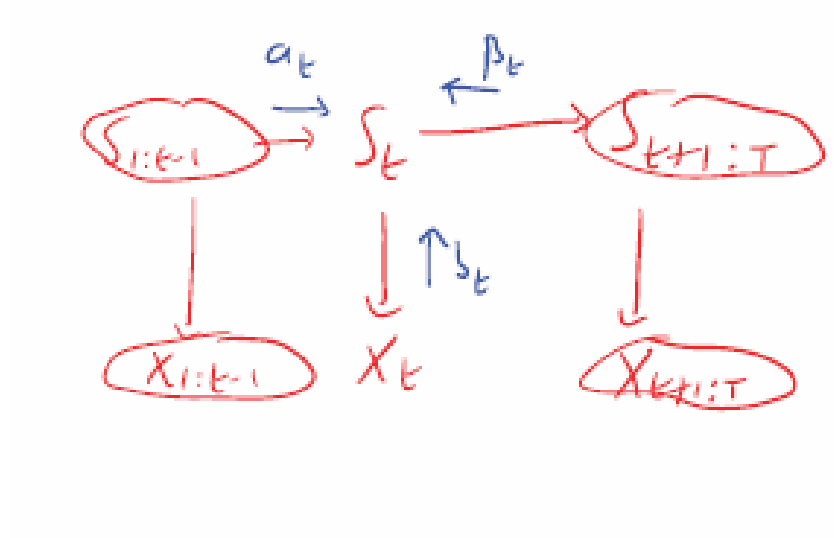


Fixed lag smoothing



nds of inference for state-space models. The shaded region is the interval for

FB



$$p(S_t|\mathbf{x}_{1:T}) \propto \sum_{\mathbf{s}_{1:t-1}} \sum_{\mathbf{s}_{t+1:T}} p(\mathbf{s}_{1:t-1}, \mathbf{x}_{1:t-1}, S_t, \mathbf{x}_t, \mathbf{s}_{t+1:T}, \mathbf{x}_{t+1:T}) \tag{1}$$

$$= \sum_{\mathbf{s}_{1:t-1}} \sum_{\mathbf{s}_{t+1:T}} p(\mathbf{s}_{1:t-1}, \mathbf{x}_{1:t-1}) p(S_t|s_{t-1}) p(\mathbf{x}_t|S_t) p(\mathbf{s}_{t+1:T}, \mathbf{x}_{t+1:T}|S_t) \tag{2}$$

$$= \sum_{s_{t-1}} p(s_{t-1}, \mathbf{x}_{1:t-1}) p(S_t|s_{t-1}) p(\mathbf{x}_t|S_t) p(\mathbf{x}_{t+1:T}|S_t) \tag{3}$$

$$\propto \sum_{s_{t-1}} p(s_{t-1}|\mathbf{x}_{1:t-1}) p(S_t|s_{t-1}) p(\mathbf{x}_t|S_t) p(\mathbf{x}_{t+1:T}|S_t) \tag{4}$$

Matrix vector form

Let us define the following notation

$$\alpha_t(j) \overset{\text{def}}{=} p(S_t = j|\mathbf{x}_{1:t}) \quad (1)$$

$$\beta_t(j) \overset{\text{def}}{=} p(\mathbf{x}_{t+1:T}|S_t = j) \quad (2)$$

$$\gamma_t(j) \overset{\text{def}}{=} p(S_t = j|\mathbf{x}_{1:T}) \quad (3)$$

Then we can rewrite the above equation as

$$\gamma_t(j) \propto \sum_i \alpha_{t-1}(i) A_{ij} b_t(j) \beta_t(j) \quad (4)$$

Furthermore, let us define the one-step ahead predictive density

$$\mathsf{a}_t(j) \overset{\text{def}}{=} p(S_t = j|\mathbf{x}_{1:t-1}) = \sum_i \alpha_{t-1}(i) A_{ij} \quad (5)$$

Then we can rewrite the above equation as

$$\gamma_t(j) \propto \mathsf{a}_t(j) b_t(j) \beta_t(j) \quad (6)$$

Backwards algorithm

$$\beta_{t-1}(i) = p(\mathbf{x}_{t+1:T}|S_{t-1}=i) \quad (1)$$

$$= \sum_j p(S_t=j, \mathbf{x}_t, \mathbf{x}_{t+1:T}|S_{t-1}=i) \quad (2)$$

$$= \sum_j p(S_t=j|S_{t-1}=i)p(\mathbf{x}_t|S_t=j, S_{t-1}=i)p(\mathbf{x}_{t+1:T}|S_t=j, S_{t-1}=i) \quad (3)$$

$$= \sum_j p(S_t=j|S_{t-1}=i)p(\mathbf{x}_t|S_t=j)p(\mathbf{x}_{t+1:T}|S_t=j) \quad (4)$$

$$= \sum_j A_{ij}b_t(j)\beta_t(j) \quad (5)$$

where Equation **??** is justi¬ ed since $X_t \perp X_{t+1:T}|S_t$ and Equation **??** is justi¬ ed since $X_t \perp S_{t-1}|S_t$ and $X_{t+1:T} \perp S_{t-1}|S_t$. We can write the resulting equation in matrix-vector form as

$$\boldsymbol{\beta}_{t-1} = \mathbf{A}(\mathbf{b}_t .* \boldsymbol{\beta}_t) \quad (6)$$

The base case is

$$\beta_T(i) = p(\mathbf{x}_{T+1:T}|S_T=i) = p(\emptyset|S_T=i) = 1 \quad (7)$$

Matlab

Listing 1: Listing of `hmmBackwards`

```
function [beta] = hmmBackwards(transmat, obslik)
% beta(i,t) propto p(y(t+1:T) | Q(t=i))
[K T] = size(obslik);
beta = zeros(K,T);
beta(:,T) = ones(K,1);
for t=T-1:-1:1
 beta(:,t) = normalize(transmat * (beta(:,t+1) .* obslik(:,t+1)));
end
\end{codeCap

\begin{codeCap}{Listing of \codename{hmmFwdBack}}
function [gamma, alpha, beta, loglik] = hmmFwdBack(initDist, transmat, obslik)
% gamma(i,t) = p(Q(t)=i | y(1:T))
[alpha, loglik] = hmmFilter(initDist, transmat, obslik);
beta = hmmBackwards(transmat, obslik);
gamma = normalize(alpha .* beta, 1);% make each column sum to 1
```

Avoiding underflow

$$\alpha_t(j) = p(S_t = j|\mathbf{x}_{1:T}) = \frac{1}{c_t} b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \quad (1)$$

$$c_t = \sum_j b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \quad (2)$$

$$\hat{\beta}_{t-1}(i) = \frac{1}{d_{t-1}} \sum_j A_{ij} b_t(j) \hat{\beta}_t(j) \quad (3)$$

$$d_{t-1} = \sum_i A_{ij} b_t(j) \hat{\beta}_t(j) \quad (4)$$

$$p(S_t = j, \mathbf{x}_{1:t}) = p(S_t = j|\mathbf{x}_{1:t}) p(\mathbf{x}_{1:t}) = \alpha_t(j) (\prod_{\tau=1}^{t} c_\tau) \quad (5)$$

$$p(\mathbf{x}_{t+1:T}|S_t = j) = \hat{\beta}_t(j) (\prod_{\tau=t}^{T} d_\tau) \quad (6)$$

Avoiding underflow

$$
\begin{aligned}
\gamma_t(j) &= p(S_t = j | x_{1:T}) &(1)\\
&= \frac{p(x_{t+1:T} | S_t = j) p(S_t = j, x_{1:t})}{p(x_{1:T})} &(2)\\
&= \frac{(\prod_{\tau=t}^{T} d_\tau) \hat{\beta}_t(j) (\prod_{\tau=1}^{t} c_\tau) \alpha_t(j)}{\sum_{j'} (\prod_{\tau=t}^{T} d_\tau) \hat{\beta}_t(j') (\prod_{\tau=1}^{t} c_\tau) \alpha_t(j')} &(3)\\
&= \frac{\beta_t(j) \alpha_t(j)}{\sum_{j'} \hat{\beta}_t(j') \alpha_t(j')} &(4)
\end{aligned}
$$

Two-slice marginals

$$N_{ij} = \sum_{t=1}^{T-1} E[I(S_t = i, S_{t+1} = j)|\mathbf{x}_{1:T}] = \sum_{t=1}^{T-1} p(S_t = i, S_{t+1} = j|\mathbf{x}_{1:T}) \qquad (1)$$

$$\begin{aligned}
\xi_{t-1,t}(i,j) &\stackrel{\text{def}}{=} p(S_{t-1} = i, S_t = j|x_{1:T}) \\
&\propto p(S_{t-1} = i|\mathbf{x}_{1:t-2})p(\mathbf{x}_{t-1}|S_{t-1} = i)p(S_t = j|S_{t-1} = i)p(\mathbf{x}_t|S_t = j)p(\mathbf{x}_{t+1:T}|S_t = j) \\
&= a_{t-1}(i)b_{t-1}(i)A_{ij}b_t(j)\beta_t(j)
\end{aligned}$$

$$\boldsymbol{\xi}_{t-1,t} \propto \mathbf{A}. * (\boldsymbol{\alpha}_{t-1} * (\mathbf{b}_t. * \boldsymbol{\beta}_t)^T) \qquad (2)$$

Time and space complexity

- $O(T\ K\ b)$ time, b = branching factor
- In discretization of cts space,
  $O(T\ K \log K)$ or $O(T\ K)$ – Felzenswalb & Huttenlocher
- $O(T\ K)$ space, $O(T\ K^2)$ time
- $O(K \log T)$ space, $O(T \log T\ K^2)$ time (island algorithm)

Viterbi

MAP path

$$s^*_{1:T} = \arg\max_{s_{1:T}} p(s_{1:T}|x_{1:T}) \quad (1)$$

Max marginals

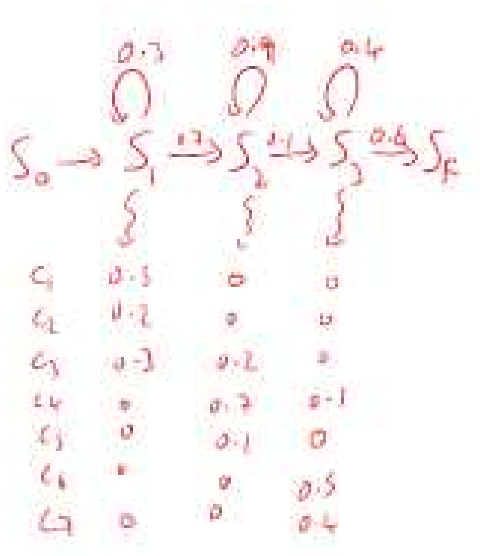
$$s^*_t = \arg\max_i p(S_t = i|\mathbf{x}_{1:T}) = \arg\max_i \sum_{\mathbf{s}_{-t}} p(S_t = i, \mathbf{s}_{-t}|\mathbf{x}_{1:T}) \quad (2)$$

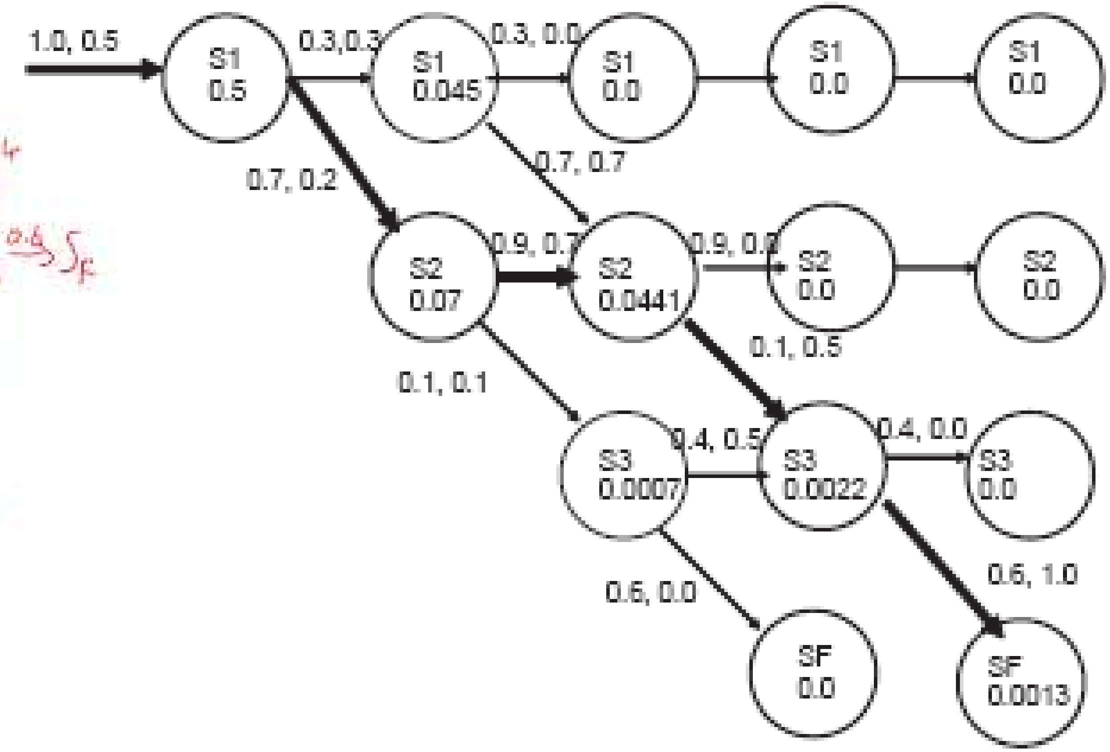
$$\begin{aligned}
\delta_t(i) &\stackrel{\text{def}}{=} \max_{s_1,\ldots,s_{t-1}} p(\mathbf{s}_{1:t-1}, s_t = i, \mathbf{x}_{1:t}|\boldsymbol{\theta}) \\
\delta_{t+1}(j) &= \max_i \delta_t(i) A_{ij} b_{t+1}(j) \\
\psi_{t+1}(j) &= \arg\max_i \delta_t(i) A_{ij} b_{t+1}(j) \\
\delta_1(j) &= \pi_j b_1(j)
\end{aligned}$$

Traceback

$$\begin{aligned}
S^*_T &= \arg\max_i \delta_T(i) \\
S^*_t &= \psi_{t+1}(s^*_{t+1})
\end{aligned}$$

Viterbi example





$$
\begin{aligned}
\delta_1(1) &= 0.5 \\
\delta_2(1) &= \delta_1(1) A_{11} b_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \\
\delta_2(2) &= \delta_1(1) A_{12} b_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07
\end{aligned}
$$

Top N list
Discrim. reranking

Fwd filtering, back sampling

$$s_{1:T}^* \sim p(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}, \boldsymbol{\theta}) \quad (1)$$

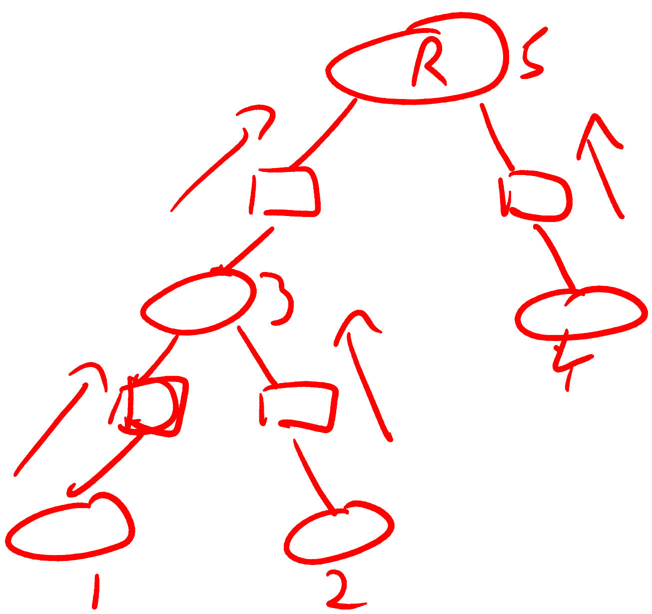
$$\begin{aligned}
s_t^* &\sim p(S_t|s_{t+1:T}^*, \mathbf{x}_{1:T}) && (2)\\
&\propto p(S_t|s_{t+1}^*, \mathbf{x}_{1:t}) && (3)\\
p(S_t = i|S_{t+1} = j, x_{1:t}) &= p(S_t = i|S_{t+1} = j, x_{1:t}, x_{t+1}) && (4)\\
&= \frac{p(S_t = i, S_{t+1} = j|x_{1:t+1})}{p(S_{t+1} = j|x_{1:t+1})} && (5)\\
&= \frac{p(\mathbf{x}_t|S_t = j)p(S_t = j|S_{t-1} = i)p(S_{t-1} = i|\mathbf{x}_{1:t-1})}{p(S_{t+1} = j|x_{1:t+1})} && (6)\\
&= \frac{A_{ij}\alpha_t(i)b_{t+1}(j)}{\alpha_{t+1}(j)} && (7)
\end{aligned}$$

Listing 1: Listing of `hmmSamplePost`

```
function [samples] = hmmSamplePost(initDist, transmat, obslik, nsamples)
% samples(t,s) = value of S(t) in sample s
[K T] = size(obslik);
alpha = hmmFilter(initDist, transmat, obslik);
samples = zeros(T, nsamples);
dist = normalize(alpha(:,T));
samples(T,:) = sample(dist, nsamples);
for t=T-1:-1:1
  tmp = obslik(:,t+1) ./ (alpha(:,t+1)+eps); % b_{t+1}(j) / alpha_{t+1}(j)
  xi_filtered = transmat .* (alpha(:,t) * tmp');
  for n=1:nsamples
    dist = xi_filtered(:,samples(t+1,n));
    samples(t,n) = sample(dist);
  end
end
```

Message passing on a clique tree

- To compute p(X_i), find a clique that contains X_i, make it the root, and send messages to it from all other nodes.
- A clique cannot send a node to its parent until it is ready, ie. Has received msgs from all its children.
- Hence we send from leaves to root.

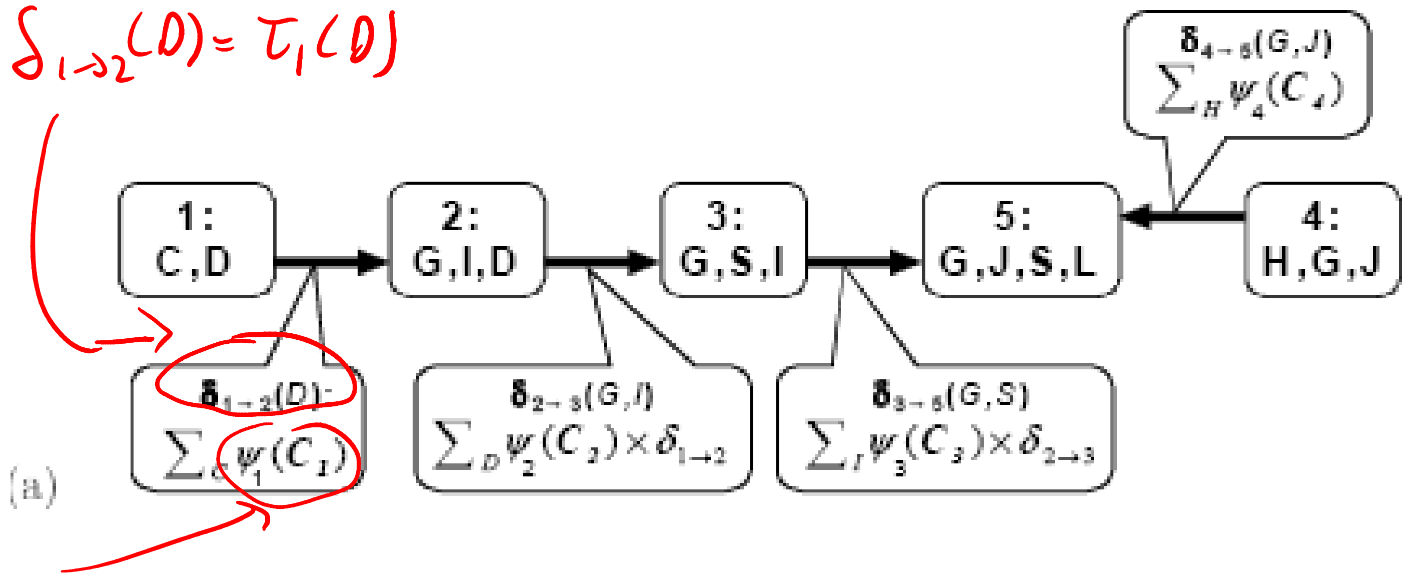


Message passing on a clique tree

- .

$$
\begin{aligned}
P(J) &= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_G(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)} \\
&= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_G(G,I,D)\tau_1(D)}_{\tau_2(G,I)}
\end{aligned}
$$

$\delta_{1\to 2}(D) = \tau_1(D)$



(a)

$\psi_1(C_1) = \psi_C(C)\,\psi_D(D,C)$

Multiply terms in bucket (local & incoming),
sum out those that are not in sepset,
send to nbr upstream

Upwards pass (collect to root)

Procedure CTree Sum Product Up (
  Φ, // Set of factors
  $\mathcal{T}$, // Clique tree over Φ
  α, // Initial assignment of factors to cliques
  C_r // Some selected root clique
)

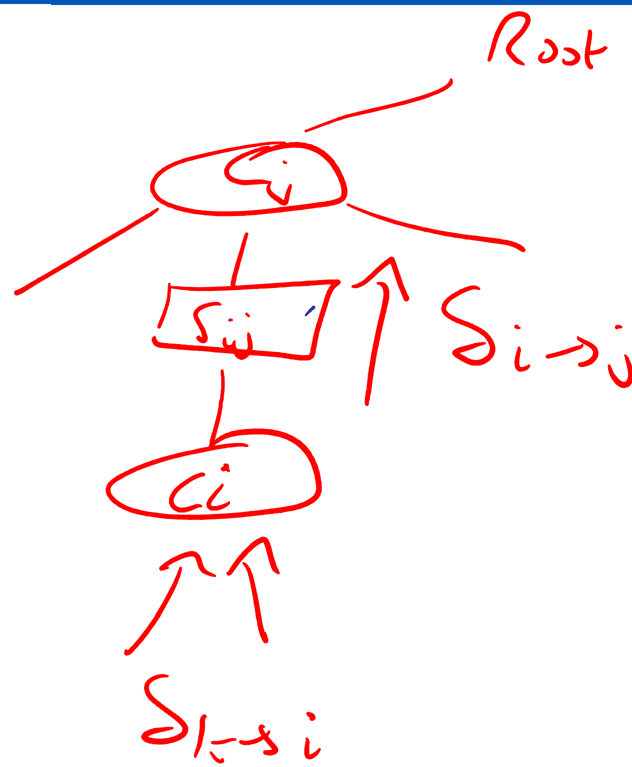
Initialize Cliques
while C_r is not ready
  Let C_i be a ready clique
  $\delta_{i \to p_r(i)}(S_{i,p_r(i)}) \leftarrow$ SP Message$(i, p_r(i))$
$\beta_r \leftarrow \psi_r \cdot \prod_{k \in \mathrm{Nb}_{C_r}} \delta_{k \to r}$
return β_r

Procedure Initialize Cliques (
)

for each clique C_i
  $\psi_i[C_i] \leftarrow \prod_{\phi_j \,:\, \alpha(\phi_j)=i} \phi$

Procedure SP Message (
  i, // sending clique
  j // receiving clique
)

$\psi(C_i) \leftarrow \psi_i \cdot \prod_{k \in (\mathrm{Nb}_i - \{j\})} \delta_{k \to i}$
$\tau(S_{i,j}) \leftarrow \sum_{C_i - S_{i,j}} \psi(C_i)$
return $\tau(S_{i,j})$

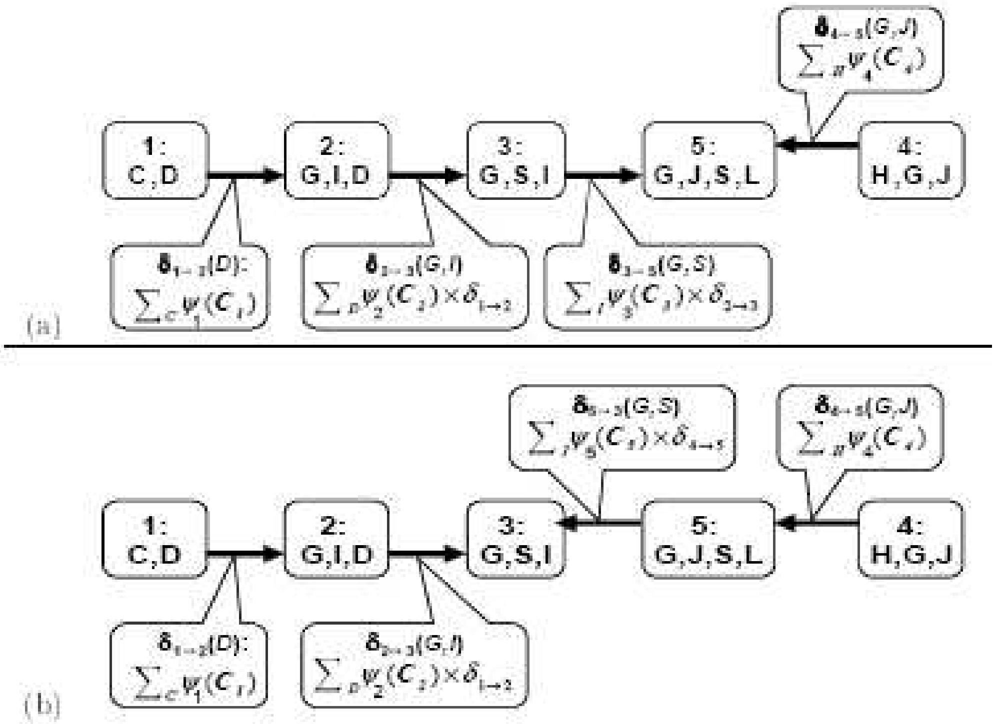


$$\beta_i(C_i) = \phi_i(C_i) \prod_{k \in n_i, k \neq j} \delta_{k \to i}(S_{k,i})$$

$$\delta_{i \to j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \beta_i(C_i)$$

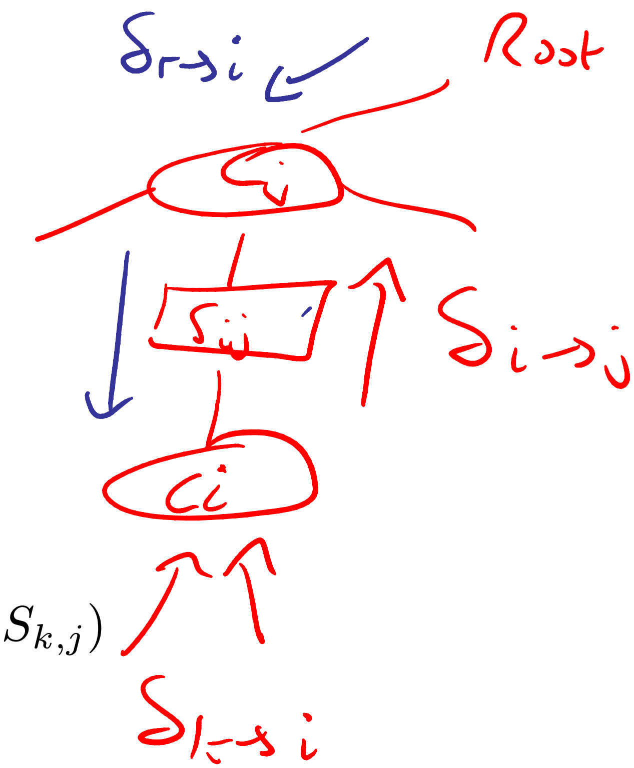
Message passing to a different root

- If we send messages to a different root, many of them will be the same
- Hence if we send messages to all the cliques, we can reuse the messages- dynamic programming!



Downwards pass (distribute from root)

- At the end of the upwards pass, the root has seen all the evidence.
- We send back down from root to leaves.



$$\beta_j(C_j) = \phi_j(C_j) \prod_{k\in n_j} \delta_{k\to j}(S_{k,j})$$

$$\delta_{j\to i}(S_{ij}) = \sum_{C_j \setminus S_{ij}} \phi_j(C_j) \prod_{k\in n_j, i\neq k} \delta_{k\to j}(S_{k,j})$$

$$= \sum_{C_j \setminus S_{ij}} \frac{\beta_j(C_j)}{\delta_{i\to j}(S_{ij})}$$

Use division operator to avoid double counting

Beliefs

- Thm 10.2.7. After collect/distribute, each clique potential represents a marginal probability (conditioned on the evidence)

$$\beta_i(C_i) = \sum_{\mathbf{x} \setminus C_i} \tilde{P}(\mathbf{x})$$

- If we get new evidence on X_i, we can multiply it in to any clique containing i, and then distribute messages outwards from that clique to restore consistency.

MAP configuration

- We can generalize the Viterbi algorithm to find a MAP configuration as follows.
- On the upwards pass, replace sum with max.
- At the root, find the most probable joint setting and send this as evidence to the root's children.
- Each child finds its most probable setting and sends this to its children.
- The jtree property ensures that when the state of a variable is fixed in one clique, that variable assumes the same state in all other cliques.

Samples

- We can generalize forwards-filtering backwards-sampling to draw exact samples from the joint as follows.
- Do a collect pass to the root as usual.
- Sample xR from the root marginal, and then enter it as evidence in all the children.
- Each child then samples itself from its updated local distribution and sends this to its children.

Calibrated clique tree

- Def 102.8. A clique tree is calibrated if, for all pairs of neighboring cliques, we have

$$\sum_{C_i \backslash S_{i,j}} \beta_i(C_i) = \sum_{C_j \backslash S_{i,j}} \beta_j(C_j) = \mu_{i,j}(S_{i,j})$$

- Eg. A-B-C clq tree AB – [B] – BC. We require

$$\sum_a \beta_{ab}(a,b) = \sum_c \beta_{bc}(b,c)$$

- Thm. After collect/distribute, all cliques are calibrated.
- Thm 10.2.12. A calibrated tree defines a joint distribution as follows $p(x) = \frac{\prod_i \beta_i(C_i)}{\prod_{<ij>} \mu_{i,j}(S_{ij})}$

eg $$p(A,B,C) = \frac{p(A,B)p(B,C)}{p(C)} = p(A,B)p(C|B) = p(A|B)p(B,C)$$

Clique tree invariant

- Suppose at every step, clique i sends a msg to clique j, and stores it in $\mu_{i,j}$:

```
Procedure Send-BU-Msg (
    i,   // sending clique
    j    // receiving clique
)
    σ_{i→j} ← Σ_{C_i − S_{i,j}} β_i
        // marginalize the clique over the sepset
    β_j ← β_j · σ_{i→j} / μ_{i,j}
    μ_{i,j} ← σ_{i→j}
```

- Initially $\mu_{i,j}=1$ and $\beta_i = \prod_{f:\ f \text{ ass to } i} \phi_f$. Hence the following holds.

$$p(x) = \frac{\prod_i \beta_i(C_i)}{\prod_{<ij>} \mu_{i,j}(S_{ij})}$$

- Thm 10.3.4. This property holds after every belief updating operation.

Out of clique queries

- We can compute the distribution on any set of variables inside a clique. But suppose we want the joint on variables in different cliques. We can run VE on the calibrated subtree
- eg

$A - B - C - D \qquad AB - BC - CD$

$$p(B,D) = \sum_C p(B C D)$$

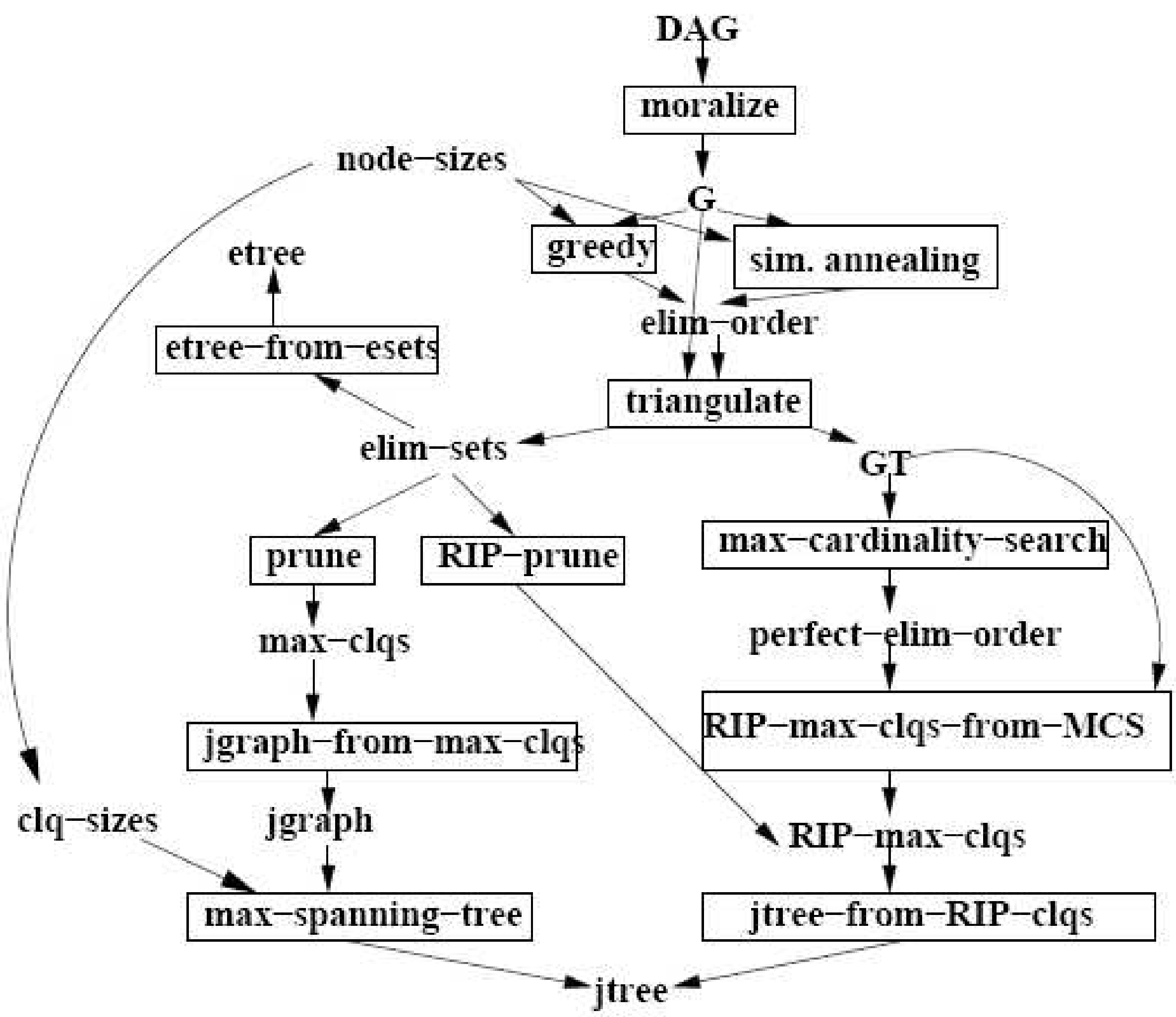
$$= \sum_C \frac{\beta_2(BC)\,\beta_3(CD)}{\mu_{23}(C)}$$

$$= \sum_C p(B|C)\,p(C,D)$$

Out of clique inference

Procedure CTree-Query (
  $\mathcal{T}$, // Clique tree over Φ
  $\{\beta_i\}, \{\mu_{i,j}\}$, // Calibrated clique and sepset beliefs for $\mathcal{T}$
  $\boldsymbol{Y}$ // A query
)
  Let $\mathcal{T}'$ be a subtree of $\mathcal{T}$ such that $\boldsymbol{Y} \subseteq Scope[\mathcal{T}']$
  Select a clique $r \in \mathcal{V}_{\mathcal{T}'}$ to be the root
  $\Phi \leftarrow \beta_r$
  for each $i \in \mathcal{V}'_{\mathcal{T}}$
    $\phi \leftarrow \frac{\beta_i}{\mu_{i,Pr(i)}}$
    $\Phi \leftarrow \Phi \cup \{\phi\}$
  $\boldsymbol{Z} \leftarrow Scope[\mathcal{T}'] - \boldsymbol{Y}$
  Let $\prec$ be some ordering over $\boldsymbol{Z}$
  return Sum-Product-Variable-Elimination($\Phi, \boldsymbol{Z}, \prec$)

Creating a Jtree



Murphy PhD thesis (2002) p140

Max cliques from a chordal graph

- Triangulate the graph according to some ordering.

  - Start with all vertices unnumbered, set counter $i := N$.
  - While there are still some unnumbered vertices:
    - Let $v_i = \pi(i)$.
    - Form the set C_i consisting of v_i and its (unnumbered/ uneliminated) neighbors.
    - Fill in edges between all pairs of vertices in C_i.
    - Eliminate v_i and decrement i by 1.

- At each step, keep track of the clique that is created; if it is a subset of any previously created clique, discard it (since non maximal).

Cliques to Jtree

- Build a weighted graph where
  $W_{ij} = |C_i \text{ intersect } C_j|$
- Find max weight spanning tree. This is a jtree.