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 one-step-ahead predictive density \( p(S_t|x_{1:t-1}) \) as follows:

\[
p(S_t = j|x_{1:t-1}) = \sum_i p(S_t = j, S_{t-1} = i|x_{1:t-1}) = \sum_i p(S_t = j|S_{t-1} = i)p(S_{t-1} = i|x_{1:t-1})
\]

(1)

(2)

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

2. update: compute \( p(S_t|x_t, x_{1:t-1}) \) using Bayes rule, where we use \( p(S_t|x_{1:t-1}) \) as the prior:

\[
p(S_t = j|x_t) = \frac{1}{c_t} p(x_t|S_t = j)p(S_t = j|x_{1:t-1})
\]

(3)

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

\[
c_t = p(x_t|x_{1:t-1}) = \sum_j p(x_t|S_t = j)p(S_t = j|x_{1:t-1})
\]

(4)

The base case is

\[
p(S_1 = j|x_1) \propto p(S_1 = j)p(x_1|S_1 = j) = \pi_j p(x_1|S_1 = j)
\]

(5)
Matrix vector form

\[ \alpha_t(j) = p(S_t = j | x_{1:t}) \]  
\[ b_t(j) = p(x_t | S_t = j) \]  
\[ A(i, j) = p(S_t = j | S_{t-1} = i) \]

Hence the recursion step is

\[ \alpha_t(j) \propto b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \]  

This can be rewritten in matrix-vector notation as

\[ \alpha_t \propto \text{diag}(b_t) A^T \alpha_{t-1} \]

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

\[ \alpha_t \propto b_t .* (A^T \alpha_{t-1}) \]

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

\[ \log p(x_{1:T}) = \sum_{t=1}^{T} \log p(x_t | x_{1:t-1}) = \sum_{c=1}^{T} \log c_t \]
Listing 1: Listing of `hmmFilter`

```matlab
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`

```matlab
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:

\[ \gamma_t(j) \overset{\text{def}}{=} p(S_t = j | x_{1:T}, \theta) \]  

(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

\[ \xi_{t-1,t}(i, j) \overset{\text{def}}{=} p(S_{t-1} = i, S_t = j | x_{1:T}, \theta) \]  

(2)

These are needed for parameter estimation, as described in Section ??.

- The posterior mode, or most probable path:

\[ s^*_1:T = \arg \max_{s_{1:T}} p(s_{1:T} | x_{1:T}, \theta) \]  

(3)

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

- Samples from the posterior

\[ s_{1:T} \sim p(s_{1:T} | x_{1:T}, \theta) \]  

(4)

These can be generated by the forwards-backwards algorithm, as described in Section ??.
Filtering vs smoothing vs Viterbi
Fixed lag smoothing

Ends of inference for state-space models. The shaded region is the interval for
some statistic, which is extended between all the observations until the

\begin{align*}
p(S_t | x_{1:T}) & \propto \sum_{s_{1:t-1}} \sum_{s_{t+1:T}} p(s_{1:t-1}, x_{1:t-1}, S_t, x_t, s_{t+1:T}, x_{t+1:T}) \\
& = \sum_{s_{1:t-1}} \sum_{s_{t+1:T}} p(s_{1:t-1}, x_{1:t-1}) p(S_t | s_{t-1}) p(x_t | S_t) p(s_{t+1:T}, x_{t+1:T} | S_t) \\
& = \sum_{s_{t-1}} p(s_{t-1}, x_{1:t-1}) p(S_t | s_{t-1}) p(x_t | S_t) p(x_{t+1:T} | S_t) \\
& \propto \sum_{s_{t-1}} p(s_{t-1} | x_{1:t-1}) p(S_t | s_{t-1}) p(x_t | S_t) p(x_{t+1:T} | S_t)
\end{align*}
Let us define the following notation

\[ \alpha_t(j) \overset{\text{def}}{=} p(S_t = j | x_{1:t}) \] (1)
\[ \beta_t(j) \overset{\text{def}}{=} p(x_{t+1:T} | S_t = j) \] (2)
\[ \gamma_t(j) \overset{\text{def}}{=} p(S_t = j | x_{1:T}) \] (3)

Then we can rewrite the above equation as

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

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

\[ a_t(j) \overset{\text{def}}{=} p(S_t = j | x_{1:t-1}) = \sum_i \alpha_{t-1}(i) A_{ij} \] (5)

Then we can rewrite the above equation as

\[ \gamma_t(j) \propto a_t(j) b_t(j) \beta_t(j) \] (6)
Backwards algorithm

\[ \beta_{t-1}(i) = p(x_{t+1:T}|S_{t-1} = i) \]
\[ = \sum_j p(S_t = j, x_t, x_{t+1:T}|S_{t-1} = i) \]
\[ = \sum_j p(S_t = j|S_{t-1} = i)p(x_t|S_t = j, S_{t-1} = i)p(x_{t+1:T}|S_t = j, S_{t-1} = i) \]
\[ = \sum_j p(S_t = j|S_{t-1} = i)p(x_t|S_t = j)p(x_{t+1:T}|S_t = j) \]
\[ = \sum_j A_{ij} b_t(j) \beta_t(j) \]

where Equation (2) is justified since \( X_t \perp X_{t+1:T}|S_t \) and Equation (3) is justified 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

\[ \beta_{t-1} = A(b_t \ast \beta_t) \]

The base case is

\[ \beta_T(i) = p(x_{T+1:T}|S_T = i) = p(\emptyset|S_T = i) = 1 \]
Listing 1: Listing of `hmmBackwards`

```matlab
function [beta] = hmmBackwards(transmat, obslik)
    % beta(i,t) = 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
```

```matlab
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
end
```
Avoiding underflow

\[ \alpha_t(j) = p(S_t = j | x_{1:T}) = \frac{1}{c_t} b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \]  

(1)

\[ c_t = \sum_j b_t(j) \sum_i A_{ij} \alpha_{t-1}(i) \]  

(2)

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

(3)

\[ d_{t-1} = \sum_i A_{ij} b_t(j) \hat{\beta}_t(j) \]  

(4)

\[ p(S_t = j, x_{1:t}) = p(S_t = j | x_{1:t}) p(x_{1:t}) = \alpha_t(j)(\prod_{\tau=1}^{t} c_\tau) \]  

(5)

\[ p(x_{t+1:T} | S_t = j) = \hat{\beta}_t(j)(\prod_{\tau=t}^{T} d_\tau) \]  

(6)
Avoiding underflow

\[
\gamma_t(j) = p(S_t = j|x_{1:T}) = \frac{p(x_{t+1:T}|S_t = j)p(S_t = j, x_{1:t})}{p(x_{1:T})}
\]

\[
= \frac{(\prod_{\tau=t}^{T} d_{\tau}) \hat{\beta}_t(j)(\prod_{\tau=1}^{t} c_{\tau}) \alpha_t(j)}{\sum_{j'}(\prod_{\tau=t}^{T} d_{\tau}) \hat{\beta}_t(j')(\prod_{\tau=1}^{t} c_{\tau}) \alpha_t(j')}
\]

\[
= \frac{\beta_t(j)\alpha_t(j)}{\sum_{j'}\beta_t(j')\alpha_t(j')}
\]
Two-slice marginals

\[ N_{ij} = \sum_{t=1}^{T-1} E[I(S_t = i, S_{t+1} = j)|x_{1:T}] = \sum_{t=1}^{T-1} p(S_t = i, S_{t+1} = j|x_{1:T}) \tag{1} \]

\[ \xi_{t-1,t}(i,j) \overset{\text{def}}{=} p(S_{t-1} = i, S_t = j|x_{1:T}) \]
\[ \propto p(S_{t-1} = i|x_{1:t-2})p(x_{t-1}|S_{t-1} = i)p(S_t = j|S_{t-1} = i)p(x_t|S_t = j)p(x_{t+1:T}|S_t = j) \]
\[ = a_{t-1}(i)b_{t-1}(i)A_{ij}b_t(j)\beta_t(j) \]

\[ \xi_{t-1,t} \propto A.*(\alpha_{t-1}.*(b_t.*\beta_t)^T) \tag{2} \]
Time and space complexity

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

\[ s_{1:T}^* = \arg \max_{s_{1:T}} p(s_{1:T} | x_{1:T}) \]  \hspace{1cm} (1)

Max marginals

\[ s_t^* = \arg \max_i p(S_t = i | x_{1:T}) = \arg \max_i \sum_{s_{-t}} p(S_t = i, s_{-t} | x_{1:T}) \]  \hspace{1cm} (2)

\[ \delta_t(i) \overset{\text{def}}{=} \max_{s_1, \ldots, s_{t-1}} p(s_{1:t-1}, s_t = i, x_{1:t} | \theta) \]

\[ \delta_{t+1}(j) = \max_i \delta_t(i) A_{ij} b_{t+1}(j) \]

\[ \psi_{t+1}(j) = \arg \max_i \delta_t(i) A_{ij} b_{t+1}(j) \]

\[ \delta_1(j) = \pi_j b_1(j) \]

Traceback

\[ S_T^* = \arg \max_i \delta_T(i) \]

\[ S_t^* = \psi_{t+1}(s_{t+1}^*) \]
\[ \begin{align*}
\delta_1(1) &= 0.5 \\
\delta_2(1) &= \delta_1(1) A_{11} b_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \\
\delta_2(2) &= \delta_1(1) A_{12} b_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07
\end{align*} \]
Fwd filtering, back sampling

\[ s_{1:T}^* \sim p(s_{1:T} | x_{1:T}, \theta) \]  \hspace{1cm} (1)

\[ s_t^* \sim p(S_t | s_{t+1:T}^*, x_{1:T}) \]

\[ \propto p(S_t | s_{t+1}^*, x_{1:t}) \]  \hspace{1cm} (3)

\[ p(S_t = i | S_{t+1} = j, x_{1:t}) = p(S_t = i | S_{t+1} = j, x_{1:t}, x_{t+1}) \]

\[ = \frac{p(S_t = i, S_{t+1} = j | x_{1:t+1})}{p(S_{t+1} = j | x_{1:t+1})} \]  \hspace{1cm} (4)

\[ = \frac{p(x_t | S_t = j)p(S_t = j | S_{t-1} = i)p(S_{t-1} = i | x_{1:t-1})}{p(S_{t+1} = j | x_{1:t+1})} \]  \hspace{1cm} (6)

\[ = \frac{A_{ij} \alpha_t(i) b_{t+1}(j)}{\alpha_{t+1}(j)} \]  \hspace{1cm} (7)

**Listing 1: Listing of `hmmSamplePost`**

```matlab
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
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, i.e., has received messages from all its children.
- Hence we send from leaves to root.
Message passing on a clique tree

\[ P(J) = \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \sum_D \psi_G(G, I, D) \sum_{\tau_1(D)} \psi_{C}(C) \psi_D(D, C) \]

\[ = \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I) \psi_I(I) \sum_D \psi_G(G, I, D) \tau_1(D) \tau_2(G, I) \]

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

\[ \psi_i(C_i) = \psi_C(C) \psi_0(D, C) \]

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()

\( \Phi \), // Set of factors
\( T \), // Clique tree over \( \Phi \)
\( \alpha \), // Initial assignment of factors to cliques
\( C_r \), // Some selected root clique

1. Initialize Cliques
2. while \( C_r \) is not ready
   3. Let \( C_t \) be a ready clique
   4. \( \delta_{i\rightarrow p_r(i)}(S_{t,p_r(i)}) \leftarrow \text{SP Message}(i, p_r(i)) \)
   5. \( \beta_r \leftarrow \psi_r \cdot \prod_{k \in \text{Nb}_{C_r}} \delta_{k\rightarrow r} \)
6. return \( \beta_r \)

Procedure Initialize Cliques()

1. for each clique \( C_i \)
   2. \( \psi_i[C_i] \leftarrow \prod_{\phi_j : \alpha(\phi_j) = \psi} \phi \)

Procedure SP Message(i, j)

1. \( \psi(C_i) \leftarrow \psi_i \cdot \prod_{k \in (\text{Nb}_i - \{j\})} \delta_{k\rightarrow i} \)
2. \( \tau(S_{t,j}) \leftarrow \sum_{C_i-S_{t,j}} \psi(C_i) \)
3. return \( \tau(S_{t,j}) \)

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

\( \delta_{i\rightarrow j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \beta_i(C_i) \)
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.

\[
\beta_j(C_j) = \phi_j(C_j) \prod_{k \in n_j} \delta_{k \rightarrow j}(S_{k,j})
\]

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

\[
= \sum_{C_j \setminus S_{ij}} \frac{\beta_j(C_j)}{\delta_{i \rightarrow j}(S_{ij})} \quad \text{Use division operator to avoid double counting}
\]
Beliefs

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

\[ \beta_i(C_i) = \sum_{x \in C_i} \tilde{P}(x) \]

- 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 $x_R$ 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

\[
\sum_{C_i \setminus S_{i,j}} \beta_i(C_i) = \sum_{C_j \setminus S_{i,j}} \beta_j(C_j) = \mu_{i,j}(S_{i,j})
\]

- **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
                      )
  1. $\sigma_{i \rightarrow j} = \sum_{C_i} S_{i,j} \beta_i$
  2. // marginalize the clique over the sepset
  3. $\beta_j \leftarrow \beta_j \cdot \frac{\sigma_{i \rightarrow j}}{\mu_{i,j}}$
  4. $\mu_{i,j} \leftarrow \sigma_{i \rightarrow j}$
  ```

- 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

\[ A - B - C - D \quad A B - B C - C D \]

\[ P(B, D) = \sum_c P(B|C) \]

\[ = \sum_c \beta_2 (B|C) \beta_3 (C|D) \]

\[ = \sum_c \beta_2 (B|C) \beta_3 (C|D) \]

\[ = \sum_c P(B|C) P(C|D) \]
Out of clique inference

Procedure CTree-Query (
    T, // Clique tree over \( \Phi \)
    \{\beta_i\}, \{\mu_{i,j}\}, // Calibrated clique and sepset beliefs for \( T \)
    Y // A query
)

Let \( T' \) be a subtree of \( T \) such that \( Y \subseteq \text{Scope}[T'] \)
Select a clique \( r \in \mathcal{V}_{T'} \) to be the root
\( \Phi \leftarrow \beta_r \)
for each \( i \in \mathcal{V}_{T'} \)
\( \phi \leftarrow \frac{\beta_i}{\mu_{i,Pr(i)}} \)
\( \Phi \leftarrow \Phi \cup \{\phi\} \)
\( Z \leftarrow \text{Scope}[T'] - Y \)
Let \( \prec \) be some ordering over \( Z \)
return Sum-Product-Variable-Elimination(\( \Phi, Z, \prec \)
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.