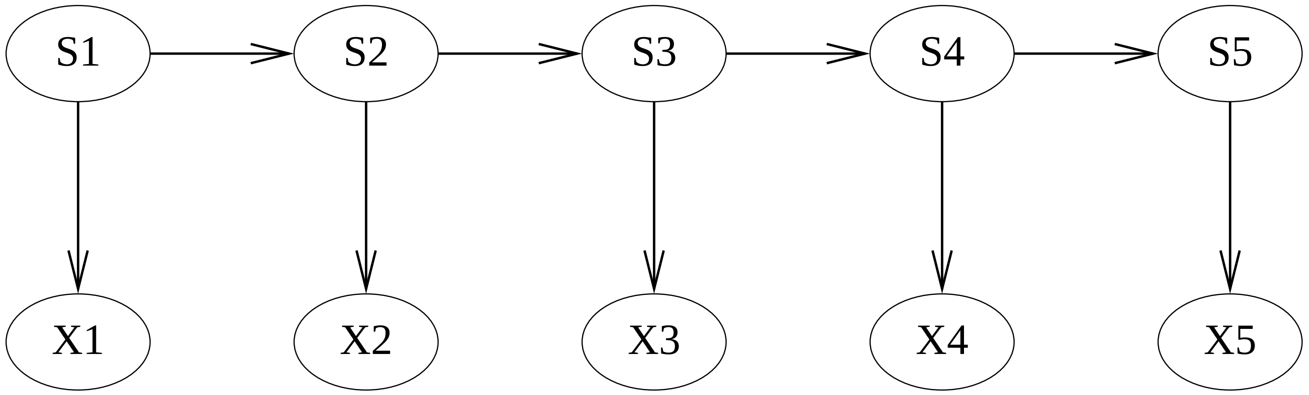
Hidden Markov Models



10 probability distributions to learn:

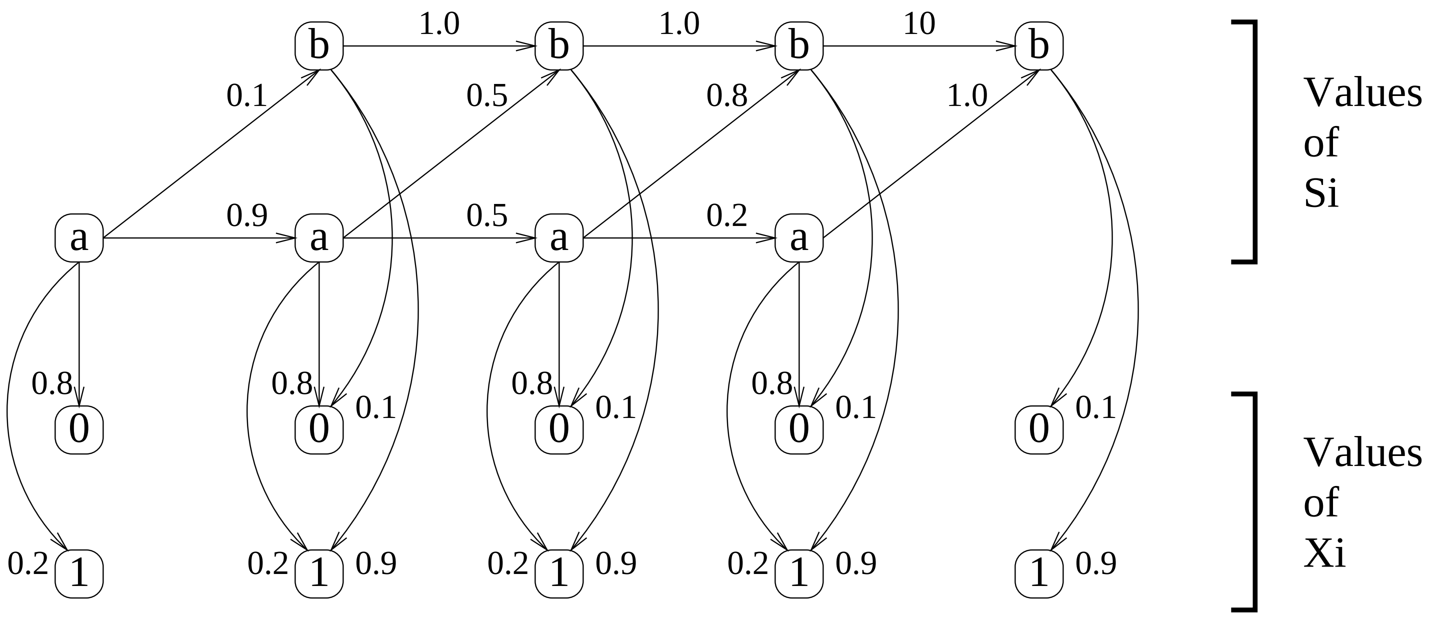
$\mathbf{P}(S_1), \mathbf{P}(S_2|S_1), \mathbf{P}(S_3|S_2), \mathbf{P}(S_4|S_3), \mathbf{P}(S_5|S_4)$

$\mathbf{P}(X_1|S_1), \mathbf{P}(X_2|S_2), \mathbf{P}(X_3|S_3), \mathbf{P}(X_4|S_4), \mathbf{P}(X_5|S_5)$

19 parameters if all variables are boolean.

At each time t, the observed output is a mixture of two binomial distributions.

Example HMM



Note that the two states are called a and b and the two outputs are called 0 and 1.

The output distribution is *stationary* (the same for all times t), but the transition distribution is *non-stationary*. In other words,

$$\mathbf{P}(X_1|S_1) = \mathbf{P}(X_2|S_2) = \mathbf{P}(X_3|S_3) = \mathbf{P}(X_4|S_4) = \mathbf{P}(X_5|S_5)$$

Sometimes we say that the output distributions are "tied together". If at learning time we *knew* this, then the number of parameters to be learned would only be 11.

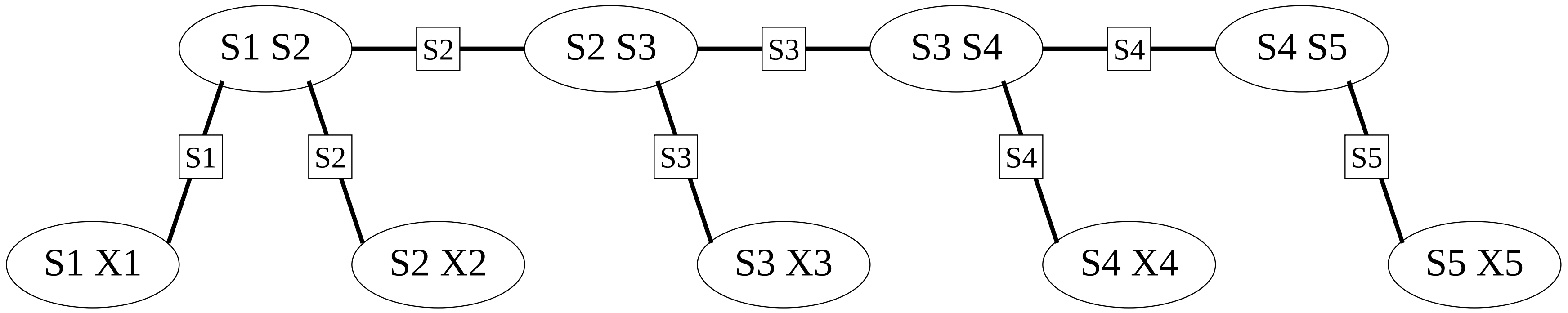
Example HMM (2)

Example strings generated by this HMM:

```
(s1 . x1)(s2 . x2)(s3 . x3)(s4 . x4)(s5 . x5)
((0 . 0)  (0 . 1)  (0 . 0)  (1 . 0)  (1 . 1))
((0 . 0)  (0 . 0)  (0 . 0)  (1 . 1)  (1 . 1))
((0 . 0)  (0 . 0)  (0 . 0)  (0 . 0)  (1 . 1))
((0 . 0)  (0 . 0)  (1 . 1)  (1 . 1)  (1 . 1))
((0 . 1)  (0 . 0)  (1 . 1)  (1 . 1)  (1 . 1))
((0 . 0)  (0 . 1)  (1 . 1)  (1 . 1)  (1 . 0))
((0 . 0)  (0 . 0)  (1 . 0)  (0 . 0)  (1 . 1))
((0 . 1)  (0 . 0)  (1 . 1)  (1 . 1)  (1 . 1))
((0 . 0)  (0 . 0)  (1 . 1)  (1 . 1)  (1 . 1))
((0 . 0)  (0 . 1)  (0 . 0)  (0 . 0)  (1 . 1))
```

The HMM Junction Tree

The HMM is a tree-structured network (each node has only one parent). So the JT is easy to compute:



HMM Inference 1: Computing $P(X)$

Suppose we would like to know the probability that the HMM will generate a particular string $X = X_1X_2X_3X_4X_5$, for example, $X = \texttt{00000}$.

What we want to compute is

$$\sum_S \mathbf{P}(S_1)\mathbf{P}(S_2|S_1)\mathbf{P}(S_3|S_2)\mathbf{P}(S_4|S_3)\mathbf{P}(S_5|S_4)\mathbf{P}(X_1|S_1)\mathbf{P}(X_2|S_2)\mathbf{P}(X_3|S_3)\mathbf{P}(X_4|S_4)\mathbf{P}(X_5|S_5)$$

for some particular X_i's.

If we eliminate the variables in the order S_1, S_2, $\ldots$, we get what is known as the "forward algorithm":

$$\begin{aligned} P(X) \;=\; & \sum_{S_5} P(X_5|S_5)\left[\sum_{S_4} P(S_5|S_4)P(X_4|S_4)\left[\sum_{S_3} P(S_4|S_3)P(X_3|S_3)\left[\sum_{S_2} P(S_3|S_2)P(X_2|S_2)\right.\right.\right. \\ & \left.\left.\left.\left[\sum_{S_1} P(S_2|S_1)P(X_1|S_1)\left[P(S_1)\right]\right]\right]\right]\right] \end{aligned}$$

HMM Inference 1: The Forward Algorithm

Let

$$\begin{aligned}\alpha(0) &= \mathbf{P}(S_1)\\ \alpha(i) &= \sum_{S_i} \mathbf{P}(S_{i+1}|S_i)P(X_i|S_i)\alpha(i-1)\end{aligned}$$

We can view this as message passing from left-to-right along the junction tree. But when we multiply by $P(X_i|S_i)$ we are only considering the one value of X_i of interest (or equivalently, we apply evidence, which sets the other values to zero and then propagate them all).

Note that we *do not* normalize at the end of the propagation.

This is equivalent to

- **Apply evidence**. Apply the evidence $X = \texttt{00000}$.
- $\textsc{collectEvidence}(S_5X_5)$
- **Sum**: $\sum_{S_5} \mathbf{P}[S_5]$

HMM Inference 1: The Forward Algorithm: Example

$P(\mathsf{00000})$:

$\alpha(0) = [1.0, 0.0]$

$\alpha(1) = \Sigma_{S_1} \mathbf{P}(S_2|S_1) \cdot P(X_1|S_1) \cdot \alpha(0)$

$\mathbf{P}(S_2\|S_1)$	$S_1 = 0$	$S_1 = 1$
$S_2 = 0$	.9	NA
$S_2 = 1$	.1	NA

$\cdot$

$\mathbf{P}(X_1\|S_1)$	$S_1 = 0$	$S_1 = 1$
	.8	NA

$\cdot$

$\alpha(0)$	$S_1 = 0$	$S_1 = 1$
	1.0	0.0

$=$

$P[S1, S2]$	$S_1 = 0$	$S_1 = 1$
$S_2 = 0$	0.72	0
$S_2 = 1$	0.08	0

Summing over S_1 gives

$\alpha(1) =$

$\alpha(1)$	$S_2 = 0$	$S_2 = 1$
	0.72	0.08

HMM Inference 1: The Forward Algorithm: Example (2)

Messages that are passed:

i	$P(S_i = 0)$	$P(S_i = 1)$
1	1.0000	0.0000
2	0.7200	0.0800
3	0.2880	0.2960
4	0.0474	0.2190
5	0.0000	0.0598

Final probability $P(\texttt{00000}) = 0.00598$.

HMM Inference 2: Computing most likely state sequence

Suppose we want to know what was the most likely sequence of states. This is a special case of the following general problem:

Given: a belief network over variables U
evidence E for some of those variables

Find: the values u of the variables U such that
$u = \operatorname*{argmax}_u P(U = u|E)$

The general algorithm is called finding the *Most Probable Explanation* (MPE)

A more efficient special case algorithm for HMMs is called the *Viterbi Algorithm*

HMM Inference 2: Example

$\mathbf{P}(A,B)$	$A=a_1$	$A=a_2$
$B=b_1$	0.3	0.2
$B=b_2$	0.1	0.4

$\mathbf{P}(B,C)$	$B=b_1$	$B=b_2$
$C=c_1$	0.1	0.35
$C=c_2$	0.4	0.15

The configuration of the $\mathbf{P}(A,B)$ table with highest probability is (a_2, b_2).

The configuration of the $\mathbf{P}(B,C)$ table with highest probability is (b_1, c_2).

How can we combine these to get the most likely configuration of the joint distribution?

$$\begin{aligned}
\max_{A,B,C} \mathbf{P}(A,B,C) &= \max_{A,B,C} \frac{\mathbf{P}(A,B)\mathbf{P}(B,C)}{\mathbf{P}(B)} \\
&= \max_{A,B} \left(\max_{C} \frac{\mathbf{P}(A,B)\mathbf{P}(B,C)}{\mathbf{P}(B)} \right) \\
&= \max_{A,B} \mathbf{P}(A,B) \frac{\max_C \mathbf{P}(B,C)}{\mathbf{P}(B)} \\
&= \max_{A,B} \mathbf{P}(A,B) \frac{[0.40\ 0.35]}{[0.50\ 0.50]} \\
&= \max_{A,B} \mathbf{P}(A,B)[0.8\ 0.7] \\
&= \max_{A,B} [0.24\ 0.16\ 0.07\ 0.28] \\
&= 0.28
\end{aligned}$$

HMM Inference 2: Max Propagation in the Junction Tree

Max Message Propagation:

- **Compute max message:** $T_s^* = \max_{V-S} \mathbf{P}[V]$
  where V is the set of variables at a cluster node, and S is the set of variables in the separator node.
- **Update:** $T_2 := T_2 \cdot (T_s^*/T_s)$
- **Update the separator table:** $T_s := T_s^*$

CollectMax: Exactly like CollectEvidence, except that it uses Max propagation instead of marginalizing propagation.

DistributeMax: Exactly like DistributeEvidence, except that it uses Max propagation instead of marginalizing propagation.

HMM Inference 2: Computing A Most Likely Configuration

FindMax:

- **Choose a cluster node** N
- **Perform** CollectMax(N)
- **Perform** DistributeMax(N)
- **Remove ties**. If there is a cluster node with two configurations v_1 and v_2 that both are maximal for that node, then (a) choose one, (b) enter it as evidence, (c) call FindMax recursively.

The maximum configuration at each node is now part of a global maximum configuration.

The Viterbi Algorithm

If we only want one most likely sequence, we can perform only the CollectMax operation, and then do a “traceback”. This leaves the junction tree in an inconsistent state, however.

Recall in our example, $\max_{A,B}$ was attained by (a_2, b_2).

We can therefore propagate the b_2 back across the separator and compute

$$c_1 = \operatorname*{argmax}_C \mathbf{P}(b_2, C)$$

In fact, we could keep back pointers so that when we computed $\max_C \mathbf{P}(B, C)$, we remember the following table:

$\mathbf{P}(B)$	$\max_C$	argmax_C
$B = b_1$	0.35	c_2
$B = b_2$	0.40	c_1

When we learn that b_2 was best, then we know that c_1 was best.

The Viterbi Algorithm (2): Formal Statement

- **Perform** CollectMax **using the final HMM state S_f**

  For each value of S_i, we remember the value of S_{i-1} that achieved the maximum.

- **Compute the value of S_f with maximum probability**

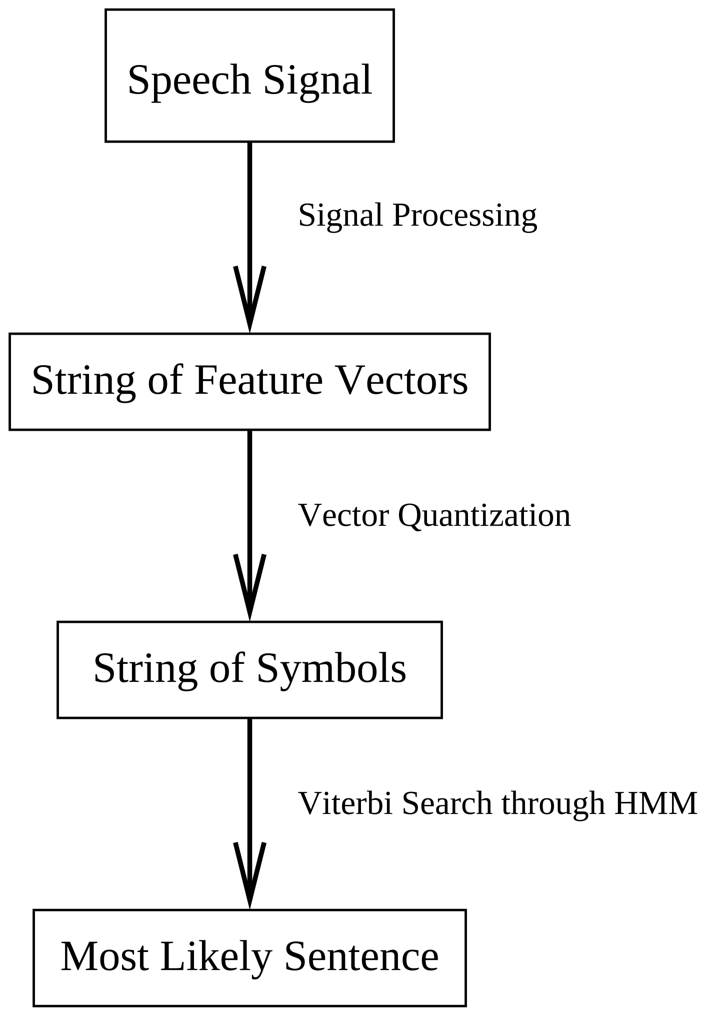
- **Follow the back pointers to determine the best path**

HMM Inference 3: The Forward-Backward Algorithm

If we perform Hugin message passing given an observed string X, then we can answer all marginals query $\mathbf{P}(S_i)$ for all i.

This is called the forward-backward or Baum-Welch algorithm, because if we perform Hugin propagation from the final state S_f, messages are first passed forward and then passed backward.

Applications of HMMs in Speech Recognition



Signal Processing

The goal is to get a feature vector with features that describe the amount of energy in each of several perceptually important frequency ranges.

This is done by sampling the speech at high frequency (e.g., 16kHz), taking 10ms "windows" of the signal, fitting an autoregressive linear model that tries to predict the signal in the current window based on the last 14 windows, and extracting the coefficients of that linear model.

These coefficients correspond to different frequency ranges, but they are evenly spaced. They can be transformed into the spacing believed to be important for human speech perception by the *cepstral* transformation. This creates a nonlinear spacing of the intensity values, so the cepstral coefficients are transformed to a mel scale.

A typical set of features is the following:

- **12 LPC cepstral coefficients**
- **the difference between the 12 coefficients in the window at $t + 2$ and the window at $t - 2$**
- **the power in this window, and the differenced power ($t + 2$ minus $t - 2$)**

This gives 26 features.

Vector Quantization

A VQ codebook is a set of (e.g., 256) vectors $c_1, \ldots, c_{256}$.
To encode a new vector v, we find the nearest vector c_i in the codebook and return its index i.

The codebook is constructed by taking a large speech sample, extracting features, and applying k-means clustering (i.e., hard EM) to find 256 cluster centers (i.e., the latent variable has 256 values).

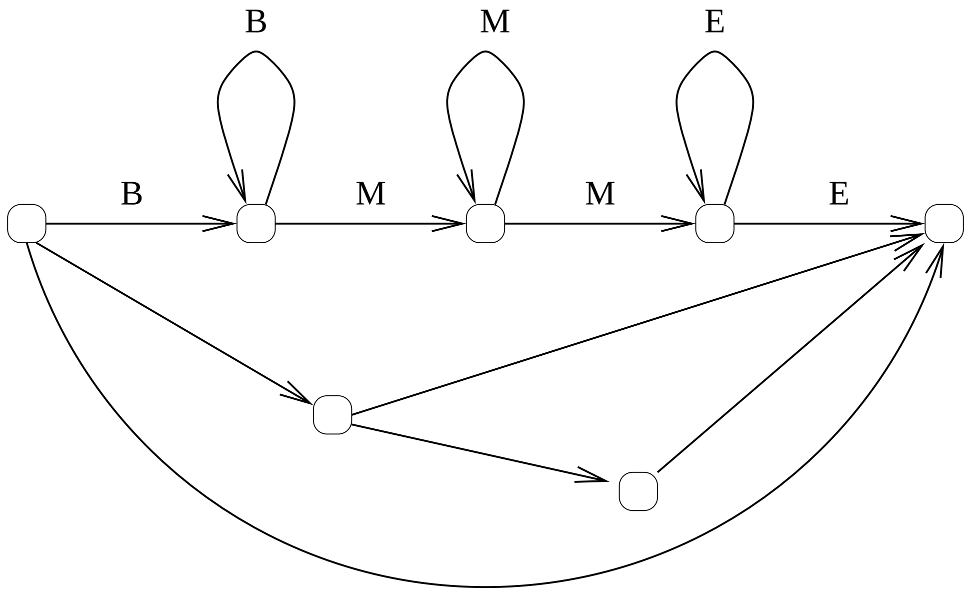
Constructing a Speech HMM

A Speech HMM is built by combining three levels of model:

- **Word sequences:** Given a large body of words (e.g., newspaper stories, closed-caption transcriptions), form a probabilistic model over strings of words.
  - *bigram model*: $P(W_i|W_j)$
    this is the probability that word W_j is immediately followed by word W_i.
  - *trigram model*: $P(W_i|W_j, W_k)$
- **Word Models:** $P(p_1, p_2, \ldots, p_k|W_i)$ is the probability that word W_i will be pronounced as the sequence of phones $p_1, p_2, \ldots, p_k$.
  These are typically learned from transcribed, labeled speech.
- **Phone HMM:** For each phone, an HMM is trained to fit the probability distribution of that phone. $P(c_1, c_2, \ldots, c_n|p_i)$ is the probability that phone i will be pronounced as the sequence of VQ symbols $c_1, c_2, \ldots, c_n$.

By concatenating the phone HMM's according to the word model and concatenating the word models according to the bigram or trigram model, we can construct a big HMM that corresponds to any particular spoken sentence.

Typical Phone HMM



Some of the edges are labeled with the name of an output probability distribution $P(C|S_t, S_{t+1})$:

- **B** Beginning
- **M** Middle
- **E** End

The distributions of the other edges are usually unique (or are tied depending on the particular phone).

Viterbi Search of the Speech HMM

The word HMM has a huge branching factor (20,000 words is typical today).

This means it is infeasible to build an actual HMM data structure by macro expanding the trigram model and the word pronunciation models.

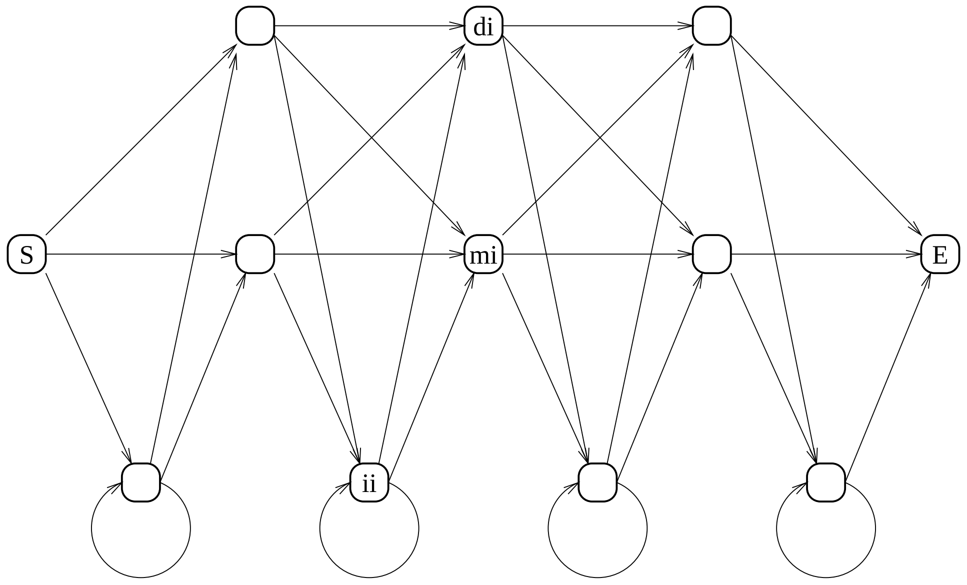
Instead, the necessary structure is constructed dynamically using a beam search.

In the forward algorithm, instead of passing the whole $P[S_i]$ table, we delete all but the top B largest values (B is called the "beam width"). The HMM is only constructed to store these B current best paths.

There are additional tricks for avoiding considering all possible words.

Applications of HMMs in Molecular Biology

Each DNA molecule can be represented as a string over a 4-letter alphabet $\{A, G, C, T\}$. A gene is a DNA string, and often several genes are closely related to each other because of evolution. Given a "family" of related genes, we would like to fit an HMM that generates all of the members of that gene family.



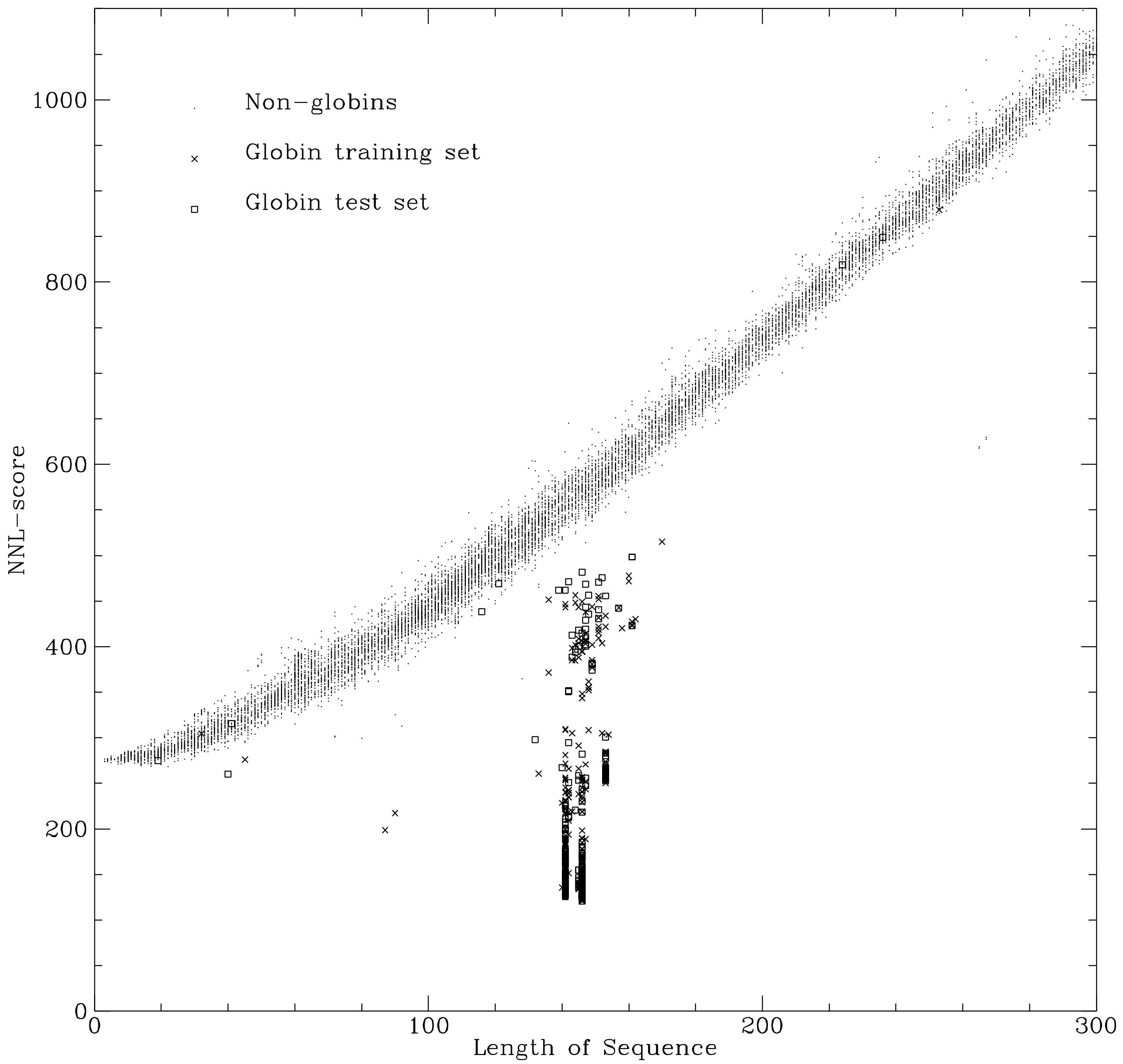
m_i = main line sequence ("consensus")

d_i = deletion states. They are "silent"

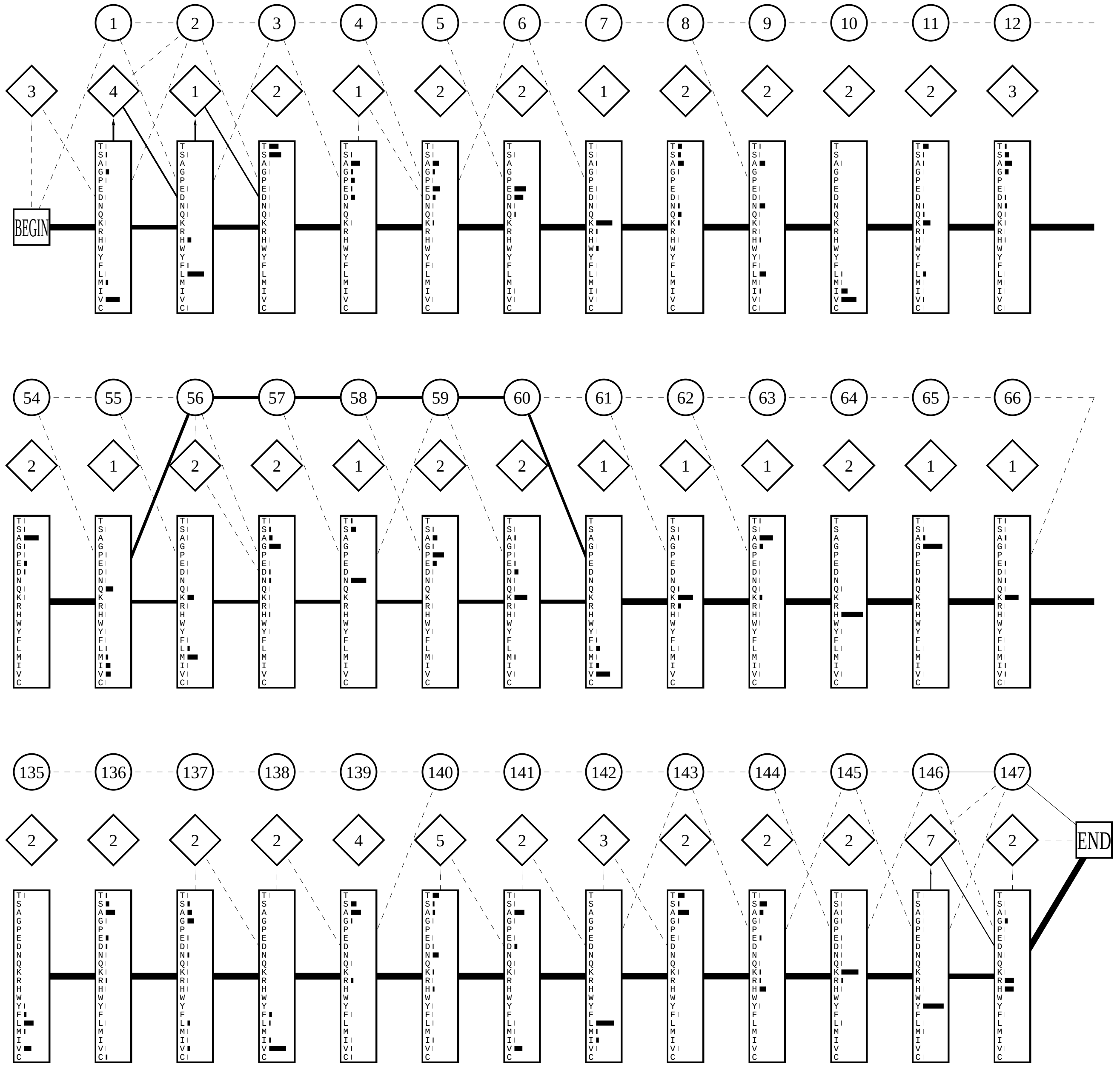
i_i = insert states.

For DNA, the output variables have 4 symbols, for proteins, 20.

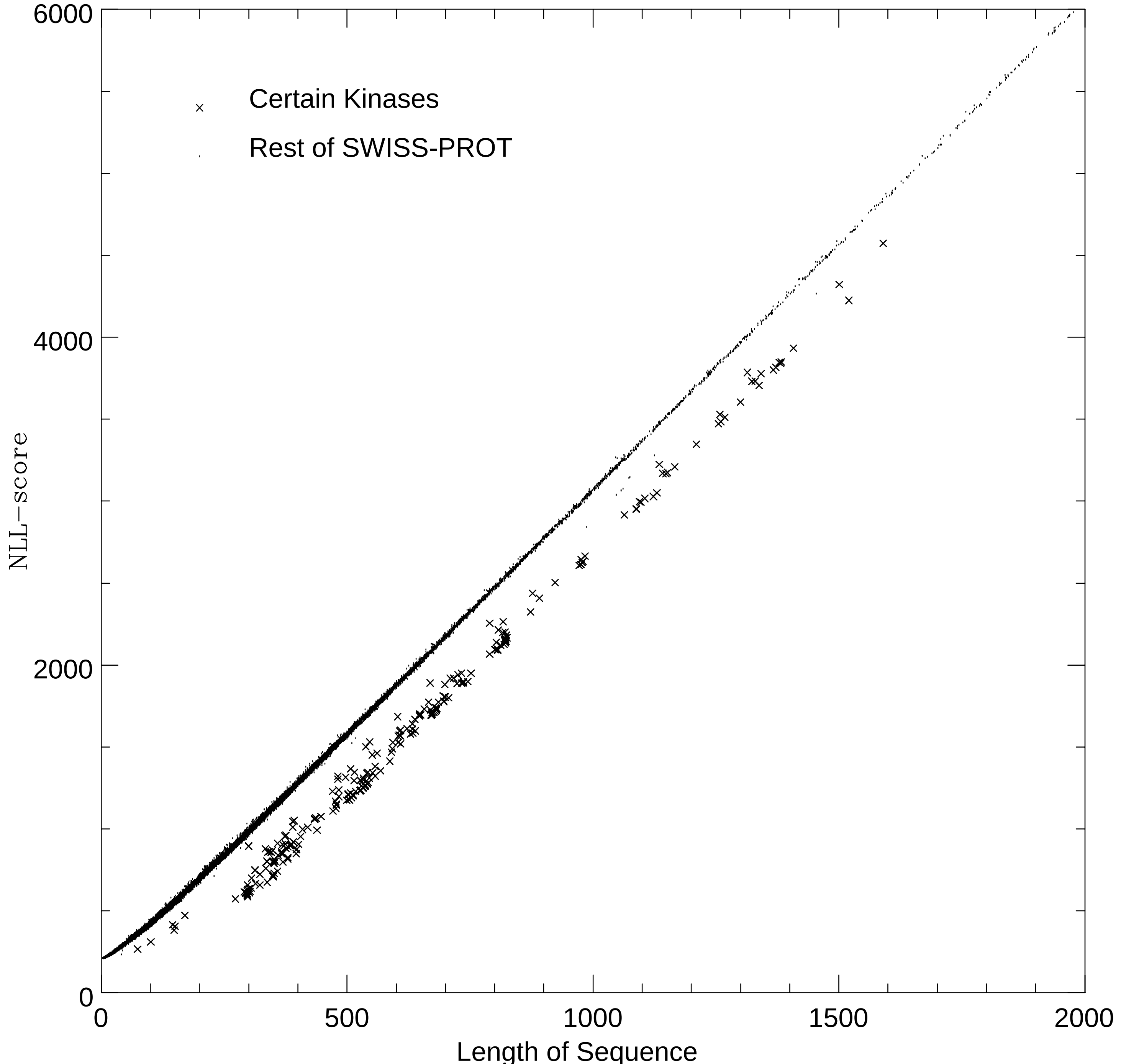
A HMM Model of the Globins (Krogh et al, 1993)



A HMM Model of the Globins (2)



A HMM Model of Kinases



Applications

- **Understand Gene Families**
- **Mutually align genes to discover where bases have been inserted and deleted**
- **Identify new members of gene families**
- **Discover highly-conserved subregions, which suggests functional importance**

And many others!

Learning in HMMs

This is an incomplete data problem, like the Naive Bayes mixture model.

The missing data consists of the "true hidden state" at each point in time.

- **EM training**

  E-Step: Run Hugin propagation (= forward-backward) to compute $\tilde{\mathbf{P}}(S_i)$

  M-Step: Directly estimate $P(S_{i+1}|S_i)$ and $P(X_i|S_i)$.

- **Viterbi training (Hard EM)**

  Hard E-Step: Run Viterbi algorithm and set the states S_i.

  Hard M-Step: Directly estimate $P(S_{i+1}|S_i)$ and $P(X_i|S_i)$.

Care must be taken to initialize well.

In speech: hand-labeled data can be used to learn initial parameters.

Incremental EM

Full EM requires making a complete pass through the training data, computing $\tilde{\mathbf{P}}(S|X)$ for each example X, and finally performing the M step.

However, EM still retains its convergence guarantees if we do partial E steps and partial M steps. For example, we can compute $\tilde{\mathbf{P}}(S|X)$ for one example, do an M step to re-estimate the parameters using the new $\tilde{\mathbf{P}}(S|X)$ for one example and the old ones for all of the rest of the examples.

Incremental EM tends to converge faster than batch EM.