Aalto University
School of Science

CS-E5865 Computational genomics

Autumn 2019, Lecture 5: HMM algorithms, Pair HMM & Profile HMM

Lecturer: Pekka Marttinen

Assistants: Alejandro Ponce de León, Zeinab Yousefi, Onur Poyraz

HMM problems and algorithms

1. Find the most likely hidden state sequence => Viterbi algorithm ✔
2. Estimate the emission and transition probabilities of the HMM => Viterbi training ✔
3. Determine the probability of a sequence s given the HMM model => Forward algorithm
4. Determine the probability of being in state k at position i => Posterior decoding

Forward, Backward & Posterior decoding

The Forward Algorithm

- Task: calculate the probability $\mathrm{P}(s)$ of sequence s, given by our HMM
- Sum over all possible hidden state paths (set Π) that could have been used to generate s:

$$P(s) = \sum_{\pi \in \Pi} P(s, \pi) = \sum_{\pi \in \Pi} P(s|\pi) P(\pi)$$

- Exponential sum, cannot enumerate over all state paths!
- Again, we will define a dynamic programming problem, and fill a table of *forward probabilities*

$$\mathrm{F_k(i)} = \mathrm{P}(s_1 ... s_i, \pi_i = \mathrm{k})$$

- Probability of emitting the prefix $s_1,...,s_i$ and ending up in state k

The Forward Algorithm – derivation

$$F_k(i) = P(s_1 \ldots s_i, \pi_i = k)$$

$$= \Sigma_l P(s_1 \ldots s_i, \pi_{i-1} = l, \pi_i = k)$$

$$= \Sigma_l P(s_1 \ldots s_{i-1}, \pi_{i-1} = l) P(\pi_i = k | \pi_{i-1} = l) P(s_i | \pi_i = k)$$

$$= \Sigma_l P(s_1 \ldots s_{i-1}, \pi_{i-1} = l) T_{lk} E_k(s_i)$$

$$= E_k(s_i) \Sigma_l F_l(i-1) T_{lk}$$

- Sum over all possibilities of emitting $s_1,\ldots,s_{i-1}$ ending up in state l, and then making a transition from l to k, and emitting s_i

The Forward Algorithm

- $F_k(i) = E_k(s_i) \Sigma_l F_l(i-1) T_{lk}$
- Dynamic programming formulation:
  - table F of size m X n where:
    - m=num of hidden states
    - n=length of the observed sequence

Initialization: (first column)

$F(k,1) = 1/m \; E_k(s_1)$, for all $k > 0$

Iteration

$F(k,i) = E_k(s_i) \Sigma_l F(l,i-1) T_{lk,}$ for all k, and for all $i=2,...,n$

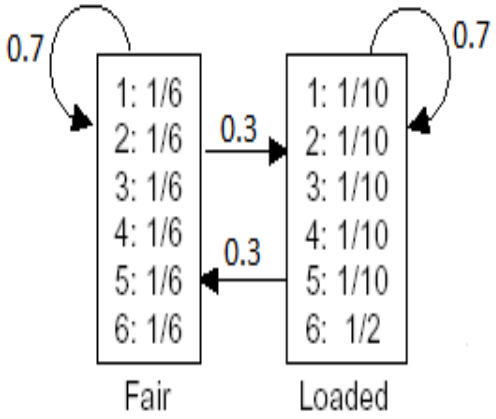
Termination: (sum all the values in the last column)

$P(s) = \Sigma_k F(k,n)$

- Difference to Viterbi: replace *max* with *sum*

Forward at the Casino

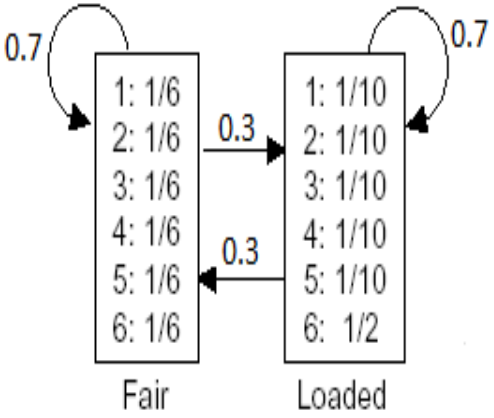
```
forward <- function(s, T, E) {

  n.states <- ncol(E)

  F <- matrix(rep(0, n.states * length(s)), nrow = n.states)
  F[,1] <- 1 / n.states * E[s[1],]

  for (i in 2:length(s)) {
    for (l in 1:n.states) {
      F[l,i] <- sum(F[,i-1] * T[,l])
      F[l,i] <- F[l,i] * E[s[i],l]
    }
  }

  prob <- sum(F[,length(s)])

  res <- list()
  res$fprob <- prob
  res$F <- F

  return(res)
}
```




```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

$$F(k,i) = E_k(s_i) \, \Sigma_l \, F(l,i-1) \, T_{lk}$$

Forward at the Casino



- Initialization

$F(1,1) = E_{Fair}(s_1)\ 0.5 = 0.1667 * 0.5 = 0.08335$

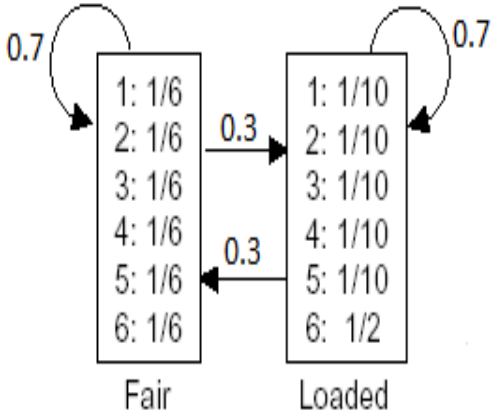
$F(2,1) = E_{Loaded}(s_1)\ 0.5 = 0.1 * 0.5 = 0.0500$

```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
> forward.res$F
             [,1]       [,2]         [,3]         [,4]         [,5]         [,6]         [,7]         [,8]         [,9]         [,10]
[1,] 0.08333333 0.01222222 0.0017259259 0.0002406914 3.342288e-05 4.634329e-06 6.422555e-07 1.452416e-07 4.637090e-08 1.679837e-08
[2,] 0.05000000 0.00600000 0.0007866667 0.0001068444 1.469985e-05 2.031676e-06 1.406236e-06 5.885209e-07 2.277686e-07 8.667463e-08
> forward.res$fprob
[1] 1.03473e-07
```

$F(k,i) = E_k(s_i)\ \Sigma_l\ F(l,i-1)\ T_{lk}$

Forward at the Casino



- Compute $F(1,2)$
- For clarity, we'll use informal notation $F(\text{fair},2)$ for $F(1,2)$
- $F(\text{fair},2) = P(s_1, s_2, \pi_2 = \text{fair})$

$= E_{\text{fair}}(s_2) \Sigma_l F(l,1) T_{l,\text{fair}}$

$=0.1667\ [F(\text{fair},1)\ T_{\text{fair,fair}} + F(\text{loaded},1) T_{\text{loaded,fair}}]$

$=0.1667[0{,}0833 * 0{,}7 + 0{,}05 * 0{,}3] = 0.0122$

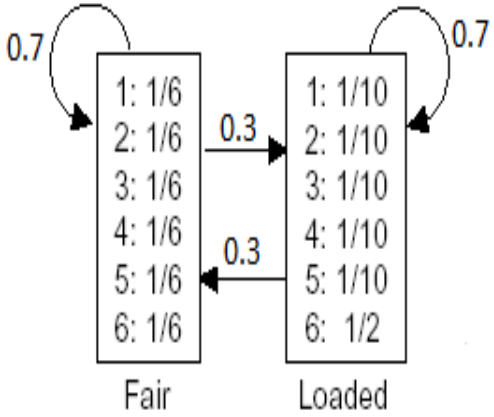
```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

And so on, until the table is filled..

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
> forward.res$F
           [,1]        [,2]         [,3]         [,4]         [,5]         [,6]         [,7]         [,8]         [,9]         [,10]
[1,] 0.08333333 0.01222222 0.0017259259 0.0002406914 3.342288e-05 4.634329e-06 6.422555e-07 1.452416e-07 4.637090e-08 1.679837e-08
[2,] 0.05000000 0.00600000 0.0007866667 0.0001068444 1.469985e-05 2.031676e-06 1.406236e-06 5.885209e-07 2.277686e-07 8.667463e-08
> forward.res$fprob
[1] 1.03473e-07
```

$F(k,i) = E_k(s_i) \Sigma_l F(l,i-1) T_{lk}$

Forward at the Casino



- The probability of the full sequence s:

$$P(s) = F(\text{fair},10) + F(\text{loaded},10)$$

- Note: working with logarithms is not as straightforward as with Viterbi (logarithm of a sum does not simplify).
- For an algorithm that deals with this issue, see e.g., Bishop: Pattern Recognition and Machine Learning, Ch. 13.2 (not required on this course).

```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
> forward.res$F
            [,1]       [,2]         [,3]         [,4]         [,5]         [,6]         [,7]         [,8]         [,9]         [,10]
[1,] 0.08333333 0.01222222 0.0017259259 0.0002406914 3.342288e-05 4.634329e-06 6.422555e-07 1.452416e-07 4.637090e-08 1.679837e-08
[2,] 0.05000000 0.00600000 0.0007866667 0.0001068444 1.469985e-05 2.031676e-06 1.406236e-06 5.885209e-07 2.277686e-07 8.667463e-08
> forward.res$fprob
[1] 1.03473e-07
```

$$F(k,i) = E_k(s_i)\ \Sigma_l\ F(l,i-1)\ T_{lk}$$

Backward Algorithm - motivation

- Posterior decoding problem: We want to compute the probability of state k for position i given sequence s: $P(\pi_i = k \mid s)$
  - e.g. “During i’th roll Casino was using the loaded dice”, “Nucleotide s_i belongs to an ORF”
  - This is different from computing the most likely path $\pi_1 \ldots \pi_n$ by Viterbi
- We compute the result by splitting the sequence into two parts and computing the probabilities of prefixes and suffixes of s, such that the hidden state at position i is k: $P(\pi_i = k, s) = P(s_1 \ldots s_i, \pi_i = k, s_{i+1} \ldots s_n)$

$$= P(s_1 \ldots s_i, \pi_i = k)\, P(s_{i+1} \ldots s_n \mid s_1 \ldots s_i, \pi_i = k)$$

$$= P(s_1 \ldots s_i, \pi_i = k)\, P(s_{i+1} \ldots s_n \mid \pi_i = k)$$

Forward, $F_k(i)$ **Backward, $B_k(i)$**

- Then, $P(\pi_i = k \mid s) = P(\pi_i = k, s) \,/\, P(s)$

The Backward Algorithm – derivation

Define the backward probability:

$$B_k(i) = P(s_{i+1}...s_n \mid \pi_i = k)$$

$$= \Sigma_l P(s_{i+1}, s_{i+2}, ..., s_n, \pi_{i+1}=l \mid \pi_i = k)$$

$$= \Sigma_l P(s_{i+1}, s_{i+2}, ..., s_n \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)$$

$$= \Sigma_l P(s_{i+2}, ..., s_n \mid \pi_{i+1} = l)\, P(s_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)$$

$$= \Sigma_l E_l(s_{i+1}) T_{kl}\, B_l(i+1)$$

The Backward Algorithm

We can compute $B_k(i)$ for all k, i, using dynamic programming

- Fill in a table B of size m X n where:
  - m=nr of hidden states
  - n=length of the observed sequence

Initialization:

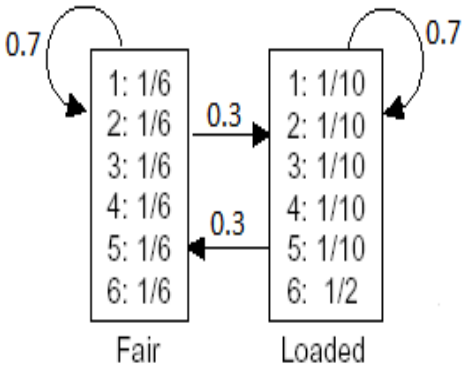
$$B(k,n) = 1, \text{ for all } k$$

Iteration: (backward from position n to 1)

$$B(k,i) = \sum_l E_l(s_{i+1})\, T_{kl}\, B(l,i+1)$$

Backward at the casino

$$B(k,i) = \sum_l E_l(s_{i+1})\, T_{kl}\, B(l,i+1)$$




```
backward <- function(s, T, E) {

  n.states <- ncol(E)

  B <- matrix(rep(0, n.states * length(s)), nrow = n.states)
  B[, length(s)] <- 1

  for (i in seq(length(s)-1,1)) {

    for (k in 1:n.states) {
      B[k,i] <- sum(E[s[i+1],] * B[,i+1] * T[k,])
    }

  }

  res <- list()
  res$B <- B

}
```

```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

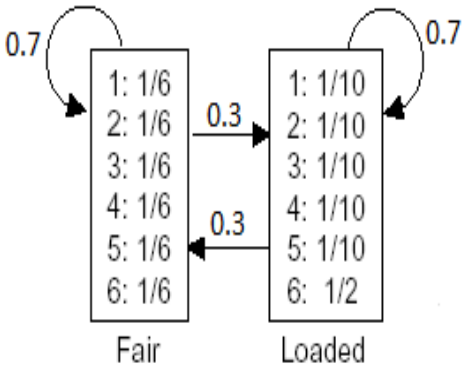
$$B(k,i) = \sum_l E_l(s_{i+1})\, T_{kl}\, B(l,i+1)$$

Backward at the casino

- Initialization

$B(1,n)=F(fair,n)= 1$

$B(2,n)=F(loaded,n)=1$

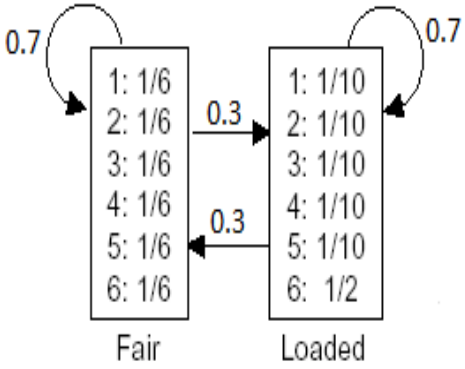



```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
> backward.res$B
             [,1]         [,2]         [,3]         [,4]        [,5]        [,6]        [,7]        [,8]      [,9] [,10]
[1,] 8.623071e-07 6.208959e-06 4.450478e-05 0.0003147900 0.002138403 0.01265679 0.03362963 0.09111111 0.2666667    1
[2,] 6.322813e-07 4.597620e-06 3.389115e-05 0.0002593093 0.002176988 0.02205926 0.05822222 0.15333333 0.4000000    1
```

$$B(k,i) = \sum_l E_l(s_{i+1})\, T_{kl}\, B(l,i+1)$$

Backward at the casino



- Recursion, for example:

$B(\text{fair},7) = P(s_8,\ldots,s_{10} \mid \pi_7 = \text{fair})$

$= T_{\text{fair,loaded}} E_{\text{loaded}}(s_8) B(\text{loaded},8) + T_{\text{fair,fair}} E_{\text{fair}}(s_8) B(\text{fair},8)$

$= T_{\text{fair,loaded}} E_{\text{loaded}}(6) B(\text{loaded},8) + T_{\text{fair,fair}} E_{\text{fair}}(6) B(\text{fair},8)$

$= 0.3 * 0.5 * 0.1533 + 0.7 * 0.1667 * 0.0911$

$= 0.0336$

```
> T
     [,1] [,2]
[1,]  0.7  0.3
[2,]  0.3  0.7
> E
          [,1] [,2]
[1,] 0.1666667  0.1
[2,] 0.1666667  0.1
[3,] 0.1666667  0.1
[4,] 0.1666667  0.1
[5,] 0.1666667  0.1
[6,] 0.1666667  0.5
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
> backward.res$B
              [,1]         [,2]         [,3]         [,4]         [,5]         [,6]         [,7]         [,8]      [,9] [,10]
[1,] 8.623071e-07 6.208959e-06 4.450478e-05 0.0003147900 0.002138403 0.0126567[illegible] 0.03362963 0.09111111 0.2666667     1
[2,] 6.322813e-07 4.597620e-06 3.389115e-05 0.0002593093 0.002176988 0.02205926 0.05822222 0.15333333 0.4000000     1
```

Posterior Decoding

We can now calculate

$$P(\pi_i = k \mid s) = \frac{F_k(i)\, B_k(i)}{P(s)}$$

$P(\pi_i = k \mid s) =$

$P(\pi_i = k , s)/P(s) =$

$P(s_1, \ldots, s_i, \pi_i = k, s_{i+1}, \ldots s_n) / P(s) =$

$P(s_1, \ldots, s_i, \pi_i = k)\, P(s_{i+1}, \ldots s_n \mid \pi_i = k) / P(s) =$

$F_k(i)\, B_k(i) / P(s)$

Posterior Decoding now gives the most likely state at position *i* of sequence:

$$\pi^*_i = \text{argmax}_k\, P(\pi_i = k \mid s)$$

Decoding problem

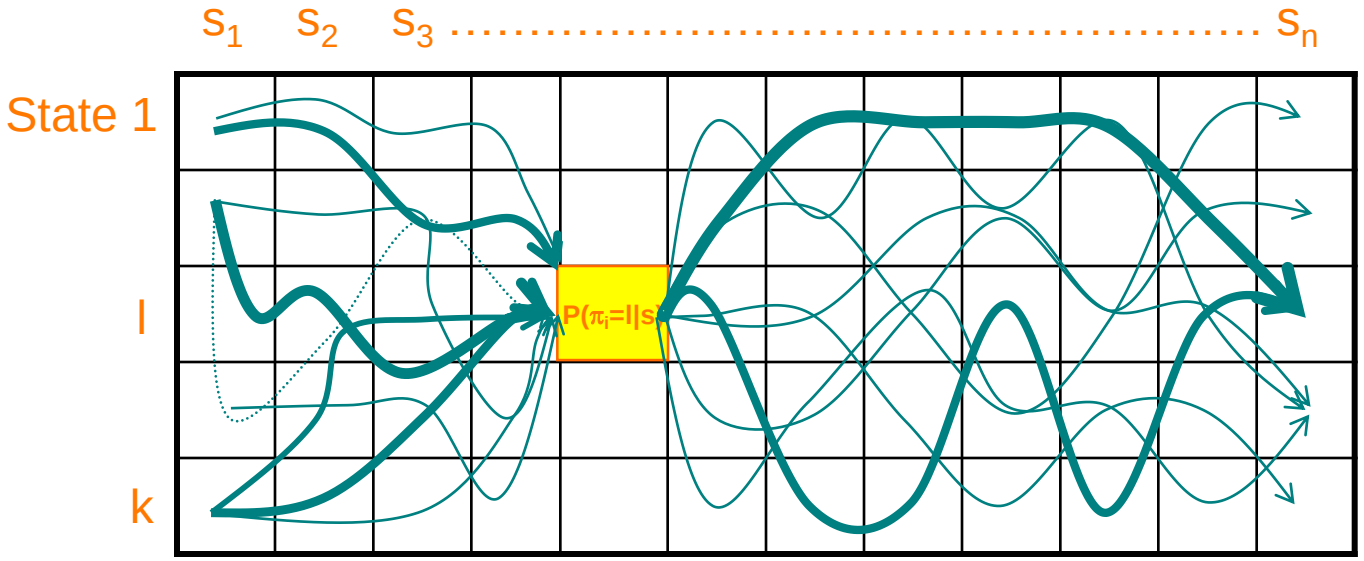
- We have now 2 methods for decoding:
  - Posterior decoding
  - Viterbi algorithm

- Which is most appropriate?

Posterior Decoding

- For each state
  - Posterior Decoding gives us a probability distribution for the state at each position
  - This is sometimes more informative than Viterbi path π^*
    - Posterior decoding takes into account all possible paths when determining the most likely state
    - Viterbi method only takes into account one path, which may end up representing a minimal fraction of the total probability

Posterior Decoding



- $P(\pi_i = k \mid s) = \Sigma_\pi P(\pi \mid s)\, \mathbf{1}(\pi_i = k)$

  $= \Sigma_{\{\pi:\pi[i] = k\}} P(\pi \mid s)$

$\mathbf{1}(\psi) = 1$, if ψ is true
0, otherwise

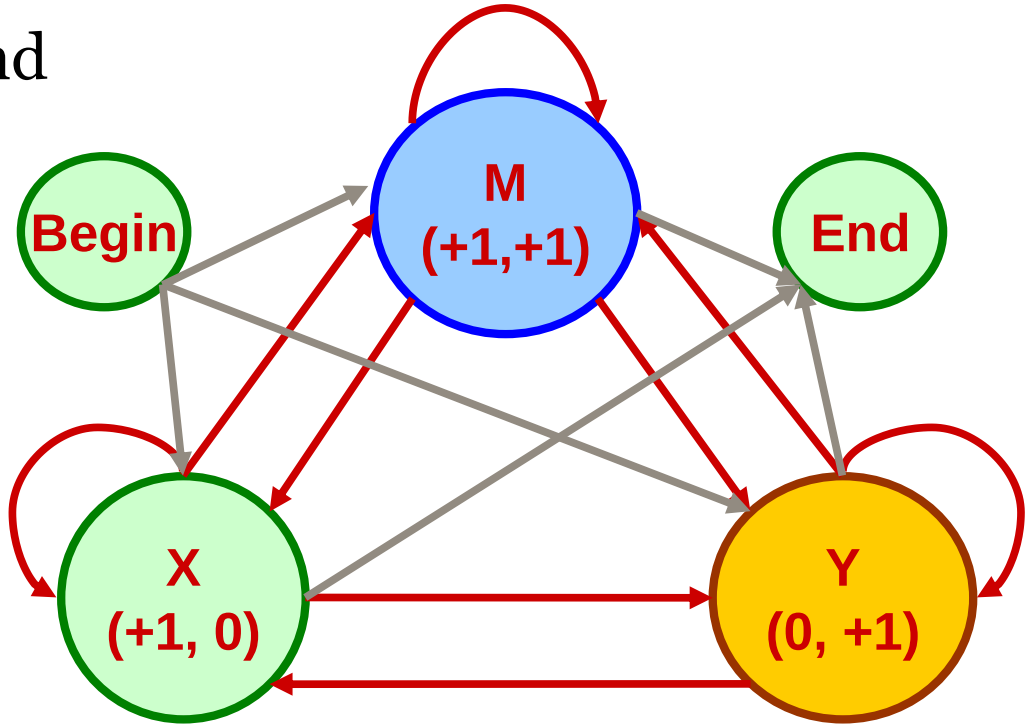
HMMs for sequence alignment

Hidden Markov Models for Sequence Alignment

- So far, we have used HMMs to detect certain regions from a single sequence
- HMMs can also be used for sequence alignment tasks
  - Pair-HMM can be used to find high-scoring alignments between two sequences, allowing gaps
  - Profile-HMM can be used to model a multiple alignment of a set of sequences

Pair HMM

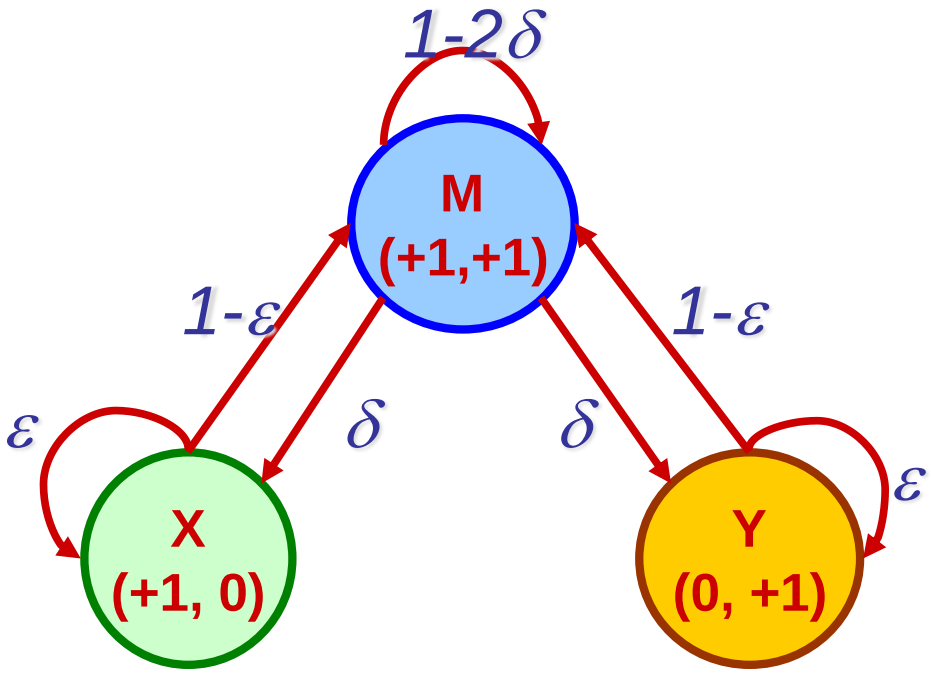
- Given 2 sequences X and Y, we want to identify their alignment
- Pair HMM consists of
  - Begin and End state which do not emit symbols
  - Three normal states
    - M (match)
    - X (gap in Y)
    - Y (gap in X)




```
X TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
Y -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  XMMYMMMMMMMYYMMMMMMYMMMMMMMXXMMMMMXXX
```

Pair HMM - Transitions

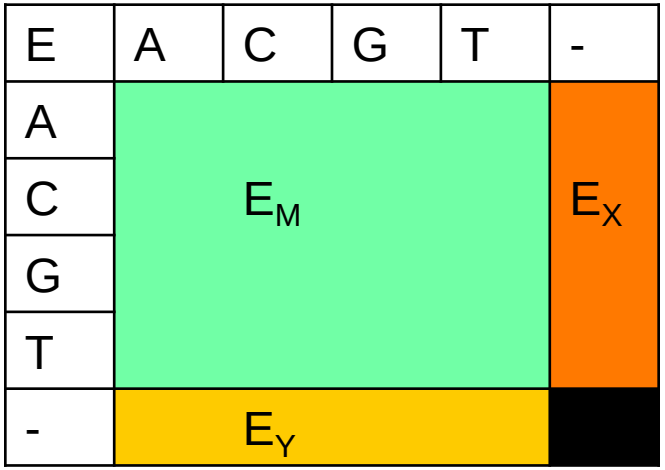
- Transition from M to X (resp. Y) opens a gap in Y (resp. X),
- Transition back to M closes the gap
  - δ ~ open gap probability
  - ε ~ extend gap probability




```
X TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
Y -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  XMMYMMMMMMMYYMMMMMMYMMMMMMMXXMMMMMXXX
```

Pair HMM - Emissions

- State M: emit (b,b') with probability $E_M(b,b')$
- State X: emit (b,-) with probability $E_x(b,-)$
- State Y: emit (-,b') with probability $E_Y(-,b')$




```
X TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
Y -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  XMMYMMMMMMMYYMMMMMMMYMMMMMMMXXMMMMMXXX
```

Pair HMMs – Finding Optimal Alignment

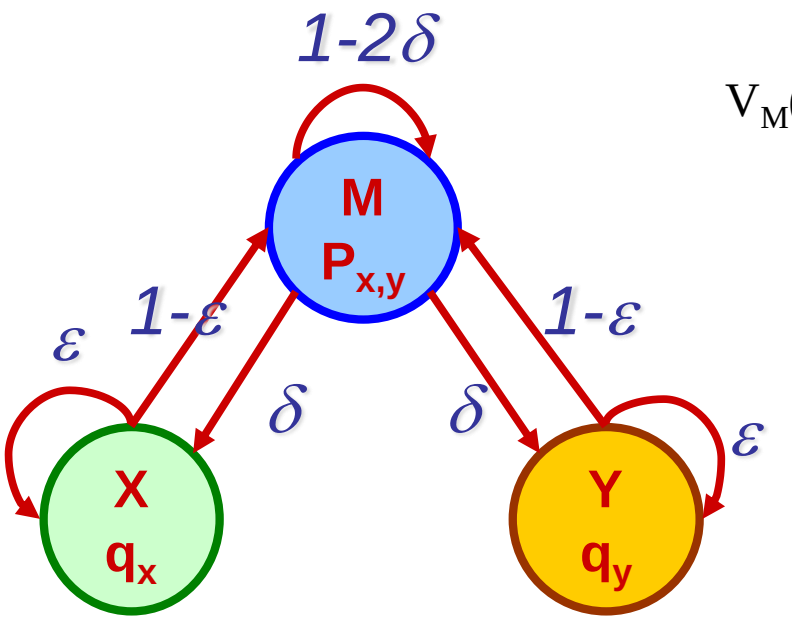
- A state sequence π from begin to end state that emits x and y gives an alignment for them
  - Transition and emission probabilities give the probability of the alignment
- The best alignment of two sequences corresponds to the most probable state sequence

$$\pi^* = \text{argmax}_{\pi} P(x,y,\pi)$$

- Can be computed by the Viterbi algorithm

```
X TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
Y -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  XMMYMMMMMMMMYYMMMMMMMYMMMMMMMXXMMMMMXXX
```

Viterbi for pair-HMMs



$$V_M(i, j) = E_M(x_i, y_j) \max \begin{cases} (1 - 2\delta)\, V_M(i - 1, j - 1) \\ (1 - \varepsilon)\, V_X(i - 1, j - 1) \\ (1 - \varepsilon)\, V_Y(i - 1, j - 1) \end{cases}$$

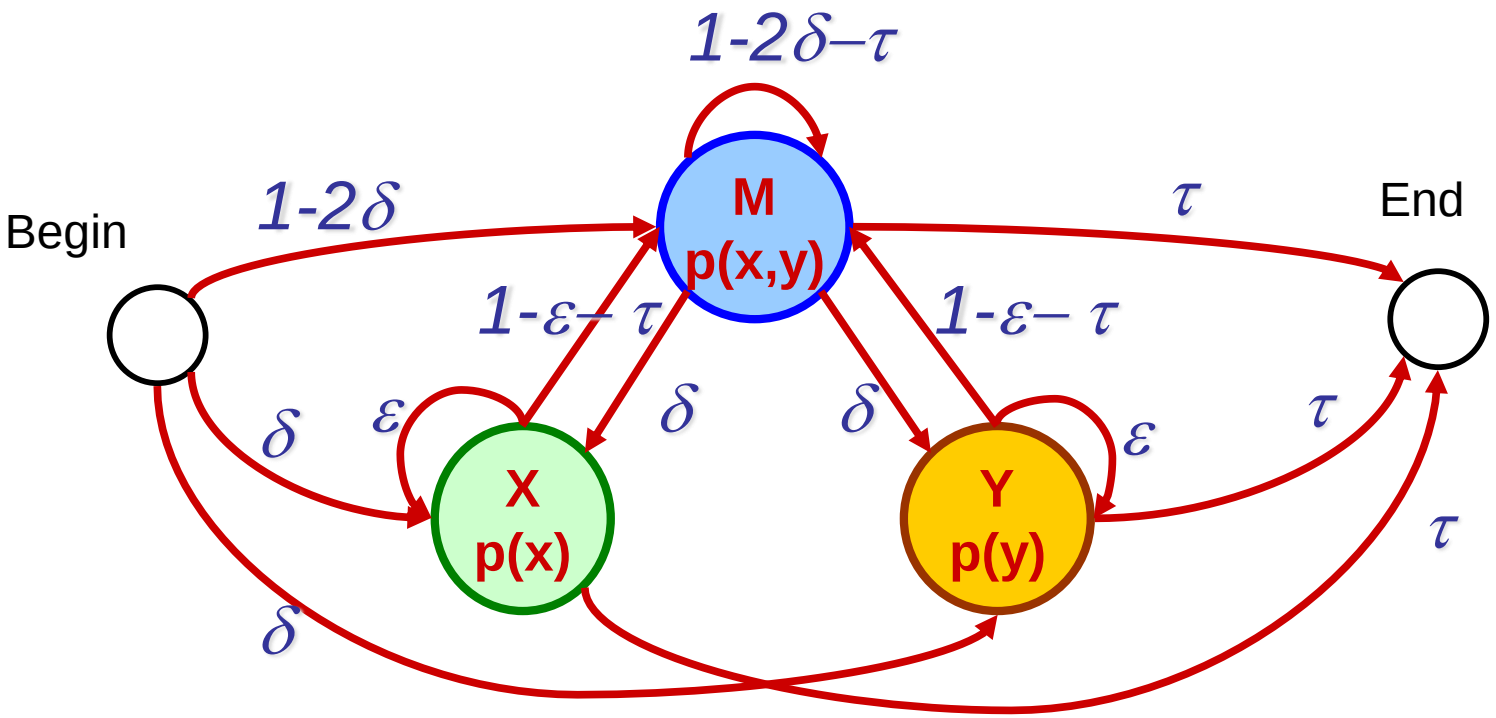
$$V_X(i, j) = E_X(x_i) \max \begin{cases} \delta\, V_M(i - 1, j) \\ \varepsilon\, V_X(i - 1, j) \end{cases}$$

$$V_Y(i, j) = E_Y(y_j) \max \begin{cases} \delta\, V_M(i, j - 1) \\ \varepsilon\, V_Y(i, j - 1) \end{cases}$$

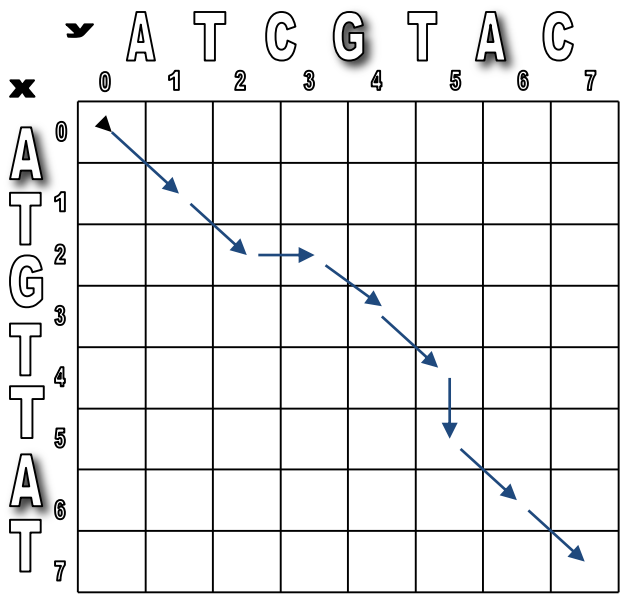
```
X TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
Y -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  XMMYMMMMMMMMYYMMMMMMMYMMMMMMMXXMMMMMXXX
```

Full model

- The complete model should also contain the transitions between the begin, end and normal states



Pair-HMM vs. Needleman-Wunsch



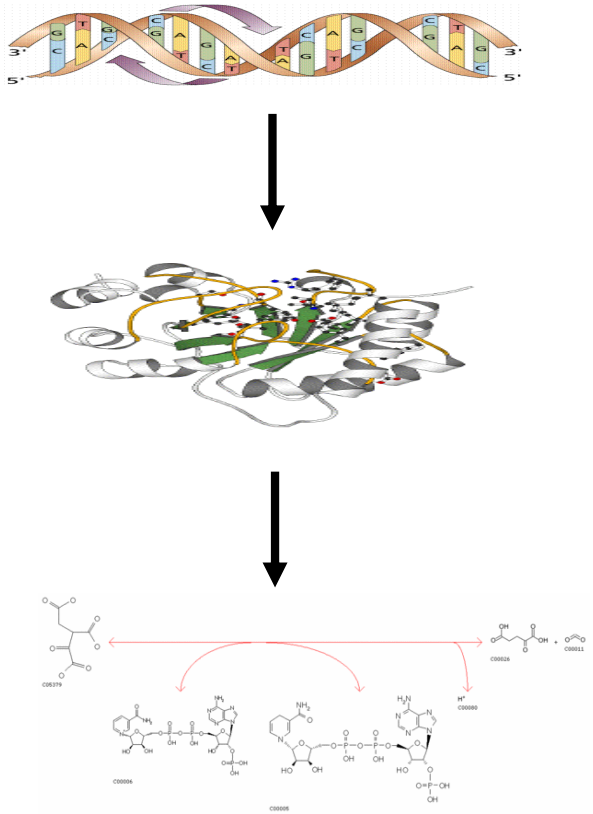
A T – G T T A T
A T C G T – A C
M M Y M M X M M

- Similarities:
  - HMM transition to a match state ~ NW diagonal move
  - HMM tr. to Y state ~ NW horizontal move
  - HMM tr. to X state ~ NW vertical move
  - HMM Emissions ~ NW substitutions
- Important difference:
  - HMM transition and emission probabilities can be trained
  - NW substitution scores fixed

Profile Hidden Markov Models

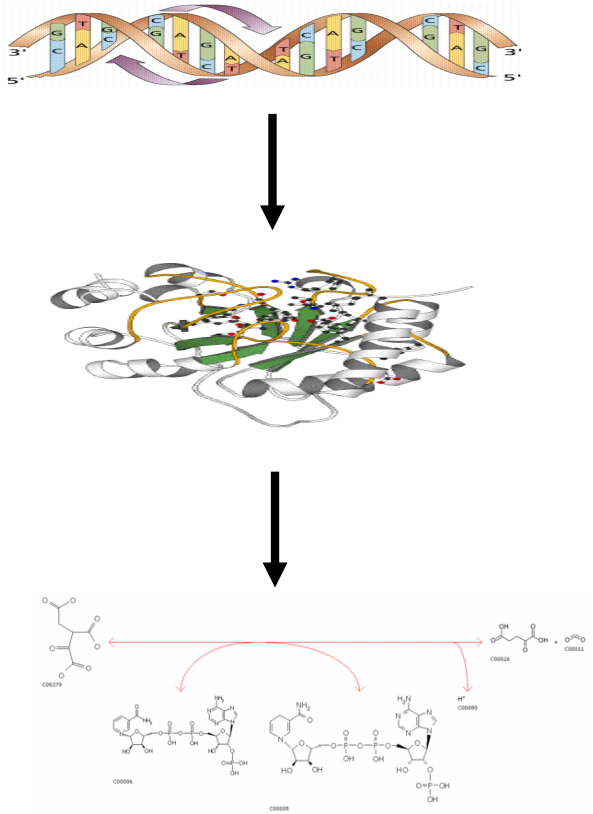
From Sequence to Structure to Function

- In functional genomics the goal is to annotate the genes by their function (e.g. catalysis of a biochemical reaction)
- In principle, possible functions of proteins are determined by their 3D structure
- 3D structure is in principle determined by the amino acid sequence
- Consequently, the amino sequence should determine the function



From Sequence to Structure to Function

- However, predicting the 3D structure of a protein (aka Protein folding problem) from the amino acid sequence is extremely difficult
  - Not fully solved yet
- Also, predicting the function from the 3D structure is not easy
  - Require molecular simulations run on supercomputers
- A shortcut is offered by Hidden Markov Models



Profile Hidden Markov Models

- Protein families:
  - Sets of related sequences and structures
  - Diverged from each other in their primary sequence during evolution
  - Some regions are more conserved than others
- Profile HMM is tailored to the family
  - by defining the HMM structure to match the family
  - by training the parameters with the sequences of that family

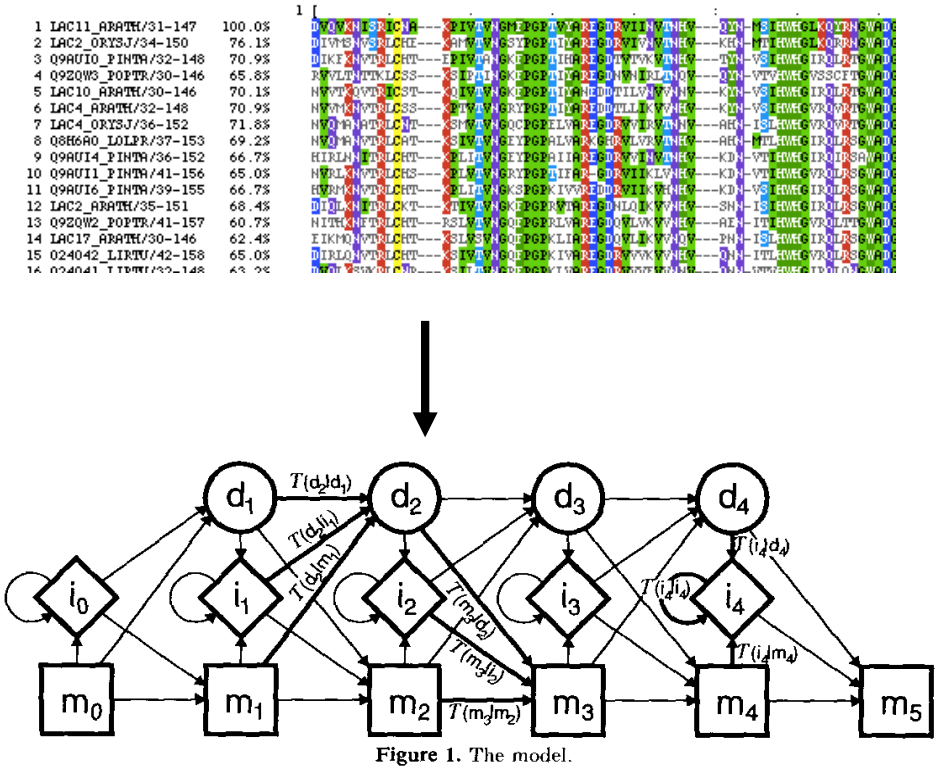


Figure 1. The model.

Profile Hidden Markov Models – the approach

- We construct a HMM of a set of proteins that share a function or structural regions (called *domains*)
- This model can be used to give a *probability* for each new protein sequence to share that same function or domain
- The same sequence can be tested against a large set of HMM models
  - high probability by a HMM indicates that our new sequence may share the domain or function modeled by that HMM

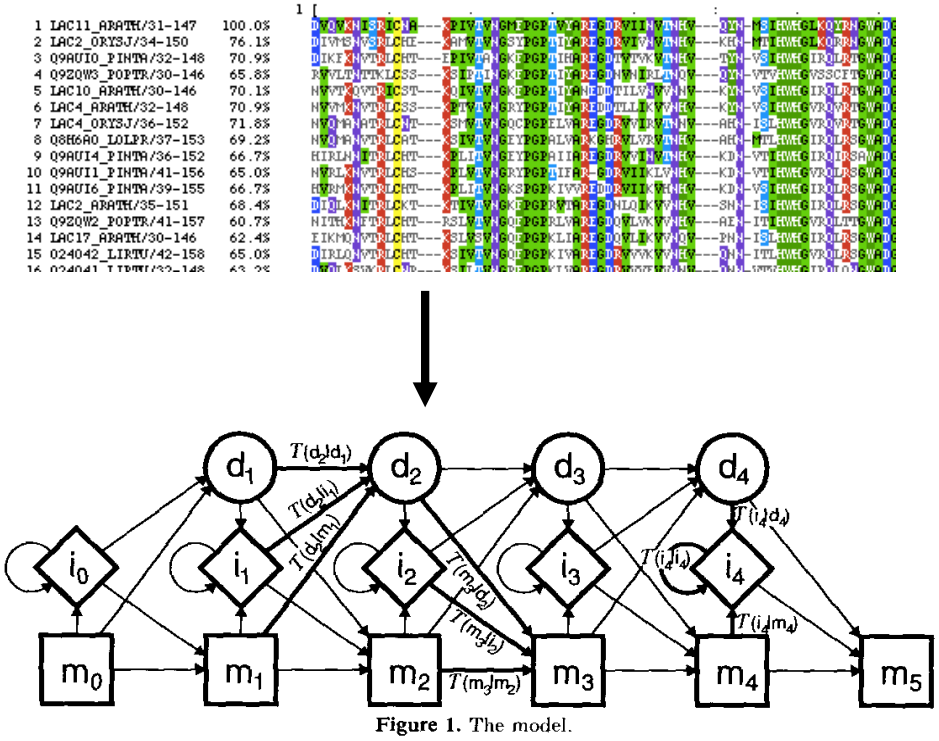
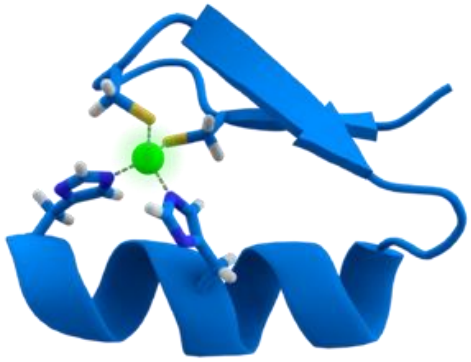


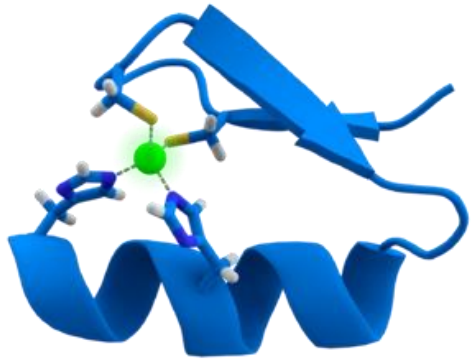
Figure 1. The model.

Example: zinc finger domain

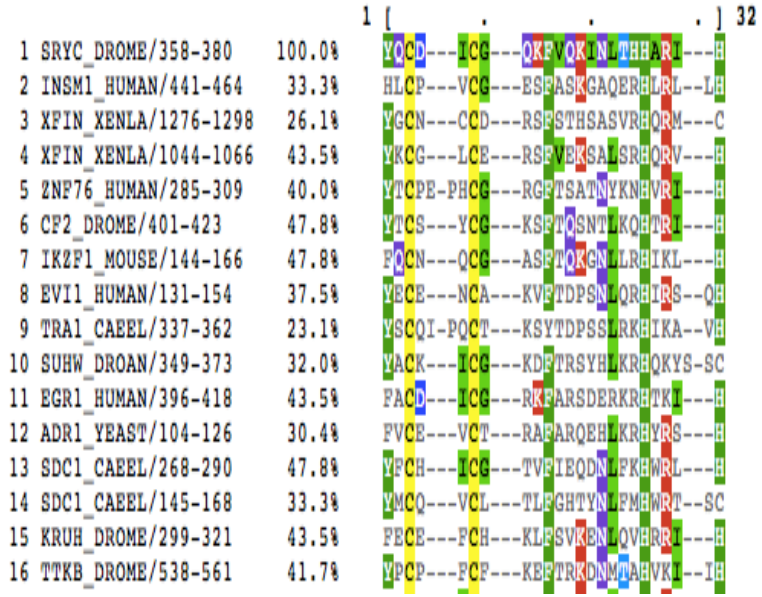
- Typically it functions as interaction module that binds DNA, RNA, proteins, or other small, useful molecules
- Several variants exist, one of which is depicted above right
- Below right a protein with three zinc finger domains embedded



Example: zinc finger domain



- Part of the multiple alignment of proteins containing the zinc finger domain is depicted below
- The full alignment has 194 proteins
- A profile HMM can be trained to recognize new members of the family
  - Does not require 3D structure
- PFAM database contains a large number of profile HMMs for different structural and functional domains or motifs



Structure of a Profile HMM

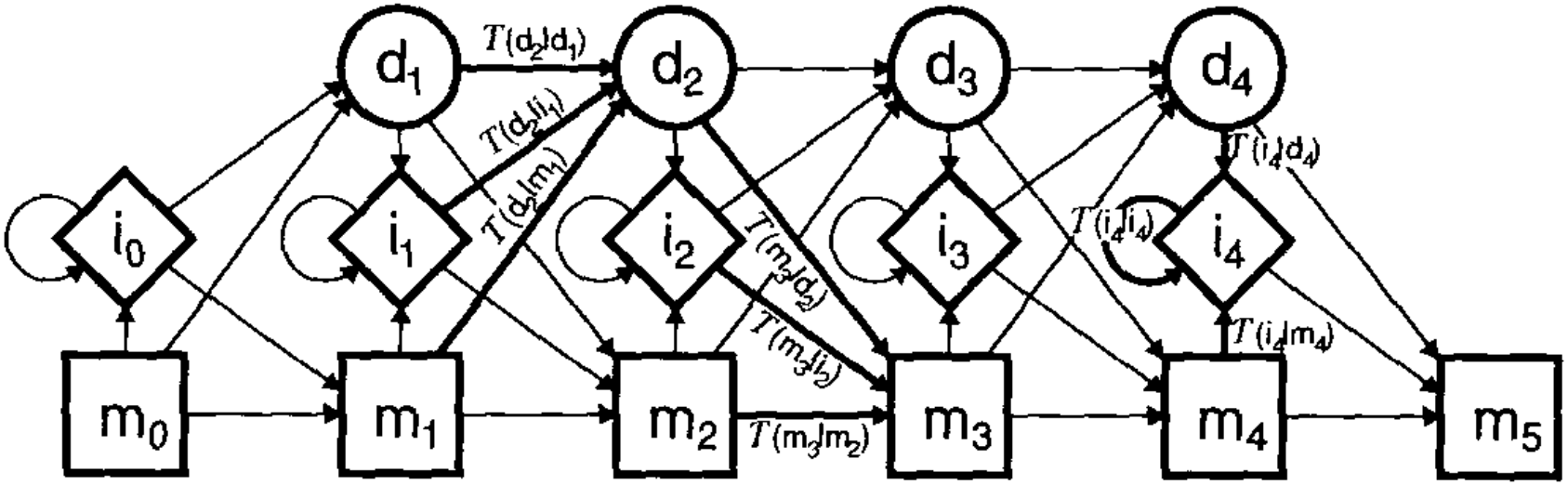
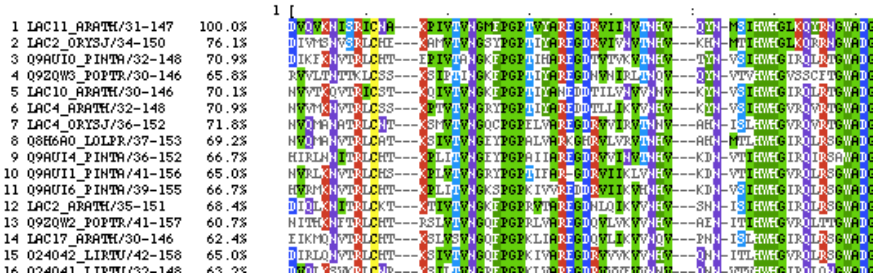


Figure 1. The model.

Match states

- Correspond to the columns of the multiple alignment
- Number of match states picked using expert knowledge e.g.
  - average length of sequences in the alignment
  - number of columns that contain at least 50% non-gap symbols
- Initial emission probabilities can be computed from the multiple alignment:
  - For each amino acid, count the times it appears in each column



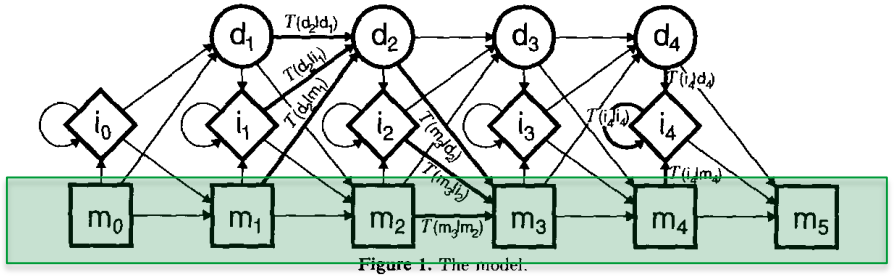
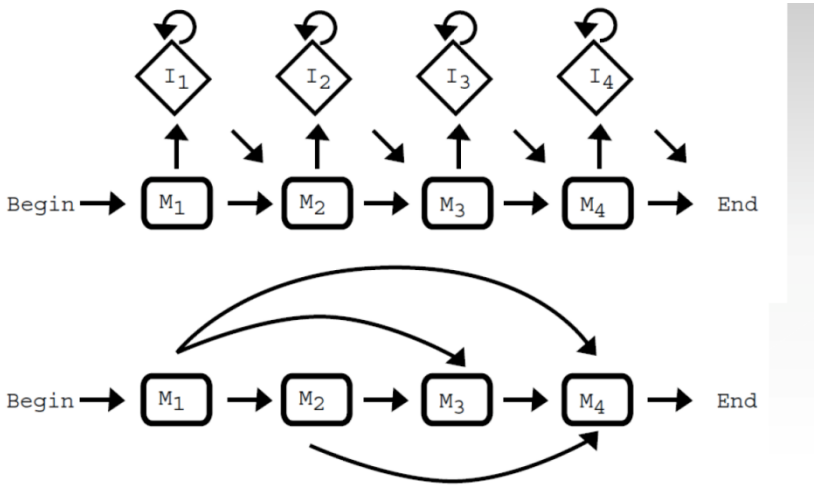


Figure 1. The model.

Insertion and deletion states

- Insertion states allow the profile HMM to model symbols in the sequences that do not match the model
  - aligning a symbol in sequence to a gap in the model
- Deletion states allow the profile HMM to model symbols deleted from the sequence
  - aligning a gap in a sequence to a symbol in the model



Summary: Building Profile HMM topology