Lecture 10

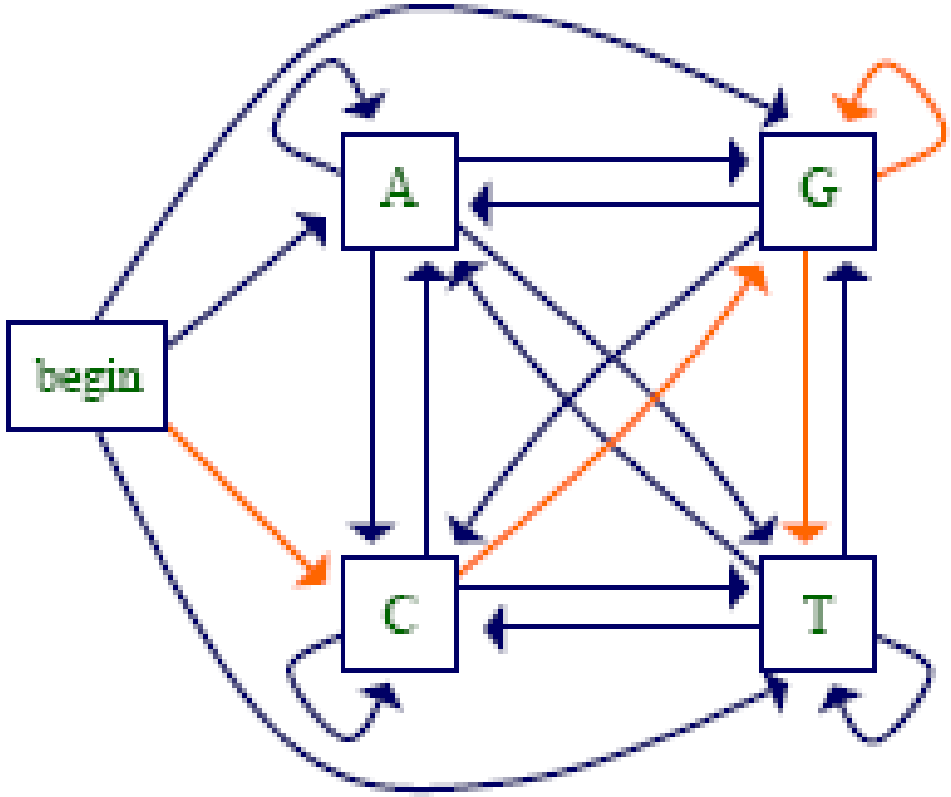
Hidden Markov models

Algorithms in Sequence Analysis

Content

- HMM introduction
  - what are the ingredients
  - A few uses
- The three main issues
  - decoding
  - evaluating
  - learning
- Overview of pair-HMMs (for pairwise sequence alignment) and profile-HMMs
- The HMM practical

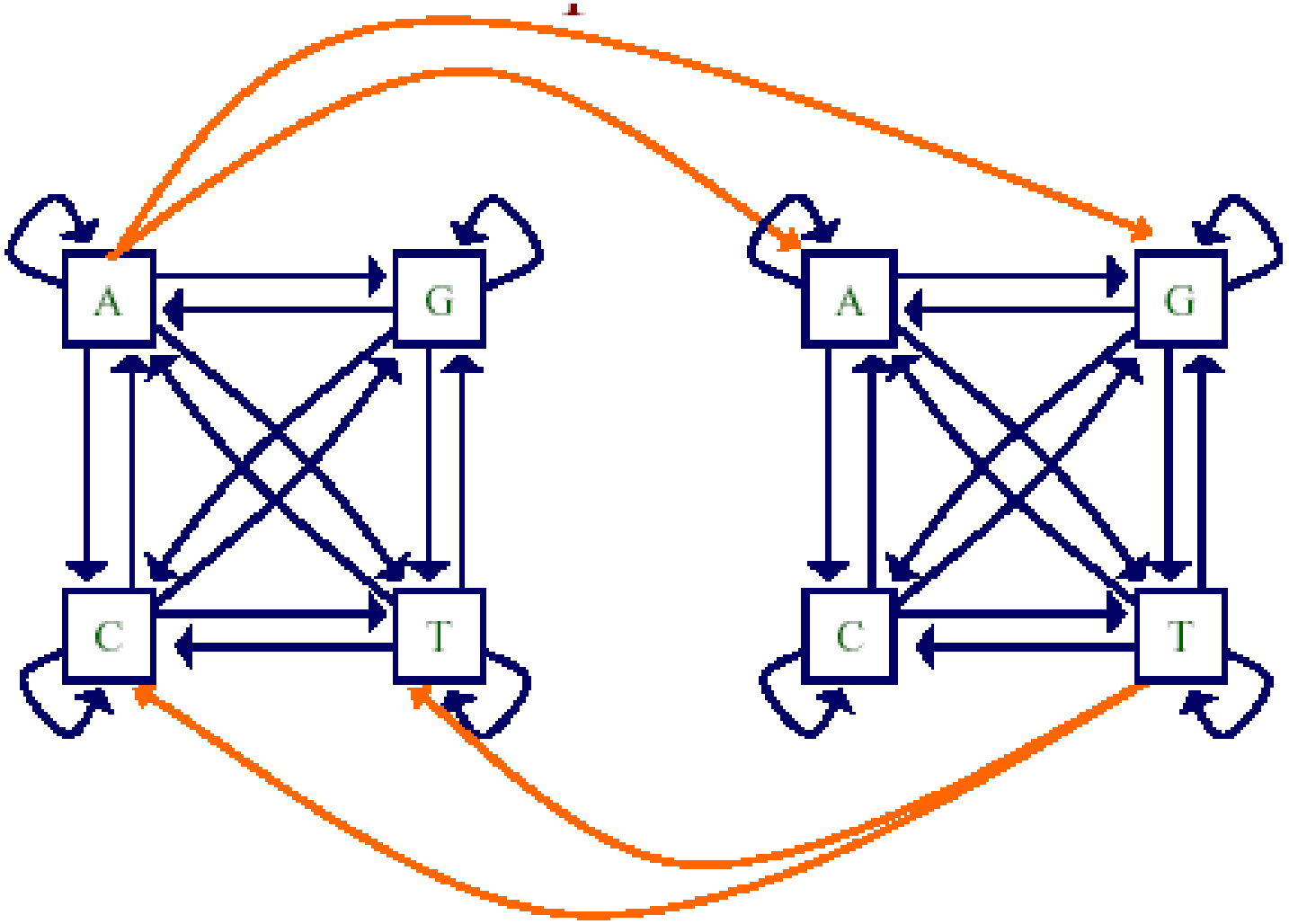
Markov Chain Models



$$\Pr(cggt) = \Pr(c)\Pr(g|c)\Pr(g|g)\Pr(t|g)$$

Given say a T in the input sequence, the state that emits it is exactly known

Hidden Markov models (HMMs)

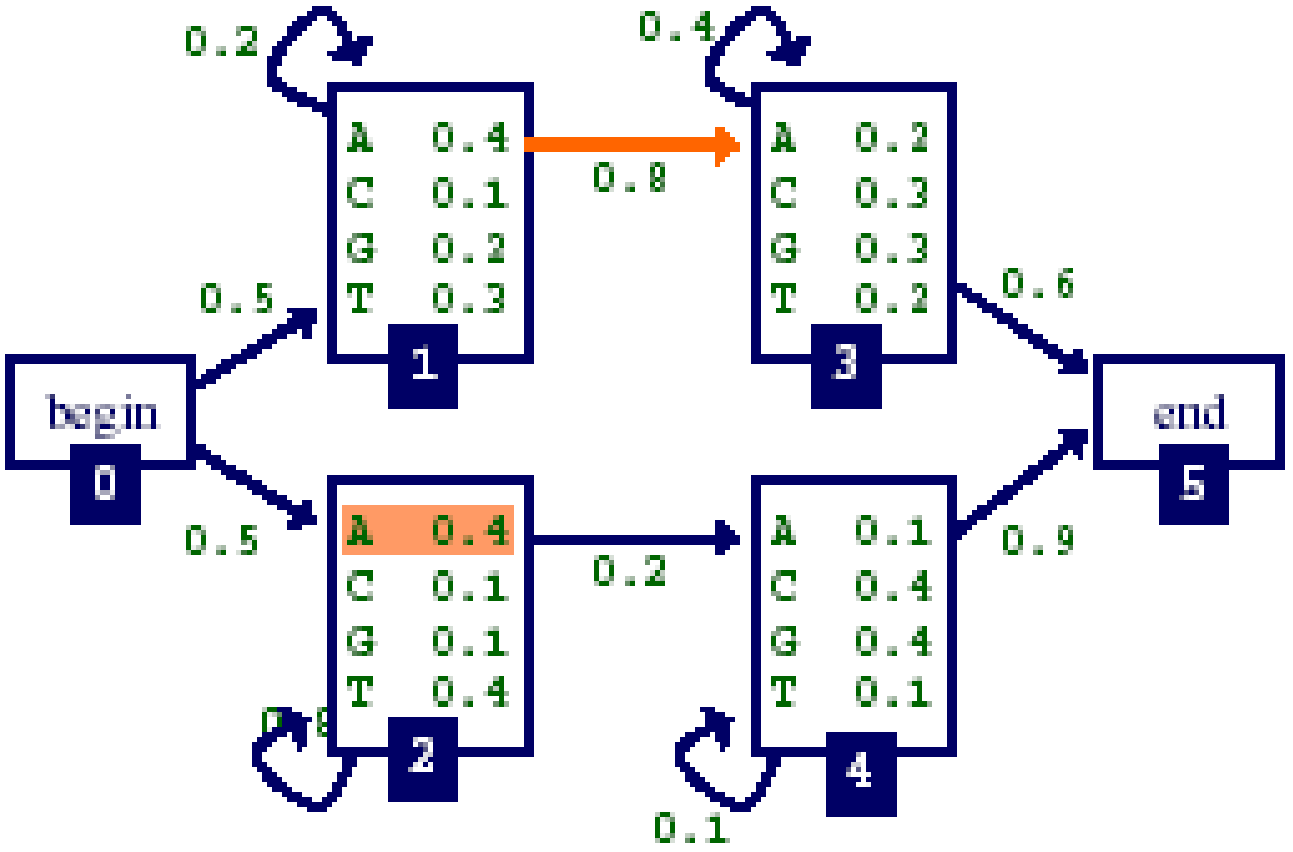


Given say a T in our input sequence, which state emitted it?

A Simple HMM

a_{13} probability of a transition from state 1 to state 3

$e_2(\mathrm{A})$ probability of emitting character *A* in state 2



Hidden Markov Models

1. S - observations
   - $x_1,\ldots,x_L$ – sequence of observations

2. Q - states
   - $\pi_1,\ldots,\pi_n$ – hidden sequence of states
   - $f=(f_1,\ldots,f_N)^T$ - initial probability of states
3. $A = (a_{ij})$ – transition matrix
4. $E = (e_i(x))$ – emission probabilities

Hidden Markov models (HMMs)

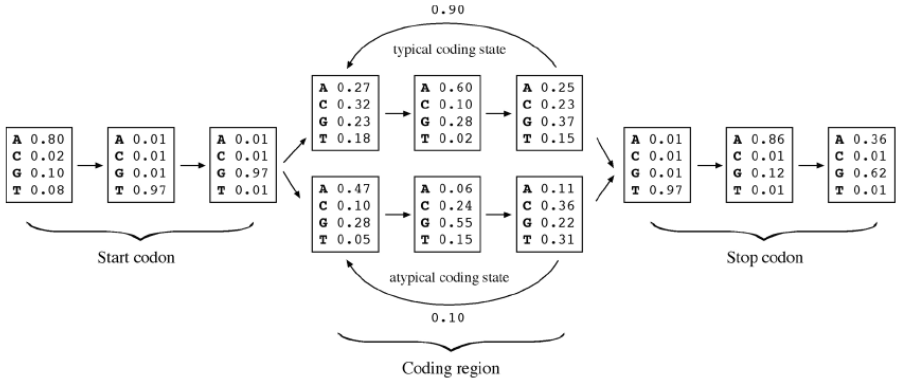
Hidden State

- We will distinguish between *observed* parts of a problem and *hidden* parts

- In the Markov models we have considered previously, it is clear which state accounts for each part of the observed sequence

- In an HMM there are multiple states that could account for each part of the observed sequence
  - this is the hidden part of the problem
  - states are **decoupled** from sequence symbols

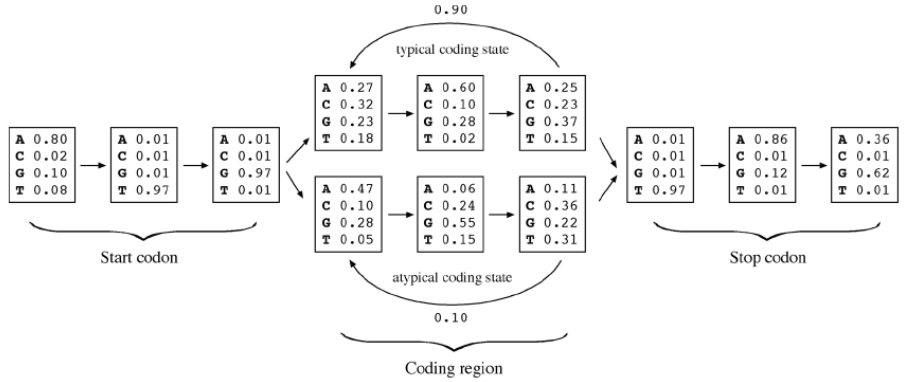
Using Hidden Markov Models



- An HMM allows the calculation of the probability of a given input sequence, i.e. you can calculate how likely it is that the sequence is *emitted* by the model.

- For a given input sequence, each sequence symbol maps to a state in the HMM and going to the next state proceeds via transition probabilities. The probability is calculated by taking the product of the **emission probabilities** (the probability of each sequence symbol in the state to which it maps) and the **transition probabilities** (when you move to a next state for the sequence)

Using Hidden Markov Models (2)



- There typically are very many different pathways for a single sequence through the HMM, one (or a number) of which will lead to an optimal score for that sequence.
  - Finding the **optimal path** (out of possibly very many) is called the **decoding problem**
  - The **total probability** of a given sequence is calculated by summing the probabilities of all different pathways through the model for that sequence. This is called the **evaluation problem**.
  - **Training** HMMs (using a set of training sequences) is required to find good values for the emission and transition probabilities

The Parameters of an HMM

- as in Markov chain models, we have transition probabilities

$$a_{kl} = \Pr(\pi_i = l \mid \pi_{i-1} = k)$$

probability of a transition from state k to l

π represents a path (sequence of states) through the model

- since we've decoupled states and characters, we might also have emission probabilities

$$e_k(b) = \Pr(x_i = b \mid \pi_i = k)$$

probability of emitting character b in state k

HMM example (assignment)

1. S – observations:

- X1=CGT
- X2=CTC

2. Q – states:

Q = {B (Begin), Q1, Q2, Q3, E (End)}

Initial probabilities f = {1.0, 0, 0, 0, 0}

HMM example (assignment)

3. A = (a_{ij}) – transition matrix

	B	Q1	Q2	Q3	E
B	0	1	0	0	0
Q1	0	0	0.4	0.4	0.2
Q2	0	0.8	0	0	0.2
Q3	0	0	1	0	0
E	0	0	0	0	0

3. E = ($e_i(x)$) – emission probabilities

	C	G	T
Q1	0.5	0.5	0
Q2	0.5	0	0.5
Q3	0	0.5	0.5

States B and E are silent, i.e. they don't emit symbols

Three basic problems of HMMs

Once we have an HMM, there are three problems of interest.

(**1) The Decoding Problem** – what is the most probable path?

Given a model and a sequence of observations, what is the most likely state sequence in the model that produced the observations?

(2) The Evaluation Problem – how likely is a given sequence?

Given an HMM and a sequence of observations, what is the probability that the observations are generated by the model?

(3) The Learning Problem – how to set the HMM parameters?

Given a model and a sequence of observations, how should we adjust the model parameters in order to maximize the generation of the (training) data

The Learning problem must be solved first, if we want to train an HMM for the subsequent use of recognition tasks (evaluation and decoding problems).

Three main solutions to the problems

Viterbi decoding

1) Decoding: Given A+E+(x_1, ...,x_n) what is (π_1, ..., π_n)?
the Viterbi algorithm

2) Evaluation: Given A+E what is the probability of (x_1, ...,x_n)?
the Forward algorithm

3) Learning: Given a set of (x_1, ...,x_n) what is A+E?
the Forward-Backward (Baum-Welch) algorithm

Three Important Questions

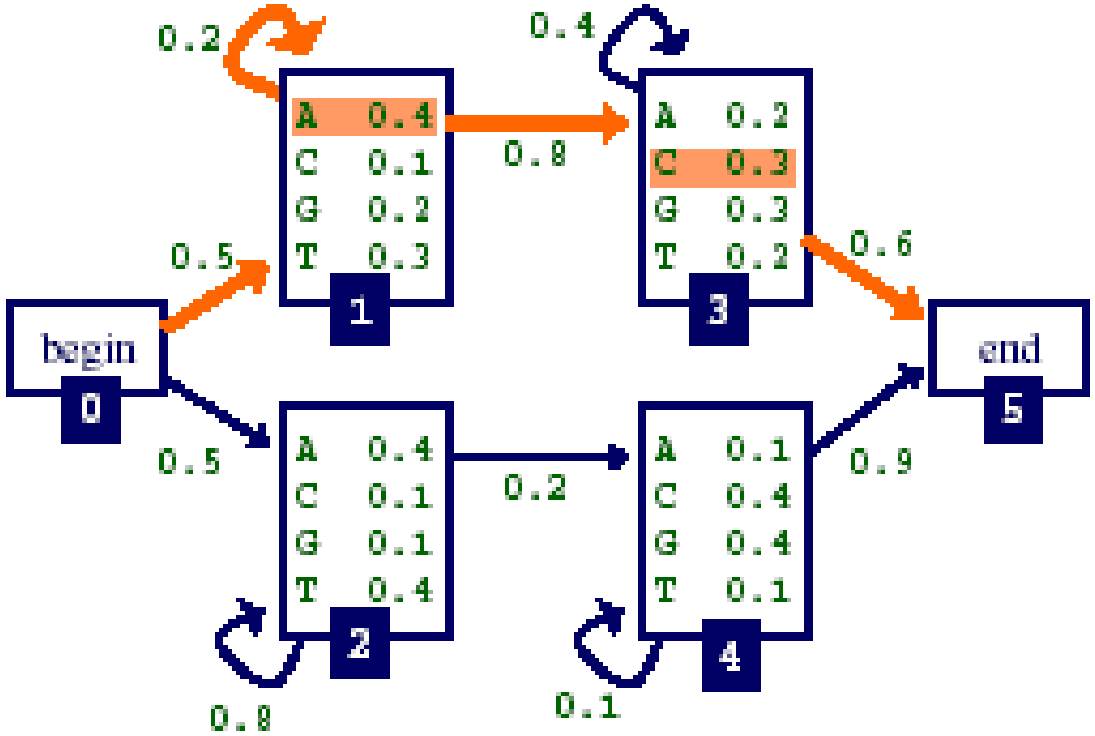
- What is the most probable “path” for generating a given sequence?

  → Viterbi algorithm

- How likely is a given sequence?

- How can we learn the HMM parameters given a set of sequences?

What is the most probable path?



$\Pr(AAC, \pi = \pi_0\pi_1\pi_1\pi_3\pi_5) = a_{01} \times e_1(A) \times a_{11} \times e_1(A) \times a_{13} \times e_3(C) \times a_{35}$
$= 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.6 = 0.002304$

$\Pr(AAC, \pi = \pi_0\pi_1\pi_3\pi_3\pi_5) = a_{01} \times e_1(A) \times a_{13} \times e_3(A) \times a_{33} \times e_3(C) \times a_{35}$
$= 0.5 \times 0.4 \times 0.8 \times 0.2 \times 0.4 \times 0.3 \times 0.6 = 0.002304$

$\Pr(AAC, \pi = \pi_0\pi_2\pi_2\pi_4\pi_5) = a_{02} \times e_2(A) \times a_{22} \times e_2(A) \times a_{24} \times e_4(C) \times a_{45}$
$= 0.5 \times 0.4 \times 0.8 \times 0.4 \times 0.2 \times 0.4 \times 0.9 = 0.004608$

Many alternative paths are possible, the highest scoring should be selected

Given a path, what is the score of a sequence generated by taking this path?

Multiply the probabilities

Probability that path is taken and sequence generated:

$$\Pr(x_1 \ldots x_L, \pi_0 \ldots \pi_N) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$

Or in short:

$$P(x, \pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$ (Durbin et al.)

(assuming begin (0) and end (π_{L+1}) states are the only silent states on path)

Finding the Most Probable Path: The Viterbi Algorithm

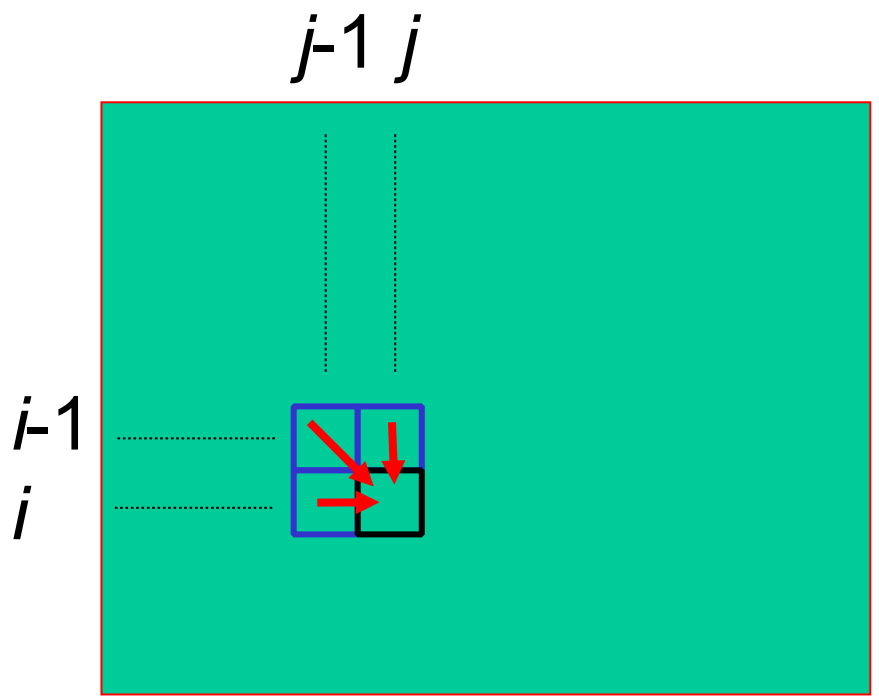
- Define $v_k(i)$ to be the probability of the single most probable path accounting for the first i characters of sequence x and ending in state k

- We want to compute $v_N(L)$, the probability of the most probable path accounting for all of the sequence and ending in the end state

- Can be defined recursively

- Can use **Dynamic Programming** to find $v_N(L)$ efficiently

Recap: Dynamic Programming



For each cell you must check three cells that 'transition' into it.

With HMMs, it is possible that many more 'cells' (now called states) should be checked.

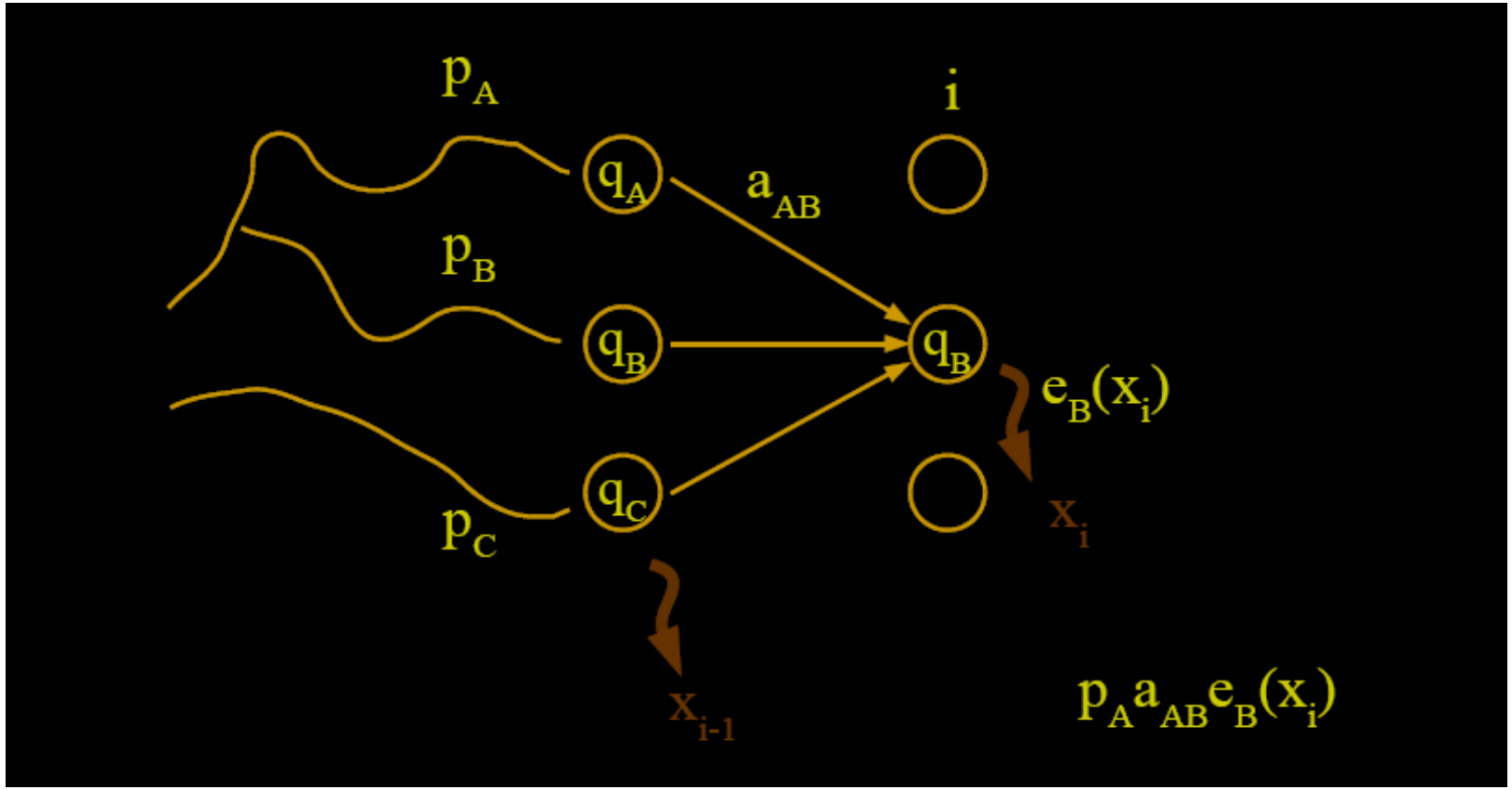
$$H(i,j) = Max \begin{cases} H(i\text{-}1,j\text{-}1) + s(i,j) & \text{diagonal} \\ H(i\text{-}1,j) - g & \text{vertical} \\ H(i,j\text{-}1) - g & \text{horizontal} \end{cases}$$

This is a recursive formula

Finding the Most Probable Path: The Viterbi Algorithm

Viterbi – recursive step

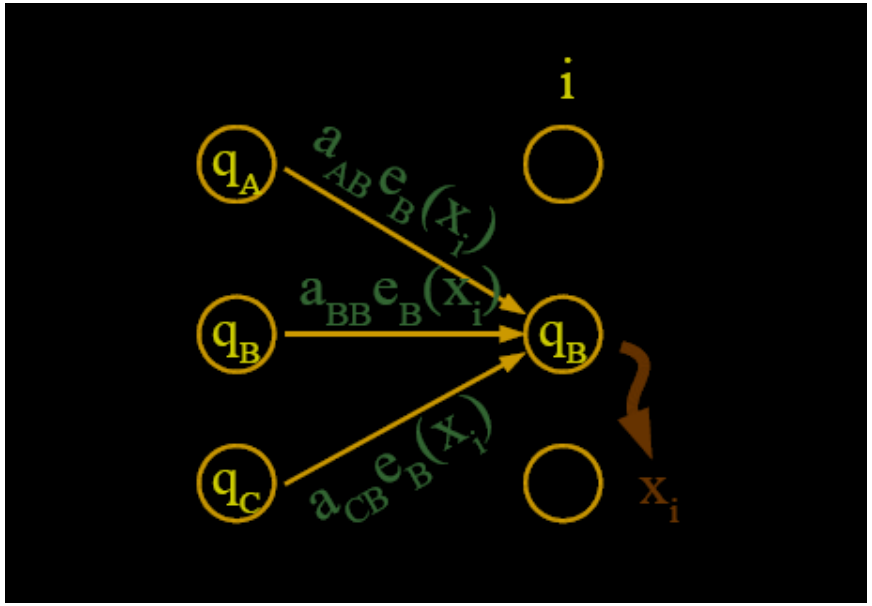
What is the probability of the path which ends with q_A->q_B and emission E_B?



Finding the Most Probable Path: The Viterbi Algorithm

Viterbi – recursive step

What is the most probable path into state B in step i?



Many more than three paths can lead into state q_a…

Just as with DP, select the one with the maximal score

If state q_B is indexed as *k:* $v_l(i) = e_l(x_i) \max_k [v_k(i-1) a_{kl}]$

Finding the Most Probable Path: The Viterbi Algorithm

Initialisation:

$v_0(0) = 1$ (start), $v_k(0) = 0$ ($k > 0$, all other emitting states)

.. can only start in start state v_0

Have a begin state, can only begin at the begin state

Recursion for emitting states ($i = 1 \ldots L$):

$$v_l(i) = e_l(x_i) \max_k [v_k(i-1)a_{kl}]$$

Emission of state to output times max (previous score * transition from previous state to this state)

Index for state π_l

$$\text{ptr}_l(i) = \arg\max_k [v_k(i-1)a_{kl}]$$

..remember path for traceback

After computing, record direction comes from, and backtrack in the end.

Recursion for silent states:

Skipped by Durbin..

$$v_l(i) = \max_k [v_k(i)a_{kl}]$$

.. π_l is a silent state, so no emission probability and sequence stays at same position

$$\text{ptr}_l(i) = \arg\max_k [v_k(i)a_{kl}]$$

Finding the Most Probable Path: The Viterbi Algorithm

- Termination:

this is the optimal (highest scoring) path

$$P(x, \pi^*) = \max_k (v_k(L) a_{k0})$$

$$\pi_L^* = \text{argmax}_k (v_k(L) a_{k0})$$

this is the index of the end state (Durbin et al. convention)

- Traceback:

$$(i = L \ldots 1)\text{: } \pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$$

Finding the Most Probable Path: The Viterbi Algorithm

How to do the bookkeeping

- Andrew Viterbi used **Manhattan grid model** to solve the Decoding problem.

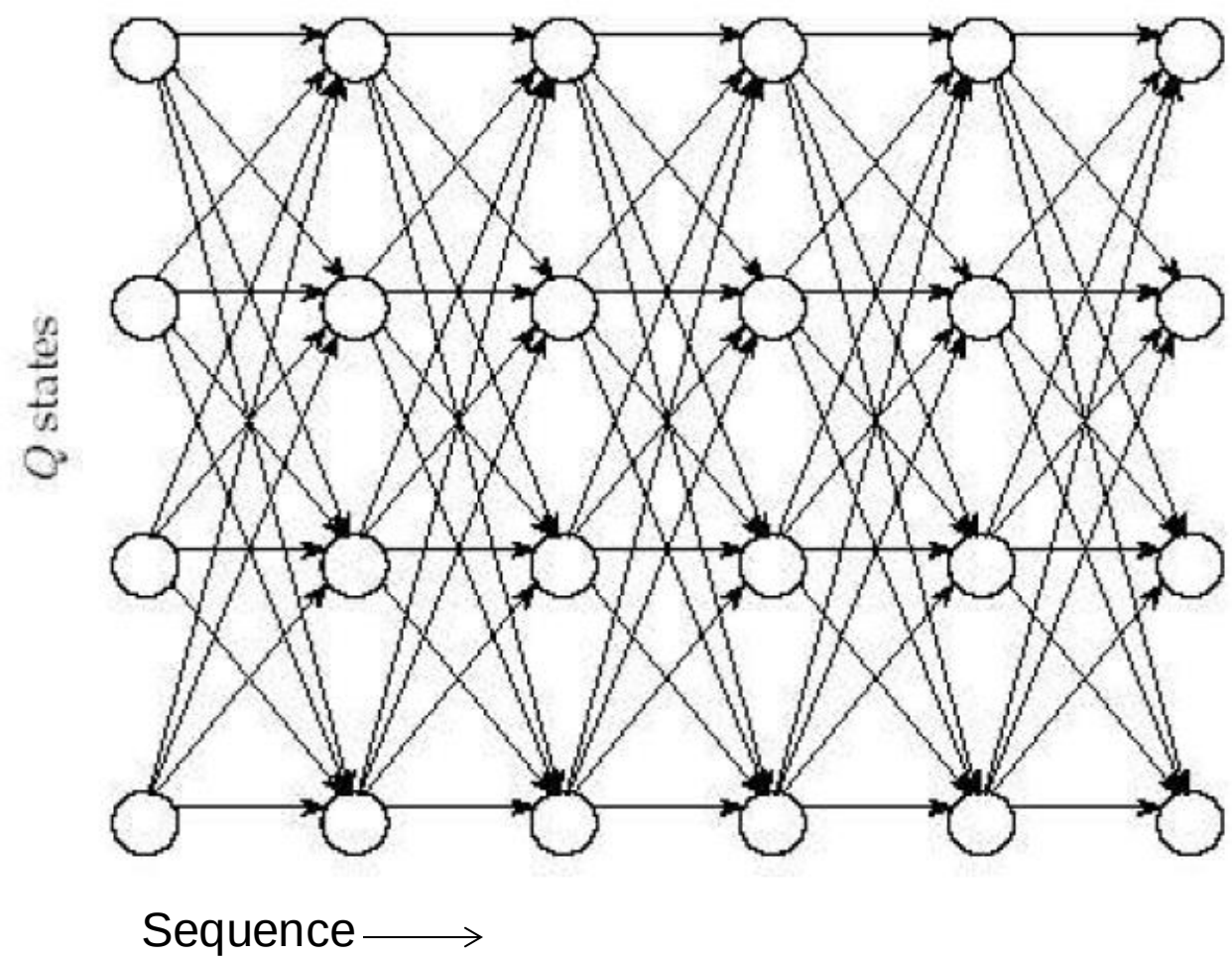
- Every choice of $\pi = \pi_1 \ldots \pi_n$ corresponds to a path in the graph.

- Only valid direction in the graph is *eastward.*

- This graph has $N^2(n\text{-}1)$ edges, where N is number of states, and n the number of sequence symbols (L) plus the number of silent states

Finding the Most Probable Path: The Viterbi Algorithm

Manhattan grid model



This diagram is also called a **trellis**. Each column in the trellis shows all HMM states; it is executed from left to right (i.e. ‘eastward’) by going through the sequence of observations.

HMM assignment: Viterbi Algorithm trellis

Consider the following simple 4-state *HMM f*or gene prediction in two main regions:

- States *Q* = {B (Begin), Q1 (Region 1), Q2 (Region 2), E (End)}
- Alphabet Σ = {A, T, G, C}
- Transition probabilities between the states *A* =

	B	Q1	Q2	E
B	0	0.5	0.5	0
Q1	0	0.7	0.1	0.2
Q2	0	0.5	0.3	0.2
E	0	0	0	0

- Emission probabilities E =

	A	T	G	C
Q1	0.25	0.25	0.25	0.25
Q2	0.1	0.1	0.5	0.3

- **Trellis to calculate optimal path for sequence ATG:**

	-	A	T	G	-
	0	1	2	3	4
B					
Q1					
Q2					
E					

Column length is number of states

Row length is pathlength of sequence, so sequence length + number of silent sites (B and E)

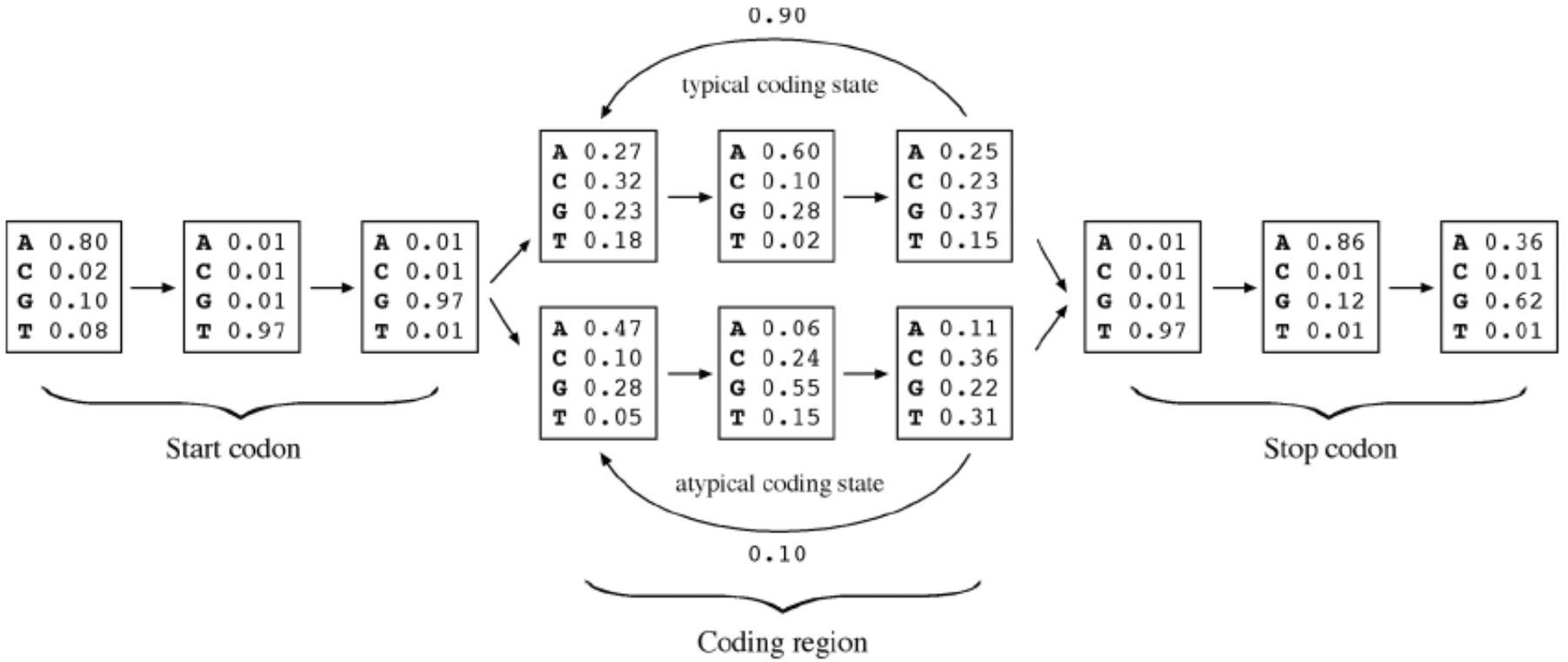
Example: Prokaryote gene prediction

Gene Prediction using Hidden Markov Models

- It has been shown that the gene content and length distribution of prokaryotic genes can be either **typical** or **atypical**.
  - Typical genes are in the range of 100 to 500 amino acids with a nucleotide distribution (codon bias) typical of the organism.
  - Atypical genes are shorter or longer with different nucleotide statistics. These genes tend to escape detection using the typical gene model.
- This means that, to make the algorithm capable of fully describing all genes in a genome, more than one Markov model is needed.
- Combining different Markov models that represent typical and atypical nucleotide distributions leads to a HMM prediction algorithm (see next slide)

Prokaryote gene prediction

Gene Prediction using Hidden Markov Models



A simplified **inhomogeneous second-order HMM** for prokaryotic gene prediction that includes a statistical model for start codons, stop codons, and the rest of the codons in a gene sequence represented by a typical model and an atypical model.

HMM prokaryote gene prediction methods

For reference

GeneMark (http://opal.biology.gatech.edu/GeneMark/) is a suite of gene prediction programs based on fifth-order HMMs.

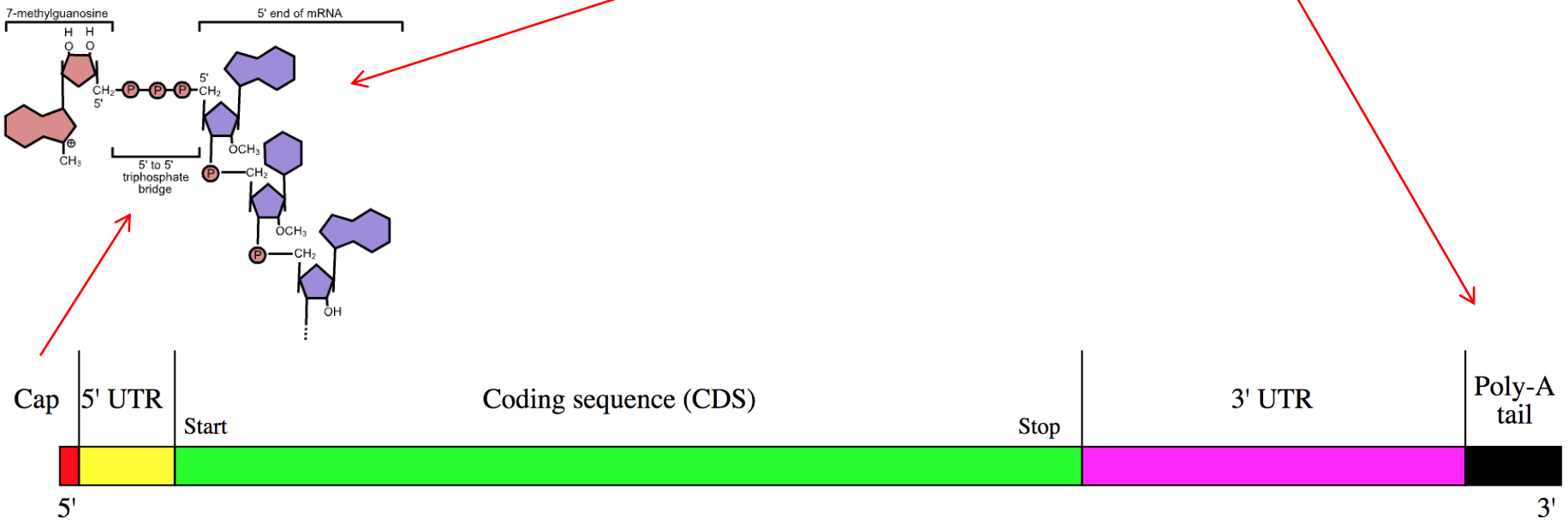
- The main program–GeneMark.hmm– is trained on a number of complete microbial genomes. If the sequence to be predicted is from a nonlisted organism, the most closely related organism can be chosen as the basis for computation.
- **GeneMarkS** is a self-trained program to predict new organisms – the user must provide at least 100 kbp of sequence on which to train the model.
  - If the query sequence is shorter than 100 kbp, a GeneMark heuristic program can be used but with some loss of accuracy.
- GeneMark also has a variant for eukaryotic gene prediction using a HMM.

FGENESB (www.softberry.com) is also based on fifth-order HMMs for detecting coding regions. The program is specifically trained for bacterial sequences.

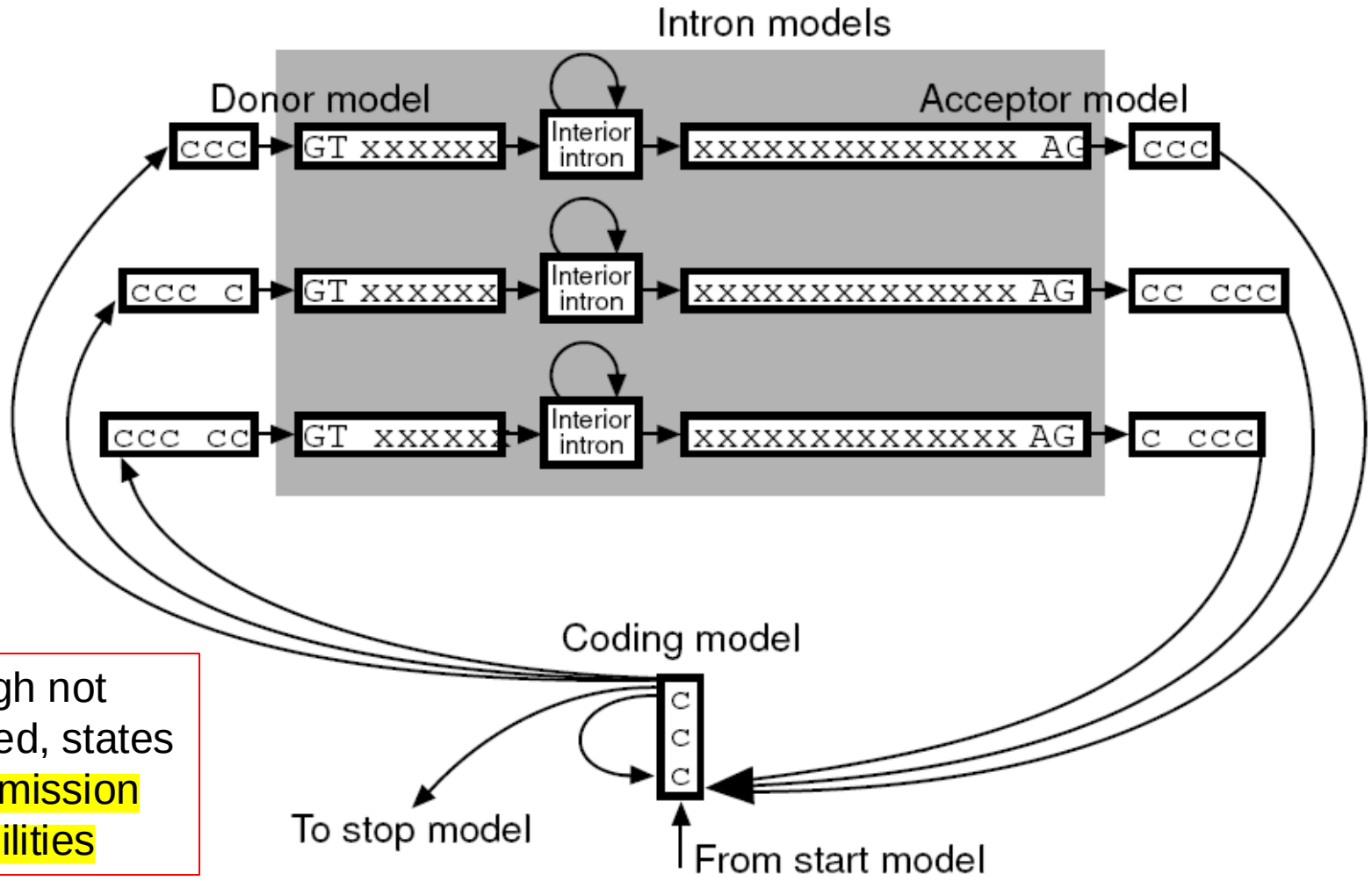
- The method finds the optimal match for the query sequence with the intrinsic model (the optimal path).
- Linear discriminant analysis (LDA) is then used to further distinguish coding signals from noncoding signals.

Eukaryotic Gene Prediction

- The most important difference between eukaryotic and prokaryotic gene prediction is *splicing* involving **exons** and **introns** on eukaryotic mRNA molecules.
- Splicing is one of three operations that happen on an *immature* eukaryotic mRNA molecule, which is then converted to a *mature* mRNA ready for translation.
  - The other two operations are **5' mRNA capping** and **poly-adenylation**



HMM for Eukaryotic Gene Finding



Although not indicated, states have emission probabilities

To allow for splicing in three different frames, three intron models are needed. To get the frame correct, 'spacer states' are added before and after the intron models.

Figure from A. Krogh, An Introduction to Hidden Markov Models for Biological Sequences

GENSCAN HMM for Eukaryotic Gene Finding (Burge & Karlin, 1997)

Each shape represents a functional unit of a gene or genomic region

Pairs of intron/exon units represent the different ways an intron can interrupt a coding sequence (after 1st base in codon, after 2nd base or after 3rd base)

Complementary submodel (not shown) detects genes on opposite DNA strand (same model)

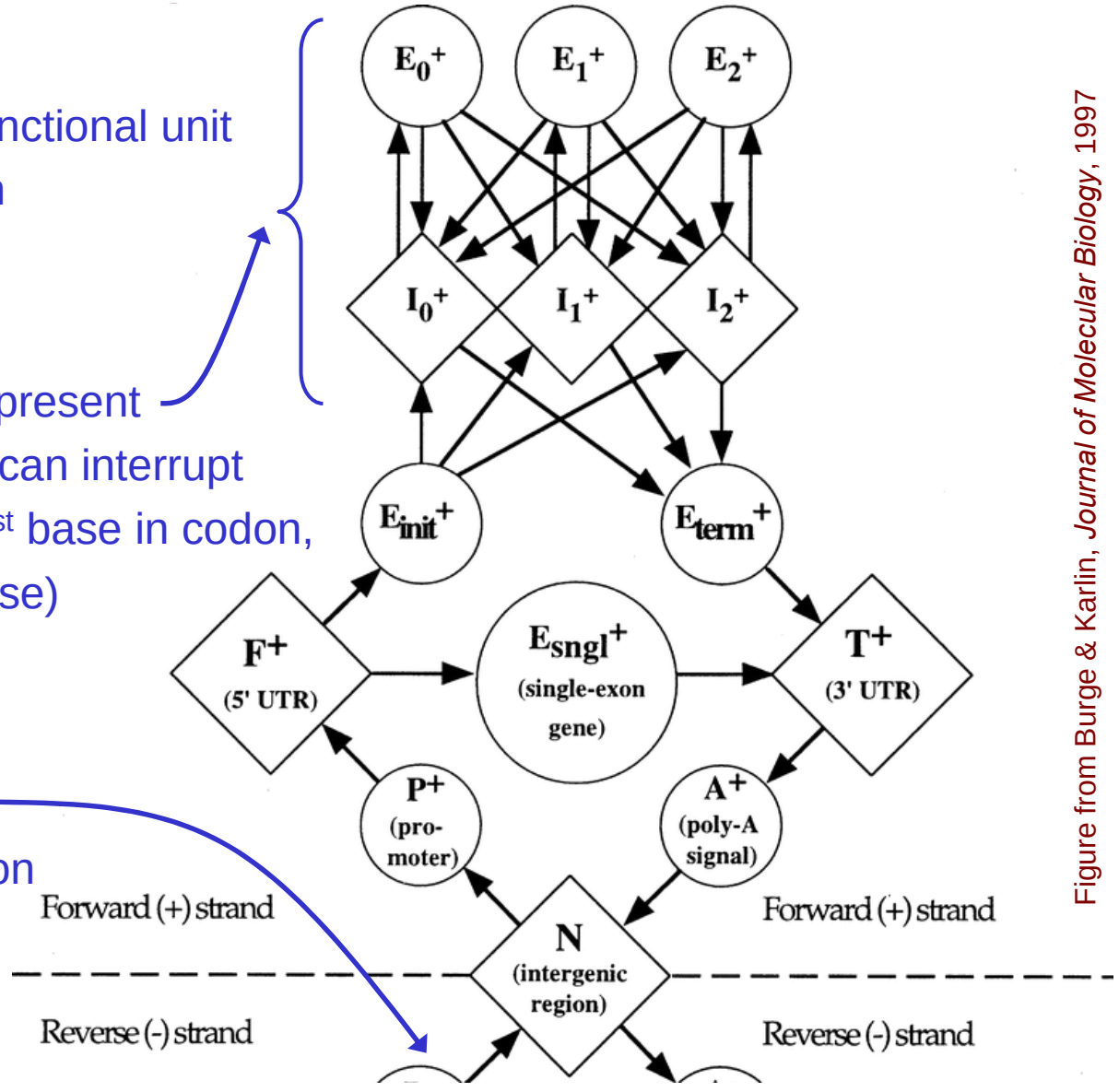
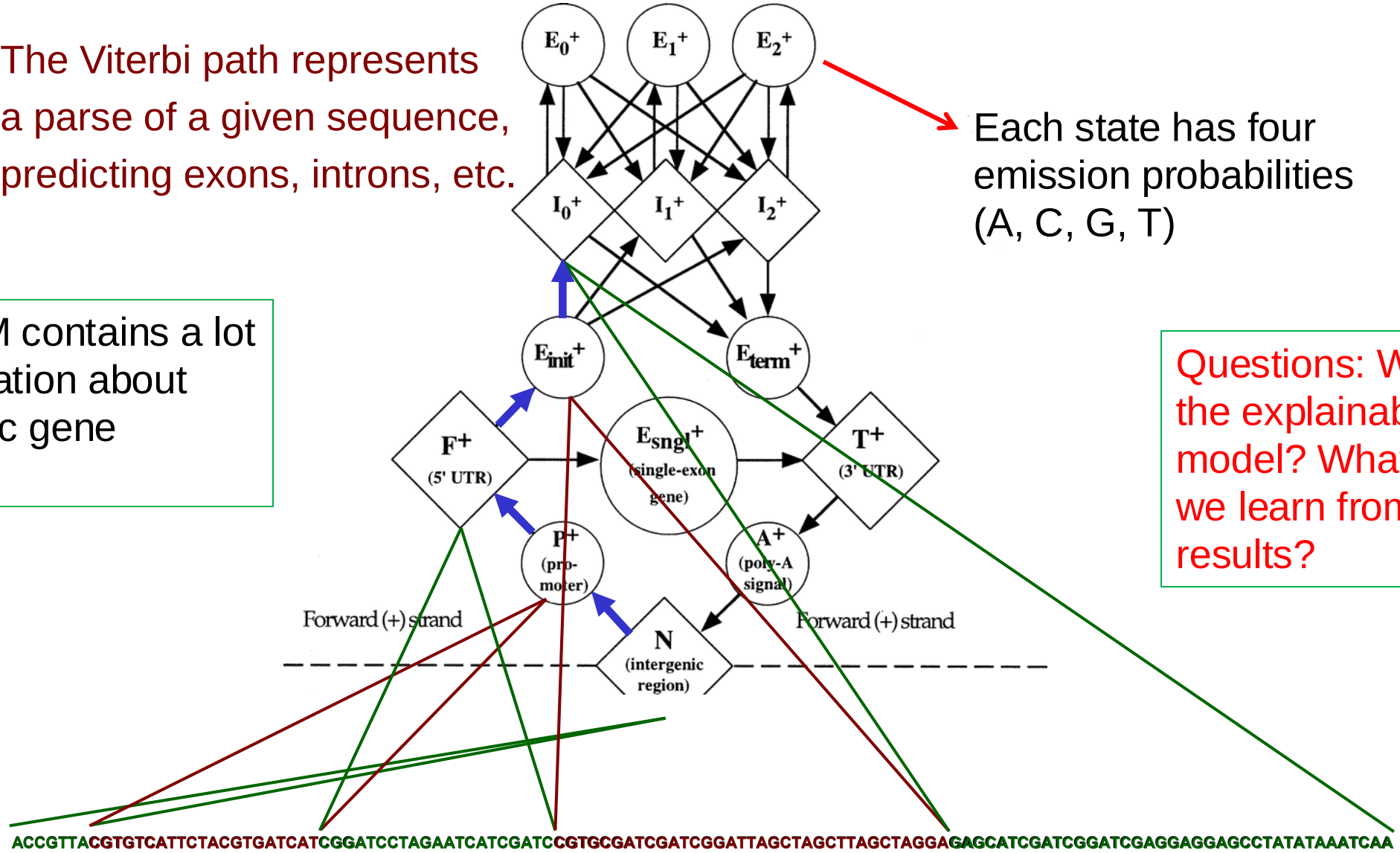


Figure from Burge & Karlin, *Journal of Molecular Biology*, 1997

GENSCAN HMM for Eukaryotic Gene Finding (Burge & Karlin, 1997)

The Viterbi path represents a parse of a given sequence, predicting exons, introns, etc.



Each state has four emission probabilities (A, C, G, T)

The HMM contains a lot of information about eukaryotic gene structure.

Questions: What about the explainability of the model? What/how can we learn from the results?

Calculating probabilities using the Viterbi algorithm

- Multiplying many (small) probabilities will lead to underflow (i.e. number cannot be distinguished from zero) and other computational problems
- To deal with this, take the log of all probabilities and use addition instead of multiplication
  - Remember that log(ab) = log(a) + log(b)
- Many CPUs are faster at adding numbers than multiplying them
- **We won't go to log space in the HMM assignment**

Three Important Questions

- What is the most probable “path” for generating a given sequence?

- How likely is a given sequence?

  → Forward algorithm

- How can we learn the HMM parameters given a set of sequences?

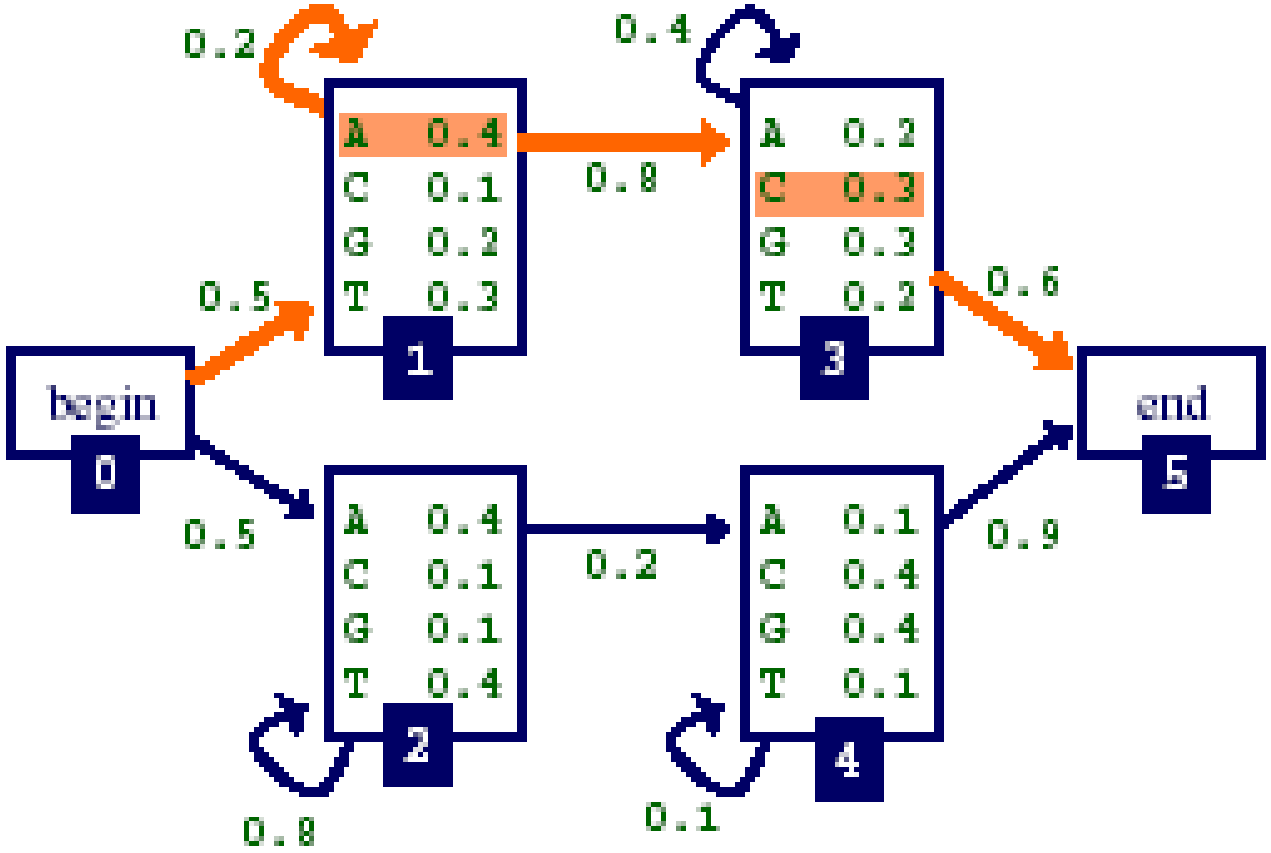
How Likely is a Given Sequence through a Given Path?

- The probability that path is taken and the sequence is generated:

$$P(x,\pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$

- (assuming begin/end are the only silent states on path)

How Likely is a Given Sequence?



$$\Pr(\mathrm{AAC}, \pi) = a_{01} \times e_1(\mathrm{A}) \times a_{11} \times e_1(\mathrm{A}) \times a_{13} \times e_3(\mathrm{C}) \times a_{35}$$

$$= 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.6$$

This is the probability of sequence AAC using path {0,1,1,3,5} – but many alternative paths are possible

How Likely is a Given Sequence?

The probability over *all* paths is:

$$\Pr(x_1 \ldots x_L) = \sum_{\pi} \Pr(x_1 \ldots x_L, \underbrace{\pi_0 \ldots \pi_N}_{\pi})$$

but the number of paths can be exponential in the length of the sequence...

- Nonetheless, we need to calculate the probability of a given sequence by summing the probabilities over **all** paths (producing the query sequence) through the HMM

- Fortunately, the **Forward algorithm** enables us to compute this efficiently

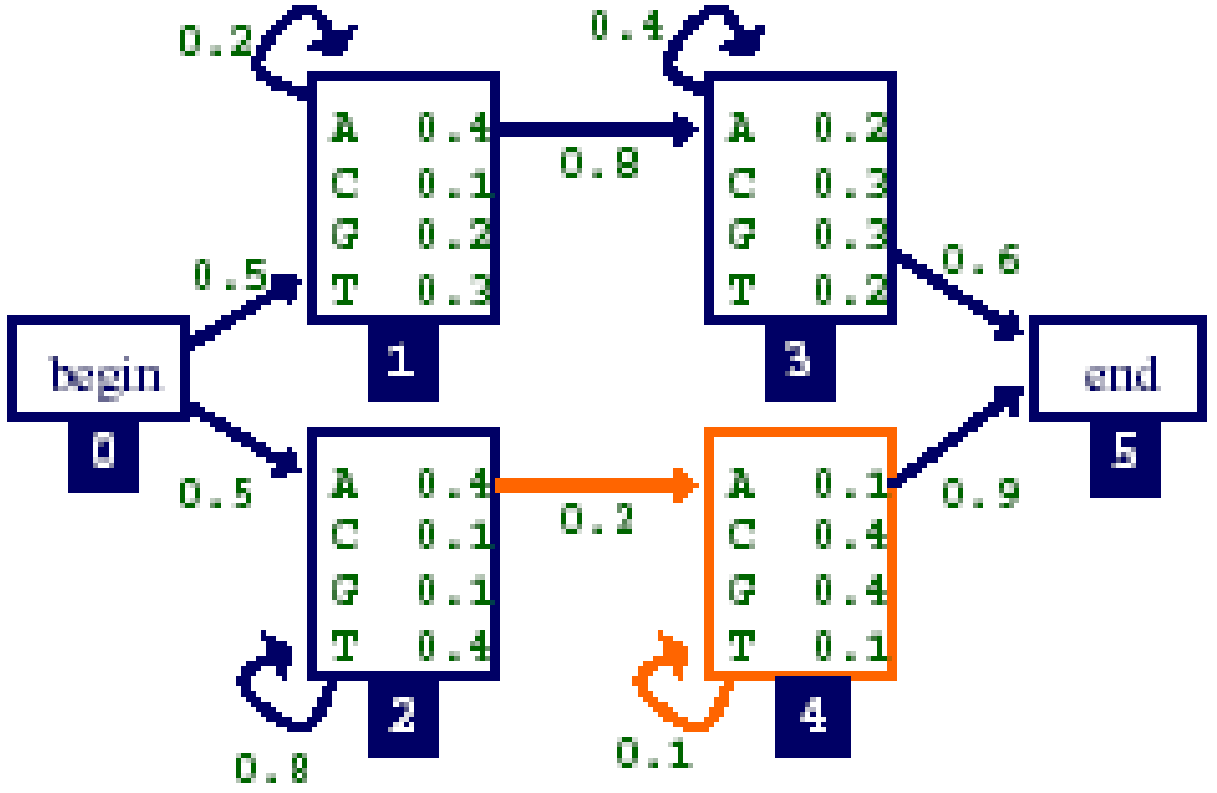
How Likely is a Given Sequence: The Forward Algorithm

Change max to sum in the Viterbi decoding algorithm to obtain the forward algorithm

- Define $f_k(i)$ to be the probability of being in state k
- Having observed the first i characters of x we want to compute $f_N(L)$, the probability of being in the end state having observed all characters of x (the whole sequence)
- As for the Viterbi algorithm, we can define this recursively by using Dynamic Programming

How Likely is a Given Sequence:

- because of the Markov property, don't have to explicitly enumerate every path – use dynamic programming instead



- e.g. compute $f_4(i)$ using $f_2(i-1),\ f_4(i-1)$

The forward algorithm

- Initialisation:

probability that we're in start state and have observed 0 characters from the sequence

$$f_0(0) = 1 \text{ (start)},$$

$$f_k(0) = 0 \text{ for } k > 0$$

- Recursion $(i = 1 \ldots L)$:

$$f_l(i) = e_l(x_i)\Sigma_k f_k(i\text{-}1)a_{kl} \quad \text{(emitting states)},$$

$$f_l(i) = \Sigma_k f_k(i)a_{kl} \quad \text{(silent states)}$$

Skipped by Durbin..

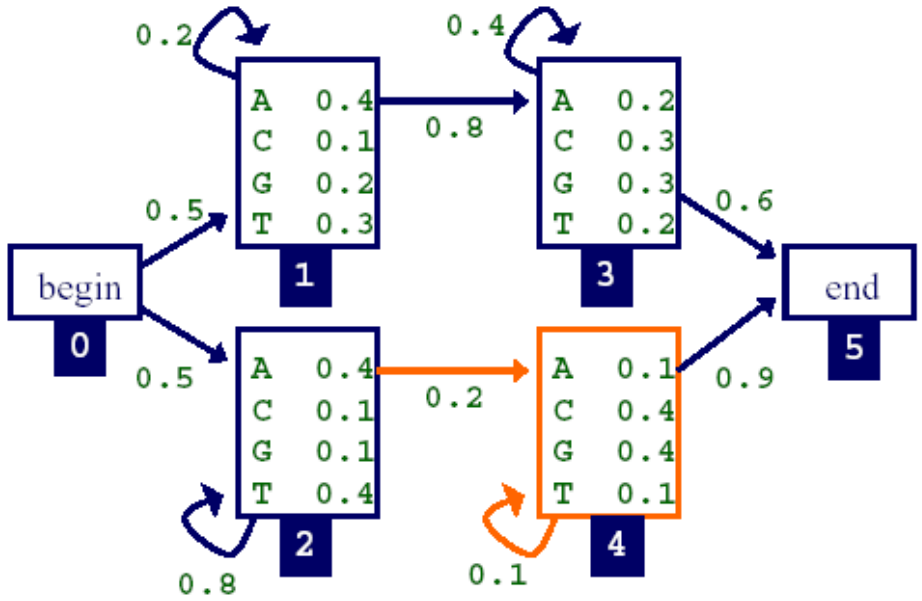
- Termination:

this is the index of the end state (Durbin book)

$$P(x) = P(x_1 \ldots x_L) = f_0(L) = \Sigma_k f_k(L)a_{k0}$$

probability that we are in the end state and have observed the entire sequence

Forward algorithm example



- given the sequence x = TAGA
- initialization

$$f_0(0) = 1 \qquad f_1(0) = 0 \quad \cdots \quad f_5(0) = 0$$

- computing other values

$$f_1(1) = e_1(T) \times (f_0(0) \times a_{01} + f_1(0) a_{11}) =$$

this is the T in the sequence

$$0.3 \times (1 \times 0.5 + 0 \times 0.2) = 0.15$$

$$f_2(1) = 0.4 \times (1 \times 0.5 + 0 \times 0.8)$$

$$f_1(2) = e_1(A) \times (f_0(1) \times a_{01} + f_1(1) a_{11}) =$$

this is the first A in the sequence

$$0.4 \times (0 \times 0.5 + 0.15 \times 0.2)$$

• • •

$$\Pr(TAGA) = f_5(4) = (f_3(4) \times a_{35} + f_4(4) a_{45})$$

..all the time calculate over all possible ways to get to state considered, and sum the probabilities

..the end state (5) is a silent state

Thinking about the Viterbi and Forward Algorithm

- The Viterbi algorithm is to find the optimal path of the sequence through the model (decoding problem)
- The Forward algorithm is to calculate the total probability of a sequence x being generated by the model (by summing over all paths)
- Both algorithms can be implemented using Dynamic Programming
  - With only one small difference:

    Change max to Σ (i.e., use max in Viterbi and summation in Forward algorithm)

Calculating probabilities using the Forward Algorithm

- Multiplying many (small) probabilities will lead to underflow and other computational problems
- However, just taking the log of probabilities as for Viterbi algorithm is not possible because the log of a **sum** of probabilities must be calculated for the Forward algorithm (so exponents and log function should be calculated which are computationally expensive)
- Durbin et al. (Section 3.6) give two ways to deal with this problem
- Again, we won't be doing this in the assignment.

Three Important Questions

- What is the most probable “path” for generating a given sequence?

- How likely is a given sequence?

- How can we learn the HMM parameters given a set of sequences?

  → Forward-Backward (Baum-Welch) algorithm

The Learning Problem

- Learning problem is how to adjust the HMM parameters, so that the given set of observations (called the training set) is represented by the model in the best way for the intended application.
- This means that the ‘quantity’ we wish to optimize during the learning process can be different from application to application. In other words, there may be several optimization criteria for learning, out of which a suitable one should be selected depending on the application.
- There are two main optimization criteria found in the literature; Maximum Likelihood (ML) and Maximum Mutual Information (MMI).
  - We will look at ML.

The Learning Task

- Given:
  – a model
  – a set of sequences (the training set)

- Do:
  – find the most likely parameters to explain the training sequences

- The goal is to find a model that *generalizes* well to sequences we haven't seen before

Learning Parameters

- If we know the state path for each training sequence, learning the model parameters is simple
  - no hidden state during training
  - count how often each parameter is used
  - normalize to get probabilities (next slide)
  - process just like for Markov chain models

- If we don't know the path for each training sequence, how can we determine the counts?
  - key idea: estimate the counts by considering every path weighted by its probability

Learning Parameters if state paths are known

- If we know the state path for each training sequence, learning the model parameters is simple:

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

.. just generate the training sequences and count the number of emissions of a given symbol in state *k* and make it a probability by normalising using the total number of emissions of state *k* (the number of state visits)

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}}$$

..just generate the training sequences and count the number of transitions from state *k* to *l* and make it a probability through dividing by the total number of transitions out of state *k*

$e_k(b)$ and a_{kl} are estimated here using **maximum likelihood**

Maximum Likelihood Estimation

- If there are not many training data, ML estimates can easily be *overfitted*, reducing the genericity of the model or leading to undefined values (e.g. if a state is never visited by the training sequences, leading to zero divide)
- To remedy this, predetermined pseudocounts can be added to $E_k(b)$ and A_{kl} values:

$$E_k(b) \longleftarrow E_k(b) + r_k(b)$$
$$A_{kl} \longleftarrow A_{kl} + r_{kl}$$

pseudocounts

- Pseudocounts should reflect preconceived ideas about biases.

Learning Parameters if we don't know the state paths

The Baum-Welch Algorithm

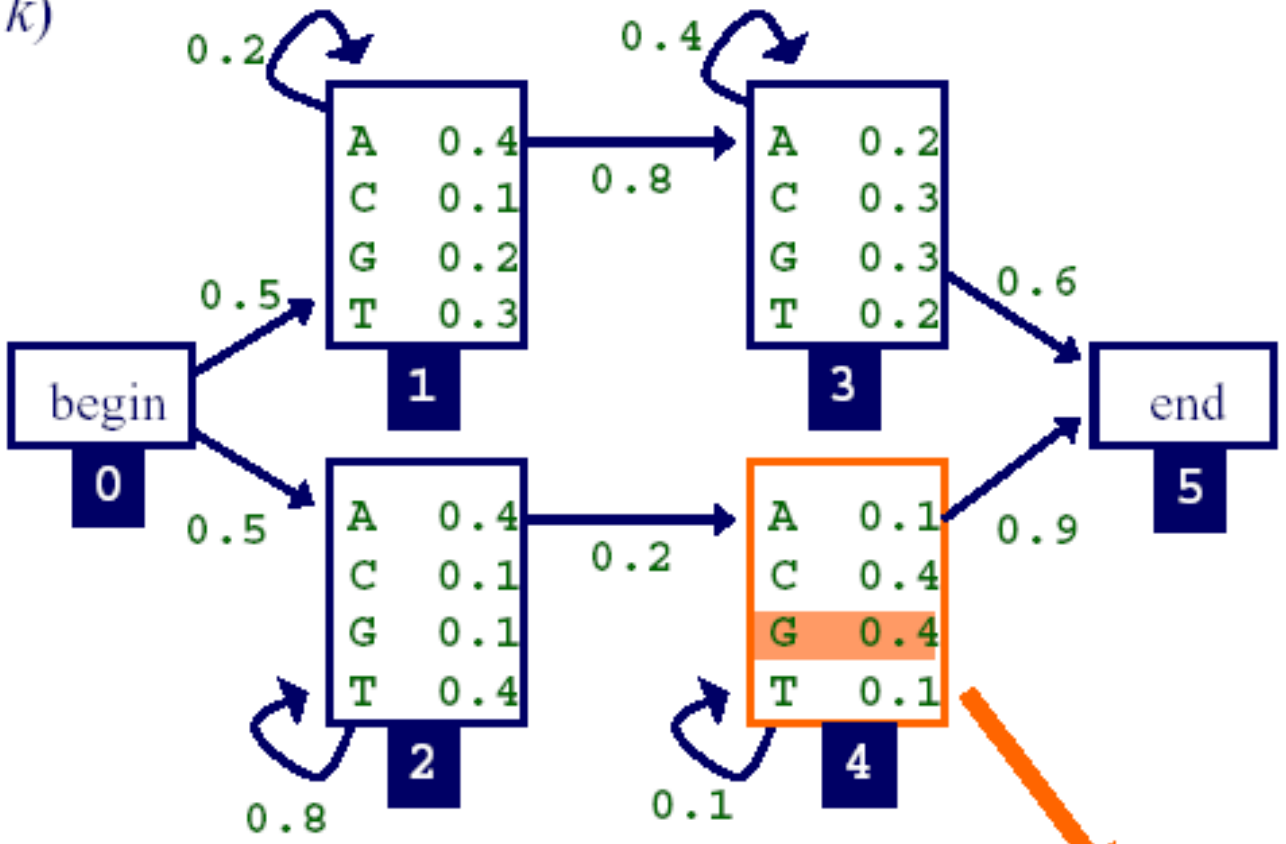
- An EM (expectation maximization) approach, iteratively applying the Forward-Backward algorithm and Maximum Likelihood estimation

- Algorithm sketch:
  - **initialize parameters of model (e.g. using a prior)**
  - **iterate until convergence:**
    - Calculate the *expected* number of times each transition or emission is used by applying the Forward-Backward algorithm to the training data (**expectation step**)
    - Adjust the parameters to *maximize* the likelihood of these expected values, using ML estimates as if state paths are known (**maximization step**)

- Baum-Welch has as important feature that it always converges
  - At each iteration the posterior probabilities calculated in the preceding iteration are used as priors.

The Expectation step

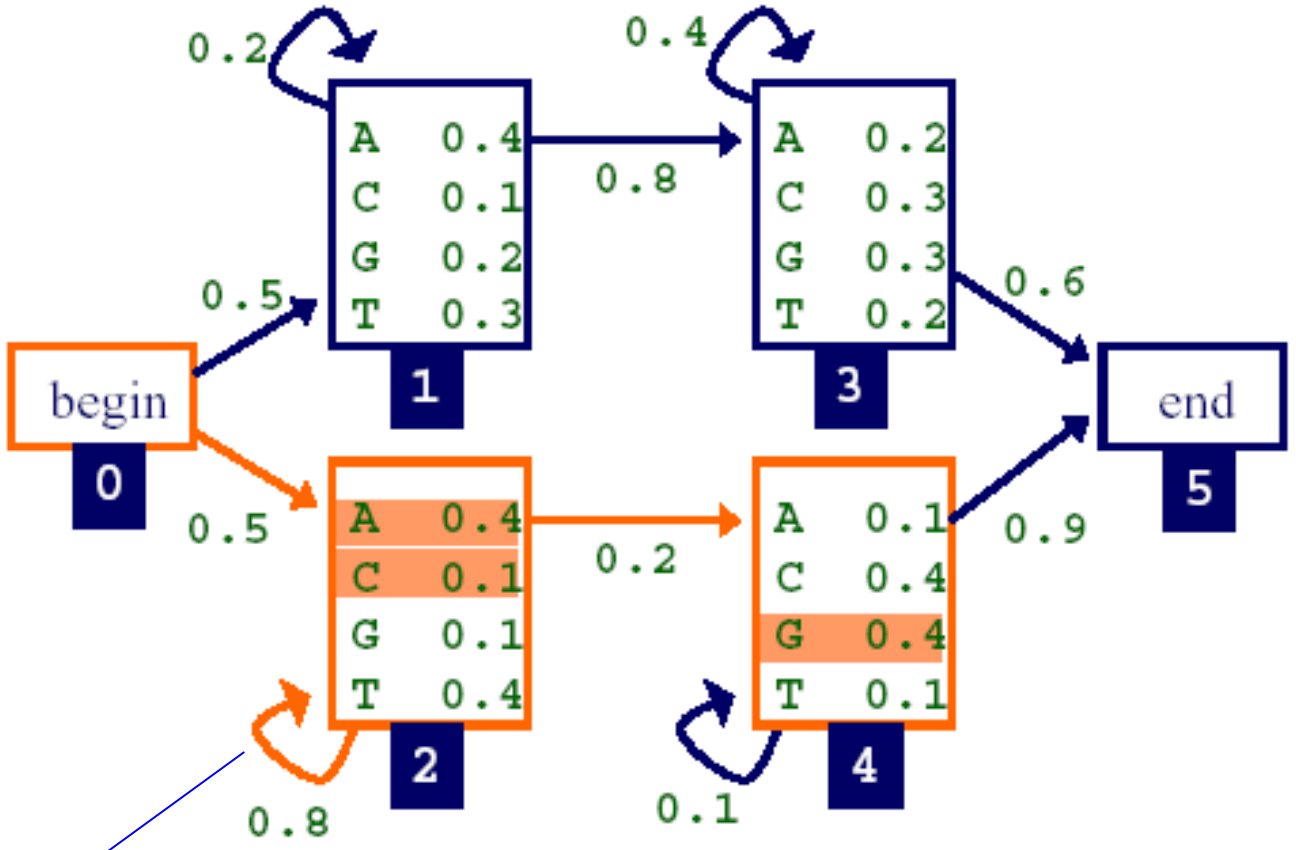
- we want to know the probability of producing sequence x with the i th symbol being produced by state k (for all x, i and k)



How likely is emitting G from state 4 if the sequence CAGT is produced

The Expectation step

- the forward algorithm gives us $f_k(i)$, the probability of being in state k having observed the first i characters of x

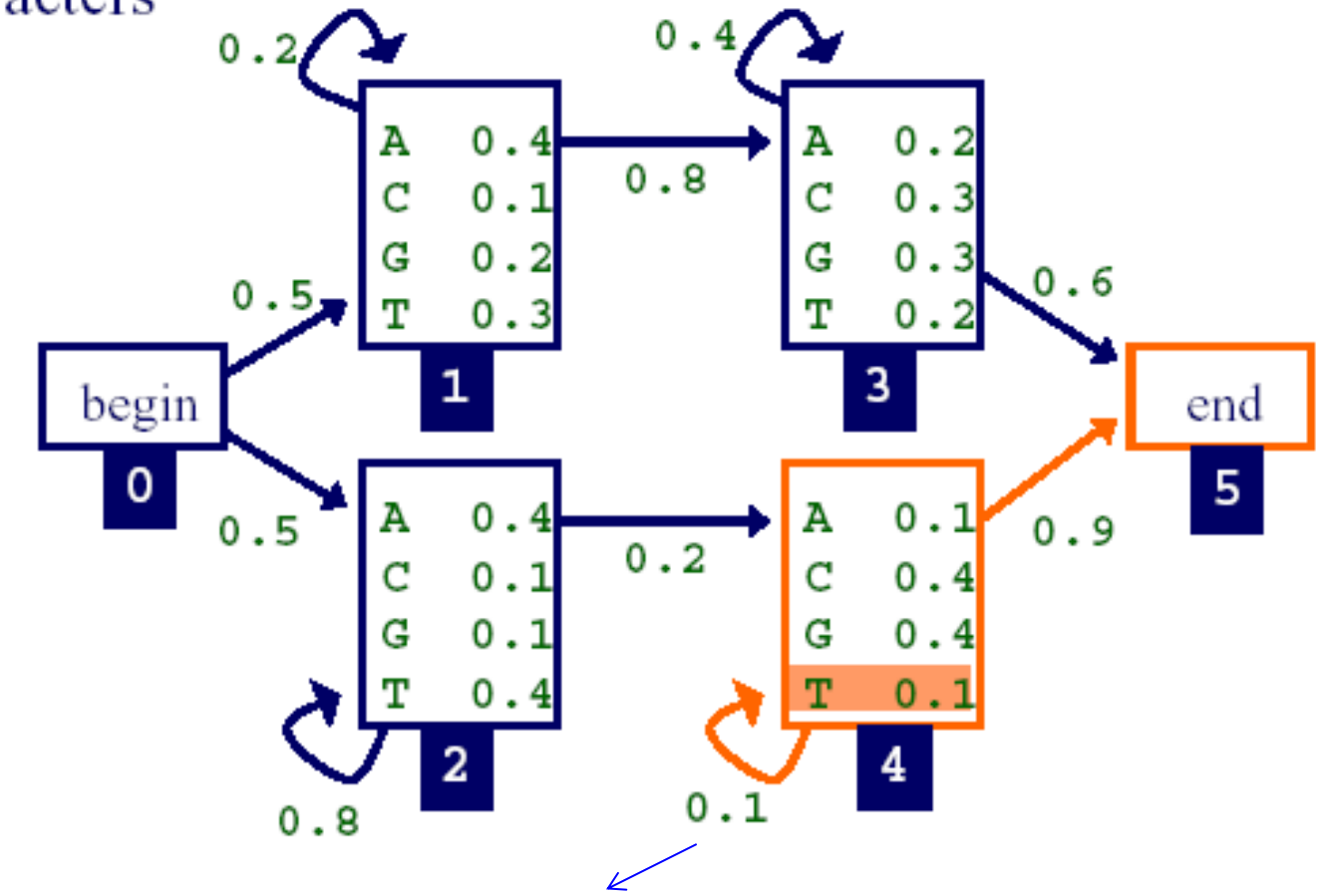


We may use path $\pi_0\ \pi_2\ \pi_2\ \pi_4$ so its probability should be included

C A G T

The Expectation step

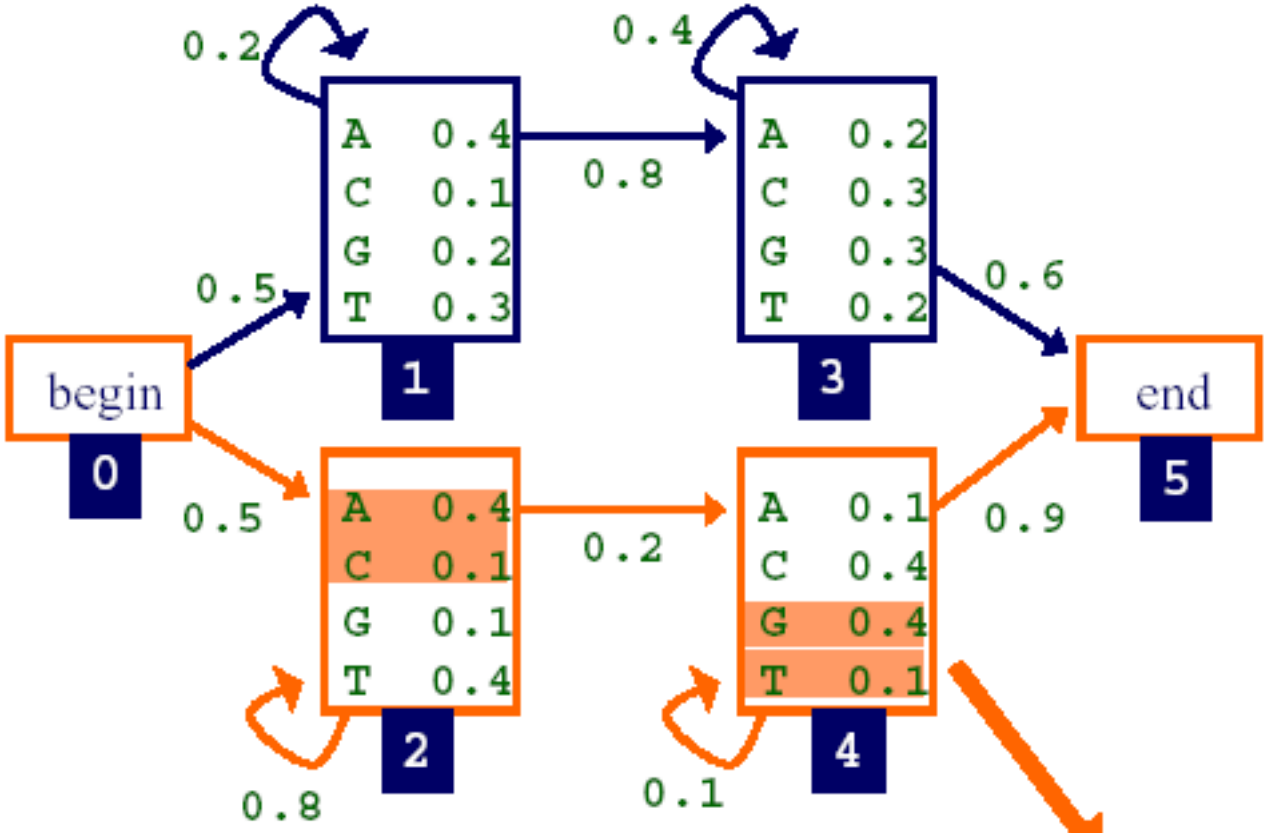
- the *backward algorithm* gives us $b_k(i)$, the probability of observing the rest of x, given that we're in state k after i characters



We have to emit the last symbol T by reentering π_4 C A G T

The Expectation step

- putting forward and backward together, we can compute the probability of producing sequence x with the i th symbol being produced by state q



We use path $\pi_0\ \pi_2\ \pi_2\ \pi_4\ \pi_4\ \pi_5$

The Backward algorithm

just like the Forward algorithm but in reverse..

- Initialisation ($i = L$):

$$b_k(L) = a_{k0} \text{ for all } k$$

set transition probabilities for all states transiting to the end state

- Recursion ($i = L\text{-}1,\ldots,1$):

Durbin et al. skips silent states

- recursion ($i = L \ldots 1$):

$$\begin{cases} a_{kl} b_l(i) & \text{if } l \text{ is silent state} \end{cases}$$

Termination:

$$P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$$

probability that we are in the begin state and have observed the entire sequence

The termination step is rarely needed since the total probability of a sequence is normally calculated using the Forward algorithm, but can be a nice test to check your algorithms

Note on Forward and Backward algorithm

- The forward algorithm score $f_k(i)$ includes the emission probability $e_k(x_i)$
- The backward algorithm score $b_k(i)$ does **not** include emission probability $e_k(x_i)$
  - The algorithm's phasing is 'one behind' since it includes $e_l(x_{i+1})$
  - This is also reflected in the termination steps of the forward and backward algorithms (where the backward algorithm has $e_l(x_1)$ as a factor)
- This slight asymmetry is introduced to make combining the forward and backward algorithms for learning the *E* + *A* parameters more elegant (next slide)

The Expectation step - emissions

- The probability of generating the entire sequence *x* with the *i*th symbol being produced by state *k:*

$$P(x, \pi_i = k) = f_k(i) b_k(i)$$

- Now we can calculate the probability of the *i*th symbol being produced by state *k*, given sequence *x:*

$$P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{P(x)}$$

What 'fraction' of total paths of sequence *x* through HMM goes through state *k* with the *i*th symbol emitted

P(*x*) can be calculated using the Forward (or Backward) algorithm

The Expectation step - emissions

- Now we can calculate the expected number of times symbol *b* is emitted by state *k* over all training sequences:

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i \mid x_i^j = b\}} f_k^j(i) b_k^j(i)$$

Sum over all training sequences j

Sum only over sequence positions i (of sequence j) for which state k emits symbol b

The Expectation step - transitions

- The probability that transition a_{kl} at position i in sequence x is used:

$$P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{f_k(i) a_{kl} e_l(x_{i+1}) b_l(i+1)}{P(x)}.$$

What 'fraction' of total paths of sequence x through HMM goes through transition a_{kl}

θ = A + E (i.e. all current transition and emission parameters in the model)

$P(x)$ can be calculated using the Forward (or Backward) algorithm

The Expectation step - transitions

- Now we can calculate the expected number of times that transition a_{kl} is used by all training sequences:

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1)$$

Sum over all training sequences j

Forward variable for symbol i of sequence j emitted by state k

Backward variable for symbol i+1 of sequence j emitted by state l

The maximization step

- Having calculated the expected numbers for $E_k(b)$ and A_{kl}, we can convert these (as before) to probabilities by maximum likelihood estimation:

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')} \qquad a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}}$$

.. convert the expected number of emissions of symbol *b* in state *k* into a probability by normalising using the total expected number of emissions of state *k*

..convert the expected number of transitions from state *k* to *l* into a probability through dividing by the total number of expected transitions out of state *k*

..as before, one might think about adding pseudocounts to $E_k(b)$ and A_{kl} to avoid overfitting (see earlier slide)

The Baum-Welch Algorithm

Iterative training using the forward-backward algorithm

- Initialize parameters of HMM (for example using priors) and iterate (expectation and maximization - EM) until convergence (using the forward-backward algorithm)
- Upon each next iteration, the *posterior* probabilities learned are used as *priors*
- Convergence is reached usually in a fairly small number of iterations
- The BW algorithm will likely converge to a ***local* maximum**. To avoid this problem and increase the chance of finding the true maximum, train multiple times with different initial parameter settings (priors)

Characteristics of EM

- Converges to a local minimum of the likelihood
- Sensitive to the initial starting point
- Usually converges after few iterations

… from what hilltop can you reach the deepest valley?
… optimisation here is looking for a minimum instead of a maximum
… so a valley rather than a hilltop (more insightful image for ‘blindness' of the method)
… finding the global minimum is depending on where you start in the landscape (i.e., do test different priors)

The Baum-Welch Algorithm

How to know how well the model is trained?
How to know when to stop training?

At each BW iteration:

- Calculate the score of each sequence in the training set
  - Use the Forward (or Backward) algorithm for this
- The sum of these probabilities tells you how well the model is 'dealing' with the sequences it is trained on
  - This cumulative probability will converge
- In practice: **log likelihood** score = $\Sigma_j \log(P_j(x))$ is used to assess the model
- What sequence set should we use to assess the optimality of the model (training, validation or test set)? How to check for overtraining?

The Baum-Welch Algorithm

What is the underlying philosophy?

- Getting an idea of a ‘count’ for each emission probability (a given letter in a given state) is done by summing all probabilities of such letters (in the given state) at any position in any training sequence (using the forward-backward algorithm). If many pathways would run through this letter in the given state, its ‘count’ becomes high.

- The same happens for each transition probability: If many of all of the above pathways go through the transition from a given state *k* to a given state *l*, its probability will become high.

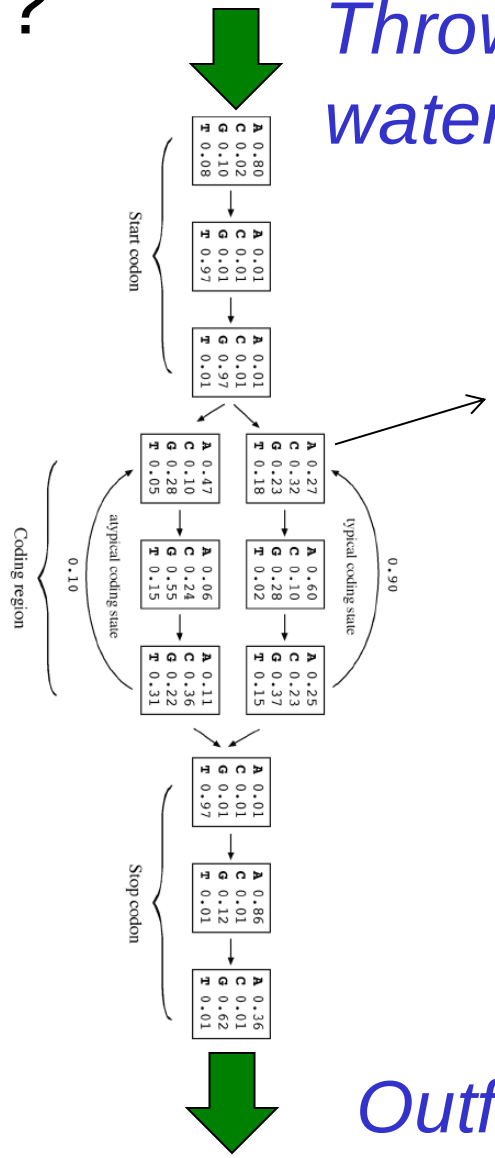
- An emission or transition probability that is ‘walked’ many times in this sense will accumulate a high *posterior* probability (starting with prior probabilities that are ‘guessed’ –at the beginning- or are the result of an earlier BW round)

The Baum-Welch Algorithm

A way to visualise?

The water model

Large A or E values mean large valves, so a lot of water gets through..



Throw in water

If a lot of water gets through A or E values, make these bigger in next iteration

Outflowing water

Viterbi training

- Training the HMM parameters (θ = A + E) can also be done iteratively using the Viterbi algorithm
- The most probable path for each of the sequences in the training set is now used to (re-)estimate the HMM parameters using iteration as before
  - We just know the states of the most probably path
- The procedure will converge precisely, because state paths are discrete
  - the algorithm can stop when the paths will not change anymore
  - the parameters will now be completely fixed

Viterbi training (2)

- Unlike Baum-Welch, Viterbi training does not maximize the true likelihood, but maximizes the contribution to the likelihood using the optimal paths for each of the sequences
- It generally trains less well than Baum-Welch
- However, if the use of the HMM is mainly to produce decodings (using the Viterbi algorithm), then Viterbi training is appropriate

Alternative training methods

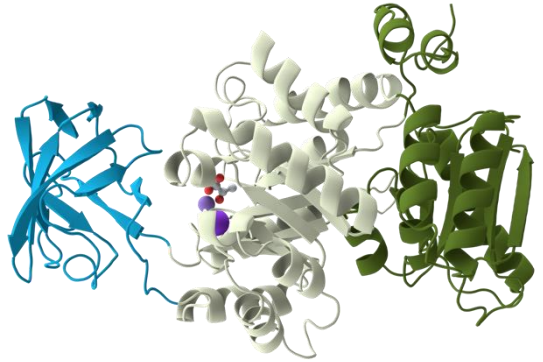
- Training HMMs using the Genetic Algorithm (GA)
  - A GA is an optimization algorithm that mimics Darwinian evolution. The algorithm was developed (in fact popularized) by Holland in the early 1970s.

- Training by Particle Swarm Optimization (PSO) technique
  - PSO is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by social behaviour of bird flocking or fish schooling.

PSO shares many similarities with evolutionary computation techniques such as the Genetic Algorithm (GA). These algorithms are less prone to falling in local maxima than Baum-Welch training.

Practical HMM

- You will construct your own HMM to predict domain/linker structures in protein sequences



- You will execute the Forward, Viterbi and Baum-Welch algorithms in a special practical
  - First on paper but then really making algorithms
  - Templates are available to get you going

- The best way to learn about an algorithm is by doing...

References

- Seminal review by Lawrence R. Rabiner (1989)
- bioalgorithms.info
- Wikipedia
- Background book *Understanding Bioinformatics*
- Course book *Biological sequence Analysis*

Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids by Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison, Cambridge University Press, ISBN 0521 62971 3 Pbk

For reference

Appendix

A few slides on pair-HMMs and profile-HMMs

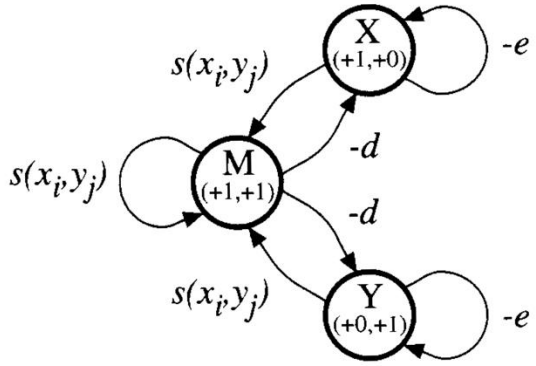
for pairwise and multiple alignments respectively

(See Durbin et al. Chapter 4)

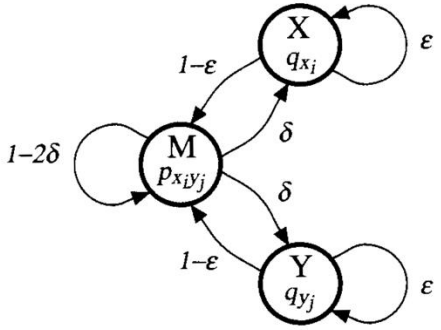
Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids by Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison, Cambridge University Press, ISBN 0521 62971 3 Pbk

Pair HMM for pairwise alignment

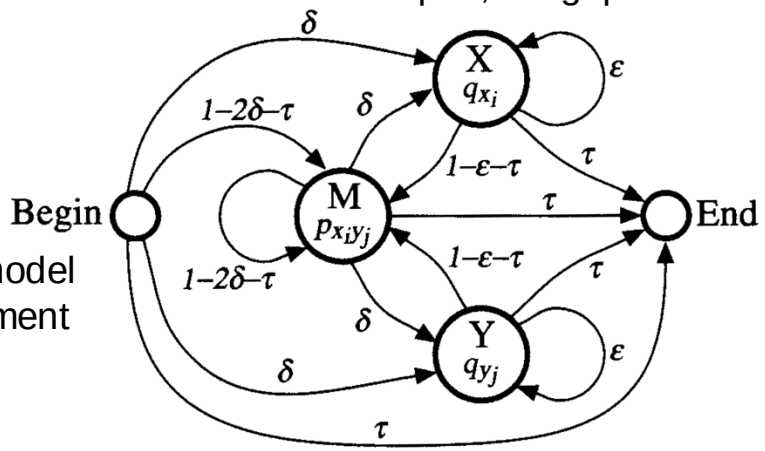
For reference



Finite state machine for pairwise alignment with affine gap penalties: M, match state; X and Y, insert states



Probabilistic model (pair HMM) for alignment with affine gap penalties (δ = gap open, ε = gap extension penalty)



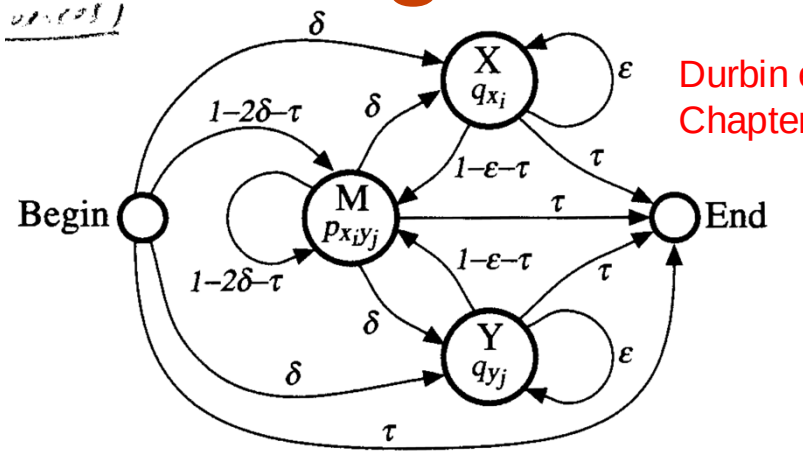
Complete probabilistic model for global pairwise alignment including Begin and End states

Durbin et al., Chapter 4

These models are called pair HMM because they emit pairs (x_i, y_j), (x_i, -) or (-, y_j) when given sequence x_1, x_2,.., x_N and sequence y_1, y_2, .., y_M to align (Durbin et al., Chapter 4)

Pair HMM for pairwise alignment

For reference



Durbin et al., Chapter 4

Complete probabilistic model for global pairwise alignment including Begin and End states

Viterbi algorithm

(Global alignment):

Initialisation:

$v^{\mathrm{M}}(0,0) = 1$. All other $v^{\bullet}(i,0), v^{\bullet}(0,j)$ are set to 0.

Recurrence: $i = 1,\ldots,n, j = 1,\ldots,m$;

$$v^{\mathrm{M}}(i,j) = p_{x_i y_j} \max \begin{cases} (1-2\delta-\tau)v^{\mathrm{M}}(i-1,j-1), \\ (1-\varepsilon-\tau)v^{\mathrm{X}}(i-1,j-1), \\ (1-\varepsilon-\tau)v^{\mathrm{Y}}(i-1,j-1); \end{cases}$$

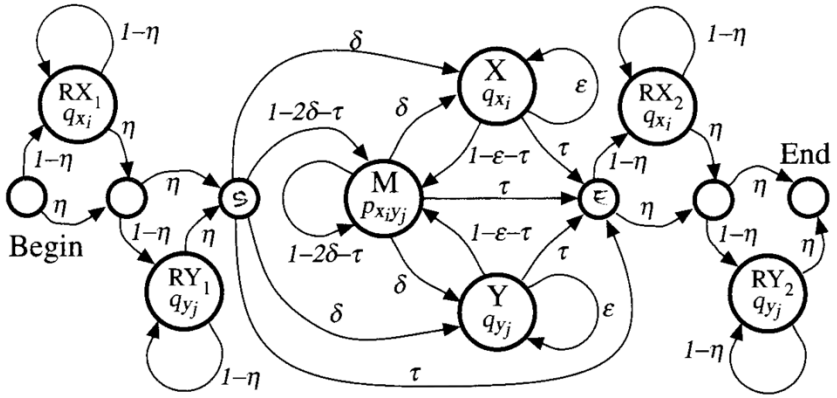
$$v^{\mathrm{X}}(i,j) = q_{x_i} \max \begin{cases} \delta v^{\mathrm{M}}(i-1,j), \\ \varepsilon v^{\mathrm{X}}(i-1,j); \end{cases}$$

$$v^{\mathrm{Y}}(i,j) = q_{y_j}, \max \begin{cases} \delta v^{\mathrm{M}}(i,j-1) \\ \varepsilon v^{\mathrm{Y}}(i,j-1). \end{cases}$$

Termination:

$$v^{\mathrm{E}} = \tau \max(v^{\mathrm{M}}(n,m), v^{\mathrm{X}}(n,m), v^{\mathrm{Y}}(n,m)).$$

Complete probabilistic model for local pairwise alignment including Begin and End states



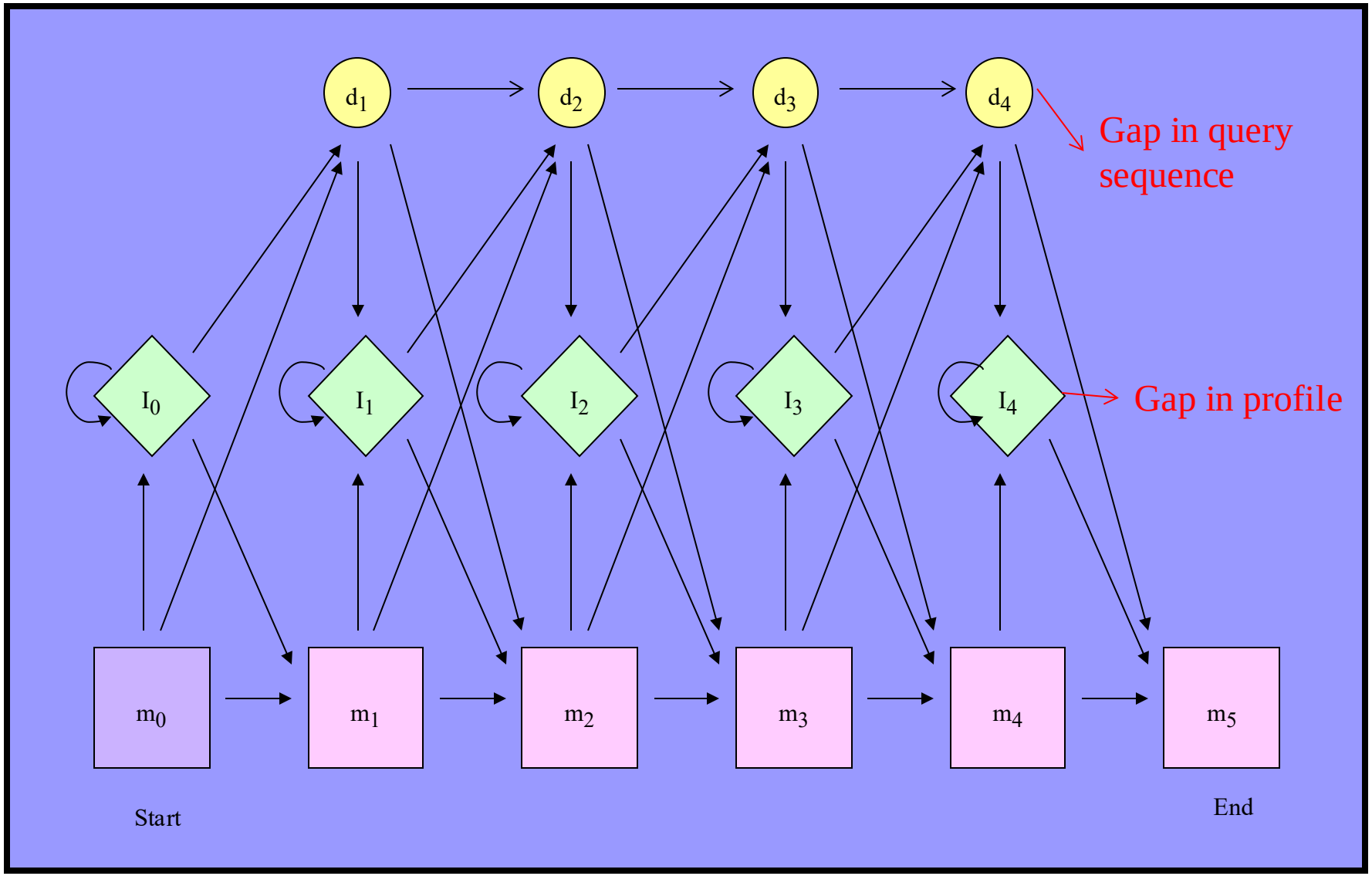
For reference

HMM-based homology searching

Gapped HMM profiles (next slide) also have insertion and deletion states to accommodate aligning a single query sequence against the HMM profile, where the alignment may show a gap in the query sequence (delete state) or a gap in the HMM profile (insert state).

Profile HMM: m=match state, I=insert state, d=delete state; go from left to right. I and m states output amino acids; d states are 'silent'.

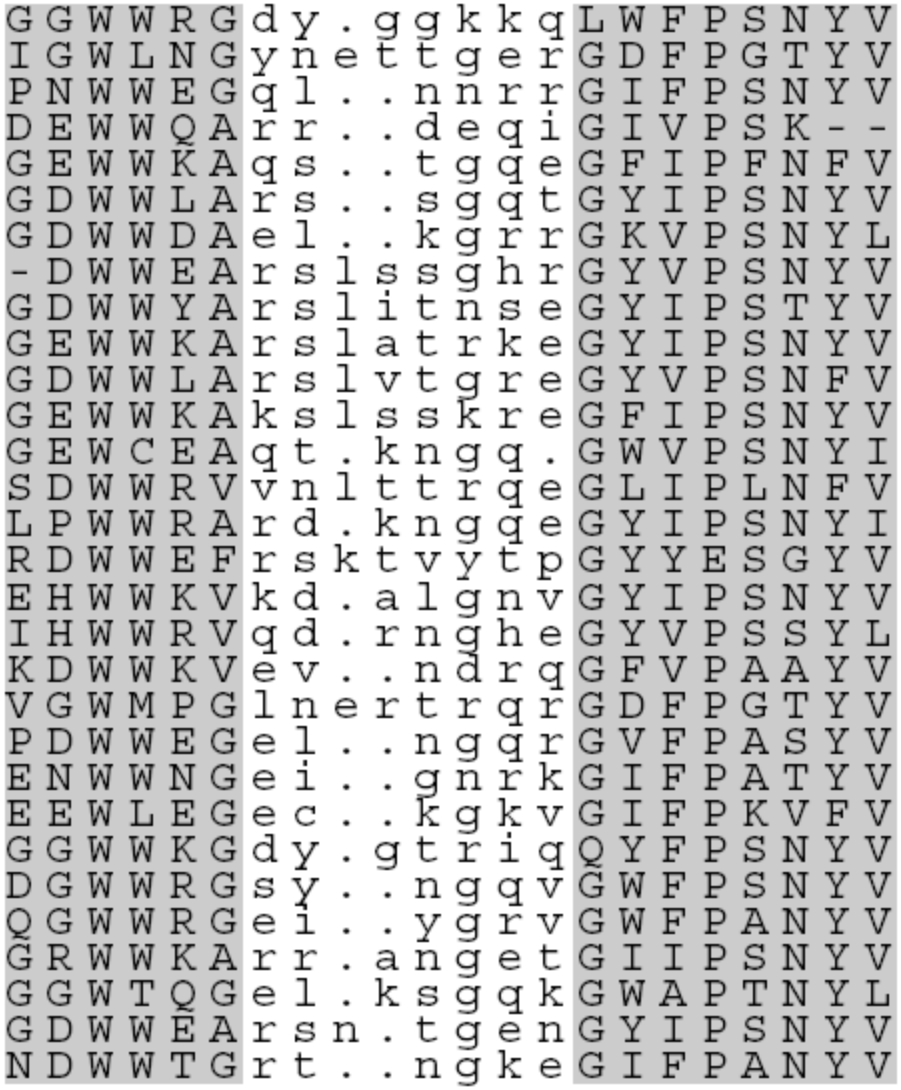
For reference



Model for alignment of a query sequence against a profile with insertions and deletions

HMM-based profile

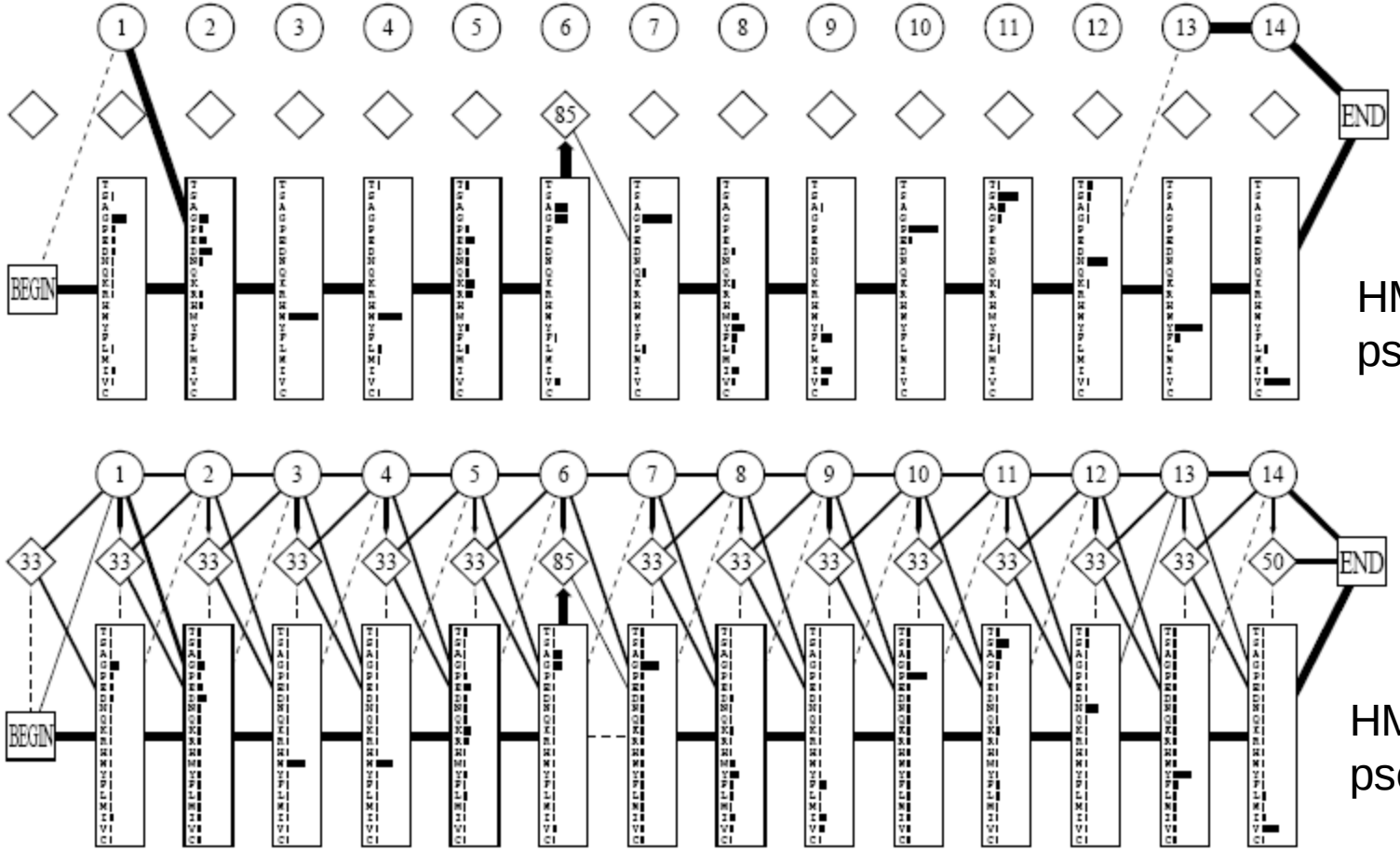
For reference



An alignment of 30 short amino acid sequences chopped out of a alignment of the SH3 domain. The shaded areas are the most conserved and were chosen to be represented by the main (match) states in the HMM. The unshaded area with lower-case letters was chosen to be represented by an insert state (Krogh, Chapter 4)

HMM-based profile

For reference



HMM without pseudocounts

HMM with pseudocounts

A profile HMM made from the preceding alignment. Transition lines with no arrow head are transitions from left to right. Transitions with probability zero are not shown, and those with very small probability are shown as dashed lines. Transitions from an insert state to itself is not shown; instead the probability times 100 is shown in the diamond. The numbers in the circular delete states are just position numbers (Krogh, Chapter 4)

HMM-based homology searching and alignment

For reference

- Most widely used HMM-based profile searching tools are SAM-T2K (Karplus *et al.*, 1998) and HMMER2 (Eddy, 1998)
- formal probabilistic basis and consistent theory behind gap and insertion scores
- HMMs are good for profile searches, but not too good for progressive alignment (due to parametrisation of the models)
- However, HMM profile-profile alignments are now state of the art (e.g. PFAM, HHalign)
- HMM training is slow