Markov Chains
and
Hidden Markov Models

= stochastic, generative models

(Drawing heavily from Durbin *et al.*, *Biological Sequence Analysis*)

Systems Biology / Bioinformatics
Edward Marcotte, Univ of Texas at Austin

1

Markov Chains and Hidden Markov Models are important probabilistic models in computational biology

Some of their applications include:

- Finding genes in genomes
- Mapping introns, exons, and splice sites
- Identifying protein domains & families
- Detecting distant sequence homology
- Identifying secondary structures in proteins
- Identifying transmembrane segments in proteins
- Aligning sequences

& outside biology, they have many uses, including:

- Speech, handwriting, and gesture recognition
- Tagging parts-of-speech
- Language translation
- Cryptanalysis

and so on….

2

The <u>key idea</u> of both of these types of models is that:

***Biological sequences can be modeled as series of stochastic (i.e., random) events*.**

It's easy to see how a random process might model stretches of DNA between genes and other important regions.

BUT, the idea of modeling something as structured and meaningful as a gene or protein sequence by a similar process might seem odd.

3

It's important to realize exactly what we're modeling.

The idea behind hidden Markov models is <u>not</u> that the sequence is random, but that the sequence we observe is <u>one of many possible instances</u> of some underlying process or object.

E.g., actin differs slightly from organism to organism.

Imagine an "ideal", but unobservable, actin, defined by specific underlying physico-chemical properties important for its function. What we see in nature is not this ideal gene, but many variants, all just a bit different.

In the hidden Markov model, the underlying process or structure is represented as hidden, unobservable ***states*** and the observed sequences represent <u>possible</u> sequences compatible with these states.

We say that the observed sequence is ***emitted*** from the hidden states.

4

Let’s start with a easier case: **Markov chains**

We’ll explore a simple non-biological model: a coin-toss

Flip a coin a bunch of times and observe the results, e.g.

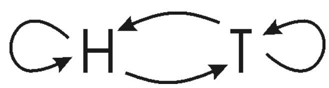
TTTHTTTHHTTHTTHTTHHTHTHHHHTTTTTTTTHTTHHTTTHTHHHTHH

We could model this process as two states:

H for heads,

T for tails,

and the probability of switching between them:



5

Let’s start with a easier case: **Markov chains**

We’ll explore a simple non-biological model: a coin-toss

Flip a coin a bunch of times and observe the results, e.g.

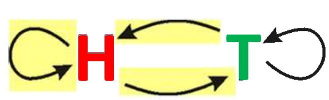
TTTHTTTHHTTHTTHTTHHTHTHHHHTTTTTTTTHTTHHTTTHTHHHTHH

We could model this process as two states:

H for heads,

T for tails,

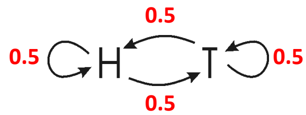
and the probability of switching between them:



A sequence is a walk along this graph:

H H T H …

6



Important: All probabilities leading out of a state add up to 1!

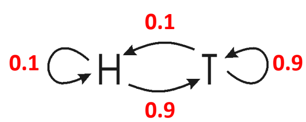
With a **fair coin:**

The chance of seeing heads or tails is equal, and
the chance of seeing heads following tails and vice versa is equal.

Therefore, the ***transition probabilities***
(corresponding to the arrows above) are:

	Position i+1:	
Position i:	**Head**	**Tail**
Head	0.5	0.5
Tail	0.5	0.5

7



Important: All probabilities leading out of a state add up to 1!

With a **biased coin** (e.g. tails comes up 90% of the time):

The chance of seeing heads or tails is not equal, nor is
the chance of seeing heads following tails and vice versa.

We might have the same model,
but with skewed ***transition probabilities*** :

	Position i+1:	
Position i:	**Head**	**Tail**
Head	0.1	0.9
Tail	0.1	0.9

8

Now, imagine a sequence of coin flips generated by these 2 coins, one fair and one biased.

To decide if a sequence of coin flips comes from the biased or fair coin, we could evaluate the **ratio of the probabilities** of observing the sequence by each model:

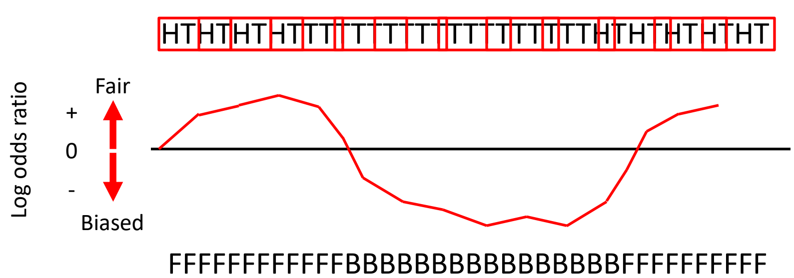
$$\frac{P(\,X \mid \text{fair coin}\,)}{P(\,X \mid \text{biased coin}\,)}$$

Does this remind you of something we've seen before?
How might we test where the fair & biased coins were swapped along a long stretch of coin flips?

9

How might we test where the fair & biased coins were swapped along a long stretch of coin flips?

One way using our current Markov chain model is to calculate the ratio of probabilities (e.g. log odds ratio) in a ***sliding window*** along the sequence:



10

How about a biological application? A classic example is ***CpG islands***

In animal genomes, the dinucleotide CG is strongly underrepresented
(note: NOT the base pair C:G, but rather 5'-CG-3')

Why? C's are often methylated, and methylated C's mutate at higher rates into T's. So, over time, CG's convert to TG's
EXCEPT around promoters, which tend not to be methylated
(DNA methylation epigenetically represses transcription*)

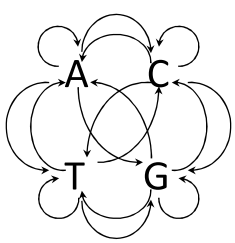
Thus, **CpG 'islands' often indicate nearby genes**. Finding them was a classic method for annotating genes.

How could we make a CpG island finding model analogous to the fair/biased coin model?

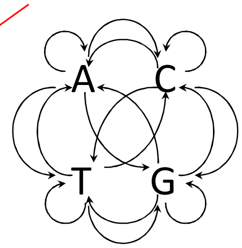
*e.g. see *J. Cell. Physiol.* 213: 384–390 (2007)

A CpG island model might look like:

(of course, need the parameters, but maybe these are the most important....)



p(C→G) is higher



p(C→G) is lower

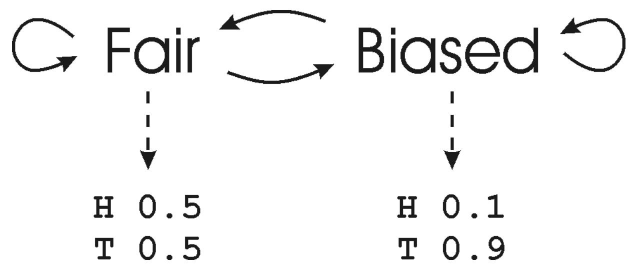
CpG island model

Not CpG island model

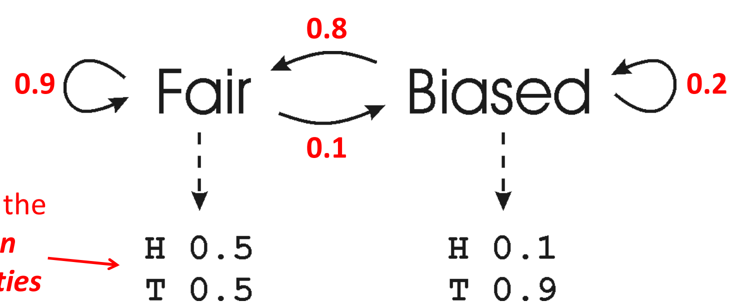
Could calculate $\dfrac{P(\,X \mid \text{CpG island})}{P(\,X \mid \text{not CpG island})}$ (or log ratio) along a sliding window, just like the fair/biased coin test

In these simple models, called ***Markov chains***,
we don't have hidden states.

BUT, we could have used a ***hidden Markov model***:



Now, the underlying *state* (the choice of coin) is hidden.
Each state *emits* H or T with different probabilities.



These are the ***emission probabilities*** for each state

The ***transition probabilities*** might be something like:

Position i:	Position i+1: Fair	Biased
Fair	0.9	0.1
Biased	0.8	0.2

Important questions we might like to ask:

1. **Given an observed sequence and a model, what is the most likely sequence of hidden states?**
   i.e., what is the path through the HMM that maximizes P(p,X|l), where p is the sequence of states)?

In our coin example, we might be given an observed sequence:

HTHTHTHTTTTTTTTTTTTTTTTTTTHTHTHTHTHT

and want to identify when the biased coin was used:

FFFFFFFFFFFFBBBBBBBBBBBBBBBBFFFFFFFFFF

Answer: Use the *Viterbi algorithm*.
We'll see this shortly.

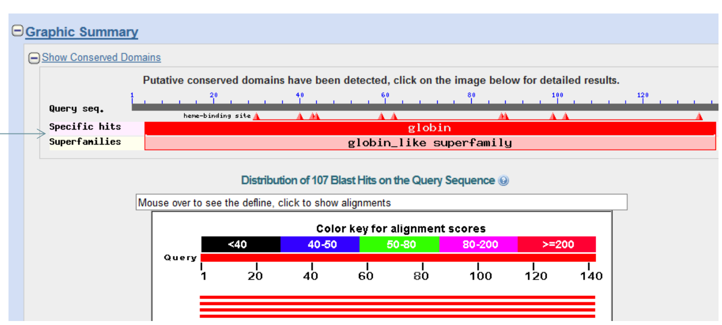
15

Important questions we might like to ask:

2. **Given a *sequence of observations*, can we calculate the <u>probability</u> that the sequence was derived from our model ?**
   i.e., can we calculate P(X|l),
   where X is our observed sequence, and l represents our HMM ?

For example, we might want to know if a given protein sequence is a member of a certain protein family.

e.g. as we saw before (although calculated a bit differently)



16

Important questions we might like to ask:

2. **Given a *sequence of observations*, can we calculate the <u>probability</u> that the sequence was derived from our model ?**
   i.e., can we calculate P(X|l),
   where X is our observed sequence, and l represents our HMM ?

For example, we might want to know if a given protein sequence is a member of a certain protein family.

Answer: Yes. Use the *forward algorithm*.
We'll see this shortly.

17

Important questions we might like to ask:

3. **Given a model, what is the <u>most likely sequence</u> of observations?**

For example, after having trained an HMM to recognize a type of protein domain, what amino acid sequence best embodies that domain?

Answer: Follow the maximum transition and emission probability at each state in the model. This will give the most likely state sequence and observed sequence.

18

Important questions we might like to ask:

4. **How do we train our HMM?**

   i.e., given some training observations, how do we set the emission and transition probabilities to maximize P(X|l)?

Answer: If the state sequence is known for your training set, just directly calculate the transition and emission frequencies. With sufficient data, these can be used as the probabilities.

This is what you will do in Problem Set #2.

With insufficient data, probabilities can be estimated from these (e.g., by adding pseudo-counts).

If the state path is unknown, use the *forward-backward algorithm* (also known as the *Baum-Welch algorithm*).

19

Important questions we might like to ask:

5. **How do we choose the best HMM topology from the many possible choices?**

Answer: Good question. No great answer.

Often trial-and-error, and understanding the essential features of the system that you are modeling.

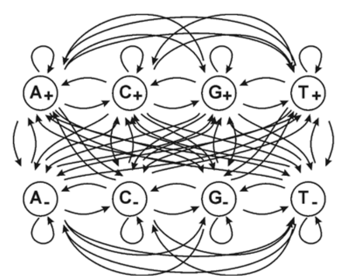
20

Each of these algorithms (the Viterbi, forward, and forward-backward) uses dynamic programming to find an optimal solution.

(just like aligning sequences)

21

Let's revisit the CpG islands using an HMM:



- 8 states: one per nucleotide inside CpG islands (+) and one per nucleotide outside CpG islands (-)
- All possible transition probabilities are represented as arrows
- This is a particularly simple model: each state emits the nucleotide indicated with probability of 1 and has zero probability of emitting a different nucleotide.

22

Just to be totally clear:

In this (greatly simplified) model, here is our table of emission probabilities:

+ means a base inside a CpG island

- means a base outside a CpG island

Emission probabilities

States	A	C	T	G
A+	1	0	0	0
A-	1	0	0	0
C+	0	1	0	0
C-	0	1	0	0
T+	0	0	1	0
T-	0	0	1	0
G+	0	0	0	1
G-	0	0	0	1

i.e. an "A" can only be emitted from an A+ or A- state, not any other state

Emission probabilities from the same state must add up to 1

This is a particularly simple model: each state emits the nucleotide indicated with probability of 1 and has zero probability of emitting a different nucleotide.

23

Given a DNA sequence *X* (e.g., CGATCGCG), how do we find the most probable sequence of states (e.g., ----++++)?

→ ***The Viterbi algorithm***

We want to find the state path that maximizes the probability of observing that sequence from that HMM model.

Viterbi does this recursively using dynamic programming.

As with sequence alignment, we'll construct a path matrix that captures the best score (*i.e.*, highest probability) along a single path through the HMM up to each position. We'll "grow" this matrix using a few simple recursion rule.

24

The rules (stated formally):

v is an entry in the Viterbi path matrix

Initialization (i=0): $v_0(0) = 1,\ v_k(0)=0$ for $k>0$

Find the best score among the alternatives at this position

x indicates an observed character

e indicates an emission probability

a gives the transition probability between previous state *k* and current state *l*

Recursion (i=1 to L):

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

$\text{pointer}_i(l) = \text{argmax}_k\ (v_k(i\text{-}1)a_{kl})$

i indicates our position in the sequence

i.e., draw the pointer back to the entry that gave rise to the current best score

Termination:

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

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

For each Viterbi matrix entry:
We try to maximize the product of prior score and transition from that state to this one.
We then multiply that score times the emission probability for the current character.

25

Step 1: Initialize the path matrix.

Observed DNA sequence

Possible states

		C	G	C	G
$\mathscr{B}$	1	0	0	0	0
A+	0				
C+	0				
G+	0				
T+	0				
A−	0				
C−	0				
G−	0				
T−	0				

For simplicity, let's assume the transition probability from $\mathscr{B}$ to each nucleotide is 1/8. We'll also ignore all transition probabilities except these for now:

	Position i+1:			
Position i:	**C+**	**G+**	**C-**	**G-**
C+	0.37	0.27	small	small
G+	0.34	0.38	small	small
C-	small	small	0.3	0.08
G-	small	small	0.25	0.3

26

Step 2: Calculate the elements of the v_k matrix for i = 1.
Then keep going for i = 2, etc..

		C	G	C	G
$\mathscr{B}$	1	0	0	0	0
A+	0	0			
C+	0	0.13			
G+	0	0			
T+	0	0			
A−	0	0			
C−	0	0.13			
G−	0	0			
T−	0	0			

For simplicity, let's assume the transition probability from $\mathscr{B}$ to each nucleotide is 1/8. We'll also ignore all transition probabilities except these for now:

Position i:	Position i+1: C+	G+	C-	G-
C+	0.37	0.27	small	small
G+	0.34	0.38	small	small
C-	small	small	0.3	0.08
G-	small	small	0.25	0.3

For example, the score $v_{C+}(i\text{=}1)$ = $1 * \max_k\{1*1/8, 0*a_{A+,C+}, 0*a_{C+,C+}, ..., 0*a_{T-,C+}\} = 1/8$

27

Let's blow up just one square of the matrix to understand a bit better:

		To fill in this entry:
$\mathscr{B}$	1	
A+	0	
C+	0	0.13
G+	0	
T+	0	
A−	0	
C−	0	
G−	0	
T−	0	

We had to evaluate 9 alternatives & their scores:

$\mathscr{B}$	1	$1*t_{\mathscr{B}\rightarrow C+} = 0.13$
A+	0	$0*\ t_{A+\rightarrow C+} = 0$
C+	0	$0*\ t_{C+\rightarrow C+} = 0$
G+	0	$0*\ t_{G+\rightarrow C+} = 0$
T+	0	$0*\ t_{T+\rightarrow C+} = 0$
A−	0	$0*\ t_{A-\rightarrow C+} = 0$
C−	0	$0*\ t_{C-\rightarrow C+} = 0$
G−	0	$0*\ t_{G-\rightarrow C+} = 0$
T−	0	$0*\ t_{T-\rightarrow C+} = 0$

Previous score * Transition probability

Take the *max* (in this case, 0.13) and only then multiply that by the emission probability (here, 1) to give a final score for that cell of 0.13. Draw an arrow back to the cell it came from (here, $\mathscr{B}$)

28

Step 3: Keep going for *i* = 2, etc..

		C	G	C	G
$\mathcal{B}$	1	0	0	0	0
A+	0	0	0	0	0
C+	0	0.13	0	0.012	0
G+	0	0	0.034	0	0.0032
T+	0	0	0	0	0
A−	0	0	0	0	0
C−	0	0.13	0	0.0026	0
G−	0	0	0.01	0	0.00021
T−	0	0	0	0	0

	Position i+1:			
Position i:	**C+**	**G+**	**C-**	**G-**
C+	0.37	0.27	small	small
G+	0.34	0.38	small	small
C-	small	small	0.3	0.08
G-	small	small	0.25	0.3

29

The maximum scoring path scores 0.0032.
The most likely state path is found by traceback from the 0.0032 to give C+G+C+G+.

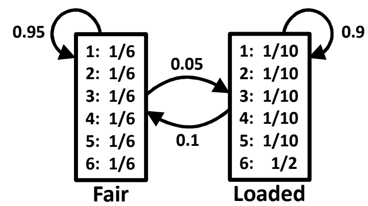
		C	G	C	G
$\mathcal{B}$	1	0	0	0	0
A+	0	0	0	0	0
C+	0	0.13	0	0.012	0
G+	0	0	0.034	0	0.0032
T+	0	0	0	0	0
A−	0	0	0	0	0
C−	0	0.13	0	0.0026	0
G−	0	0	0.01	0	0.00021
T−	0	0	0	0	0

In a longer sequence, the model would switch back & forth between CpG and non-CpG states appropriately.

30

Can this really work? Here's a real example.

An HMM model of fair and loaded dice:



Reconstructing which was used when, using the Viterbi algorithm:

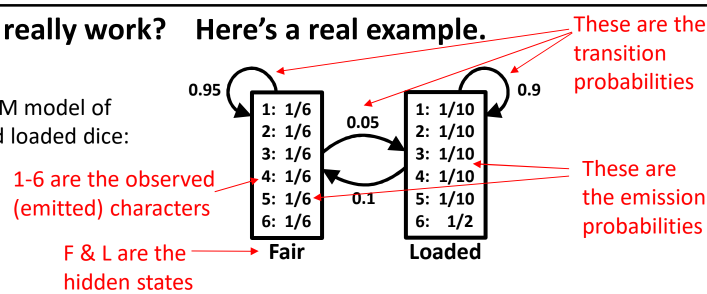
```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLFFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFF
```

from Durbin *et al.*

31

Can this really work? Here's a real example.

An HMM model of fair and loaded dice:



Reconstructing which was used when, using the Viterbi algorithm:

```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLFFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFF
```

from Durbin *et al.*

32

How do we calculate the probability of a sequence given our HMM model?

→ ***The forward algorithm***

Subtle difference from Viterbi:
Viterbi gives the probability of the sequence being derived from the model *given the optimal state path.*

The forward algorithm takes into account all possible state paths.

Again, it does this recursively using dynamic programming.

33

The rules (stated formally):

f is an entry in the forward algorithm path matrix

Initialization (i=0): $f_0(0) = 1,\ f_k(0) = 0$ for k>0

Same idea as Viterbi, but ADD the scores leading to the current position (not MAX)

Recursion (i=1 to L): $$f_l(i) = e_l(x_i)\sum_k f_k(i-1)a_{kl}$$

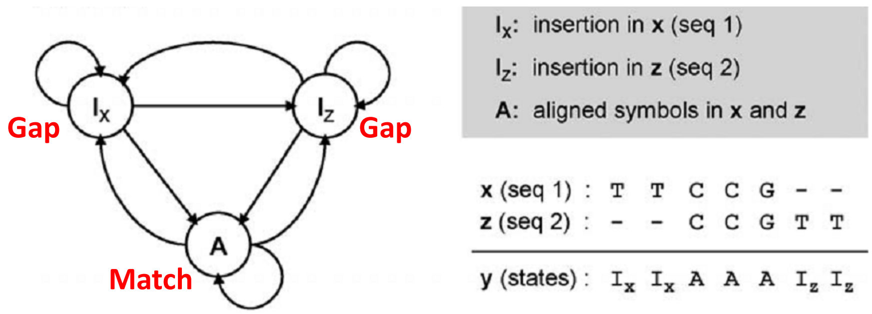
Termination: $$P(x) = \sum_k f_k(L)a_{k0}$$

Note: No pointer! Just to calculate the probability of seeing this sequence from this model.

For each Viterbi matrix entry:
We try to maximize the product of prior score and transition from that state to this one.
We then multiply that score times the emission probability for the current character.

34

A toy HMM for sequence alignment



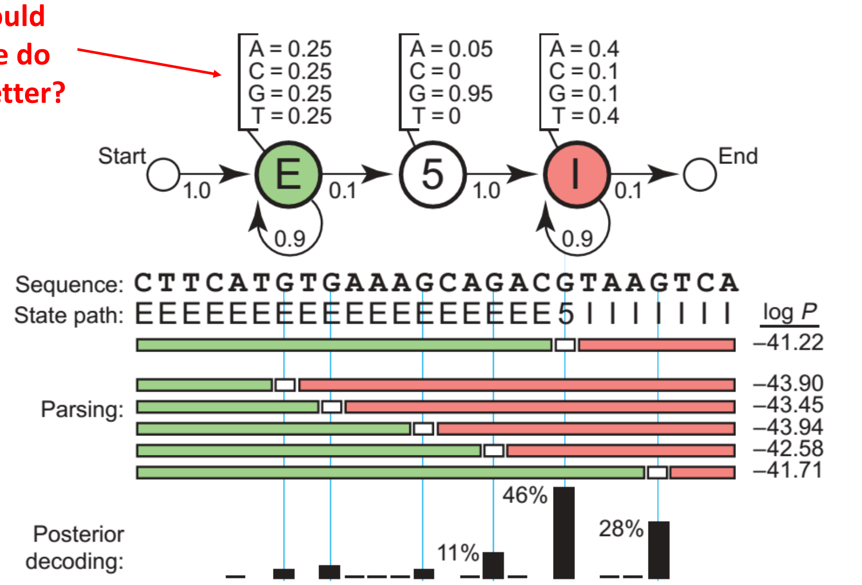
Is this global or local alignment?
How could you change the model to perform the other kind of alignment?

Curr Genomics (2009) 10(6): 402–415

35

A toy HMM for 5′ splice site recognition (from Sean Eddy's NBT primer linked on the course web page)

Could we do better?



36

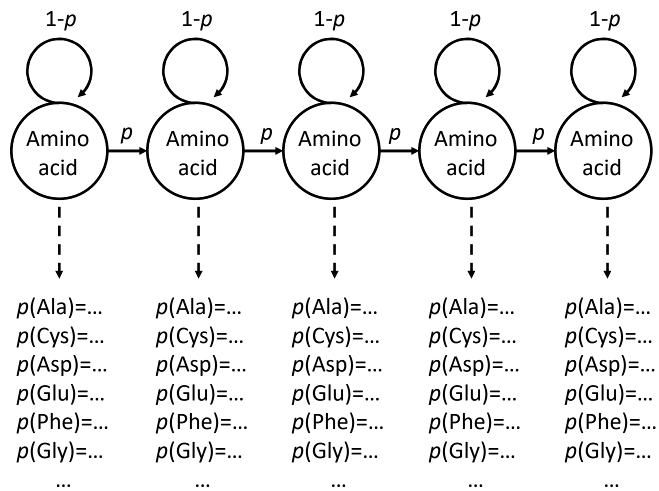
How might you design an HMM to recognize a given type of protein domain?

37

How might we design HMMs to recognize sequences of a given length?

38

What would this HMM produce?



39