Introducción a la Bioinformática:

Hidden Markov Models (HMMs)

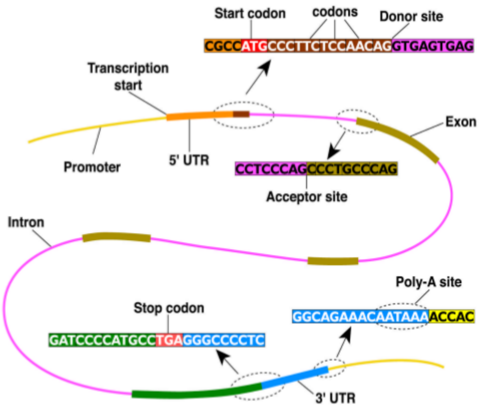
Luis Garreta

Doctorado en Ingeniería
Pontificia Universidad Javeriana – Cali

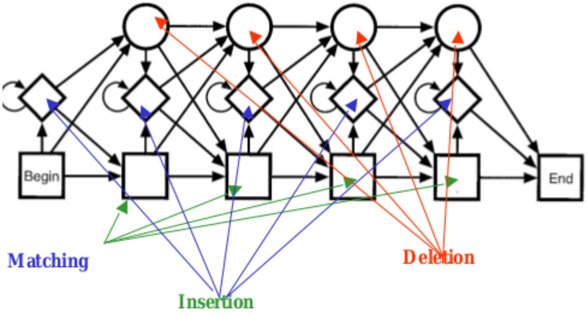
March 24, 2017

Applications of HMMs

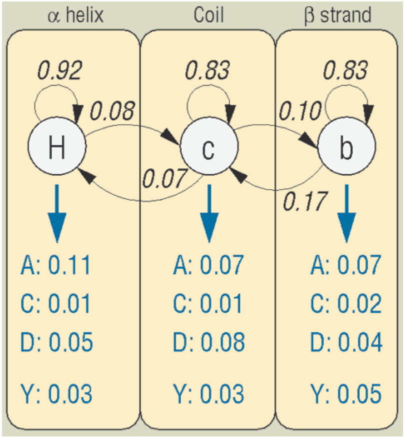
Gene Finding and Prediction



Protein- Profile HMMs



Protein- Profile HMMs



Other Applications of HMMs

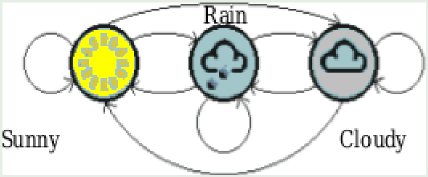
- Speech recognition
- Optical character recognition
- Spell checking

Hidden Markov Model (HMM) Architecture

Markov Chains:

Markov Assumption

Three states of weather



- Three states: Sunny, Cloudy, and Rainy
- Weather pattern instead deterministically

- Markov Assumption: The state of the model depends only upon the previous states of the model
- Order n Model (First Order): The choice of state is made purely on the basis of the previous state

Markov Chains:
State Transition Matrix (A)

$$A = \begin{array}{c|c|ccc} & & & \text{Today} & \\ & & \text{Sun} & \text{Cloud} & \text{Rain} \\ \hline \text{Yesterday} & \text{Sun} & 0.5 & 0.25 & 0.25 \\ & \text{Cloud} & 0.375 & 0.125 & 0.375 \\ & \text{Rain} & 0.125 & 0.625 & 0.375 \end{array}$$

It it was sunny yesterday, there is a probability of 0.5 that it will be sunny today, and 0.25 that it will be cloudy or rainy.

Markov Chains:

Vector of Initial Probabilities (π)

$$\Pi = \left(\begin{array}{ccc} \text{Sun} & \text{Cloud} & \text{Rain} \\ 1.0 & 0.0 & 0.0 \end{array}\right)$$

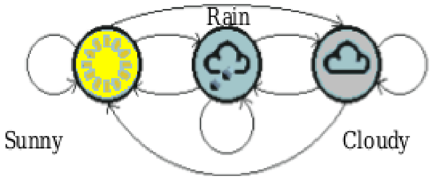
- To initialize such a system, we need to state what the weather was (or probably was) on the day after creation;
- So, we know it was sunny on day 1

Markov Chains:

First Order Markov Process

- States: Three states: sunny, cloudy, rainy
- π vector: Probability of the system in each states at time 0
- State transition Matrix: Probability of the weather given the previous day's weather

Any system that can be described in this manner is a Markov process.



$$A = \begin{array}{c|ccc} & \text{Sun} & \text{Cloud} & \text{Rain} \\ \hline \text{Sun} & 0.5 & 0.25 & 0.25 \\ \text{Cloud} & 0.375 & 0.125 & 0.375 \\ \text{Rain} & 0.125 & 0.625 & 0.375 \end{array}$$

$$\Pi = \begin{array}{ccc} \text{Sun} & \text{Cloud} & \text{Rain} \\ 1.0 & 0.0 & 0.0 \end{array}$$

Hidden Markov Models

- In some cases the patterns that we wish to find are not described sufficiently by a Markov process.
- A hermit for instance may not have access to direct weather observations, but does have a piece of seaweed.

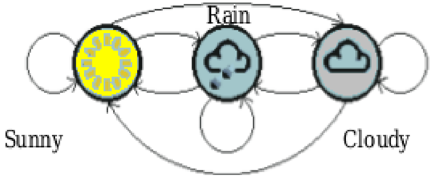
- Sea and weather lore: seaweeds are weather predictors (they absorb atmospheric moisture)
- The seaweed is probabilistically related to the state of the weather:

Hidden Markov Models: Two sets of States

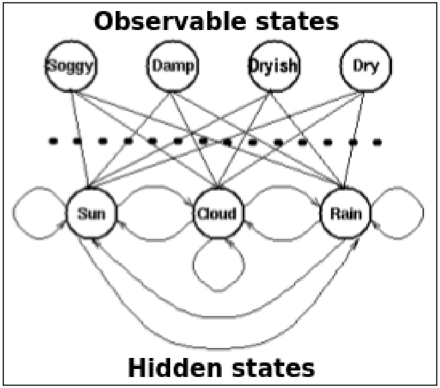
In this case we have two sets of states:

- observable states (the state of the seaweed) and
- hidden states (the state of the weather).

We wish to devise an algorithm for the hermit to forecast weather from the seaweed and the Markov assumption without actually ever seeing the weather.



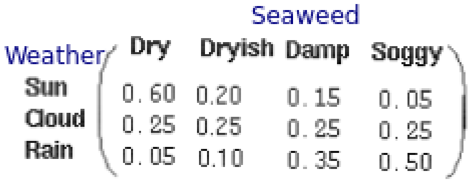
Hidden Markov Models: Hidden and Observable States



- Hidden states (the true weather) are modeled by a simple Markov process.
- So, they are all connected to each other.
- The new connections represent: *the probability of generating a particular observed state given that the Markov process is in a particular hidden state.*

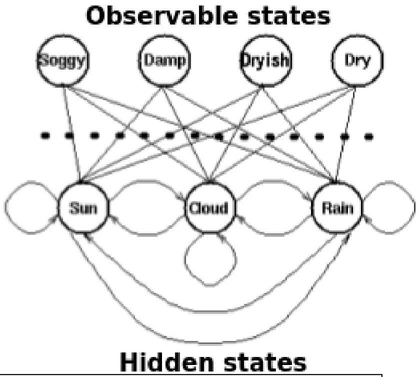
Hidden Markov Models: Emission Matrix

The probabilities of the observable states given a particular hidden state:



Seaweed

Weather	Dry	Dryish	Damp	Soggy
Sun	0.60	0.20	0.15	0.05
Cloud	0.25	0.25	0.25	0.25
Rain	0.05	0.10	0.35	0.50



All probabilities "entering" an observable state will sum to 1 :

$$Pr(Obs|Sun) + Pr(Obs|Cloud) + Pr(Obs|Rain) = 1$$

Example: The Dishonest Casino

Game:

1. You bet $1
2. You roll
3. Casino player rolls
4. Highest number wins $2

The casino has two dice:

Fair die

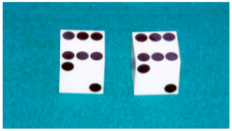
P(1) = P(2) = P(3) = P(5) = P(6) = 1/6

Loaded die

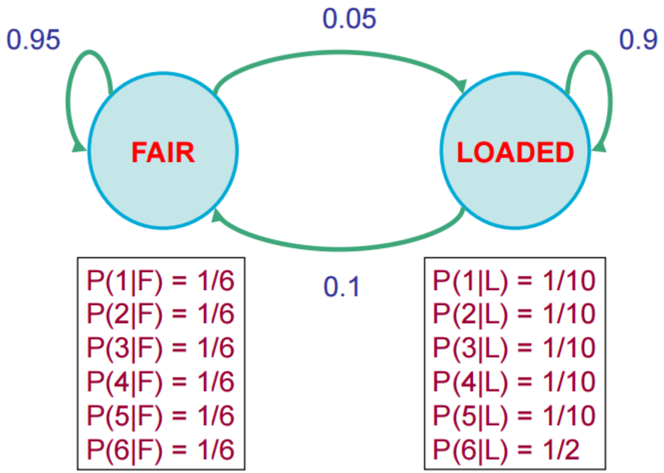
P(1) = P(2) = P(3) = P(5) = 1/10

P(6) = 1/2

Casino player switches between fair and loaded die (not too often, and not for too long)



The dishonest casino model



Question # 1 – Evaluation

GIVEN:

A sequence of rolls by the casino player

$$\underbrace{1245526462146146136136661664661636616366163616515615115146123562344}_{\boxed{\text{Prob} = 1.3 \times 10^{-35}}}$$

QUESTION:

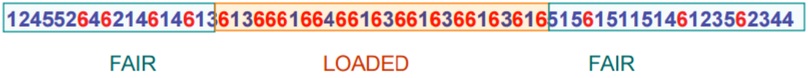
How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs

Question # 2 – Decoding

GIVEN:

A sequence of rolls by the casino player



QUESTION:

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the **DECODING** question in HMMs

Question # 3 – Learning

GIVEN:

A sequence of rolls by the casino player

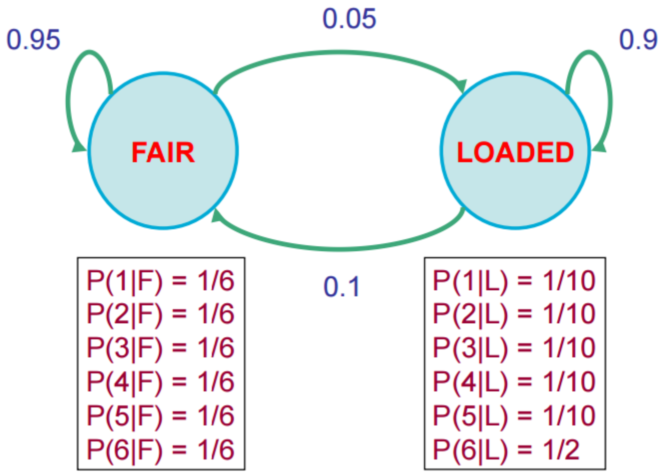
124552646214614613613666166466163661636616361651561511514612356234 4

QUESTION:

How does the casino player work: How "loaded" is the loaded die? How "fair" is the fair die? How often does the casino player change from fair to loaded, and back?

This is the **LEARNING** question in HMMs

The dishonest casino model



P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2

Definition of a hidden Markov model

- Alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of states $Q = \{ 1, \ldots, K \}$ $(K = |Q|)$
- Transition probabilities between any two states

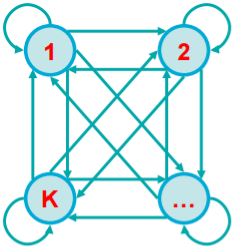
  a_{ij} = transition probability from state i to state j

  $a_{i1} + \ldots + a_{iK} = 1$, for all states i
- Initial probabilities a_{0i}

  $$a_{01} + \ldots + a_{0K} = 1$$
- Emission probabilities within each state

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

  $$e_k(b_1) + \ldots + e_k(b_M) = 1$$



Hidden states and observed sequence

At time step t,

π_t denotes the (hidden) state in the Markov chain

x_t denotes the symbol emitted in state π_t

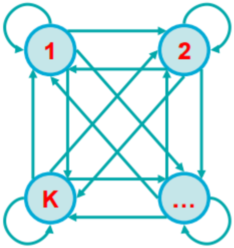
A path of length N is: $\pi_1, \pi_2, \ldots, \pi_N$

An observed sequence of length N is: $x_1, x_2, \ldots, x_N$

An HMM is "memory-less"

At time step t, the only thing that affects the next state is the current State, π_t

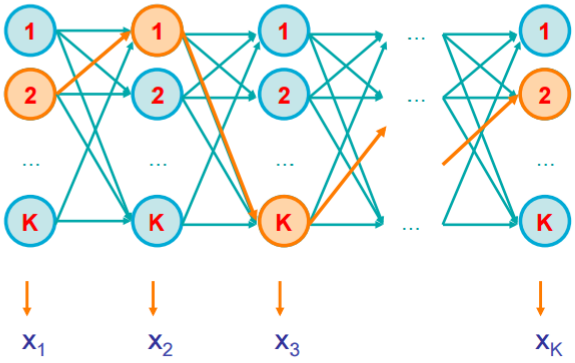
$P(\pi_{t+1} = k \mid$ "whatever happened so far")
$= P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t)$
$= P(\pi_{t+1} = k \mid \pi_t)$



A parse of a sequence

Given a sequence $x = x_1 \ldots\ldots x_N$,

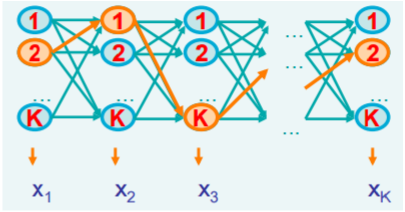
A parse of x is a sequence of states $\pi = \pi_1, \ldots\ldots, \pi_N$



Likelihood of a parse: $P(x, \pi)$

Given a sequence $x = x_1 \ldots\ldots , x_N$
and a parse $\pi = \pi_1, \ldots\ldots, \pi_N$,

How likely is the parse
(given our HMM)?



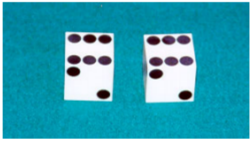
$$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots, \pi_N)$$
$$= P(x_N, \pi_N \mid x_1 \ldots x_{N-1}, \pi_1, \ldots\ldots, \pi_{N-1})\, P(x_1 \ldots x_{N-1}, \pi_1, \ldots\ldots, \pi_{N-1})$$
$$= P(x_N, \pi_N \mid \pi_{N-1})\, P(x_1 \ldots x_{N-1}, \pi_1, \ldots\ldots, \pi_{N-1})$$
$$= \ldots$$
$$= P(x_N, \pi_N \mid \pi_{N-1})\, P(x_{N-1}, \pi_{N-1} \mid \pi_{N-2}) \ldots\ldots P(x_2, \pi_2 \mid \pi_1)\, P(x_1, \pi_1)$$
$$= P(x_N \mid \pi_N)\, P(\pi_N \mid \pi_{N-1}) \ldots\ldots P(x_2 \mid \pi_2)\, P(\pi_2 \mid \pi_1)\, P(x_1 \mid \pi_1)\, P(\pi_1)$$
$$= a_{0\pi 1}\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\, e_{\pi 1}(x_1) \ldots\ldots e_{\pi N}(x_N)$$
$$= \prod_{i=1}^{N} a_{\pi_{i-1}\pi_i} e_{\pi_i}(x_i)$$

Example: the dishonest casino $P(x, \pi)$, $\pi = FFFFFF \ldots FF$

What is the probability of a sequence of rolls

x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4

and the parse



π = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs $a_{0,Fair} = ½$, $a_{0,Loaded} = ½$)

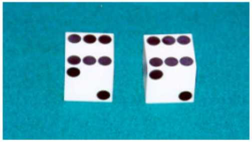
$½ \times P(1 \mid Fair)\, P(Fair \mid Fair)\, P(2 \mid Fair)\, P(Fair \mid Fair) \ldots P(4 \mid Fair) =$

$½ \times (1/6)^{10} \times (0.95)^{9} = 5.2 \times 10^{-9}$

Example: the dishonest casino $P(x, \pi)$, $\pi = LLLLLL...LL$

So, the likelihood the die is fair in all this run is 5.2×10^{-9}

What about



π = Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

½ × P(1 | Loaded) P(Loaded|Loaded) … P(4 | Loaded) =

$\frac{1}{2} \times (1/10)^8 \times (1/2)^2 (0.9)^9 = 4.8 \times 10^{-10}$

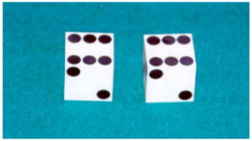
Therefore, it is more likely that the die is fair all the way, than loaded all the way

Example: the dishonest casino, loglikehood-ratio

A likelihood ratio test is a statistical test used for comparing the goodness of fit of two models, one of which (the null model) is a special case of the other (the alternative model)

$$log(\frac{P(X|\pi_{Fair})}{P(X|\pi_{Loaded})}) = log(\frac{5.2-09}{4.8e-10}) = 10.76$$

Example: the dishonest casino: Suspicion of loaded dice



Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

And let's consider π = F, F,…, F

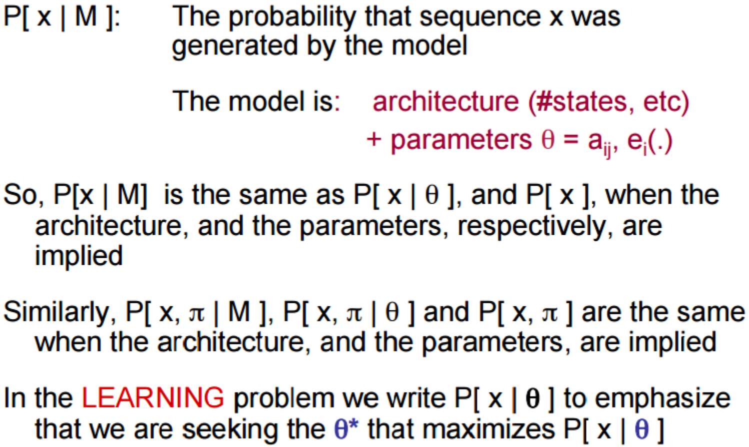
$P(x, \pi) = ½ \times (1/6)^{10} \times (0.95)^9 = 5.2 \times 10^{-9}$
(same as before)

And for π = L, L,…, L:

$P(x, \pi) = ½ \times (1/10)^4 \times (1/2)^6 (0.9)^9 = 3.02 \times 10^{-7}$

So, the observed sequence is ~100 times more likely if a loaded die is used

Clarification of notation



P[x | M]: The probability that sequence x was generated by the model

The model is: architecture (#states, etc) + parameters $\theta = a_{ij}, e_i(.)$

So, P[x | M] is the same as P[x | θ], and P[x], when the architecture, and the parameters, respectively, are implied

Similarly, P[x, π | M], P[x, π | θ] and P[x, π] are the same when the architecture, and the parameters, are implied

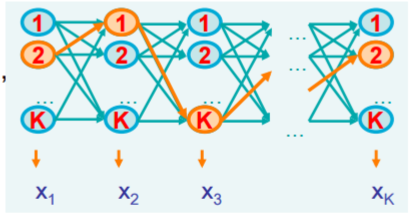
In the LEARNING problem we write P[x | θ] to emphasize that we are seeking the θ* that maximizes P[x | θ]

What we know

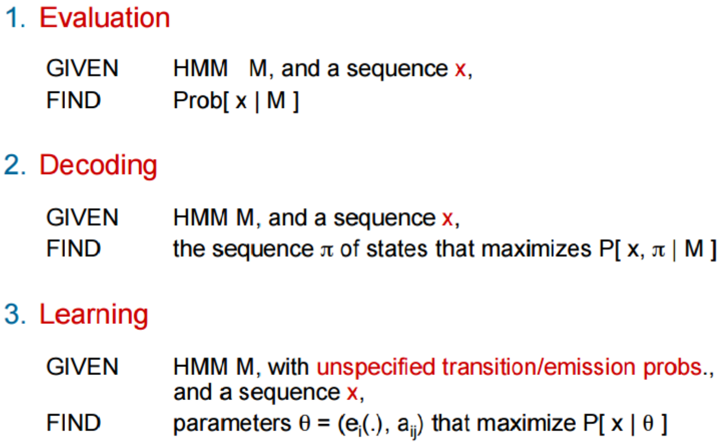
Given a sequence $x = x_1, \ldots, x_N$
and a parse $\pi = \pi_1, \ldots, \pi_N$,

we know how to compute
how likely the parse is:

$P(x, \pi)$



What we would know



1. Evaluation

GIVEN	HMM M, and a sequence x,
FIND	Prob[x | M]

2. Decoding

GIVEN	HMM M, and a sequence x,
FIND	the sequence π of states that maximizes P[x, π | M]

3. Learning

GIVEN	HMM M, with unspecified transition/emission probs., and a sequence x,
FIND	parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]

Problem 2: Decoding

Find the best parse of a sequence

Viterbi algorithm: *Notation*

Step i	i=0	i=1		i=k		i=l
Observation X:		x_1	...	x_k	...	x_l

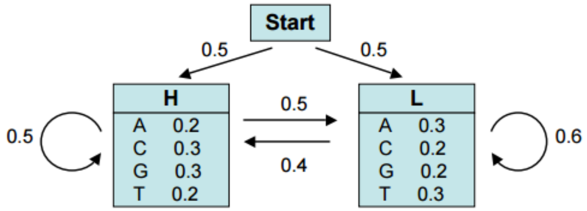
X_i : Observation at step i

$V_k(i)$: Probability of the most probable path ending in state k at position i with observation x_i

$a_{k,l}$: Probability of the transition from state l to k

$e_k(x_i)$: Probability to observe element i in state k

Viterbi algorithm Example: *Finding CpG Islands*

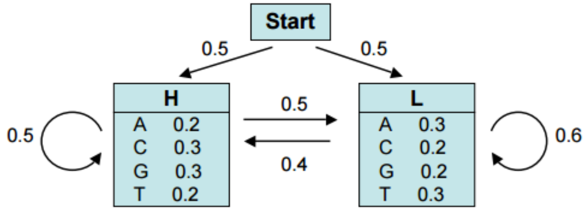


Let's consider the following simple HMM. This model is composed of 2 states, **H** (high GC content) and **L** (low GC content). We can for example consider that state H characterizes coding DNA while L characterizes a non-coding DNA.

The model can then be used to predict the region of coding DNA from a given sequence.

Sources: For the theory, see Durbin *et al* (1998);
For the example, see Borodovsky & Ekisheva (2006), pp 80-81

Viterbi algorithm: *Several Paths*



Consider the sequence S= **GGCACTGAA**

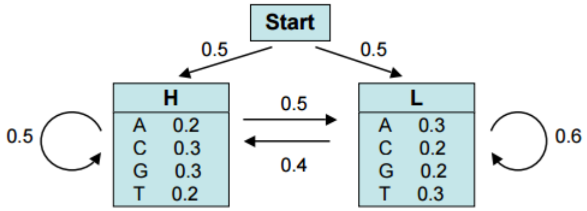
There are several paths through the hidden states (H and L) that lead to the given sequence.

Example: P = **LLHHHHLLL**

The probability of the HMM to produce sequence S through the path P is:

$$p = a_{0L} * e_L(G) * a_{LL} * e_L(G) * a_{LH} * e_H(C) * \ldots$$
$$p = 0.5 * 0.2 * 0.6 * 0.2 * 0.4 * 0.3 * \ldots$$
$$p = \ldots$$

Viterbi algorithm: *A dynamical Programming algorithm*



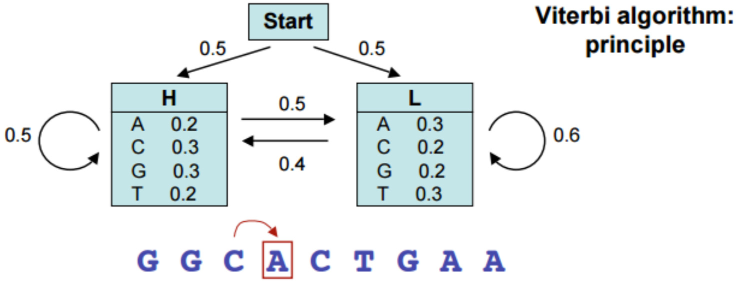
GGCACTGAA

There are several paths through the hidden states (H and L) that lead to the given sequence, but they do not have the same probability.

The **Viterbi algorithm** is a dynamical programming algorithm that allows us to compute the most probable path. Its principle is similar to the DP programs used to align 2 sequences (i.e. Needleman-Wunsch)

Source: Borodovsky & Ekisheva, 2006

Viterbi algorithm: *The most probable path $V_k(i)$*



Suppose the probability $V_k(i)$ of the most probable path ending in state k with observation i is known for all states k, then:

$$V_l(i+1) = e_l(x_i) \max_k (V_k(i) * a_{kl})$$

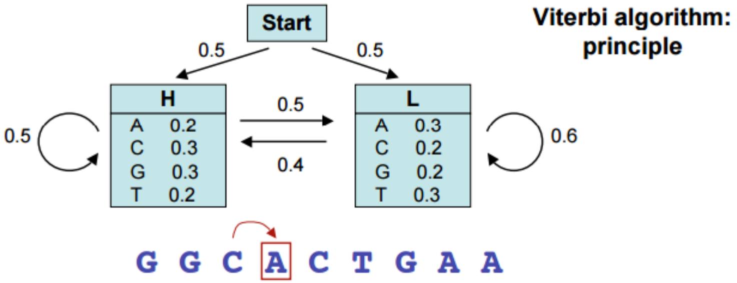
Probability of the most probable path at the end state *l*

Probability to observe element x_i in state *l*

Probability of the most probable path ending in state *k at position i*

Probability of the transition from state *k to state li*

Viterbi algorithm: *The probability of $V_H(4)$*



The probability of the most probable path ending in state **k** with observation "i" is

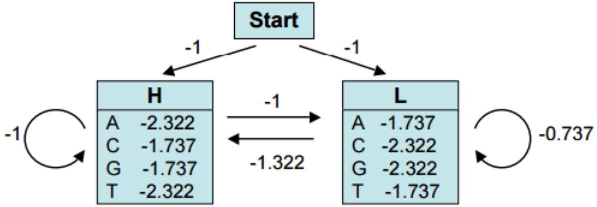
$$V_k(i) = e_k(x_i) \max_s (V_s(i-1) * a_{sl})$$

In our example, the probability of the most probable path ending in state **H** with observation "A" at the 4th position is:

$$V_H(4) = e_H(A) \max_s (V_L(3) * a_{LH}, V_H(3) * a_{HH})$$

We can thus compute recursively (from the first to the last element of our sequence) the probability of the most probable path.

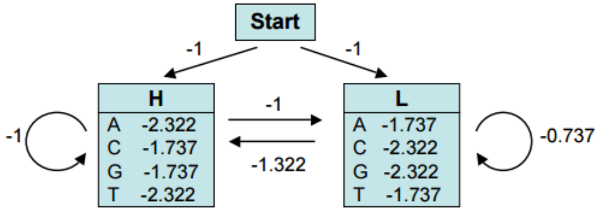
Viterbi algorithm: *Logs instead of Probabilities*



Remark: for the calculations, it is convenient to use the log of the probabilities (rather than the probabilities themselves). Indeed, this allows us to compute *sums* instead of *products*, which is more efficient and accurate.

We used here $\log_2(p)$.

Viterbi algorithm: *Maximum at the first position*



GGCACTGAA

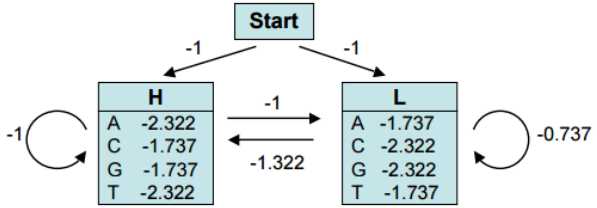
Probability (in $\log_2$) that G at the first position was emitted by state **H**

$$\boxed{V_H(1) = -1 - 1.737 = -2.737 \quad \text{(Maximum)}}$$

Probability (in $\log_2$) that G at the first position was emitted by state **L**

$$V_L(1) = -1 - 2.322 = -3.322$$

Viterbi algorithm: *Maximum at the second position*



GGCACTGAA

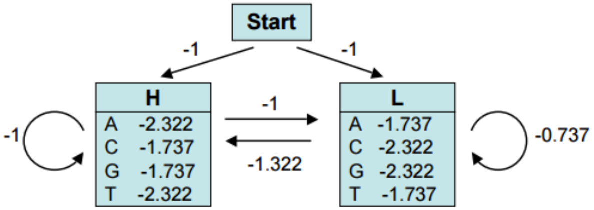
Probability (in $\log_2$) that G at the 2nd position was emitted by state **H**

$$\begin{aligned} V_H(2) &= -1.737 + max\left(V_H(1) + a_{HH}, V_L(1) + a_{LH}\right) \\ &= -1.737 + max\left(-2.737 - 1, -3.322 - 1.322\right) \\ &= -5.474 \text{ (obtained from } V_H(1) \text{)} \qquad \text{(Maximum)} \end{aligned}$$

Probability (in $\log_2$) that G at the 2nd position was emitted by state **L**

$$\begin{aligned} V_L(2) &= -2.322 + max\left(V_H(1) + a_{HL}, V_L(1) + a_{¿}\right) \\ &= -2.322 + max\left(-2.737 - 1, -3.322 - 0.737\right) \\ &= -6.059 \text{ (obtained from } V_H(1) \text{)} \end{aligned}$$

Viterbi algorithm: *Compute Iteratively the Probabilities*

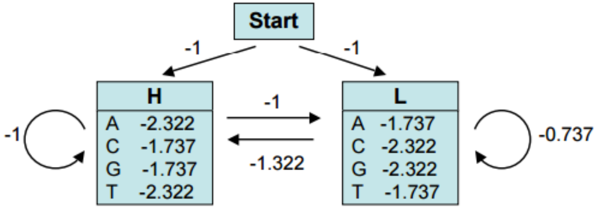


GGCACTGAA

	G	G	C	A	C	T	G	A	A
H	-2.73	-5.47	-8.21	-11.53	-14.01	...			-25.65
L	-3.32	-6.06	-8.79	-10.94	-14.01	...			-24.49

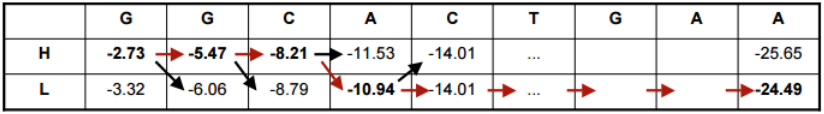
We then compute iteratively the probabilities $V_H(i)$ and $V_L(i)$ that nucleotide x_i at position i was emitted by state H or L, respectively. The highest probability obtained for the nucleotide at the last position is the probability of the most probable path. This path can be retrieved by back-tracking.

Viterbi algorithm: *Backtracking*



GGCACTGAA

back-tracking
(= finding the path which corresponds to the highest probability, -24.49)

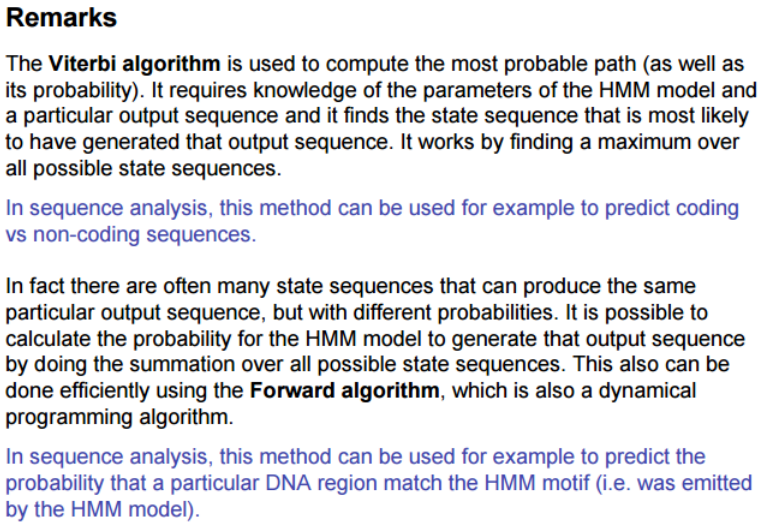


	G	G	C	A	C	T	G	A	A
H	**-2.73**	**-5.47**	**-8.21**	-11.53	-14.01	...			-25.65
L	-3.32	-6.06	-8.79	**-10.94**	-14.01	...			**-24.49**

The most probable path is: **HHHLLLLLL**

Its probability is $2^{-24.49} = 4.25E-8$
(remember that we used $\log_2(p)$)

Viterbi algorithm: *Remarks*



Remarks

The **Viterbi algorithm** is used to compute the most probable path (as well as its probability). It requires knowledge of the parameters of the HMM model and a particular output sequence and it finds the state sequence that is most likely to have generated that output sequence. It works by finding a maximum over all possible state sequences.

In sequence analysis, this method can be used for example to predict coding vs non-coding sequences.

In fact there are often many state sequences that can produce the same particular output sequence, but with different probabilities. It is possible to calculate the probability for the HMM model to generate that output sequence by doing the summation over all possible state sequences. This also can be done efficiently using the **Forward algorithm**, which is also a dynamical programming algorithm.

In sequence analysis, this method can be used for example to predict the probability that a particular DNA region match the HMM motif (i.e. was emitted by the HMM model).

The Viterbi algorithm

Input: $x = x_1,\ldots,x_N$

Initialization: $V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration: $V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$Ptr_j(i) = \text{argmax}_k\ a_{kj} V_k(i-1)$

Termination: $P(x, \pi^*) = \max_k V_k(N)$

Traceback: $\pi_N{}^* = \text{argmax}_k V_k(N)$

$\pi_{i-1}{}^* = Ptr_{\pi i}\ (i)$