EECS 4425:
Introductory Computational Bioinformatics

Fall 2018

Suprakash Datta

datta@cs.yorku.ca

Office: LAS 3043

Phone: 416-736-2100 ext 77875

Course page: http://www.cs.yorku.ca/course/4425

Next

Markov chains and hidden Markov models

Some of the following slides are based on slides from
www.bioalgorithms.info

Markov chains

- Simplest probabilistic model with states
- Finite state machine with probabilistic state transitions
- Useful for modeling many, many systems

Example 0

- Random walks on the integers
- Random walks on a cube

Example 1

- Modeling exons and introns
- What is the probability of getting a T after a A?
- Differentiates exons, introns

Example 2

- The casino problem
- Switch probabilistically between a fair coin F and an unfair (biased) coin B
- Thus, we define the probabilities:
  - $P(H|F) = P(T|F) = ½$
  - $P(H|B) = ¾$, $P(T|B) = ¼$
  - The crooked dealer changes between F and **B** with probability 10%

Example 3

- CpG islands
- CG dinucleotide typically modified by methylation; then the C is more likely to mutate to a G
- Methylation suppressed in short stretches (clusters), typically around genes, and CpG occurs more frequently in these islands
- So, finding the *CpG* islands in a genome is an important problem

The Markov property

- “Memoryless” state transitions
- Memoryless vs stateless
- Path to current node has no influence on current (or future) transitions

Probabilities of paths

- Paths are sequences of states
- The Markovian property simplifies the equation

$$P(x_1 x_2 \ldots x_n) = P(x_n | x_1 x_2 \ldots x_{n-1})\, P(x_1 x_2 \ldots x_{n-1})$$
$$= P(x_n | x_{n-1})\, P(x_1 x_2 \ldots x_{n-1})$$
$$= a_{n-1,n}\, P(x_1 x_2 \ldots x_{n-1})$$
$$= \ldots.$$
$$= P(x_1)\, a_{1,2} \ldots a_{n-3,n-2}\, a_{n-2,n-1}\, a_{n-1,n}$$

Graph-theoretic formulation

- Directed graph
- Nodes are states
- Edge weights are transition probabilities
- Graph structure determines properties of Chain

Linear Algebraic formulation

- Relationship between the state probability vectors in successive timesteps
- $X_{t+1} = A X_t$

Fundamental questions

- When is there a “steady state”?
- What is the state probability vector at steady state?
- How quickly does the Markov chain approach steady state?

Answers

- Existence of steady state can be formulated in terms of graph properties (e.g. all pairs reachable from each other)
- Steady state probability easier in terms of linear algebraic formulation:

$$\text{solve } \pi = A\,\pi$$

- Speed of convergence: second largest eigenvalue of A

Hidden Markov Models (HMM)

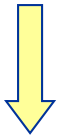
- Determining when the casino switches from fair to unfair coins or vice versa

Problem…

Fair Bet Casino Problem

Any observed outcome of coin tosses could have been generated by any sequence of states!

Need to incorporate a way to grade different sequences differently.



Decoding Problem

P(*x*|fair coin) vs. P(*x*|biased coin)

- Suppose first that dealer never changes coins. Some definitions:
  - P(*x*|fair coin): prob. of the dealer using the *F* coin and generating the outcome *x*.
  - P(*x*|biased coin): prob. of the dealer using the *B* coin and generating outcome *x*.

P(*x*|fair coin) vs. P(*x*|biased coin)

- P(*x*|fair coin)=P($x_1 \ldots x_n$|fair coin)

  $\Pi_{i=1,n}\, p\,(x_i|\text{fair coin}) = (1/2)^n$

- P(*x*|biased coin)= P($x_1 \ldots x_n$|biased coin)=

$\Pi_{i=1,n}\, p\,(x_i|\text{biased coin}) = (3/4)^k (1/4)^{n-k} = 3^k/4^n$

  - *k* - the number of ***H***eads in *x*.

P(*x*|fair coin) vs. P(*x*|biased coin)

- P(*x*|fair coin) = P(*x*|biased coin)
- $1/2^n = 3^k/4^n$
- $2^n = 3^k$
- $n = k \log_2 3$
- when $k = n / \log_2 3$ ($k \sim 0.67n$)

Log-odds Ratio

- We define ***log-odds ratio*** as follows:

$$\log_2(\mathrm{P}(x|\text{fair coin}) / \mathrm{P}(x|\text{biased coin})) = \sum_{i=1}^{k} \log_2(p^+(x_i) / p^-(x_i)) = n - k \log_2 3$$

Computing Log-odds Ratio in Sliding Windows

$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \ldots x_n$$

Consider a *sliding window* of the outcome sequence. Find the log-odds for this short window.

Hidden Markov Model (HMM)

- Can be viewed as an abstract machine with *k* *hidden* states that emits symbols from an alphabet Σ.
- Each state has its own probability distribution, and the machine switches between states according to this probability distribution.
- While in a certain state, the machine makes 2 decisions:
  - What state should I move to next?
  - What symbol - from the alphabet Σ - should I emit?

Why “Hidden”?

- Observers can see the emitted symbols of an HMM but have *no ability to know which state the HMM is currently in*.
- Thus, the goal is to infer the most likely hidden states of an HMM based on the given sequence of emitted symbols.

HMM Parameters

Σ: set of emission characters.

Ex.: Σ = {H, T} for coin tossing

Σ = {1, 2, 3, 4, 5, 6} for dice tossing

Q: set of hidden states, each emitting symbols from Σ.

Q={F,B} for coin tossing

HMM Parameters (cont'd)

A = (a_{kl}): a |Q| x |Q| matrix of probability of changing from state *k* to state *l*.

$$a_{FF} = 0.9 \quad a_{FB} = 0.1$$

$$a_{BF} = 0.1 \quad a_{BB} = 0.9$$

E = $(e_k(b))$: a |Q| x |Σ| matrix of probability of emitting symbol *b* while being in state *k*.

$$e_F(0) = ½ \quad e_F(1) = ½$$

$$e_B(0) = ¼ \quad e_B(1) = ¾$$

HMM for Fair Bet Casino

- The *Fair Bet Casino* in *HMM* terms:

  Σ = {0, 1} (0 for ***T***ails and 1 ***H***eads)

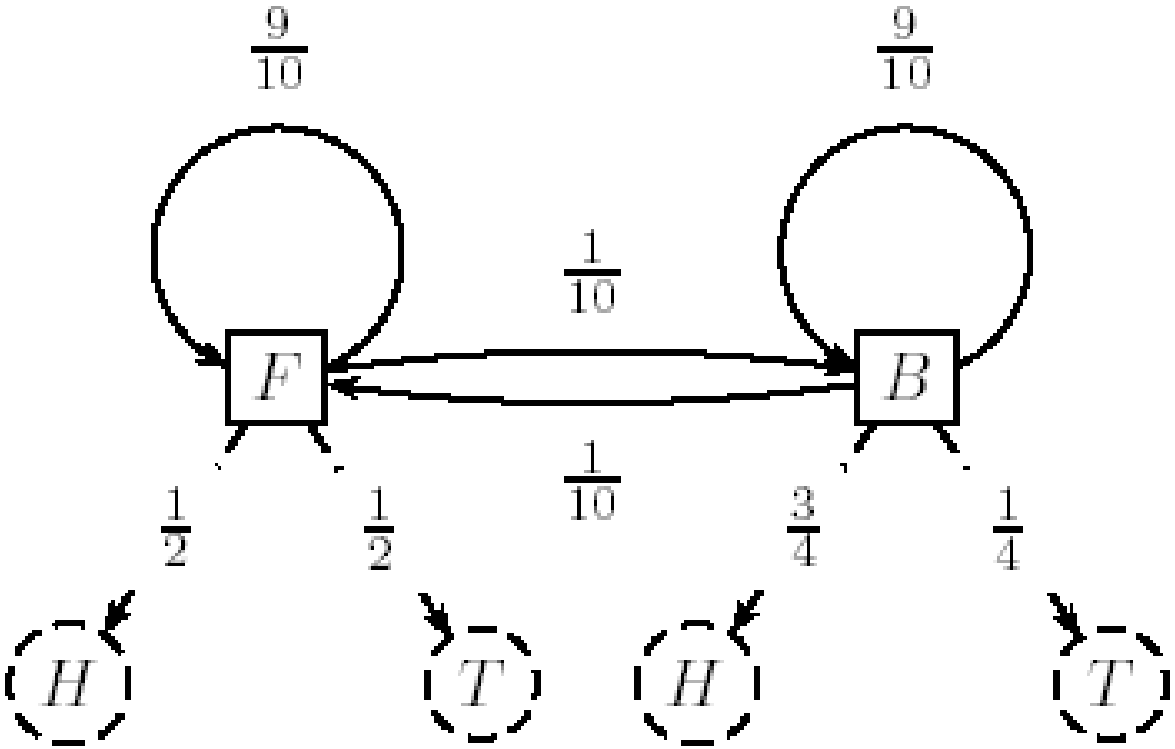
  Q = {*F*,*B*} – *F* for Fair & *B* for Biased coin.

- Transition Probabilities *A* *** Emission Probabilities *E*

	Fair	Biased
Fair	$a_{FF} = 0.9$	$a_{FB} = 0.1$
Biased	$a_{BF} = 0.1$	$a_{BB} = 0.9$

	Tails(0)	Heads(1)
Fair	$e_F(0) = ½$	$e_F(1) = ½$
Biased	$e_B(0) = ¼$	$e_B(1) = ¾$

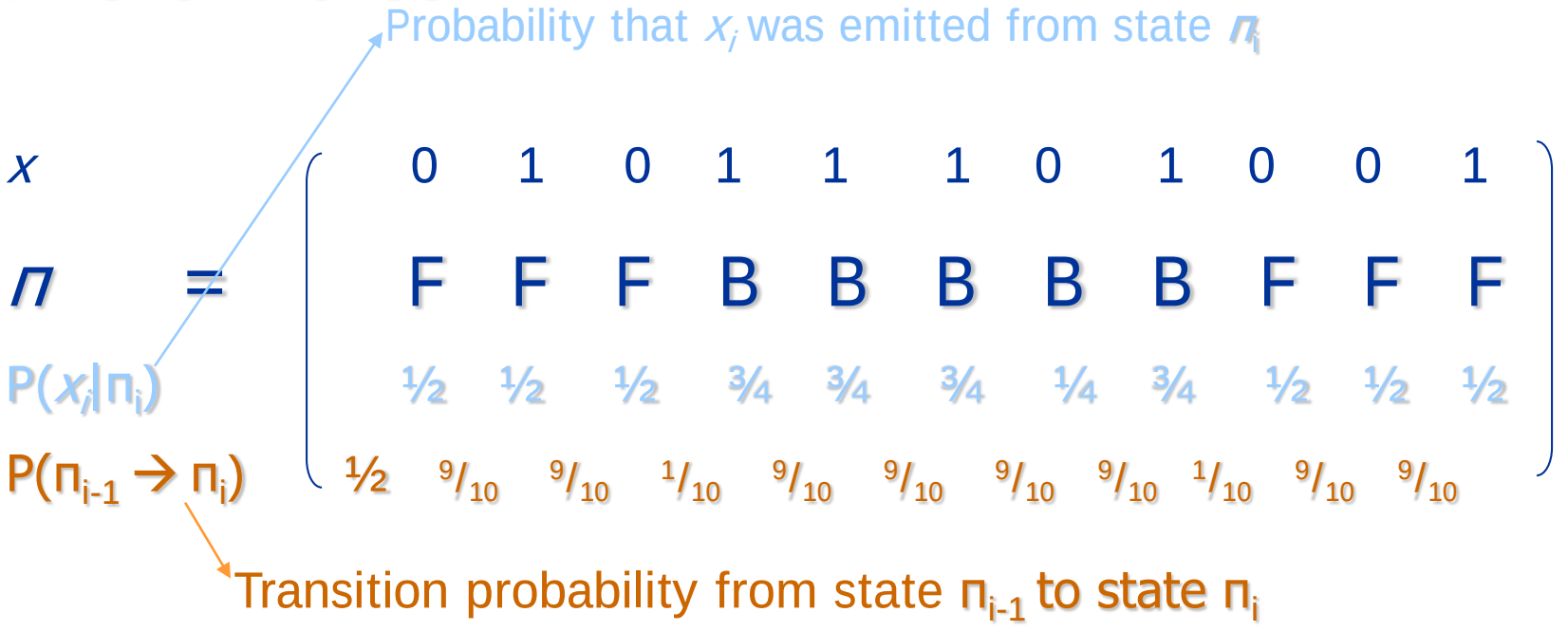
HMM for Fair Bet Casino (cont'd)



HMM model for the *Fair Bet Casino* Problem

Hidden Paths

- A *path* $\pi = \pi_1 \ldots \pi_n$ in the HMM is defined as a sequence of states.
- Consider path π = FFFBBBBBFFF and sequence x = 01011101001



P(*x*|π) Calculation

- P(*x*|π): Probability that sequence *x* was generated by the path *π:*

$$P(x|\pi) = P(\pi_0 \rightarrow \pi_1) \cdot \prod_{i=1}^{n} P(x_i|\pi_i) \cdot P(\pi_i \rightarrow \pi_{i+1})$$

$$= a_{\pi_0, \pi_1} \cdot \prod_{i=1}^{n} e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}}$$

P(*x*|π) Calculation

- P(*x*|π): Probability that sequence *x* was generated by the path *π:*

$$P(x|\pi) = P(\pi_0 \rightarrow \pi_1) \cdot \prod_{i=1}^{n} P(x_i|\pi_i) \cdot P(\pi_i \rightarrow \pi_{i+1})$$

$$= a_{\pi_0, \pi_1} \cdot \prod e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}}$$

$$= \prod e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}$$

if we count from *i=0* instead of *i=1*

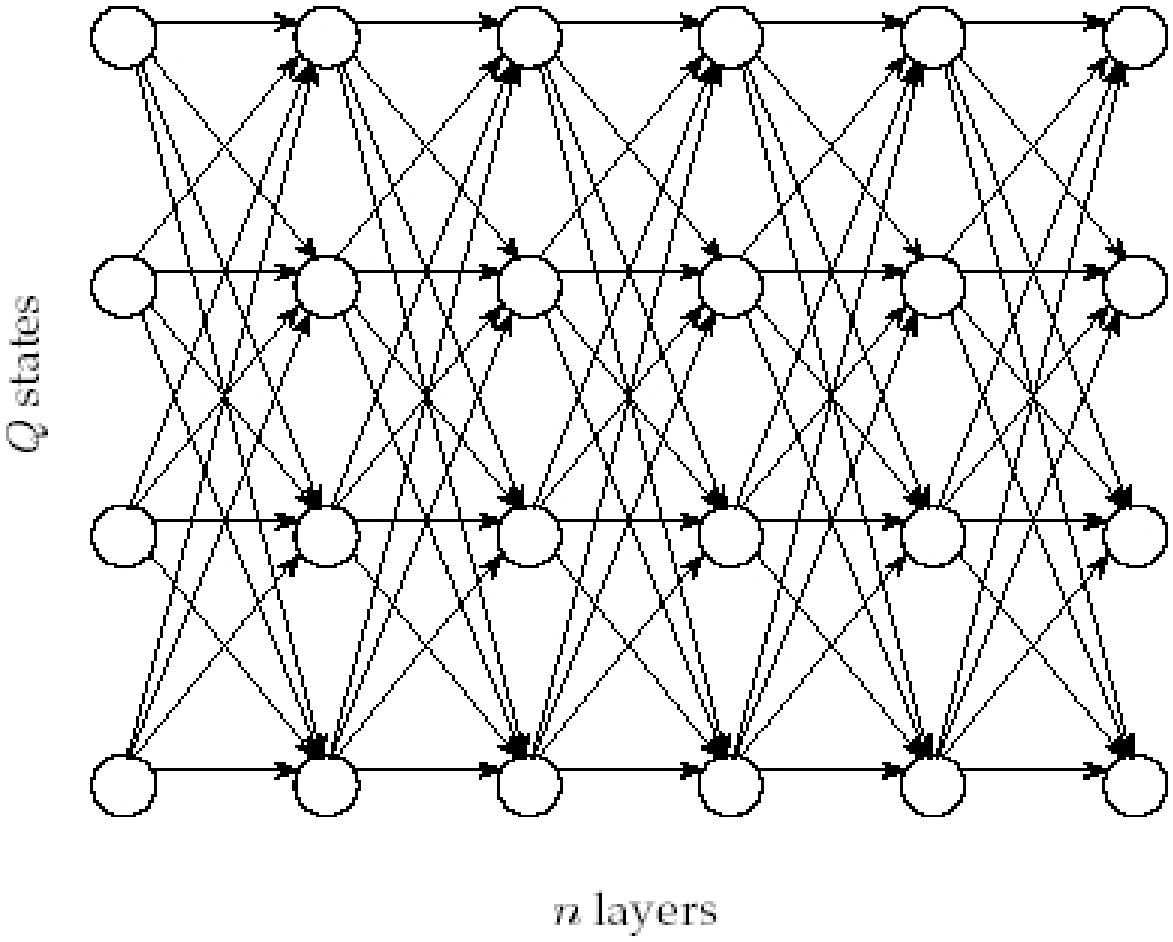
Decoding Problem

- **Goal:** Find an optimal hidden path of states given observations.
- **Input:** Sequence of observations $x = x_1 \ldots x_n$ generated by an HMM $M(\Sigma, Q, A, E)$
- **Output:** A path that maximizes $P(x|\pi)$ over all possible paths π.

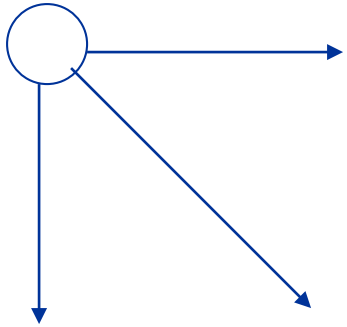
Building Manhattan for Decoding Problem

- Andrew Viterbi used the Manhattan grid model to solve the *Decoding Problem*.
- Every choice of $\pi = \pi_1 \ldots \pi_n$ corresponds to a path in the graph.
- The only valid direction in the graph is *eastward.*
- This graph has $|Q|^2(n-1)$ edges.

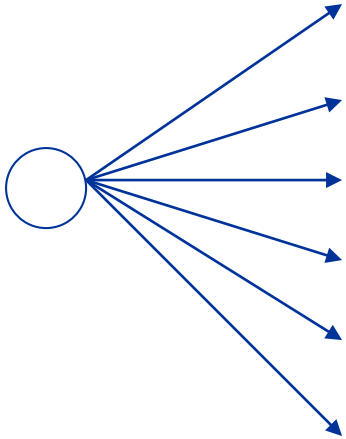
Edit Graph for Decoding Problem



Decoding Problem vs. Alignment Problem



Valid directions in the *alignment problem.*



Valid directions in the *decoding problem.*

Decoding Problem as Finding a Longest Path in a DAG

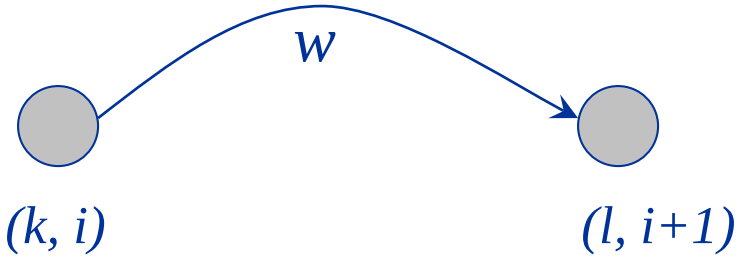
- The *Decoding Problem* is reduced to finding a longest path in the *directed acyclic graph (DAG)* above.

- **Notes:** the length of the path is defined as the *product* of its edges' weights, not the *sum.*

Decoding Problem (cont'd)

- Every path in the graph has the probability $P(x|\pi)$.
- The Viterbi algorithm finds the path that maximizes $P(x|\pi)$ among all possible paths.
- The Viterbi algorithm runs in $\boldsymbol{O(n|Q|^2)}$ time.

Decoding Problem: weights of edges

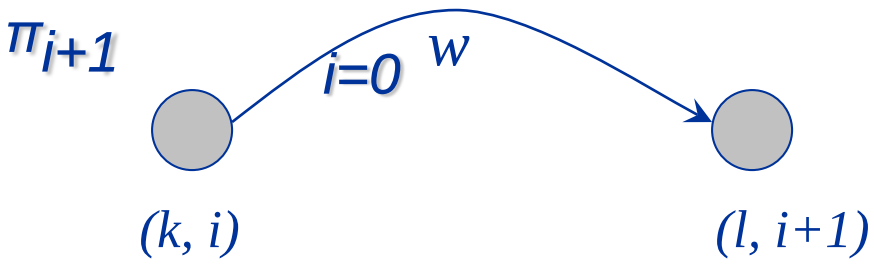


The weight ***w*** is given by:

???

Decoding Problem: weights of edges

$$P(x|\pi) = \prod_{i=0}^{n} e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i,\pi_{i+1}}$$

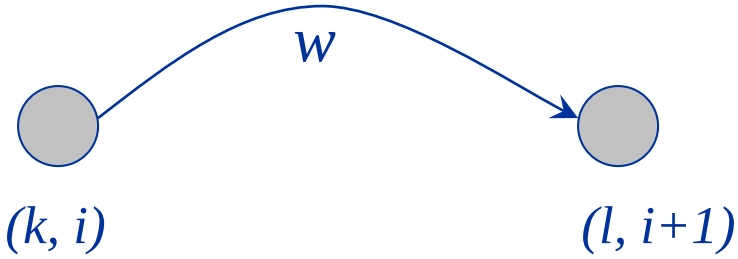


The weight ***w*** is given by:

??

Decoding Problem: weights of edges

i-th term = $e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}$

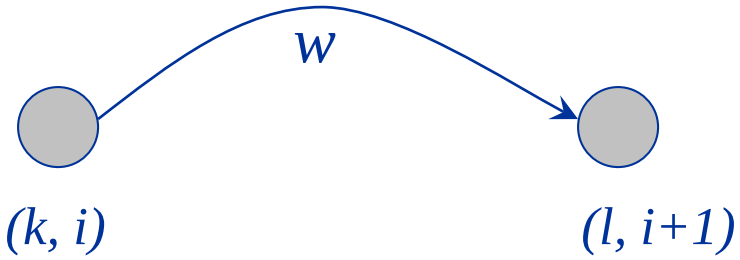


The weight ***w*** is given by:

?

Decoding Problem: weights of edges

i-th term = $e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}} = e_l(x_{i+1}) \cdot a_{kl}$ *for* $\pi_i = k$, $\pi_{i+1} = l$



The weight $w = e_l(x_{i+1}) \cdot a_{kl}$

Decoding Problem and Dynamic Programming

$s_{l,i+1} = \max_{k \in Q} \{s_{k,i} \cdot$ wt of edge between (k,i) and $(l,i+1)\}$

$= \max_{k \in Q} \{s_{k,i} \cdot a_{kl} \cdot e_l(x_{i+1})\}$

$= e_l(x_{i+1}) \cdot \max_{k \in Q} \{s_{k,i} \cdot a_{kl}\}$

Decoding Problem (cont'd)

- Initialization:
  - $s_{begin,0} = 1$
  - $s_{k,0} = 0$ for $k \neq begin$.
- Let π^* be the optimal path. Then,

$$P(x|\pi^*) = \max_{k \in Q} \{s_{k,n} \cdot a_{k,end}\}$$

Viterbi Algorithm

- The value of the product can become extremely small, which leads to overflowing.
- To avoid overflowing, use log value instead.

$$s_{k,i+1} = \log e_l(x_{i+1}) + \max_{k \in Q} \{s_{k,i} + \log(a_{kl})\}$$

Forward-Backward Problem

Given: a sequence of coin tosses generated by an HMM.

Goal: find the probability that the dealer was using a biased coin at a particular time.

Forward Algorithm

- Define $f_{k,i}$ (*forward probability*) as the probability of emitting the prefix $x_1 \ldots x_i$ and reaching the state $\pi = k$.
- The recurrence for the forward algorithm:

$$f_{k,i} = e_k(x_i) \cdot \sum_{l \in Q} f_{l,i-1} \cdot a_{lk}$$

Backward Algorithm

- However, *forward probability* is not the only factor affecting $P(\pi_i = k|x)$.

- The sequence of transitions and emissions that the HMM undergoes between π_{i+1} and π_n also affect $P(\pi_i = k|x)$.

Backward Algorithm (cont'd)

- Define *backward probability* $b_{k,i}$ as the probability of being in state $\pi_i = k$ and emitting the *suffix* $x_{i+1} \ldots x_n$.
- The recurrence for the *backward algorithm*:

$$b_{k,i} = \sum_{l \in Q} e_l(x_{i+1}) \cdot b_{l,i+1} \cdot a_{kl}$$

Backward-Forward Algorithm

- The probability that the dealer used a biased coin at any moment i:

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

$P(x)$ is the sum of $P(x, \pi_i = k)$ over all k

Profile Representation of Protein Families

Aligned DNA sequences can be represented by a $4 \cdot n$ profile matrix reflecting the frequencies of nucleotides in every aligned position.

A	.72	.14	0	0	.72	.72	0	0
T	.14	.72	0	0	0	.14	.14	.86
G	.14	.14	.86	.44	0	.14	0	0
C	0	0	.14	.56	.28	0	.86	.14

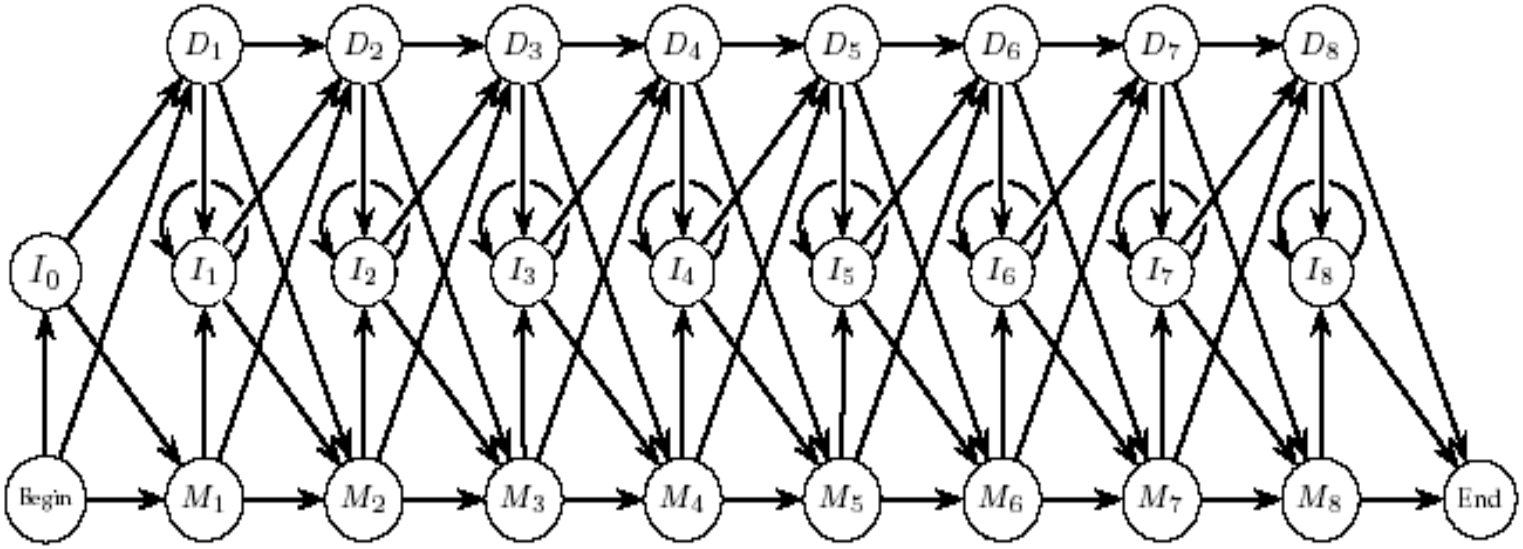
Protein family can be represented by a $20 \cdot n$ profile representing frequencies of amino acids.

Profiles and HMMs

- HMMs can also be used for aligning a sequence against a profile representing

  protein family.
- A $20 \cdot n$ profile P corresponds to n sequentially linked *match* states $M_1,\ldots,M_n$ in the profile HMM of P.

Profile HMM



A profile HMM

Building a profile HMM

- Multiple alignment is used to construct the HMM model.
- Assign each column to a *Match* state in HMM. Add I*nsertion* and *Deletion* state.
- Estimate the emission probabilities according to amino acid counts in column. Different positions in the protein will have different emission probabilities.
- Estimate the transition probabilities between *Match, Deletion* and *Insertion* states
- The HMM model gets trained to derive the optimal parameters.

States of Profile HMM

- Match states $M_1 \ldots M_n$ (plus *begin/end* states)
- Insertion states $I_0 I_1 \ldots I_n$
- Deletion states $D_1 \ldots D_n$

Transition Probabilities in Profile HMM

- $log(a_{MI})+log(a_{IM})$ = gap initiation penalty
- $log(a_{II})$ = gap extension penalty

Emission Probabilities in Profile HMM

- Probability of emitting a symbol *a* at an insertion state I_j:

$$e_{I_j}(a) = p(a)$$

where $p(a)$ is the frequency of the occurrence of the symbol *a* in all the sequences.

Profile HMM Alignment

- Define $v^M_j(i)$ as the logarithmic likelihood score of the best path for matching $x_1..x_i$ to profile HMM ending with x_i emitted by the state M_j.

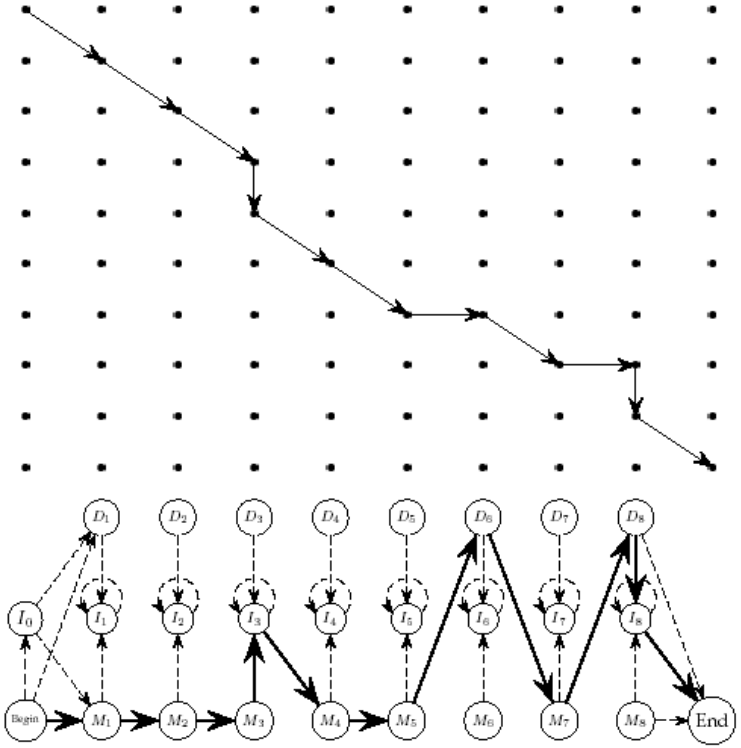
- $v^I_j(i)$ and $v^D_j(i)$ are defined similarly.

Profile HMM Alignment: Dynamic Programming

$$v^M_j(i) = \log\left(e_{M_j}(x_i)/p(x_i)\right) + \max \begin{cases} v^M_{j-1}(i-1) + \log(a_{M_{j-1},M_j}) \\ v^I_{j-1}(i-1) + \log(a_{I_{j-1},M_j}) \\ v^D_{j-1}(i-1) + \log(a_{D_{j-1},M_j}) \end{cases}$$

$$v^I_j(i) = \log\left(e_{I_j}(x_i)/p(x_i)\right) + \max \begin{cases} v^M_j(i-1) + \log(a_{M_j}, I_j) \\ v^I_j(i-1) + \log(a_{I_j}, I_j) \\ v^D_j(i-1) + \log(a_{D_j}, I_j) \end{cases}$$

Paths in Edit Graph and Profile HMM



A path through an edit graph and the corresponding path through a profile HMM

HMM Parameter Estimation

- So far, we have assumed that the transition and emission probabilities are known.

- However, in most HMM applications, the probabilities are not known. It's very hard to estimate the probabilities.

HMM Parameter Estimation Problem

- Given
  - HMM with states and alphabet (emission characters)
  - Independent training sequences $x^1, \ldots x^m$
- Find HMM parameters Θ (that is, a_{kl}, $e_k(b)$) that maximize

$$P(x^1, \ldots, x^m \mid \Theta)$$

the joint probability of the training sequences.

Maximize the likelihood

$P(x^1, \ldots, x^m \mid \Theta)$ as a function of Θ is called the likelihood of the model.

The training sequences are assumed independent, therefore

$$P(x^1, \ldots, x^m \mid \Theta) = \Pi_i P(x^i \mid \Theta)$$

The parameter estimation problem seeks Θ that realizes

$$\max_{\Theta} \prod_i P(x^i \mid \Theta)$$

In practice the log likelihood is computed to avoid underflow errors

Two situations

Known paths for training sequences

- CpG islands marked on training sequences
- One evening the casino dealer allows us to see when he changes dice

Unknown paths

- CpG islands are not marked
- Do not see when the casino dealer changes dice

Known paths

A_{kl} = # of times each $k \rightarrow l$ is taken in the training sequences

$E_k(b)$ = # of times b is emitted from state k in the training sequences

Compute a_{kl} and $e_k(b)$ as maximum likelihood estimators:

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

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

Pseudocounts

- Some state k may not appear in any of the training sequences. This means $A_{kl} = 0$ for every state l and a_{kl} cannot be computed with the given equation.
- To avoid this overfitting use predetermined pseudocounts r_{kl} and $r_k(b)$.

  A_{kl} = # of transitions $k \rightarrow l$ + r_{kl}

  $E_k(b)$ = # of emissions of b from k + $r_k(b)$

The pseudocounts reflect our prior biases about the probability values.

Unknown paths: Baum-Welch

Idea:

1. Guess initial values for parameters.

   art and experience, not science

2. Estimate new (better) values for parameters.

   how ?

3. Repeat until stopping criteria is met.

   what criteria ?

Better values for parameters

Would need the A_{kl} and $E_k(b)$ values but cannot count (the path is unknown) and do not want to use a most probable path.

For all states *k,l,* symbol *b* and training sequence *x*

Compute A_{kl} and $E_k(b)$ as expected values, given the current parameters

Notation

For any sequence of characters x emitted along some unknown path π, denote by $\pi_i = k$ the assumption that the state at position i (in which x_i is emitted) is k.

The Baum-Welch algorithm

Initialization:

Pick the best-guess for model parameters (or arbitrary)

Iteration:

1. Forward for each *x*
2. Backward for each *x*
3. Calculate A_{kl}, $E_k(b)$
4. Calculate new a_{kl}, $e_k(b)$
5. Calculate new log-likelihood

Until log-likelihood does not change much

Baum-Welch analysis

- Log-likelihood is increased by iterations

  Baum-Welch is a particular case of the EM (expectation maximization) algorithm
- Convergence to local maximum. Choice of initial parameters determines local maximum to which the algorithm converges