Lecture 24: Hidden Markov Models (Continued)

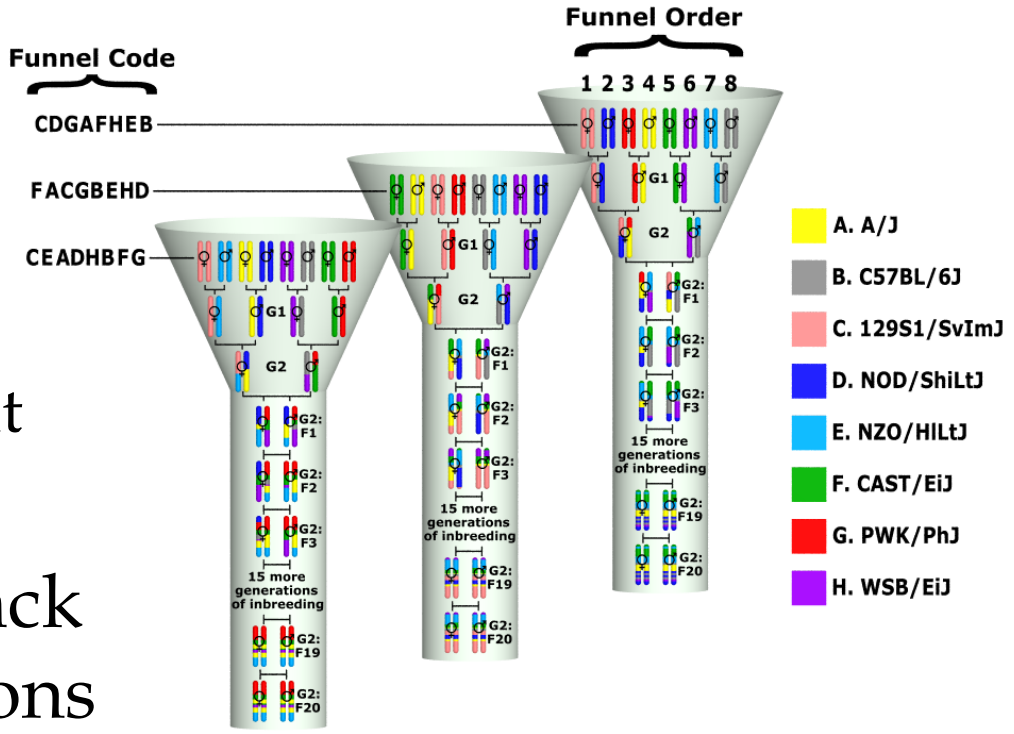
Not in book

HMMs in Biology

- Inferring ancestral contributions to a descendant
- Collaborative Cross project
- Maintained at UNC since 2006
- Objective:
  - Create new *reproducible* mouse strains by *randomly* combining the genomes of eight *diverse* mice strains
- Problem:
  - Given an extant strain, which parts of its genome came from which founder

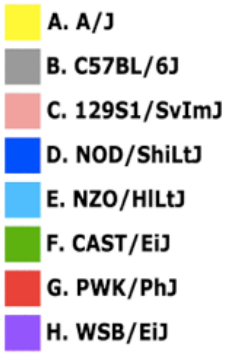
Mixing Genomes

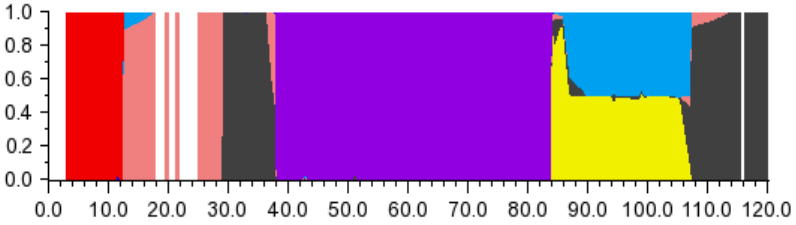
- A randomized breeding scheme was used to
  - Mix the genomes by recombination
  - Fix the genomes by selective inbreeding
- A breeding funnel
  - 8 genomes go in
  - A mosaic comes out
- Genotyping was to used monitor to track founder contributions



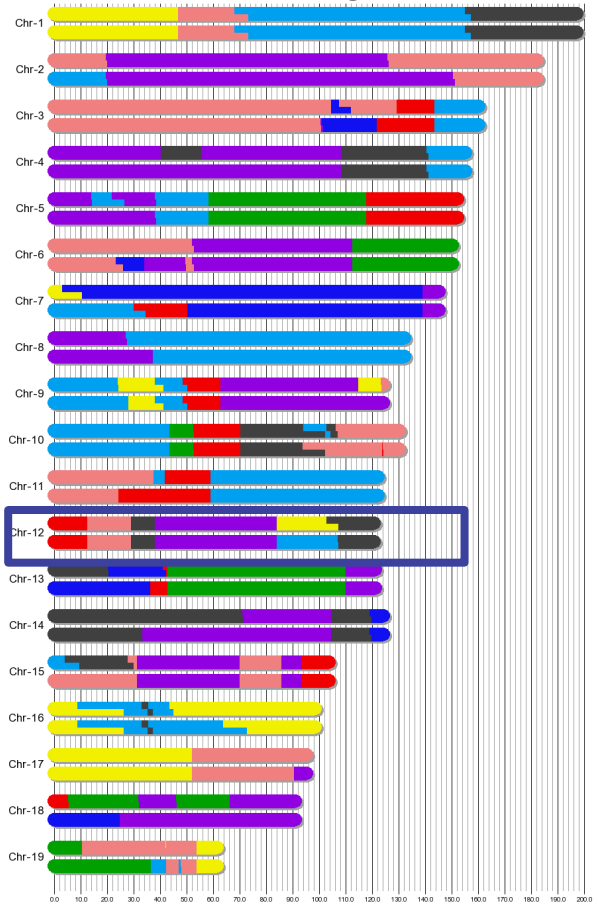
A Genome Mosaic

- A Hidden Markov Model is used to infer the "hidden" state of which of the 8 founders contributed to which parts of the genome
- A Viterbi Solution finds the most likely mosaic given a set of genotypes



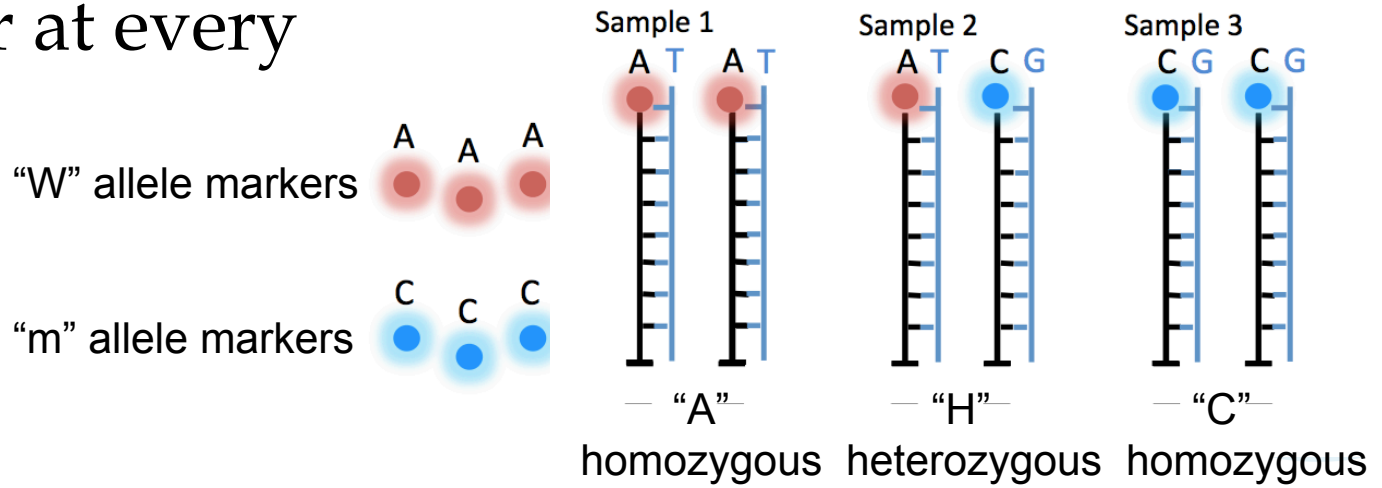


Mosaic after 7 generations



Genotyping Microarrays

- DNA *probes* to query the state of specific "known" and "informative" Single Nucleotide Polymorphisms SNPs
- Each probe distinguishes 4 cases ("W","m","H","N")
- From these observations we infer the founder at every marker



Example Genotypes

- Genotypes for a chromosome
- 8112 probes with position of variant
- Alleles are indicated by the nucleotide
- Rarely can a single maker resolve the founder
- Which strain would you guess for the beginning?

Probe info | Founder Genotypes | Target

chromosome	positionB38	A/J	C57BL/6J	129S1/SvImJ	NOD/ShiLtJ	NZO/H1LtJ	CAST/EiJ	PWK/PhJ	WSB/EiJ	OR3199m266
2	3176721	G	T	G	T	G	G	T	T	T
2	3180256	G	G	G	G	G	A	A	G	G
2	3182308	A	G	A	G	A	G	G	A	A
2	3183784	T	G	T	G	T	G	G	T	T
2	3233750	G	G	G	G	G	A	G	G	G
2	3350920	A	A	A	A	A	G	G	A	A
2	3353380	T	T	C	T	C	C	C	C	C
2	3362696	T	T	T	T	T	T	C	T	T
2	3420272	C	C	T	C	T	T	T	C	C
2	3433708	G	G	G	G	G	A	A	G	G
2	3438642	C	C	T	C	T	C	T	C	C
2	3456515	C	C	C	C	C	T	C	C	C
2	3503822	T	T	T	T	C	T	C	T	T
2	3557793	A	A	A	A	A	G	G	A	A
2	3595443	T	T	G	T	G	G	G	T	T
2	3613854	A	A	A	A	G	G	G	A	A
2	3663247	T	T	T	T	T	C	C	T	T
2	3666094	G	G	G	G	G	G	T	T	T
2	3681891	G	G	G	G	G	A	G	G	G
2	3715097	G	G	G	G	G	T	T	G	G

Noise

- One last issue, between 1% and 5% of genotypes are simply wrong
- Technical errors
  - A probe didn't glow bright enough
  - A section of the array was damaged (fingerprints, cracks, hair, etc.)
  - Messed up fabricating a probe's sequence
  - DNA was contaminated
- Error types:
  - Unexpected calls (observation is uninformative)
  - A possible, but incorrect call

Read Genotypes

```
fp = open("genotypes.csv", 'rU')
data = fp.read().split('\n')          # break file into lines
fp.close()
header = data.pop(0).split(',')       # First line is header
while (len(data[-1].strip()) < 1):    # remove extra lines
    data.pop()
for i, line in enumerate(data):       # make a list from each row
    field = line.split(',')
    field[1] = int(field[1])          # convert position to integer
    data[i] = field
fp.close()

print ”Number of probes”, len(data)


Number of Probes: 8112

data[1000] = ['2', 25896880, 'T', 'C', 'C', 'C', 'T', 'C', 'T', 'T', 'T']
data[1001] = ['2', 25914367, 'A', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
data[1002] = ['2', 25936735, 'T', 'T', 'T', 'T', 'C', 'C', 'T', 'T', 'T']
data[1003] = ['2', 25940660, 'G', 'A', 'A', 'A', 'G', 'G', 'G', 'G', 'G’]
```

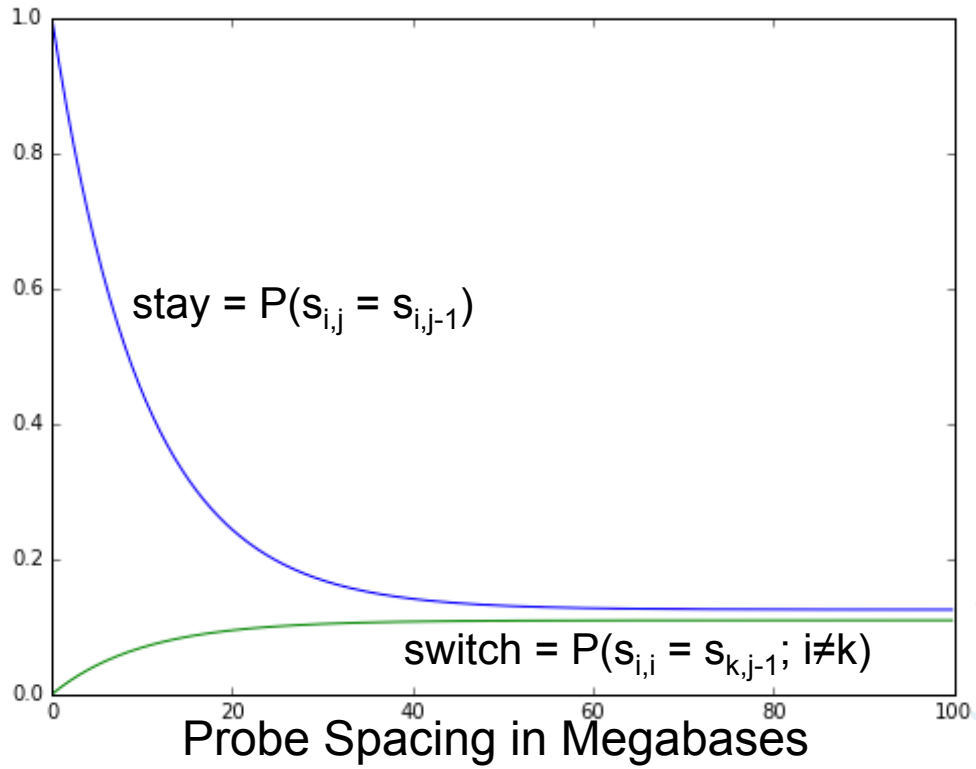
Viterbi Dynamic Program

```
from math import log10

Nstates = 8
prevpos = 1
state = [[(float(len(data)),i) for i in xrange(Nstates)]]   # (log(p), PathToHere)
for i in xrange(len(data)):
    # Count expected genotypes
    count = dict([(call, data[i][2:2+Nstates].count(call)) for call in "ACGTHN"])
    # Get the target genotype at this probe
    observed = data[i][-1]
    # Compute emission probability, assuming 5% error rate
    if (count[observed] == 0):
        emission = [1.0/Nstates for j in xrange(2,2+Nstates)]   # unexpected
    else:
        emission = [0.95/count[data[i][j]] if data[i][j] == observed else 0.05/count[data[i][j]]
                    for j in xrange(2,2+Nstates)]
    # compute transition probability
    position = data[i][1]
    delta = position - prevpos
    prevpos = position
    stay = ((Nstates - 1.0)*math.exp(-delta/1000000.0) + 1.0)/Nstates
    switch = (1.0 - stay)/(Nstates - 1.0)
    # update state probailities for all paths leading to the ith state
    path = []
    for j in xrange(Nstates):
        choices = [(log10(emission[j])+(log10(stay) if (k==j) else log10(switch))+state[-1][k][0],k)
                   for k in xrange(Nstates)]
        path.append(max(choices))
    state.append(path)
print "Length of paths:", len(state)
```

Transition Probability

- Recombination likelihood is modeled using an exponential distribution
- Recombinations between nearby probes are unlikely
- Distant probes can be from other founders



Backtracking

```
    # backtrack
    path = state[-1]
    maxi = 0
    maxp = path[0][0]
    for i in xrange(1,Nstates):
        if (path[i][0] > maxp):
            maxp = path[i][0]
            maxi = i
    print maxi, path[maxi], header[2+maxi]

    for j in xrange(len(state)-2,-1,-1):
        data[j].append(header[2+maxi])
        maxi = state[j+1][maxi][1]

    header.append("Founder")

    5 (2686.5234854121827, 5) CAST/EiJ


fp = open("result.csv", 'w')
fp.write(','.join(header)+'\n')
for row in data:
    fp.write(','.join([str(v) for v in row])+'\n')
fp.close()
```

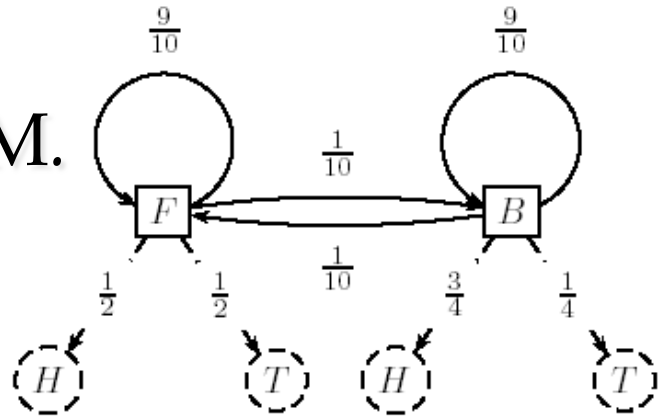
Output

- The inferred Mosaic
- Repeat for every chromosome
- Most likely, but how likely?
- Other approaches

chromosome	position	A/J	C57BL/6J	129S1/SvImJ	NOD/ShiLtJ	NZO/H1LtJ	CAST/EiJ	PWK/PhJ	WSB/EiJ	OR3199m266	Founder
2	3176721	G	T	G	T	G	G	T	T	T	WSB/EiJ
2	3180256	G	G	G	G	G	A	A	G	G	WSB/EiJ
2	3182308	A	G	A	G	A	G	G	A	A	WSB/EiJ
2	3183784	T	G	T	G	T	G	G	T	T	WSB/EiJ
2	3233750	G	G	G	G	G	A	G	G	G	WSB/EiJ
⋮											
2	132621710	T	T	T	T	T	C	C	T	T	WSB/EiJ
2	132624885	G	G	G	G	G	A	A	A	A	WSB/EiJ
2	132655807	A	A	A	A	A	G	G	A	A	NZO/H1LtJ
2	132658252	T	T	C	T	T	C	T	C	T	NZO/H1LtJ
⋮											
2	161893676	A	A	G	A	A	G	A	A	A	NZO/H1LtJ
2	161895302	T	C	T	C	C	C	C	C	C	NZO/H1LtJ
2	161922951	T	T	T	T	C	C	T	T	C	CAST/EiJ
2	161938620	G	A	G	A	G	A	A	G	A	CAST/EiJ
⋮											
2	172257444	C	C	C	C	C	A	C	C	A	CAST/EiJ
2	172287540	C	C	C	C	C	T	T	T	T	CAST/EiJ
2	172321479	G	A	G	A	A	G	G	A	G	PWK/PhJ
2	172352159	C	T	C	T	T	C	C	T	C	PWK/PhJ
⋮											
2	180938391	G	G	G	G	G	A	A	G	A	PWK/PhJ
2	180965832	T	T	C	C	T	C	C	T	C	PWK/PhJ
2	181009379	T	T	T	T	T	C	C	T	C	CAST/EiJ
2	181011845	C	C	C	C	C	C	T	T	C	CAST/EiJ

Forward-Backward Problem

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



Goal: find the most probable coin that the house was using at a particular flip.

$$P(\pi_i = k \mid x) = \frac{P(x, \pi_i = k)}{P(x)}$$

Probabilities of all paths in state k at i

Probability of sequence over all paths

Illustrating the difference

Not a lot worse than the best solution

x =	THHH	p
	FFFF	(0.0228)
	BFFF	(0.0013)
	FBFF	(0.0004)
	BBFF	(0.0019)
	FFBF	(0.0004)
	BFBF	(0.0000)
	FBBF	(0.0006)
	BBBF	(0.0028)
	FFFB	(0.0038)
	BFFB	(0.0002)
	FBFB	(0.0001)
	BBFB	(0.0003)
	FFBB	(0.0057)
	BFBB	(0.0003)
	FBBB	(0.0085)
	BBBB	(0.0384)

$P(x) = 0.0877$

Veterbi solution, the most likely sequence states.

High probability output (>0.0625)

x =	THHH	p
	FFFF	(0.0228)
	FFBF	(0.0004)
	FFFB	(0.0038)
	FFBB	(0.0057)
	BFFF	(0.0013)
	BFBF	(0.0000)
	BFFB	(0.0002)
	BFBB	(0.0003)

$P(\pi_2=F|x) = 0.0345/0.0877 = 0.3936$

	FBFF	(0.0004)
	FBBF	(0.0006)
	FBFB	(0.0001)
	FBBB	(0.0085)
	BBFF	(0.0019)
	BBBF	(0.0028)
	BBFB	(0.0003)
	BBBB	(0.0384)

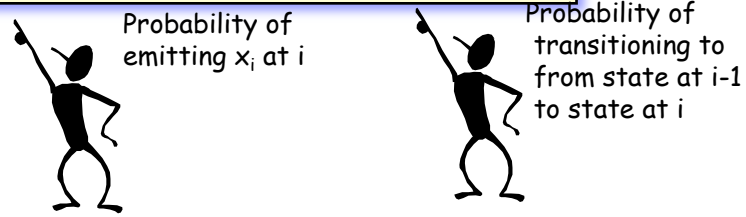
$P(\pi_2=B|x) = 0.0532/0.0877 = 0.6064$

The forward-backward algorithm tells us how likely we were using the biased coin at the second flip.

Forward Algorithm

- Defined $f_{k,i}$ (*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 is:

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



- Same as Viterbi except stop at k

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 and π_{i+1} also affect $P(\pi_i = k|x)$.

Backward Algorithm (cont'd)

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

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

This is the same as computing the probability of a specific path or suffix in this case except the initial probability is not $\frac{1}{2}$.

Backward-Forward Algorithm

- The probability that the dealer used a biased coin at any moment *i* is as follows:

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

- So, to find $P(\pi_i = k|x)$ for all I, we solve two dynamic programs
  - One from beginning to end
  - One from the end to the beginning
  - Combine the corresponding states

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.

- Parameter estimation is much harder than state estimation

HMM Parameter Estimation (cont'd)

- Let Θ be a vector containing all of the unknown transition and emission probabilities.
- Given training sequences $x^1, \ldots, x^m$, let $P(x \mid \Theta)$ be the max. prob. of x given the assignment of param.'s Θ.
- Then our goal is to find

$$max_{\Theta} \prod_{j=1}^{m} P(x_i|\Theta)$$

A Parameter Estimation Approach

- If hidden states were known, we could use our training data to estimate parameters

$$a_{kl} = \frac{A_{kl}}{\sum_{q \in Q} A_{kq}} \qquad e_k(b) = \frac{E_k(b)}{\sum_{\sigma \in \Sigma} E_k(\sigma)}$$

- In all likelihood we wouldn't be given the hidden state sequence, π, but only the observed output stream, x
- An alternative is to *make an intelligent guess of* π, use the equations above to estimate parameters, then run Viterbi to estimate the hidden state, then reestimate the parameters and repeat until the state assignments or parameter values converge.
- Such iterative approaches are called Expectation Maximization (EM) methods of parameter estimation

Profile Alignment using HMMs

- Distant species of functionally related sequences may have weak pairwise similarities with known species, and thus fail individual pairwise significance tests.
- However, they may have weak similarities with ***many*** known species.
- The goal is to consider sequences at once. (Multiple alignment)
- Related sequences are often better represented by a *consensus profile* that any multiple alignment.

Profile Representations

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

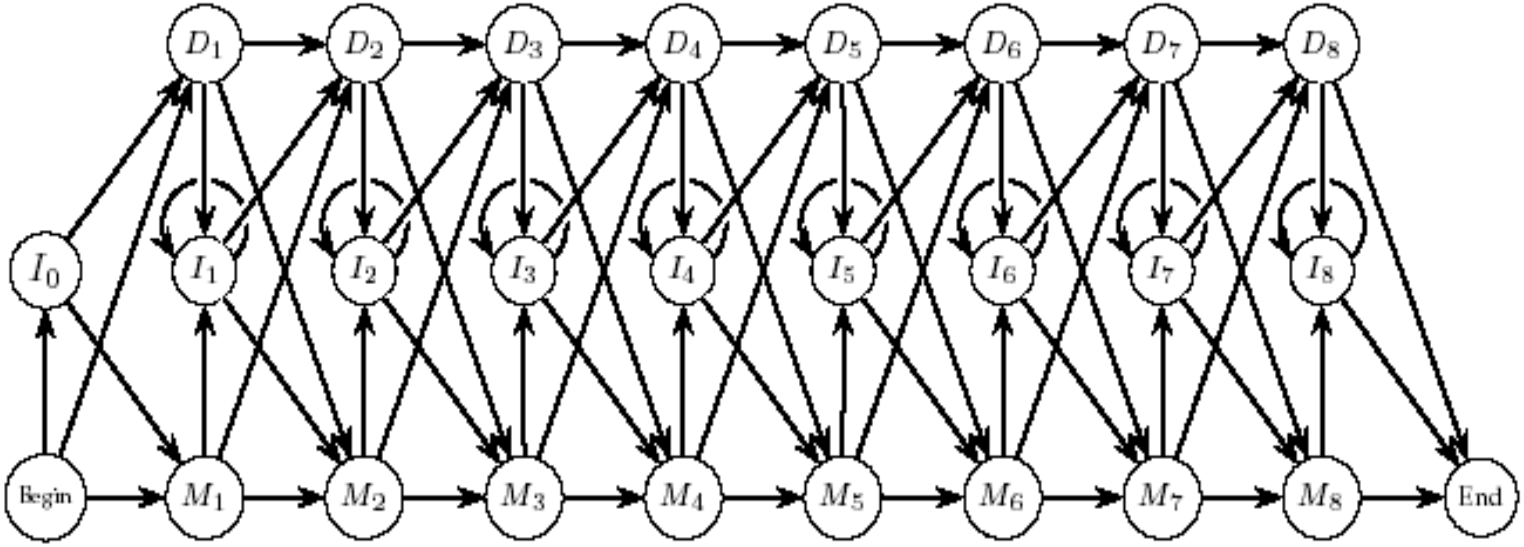
A	.72	.14	0	0	.72	.72	0	0
T	.14	.72	0	0	0	.14	.14	.86
G	.14	.14	.86	.44	0	.14	0	0
C	0	0	.14	.56	.28	0	.86	.14

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

HMM Alignment

- One method of performing sequence comparisons to a profile is to use a HMM
- Emission probabilities, $e_i(a)$, from the profile
- Transition probabilities from our match -mismatch matrix δ_{ij}.
- Or we can explicitly represent the insertion and deletion states

Profile HMM



A profile HMM

M_i, Match States
D_i, Delete States
I_i, Insertion States

Recall this is a model of a process, not a graph of a solution approach. However, every alignment is a path in this graph.

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 $D_1 \ldots D_n$

- Assumption:

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

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

Transition Probabilities in Profile HMM

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

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.

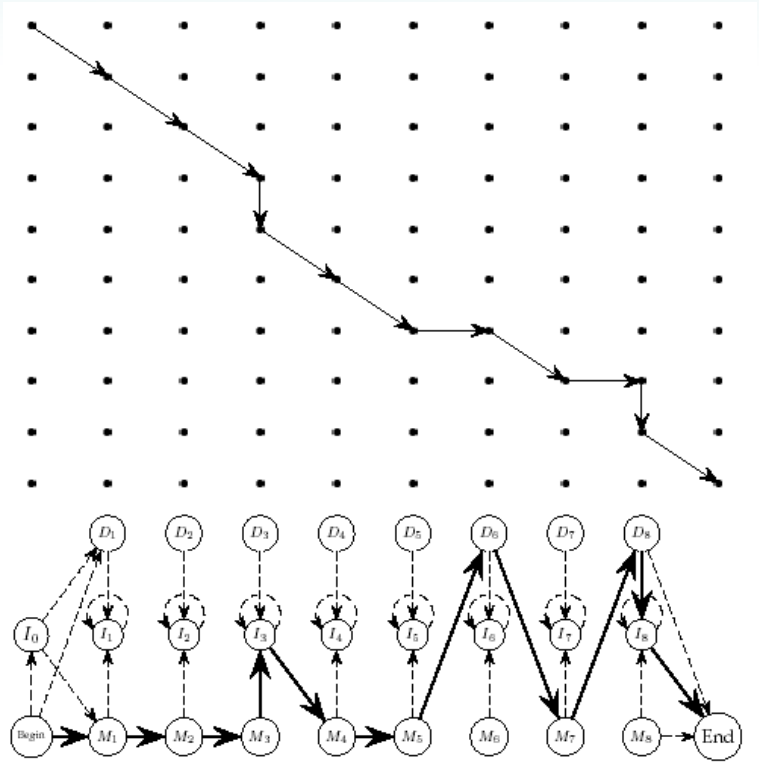
- $v^I_j(i)$ and $v^D_j(i)$ are defined similarly.

Profile HMM Alignment: Dynamic Programming

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

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

Paths in Edit Graph and Profile HMM



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