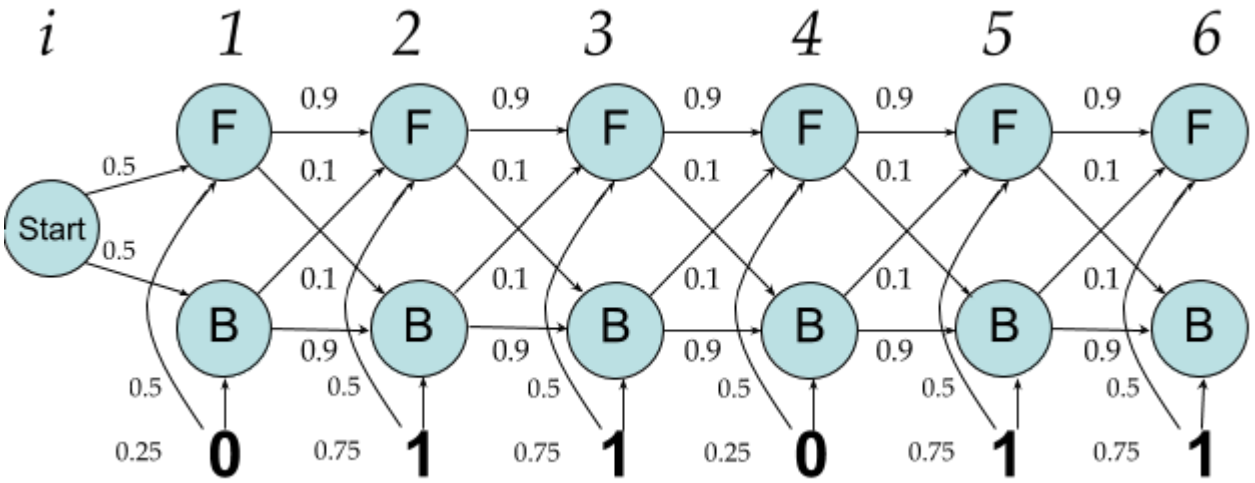
Inferring Ancestory using HMMs

- This lecture is key for Problem Set #5

Decoding Problem Solution

- The *Decoding Problem* is equivalent to finding a longest path in the *directed acyclic graph* (DAG), where "longest" is defined as the maximum product of the probabilities along the path.



Viterbi Decoding Algorithm

- Since the *longest path* is a product of edge weights, if we use the **log** of the weights we can make it a sum again!
- The value of the product can become extremely small, which leads to underflow.
- Logs avoid underflow (precision loss due to adding numbers of vastly different magnitudes)

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

Viterbi Decoding Problem (cont)

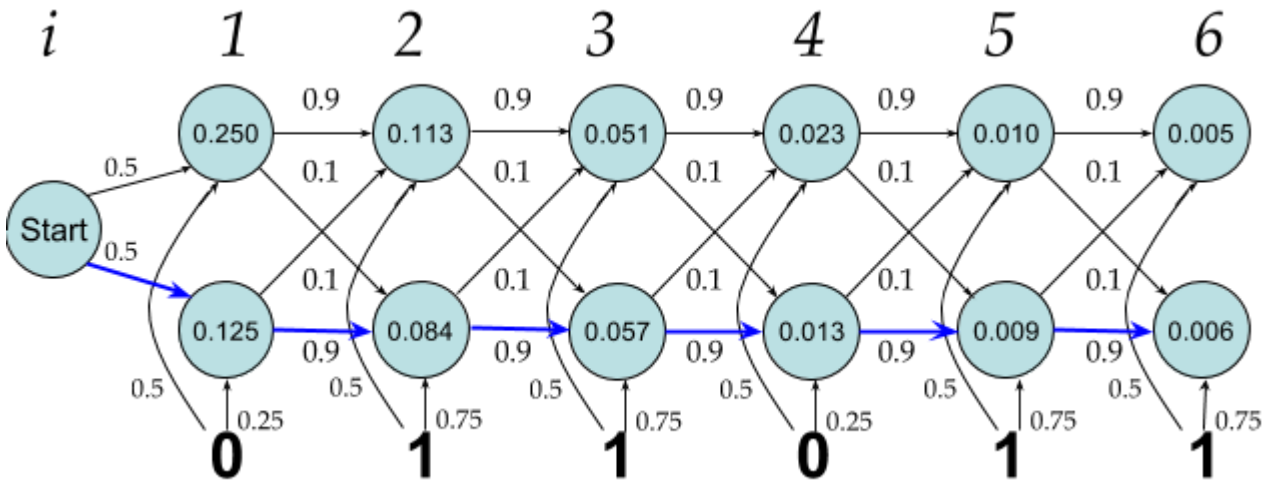
- Every path in the graph has the probability $P(x|\pi)$.
- The Viterbi decoding algorithm finds the path that maximizes P(x|π) among all possible paths.
- The Viterbi decoding algorithm runs in $O(n|Q|^2)$ time (length of sequence times number of states squared).
- The Viterbi decoding algorithm can be efficiently implemented as a dynamic program

Dynamic Program's Recursion

$$
\begin{aligned}
s_{l,i+1} &= \max_{k \in Q} \{ s_{k,i} \cdot \text{weight of edge between } (k,i) \text{ and } (l,i+1) \} \\
&= \max_{k \in Q} \{ s_{k,i} \cdot a_{kl} \cdot e_l(x_{i+1}) \} \\
&= e_l(x_{i+1}) \cdot \max_{k \in Q} \{ s_{k,i} \cdot a_{kl} \}
\end{aligned}
$$

Viterbi Example

- Solves all subproblems implied by observed sequence
- How likely is this path? *0.006*
- What is it? ***BBBBBB***



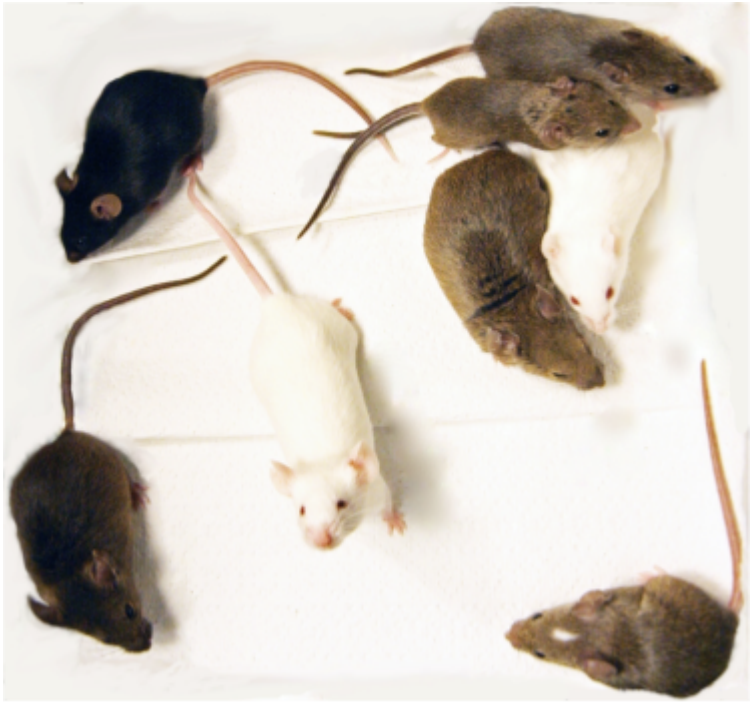
How likely is most likely?

- The "most likely path" may not be a lot more likely than a 2nd or 3rd most likely paths (more so in more realistic cases than this one).
- Actual probability of the "most likely path" is not that high.

P	π	P	π	P	π	P	π
0.0058	BBBBBB	0.0001	BBBFFB	0.0000	FFFBFF	0.0000	FBBFBF
0.0046	FFFFFF	0.0001	FFFFBF	0.0000	FFBFBB	0.0000	BFBBFF
0.0013	FBBBBB	0.0001	FFBFFF	0.0000	FBFFBB	0.0000	BFFBBF
0.0012	FFFFBB	0.0001	FBFFFF	0.0000	FBBFFB	0.0000	BBFBFF
0.0009	FFBBBB	0.0001	FFBBBF	0.0000	FFBFFB	0.0000	FFBFBF
0.0008	FFFFFB	0.0001	BFFFBB	0.0000	FBFFFB	0.0000	FBFFBF
0.0006	FFFBBB	0.0001	FBBBFF	0.0000	FBFBBB	0.0000	BFFBFF
0.0006	BBBFFF	0.0001	BBFFFB	0.0000	FBBBFB	0.0000	BFBFBB
0.0004	BBBBBF	0.0000	BFBBBB	0.0000	BBBFBF	0.0000	FBFBBF
0.0004	BBFFFF	0.0000	BBBBFB	0.0000	FFBBFB	0.0000	BFBFFB
0.0003	BBBBFF	0.0000	BBFBBB	0.0000	BBFFBF	0.0000	FBFBFF
0.0003	BFFFFF	0.0000	BFFFFB	0.0000	BFFFBF	0.0000	BFBBFB
0.0001	BBBFBB	0.0000	FFFBBF	0.0000	BFBFFF	0.0000	BBFBFB
0.0001	FBBFFF	0.0000	FFBBFF	0.0000	FFFBFB	0.0000	BFFBFB
0.0001	FBBBBF	0.0000	FBBFBB	0.0000	BFBBBF	0.0000	FBFBFB
0.0001	BBFFBB	0.0000	BFFBBB	0.0000	BBFBBF	0.0000	BFBFBF

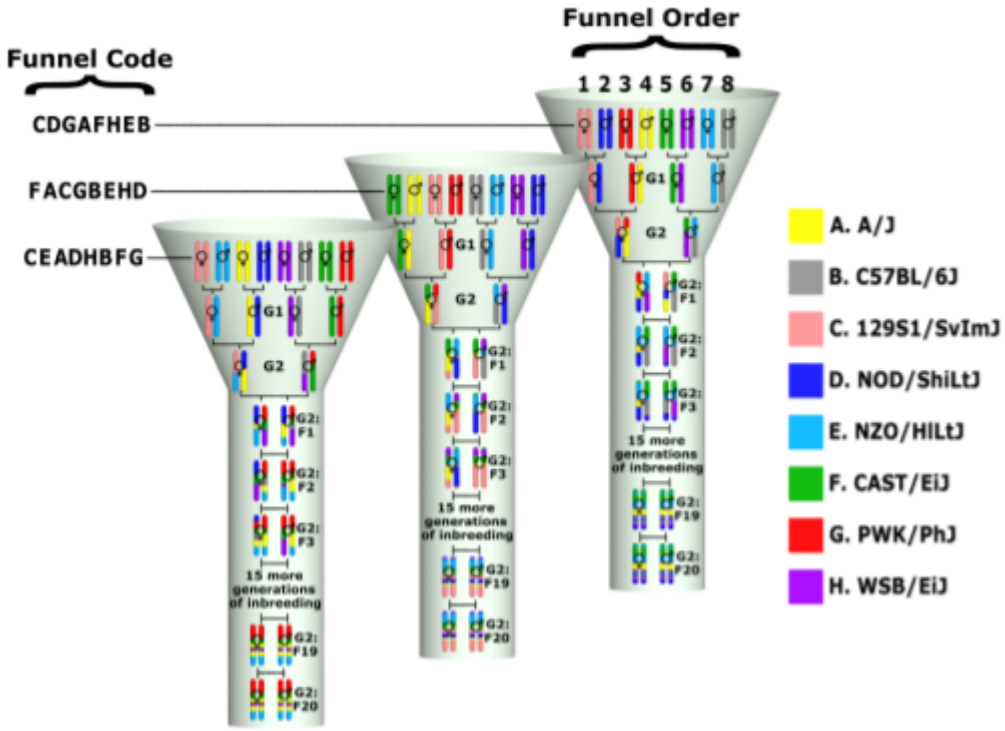
HMMs in Biology

- 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



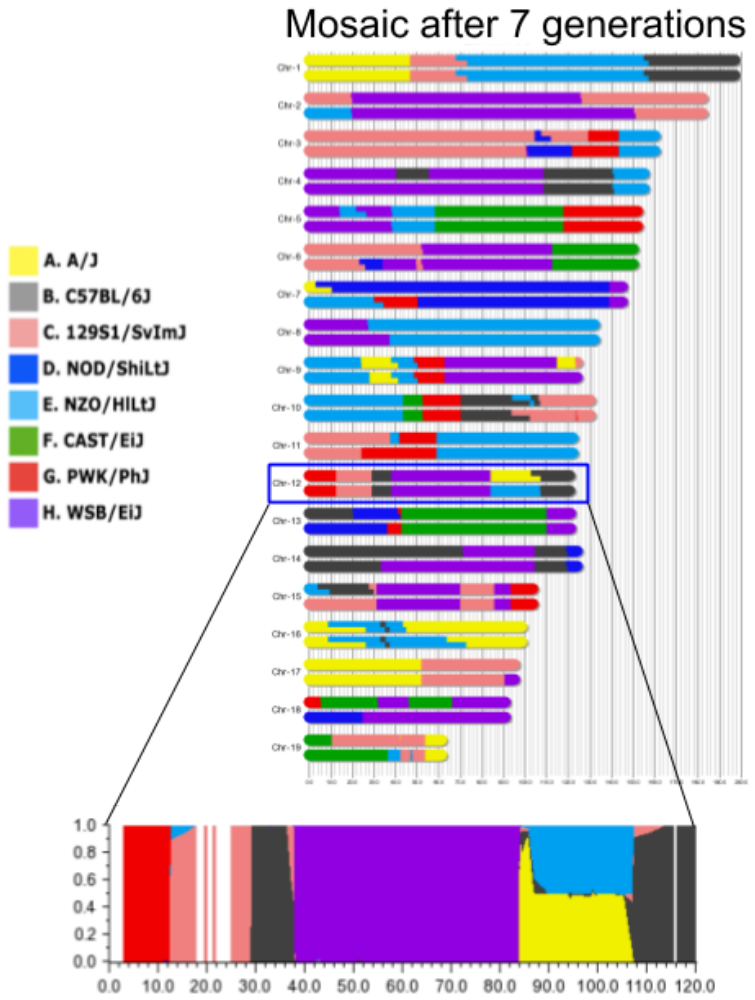
Mixing Genomes

- 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 used to track founder contributions



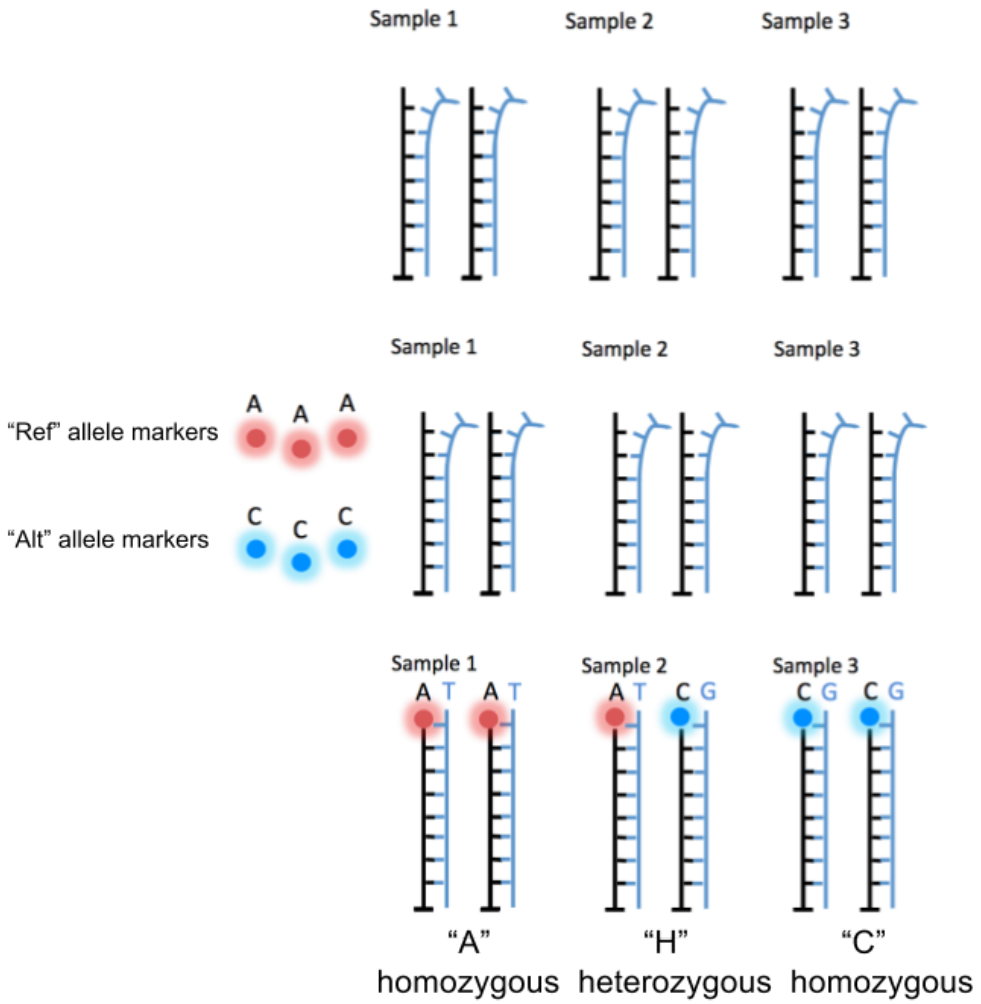
A Genome Mosaic

- 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



Genotyping Microarrays

- DNA probes to query the state of specific “known” and “informative” Single Nucleotide Polymorphisms (SNPs)
- Each probe distinguishes 4 cases ("Ref", "Alt", "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("data/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 header
print "Number of probes", len(data)
for i in xrange(1000,1005):
    print "data[%d] = %s" % (i, data[i])
```

```
['chromosome', 'position', 'A/J', 'C57BL/6J', '129S1/SvImJ', 'NOD/ShiLtJ', 'NZO/H1LtJ', 'CAST/Ei
J', 'PWK/PhJ', 'WSB/EiJ', 'OR3199m266']
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']
data[1004] = ['2', 25947335, 'C', 'C', 'C', 'C', 'C', 'C', 'A', 'C', 'C']
```

Viterbi Dynamic Program

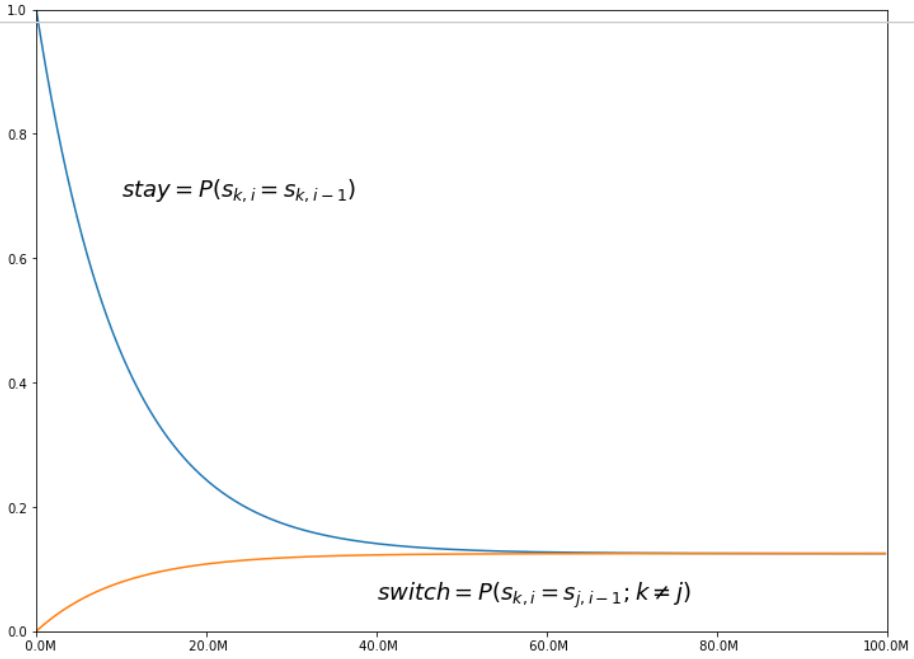
```
from math import exp, log10
```

```
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)*exp(-delta/10000000.0) + 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)
```

```
Length of paths: 8113
```

Transition Probability

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



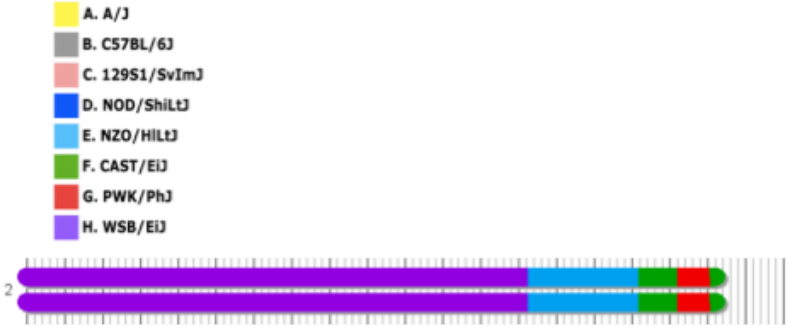
Backtracking

```
# 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")
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()
```

```
5 (2743.7737749674434, 5) CAST/EiJ
```

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

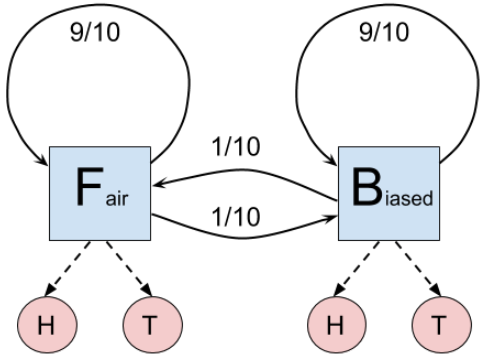
Back to the Casino with new questions

- Are there common aspects of the most likely solutions?
- Which coin was I most likely using on the 4^{th} roll

P	π	P	π	P	π	P	π
0.0058	BBBBBB	0.0001	BBBFFB	0.0000	FFFBFF	0.0000	FBBFBF
0.0046	FFFFFF	0.0001	FFFFBF	0.0000	FFBFBB	0.0000	BFBBFF
0.0013	FBBBBB	0.0001	FFBFFF	0.0000	FBFFBB	0.0000	BFFBBF
0.0012	FFFFBB	0.0001	FBFFFF	0.0000	FBBFFB	0.0000	BBFBFF
0.0009	FFBBBB	0.0001	FFBBBF	0.0000	FFBFFB	0.0000	FFBFBF
0.0008	FFFFFB	0.0001	BFFFBB	0.0000	FBFFFB	0.0000	FBFFBF
0.0006	FFFBBB	0.0001	FBBBFF	0.0000	FBFBBB	0.0000	BFFBFF
0.0006	BBBFFF	0.0001	BBFFFB	0.0000	FBBBFB	0.0000	BFBFBB
0.0004	BBBBBF	0.0000	BFBBBB	0.0000	BBBFBF	0.0000	FBFBBF
0.0004	BBFFFF	0.0000	BBBBFB	0.0000	FFBBFB	0.0000	BFBFFB
0.0003	BBBBFF	0.0000	BBFBBB	0.0000	BBFFBF	0.0000	FBFBFF
0.0003	BFFFFF	0.0000	BFFFFB	0.0000	BFFFBF	0.0000	BFBBFB
0.0001	BBBFBB	0.0000	FFFBBF	0.0000	BFBFFF	0.0000	BBFBFB
0.0001	FBBFFF	0.0000	FFBBFF	0.0000	FFFBFB	0.0000	BFFBFB
0.0001	FBBBBF	0.0000	FBBFBB	0.0000	BFBBBF	0.0000	FBFBFB
0.0001	BBFFBB	0.0000	BFFBBB	0.0000	BBFBBF	0.0000	BFBFBF

Forward-Backward Problem

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



Goal: Find the most probable coin that was in use at a particular flip.

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

Where $P(x, \pi_i = k)$ is the probabilites of all paths in state k at i, and $P(x)$ is the probability of sequence x.

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

Viterbi 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

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

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

- Same as Viterbi, excepts stops at k

Backward Algorithm

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

- 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 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:

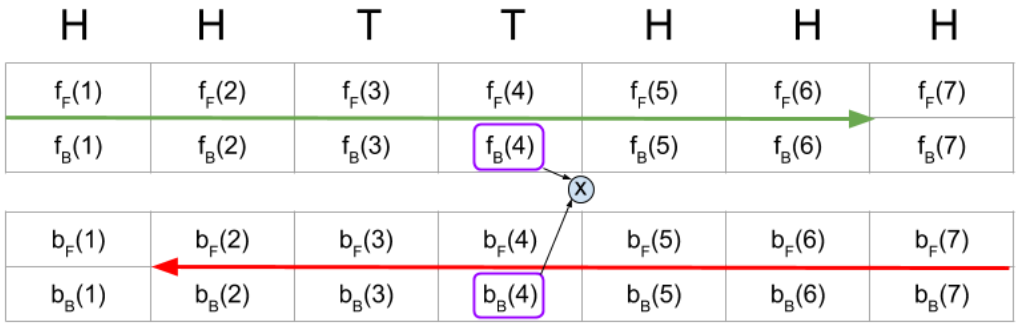
$$b_{k,i} = \sum_{l \in Q} e_l(x_{i+1}) \cdot b_{l,i+1} \cdot A_{k,l}$$

Forward-Backward Algorithm

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

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

- 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



Next Time

Genome Rearrangements