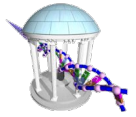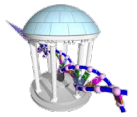# Comp 555 - BioAlgorithms - Spring 2021
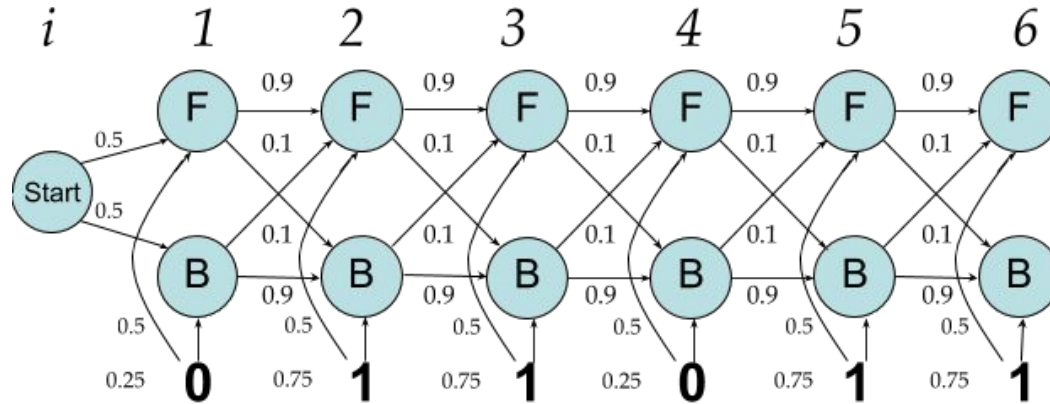
- **Problem Set #4 is due tonight**
- **Problem set #5 is due one week from today**

Inferring Ancestry using HMMs

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.
- Many common probability distributions have an exponential form. Taking their log simplifies these distributions.
- Improves numerical accurracy and stability.

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

Viterbi Decoding Algorithm (cont)

- Every path in the graph has the probability $P(x|\pi)$.
- The Viterbi decoding algorithm finds the path that maximizes $P(x|\pi)$ 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

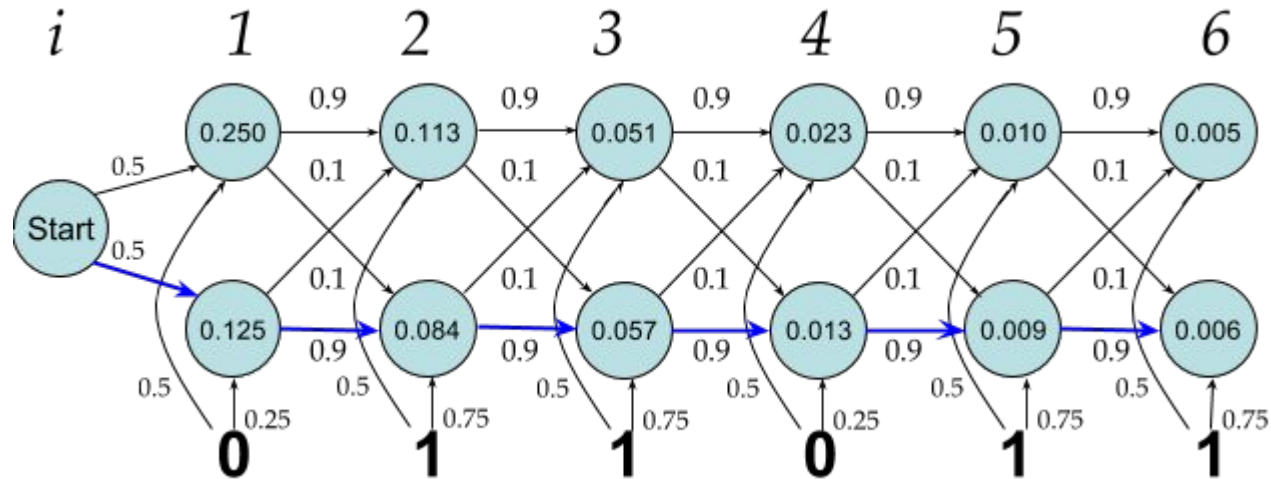Dynamic Program's Recursion:
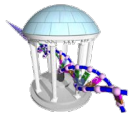
$$s_{l,i+1} = \max_{k \in Q}\{s_{k,i} \cdot \text{weight of edge between } (k, i) \text{ and } (l, i + 1)\}$$

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

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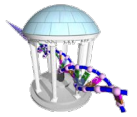
Viterbi Example

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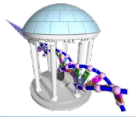
How likely is "most likely?

- The "most likely path" may not be a lot more likely than a 2nd or 3rd most likely paths (even 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

"**FFFFFF**" is nearly as good as "**BBBBBB**"

HMMs in Biology

- Inferring ancestral contributions of a descendant
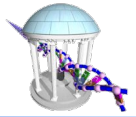- Collaborative Cross project
- Maintained at UNC since 2006

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 Genome

- A randomized breeding scheme was used to

  - Mix the genomes by recombination

  - Fix the genomes by inbreeding

- A breeding funnel - 8 genomes go in a mosaic comes out

- Process was repeated 100s of times to generate independent mosaic lines

- Genotyping was used to track founder contributions

Instead of "Birds and Bees," Mice and Flies

- Recombination mixes the genomes of the two chromosomes
- Sib-mating causes the genomes to fix
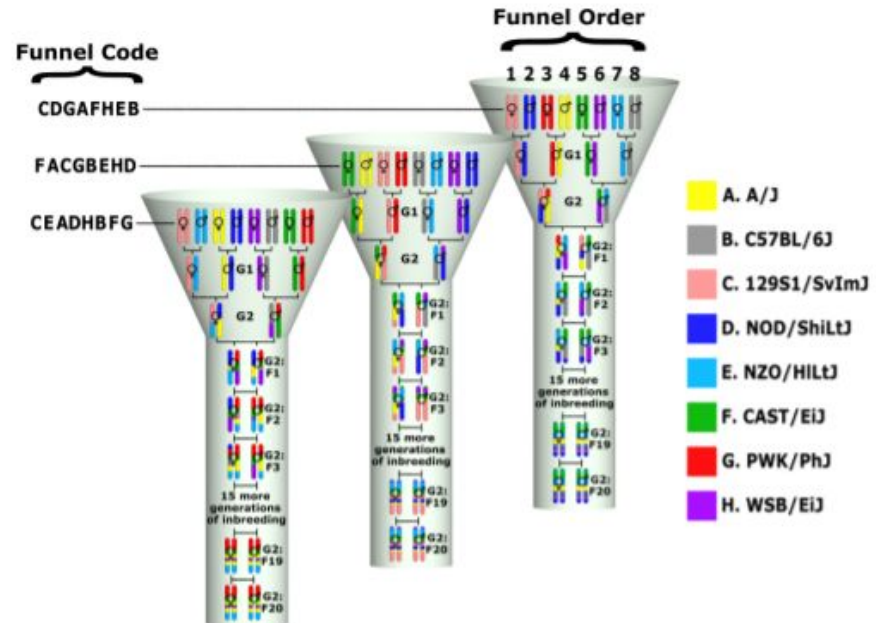
A Genome Mosaic

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

- # Coin Flips are "variants"
- # Hidden State is which "founder"

Mosaic after 7 generations

A. A/J
B. C57BL/6J
C. 129S1/SvImJ
D. NOD/ShiLtJ
E. NZO/HlLtJ
F. CAST/EiJ
G. PWK/PhJ
H. WSB/EiJ
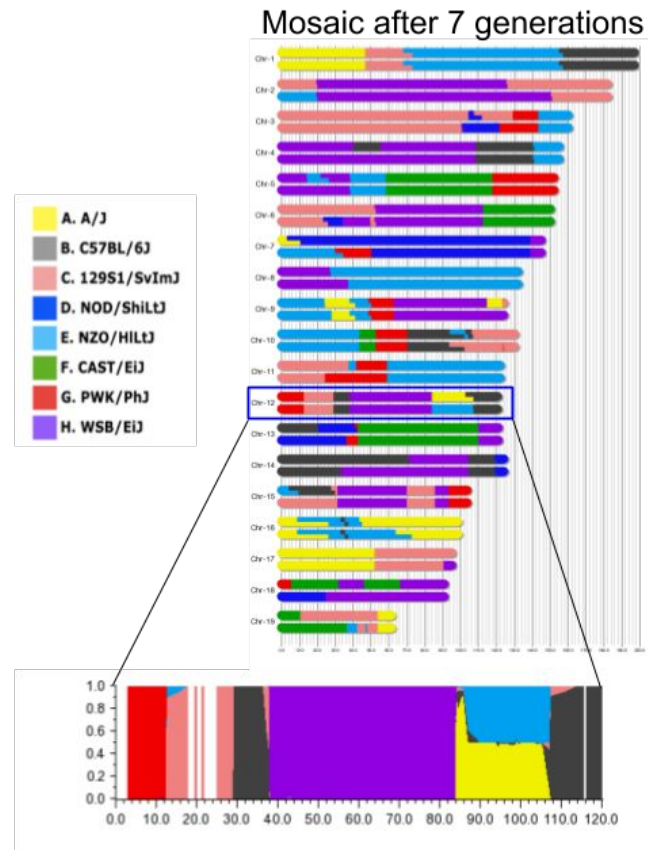
Genotyping Microarrays
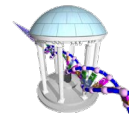
- DNA probes to query the state of specific "known" and "informative" Single Nucleotide Polymorphisms (SNPs)

  Bases in the genome that vary within a population

- Each probe distinguishes 4 cases ("Ref", "Alt", "H", "N")

- From these observations we infer the founder at every marker
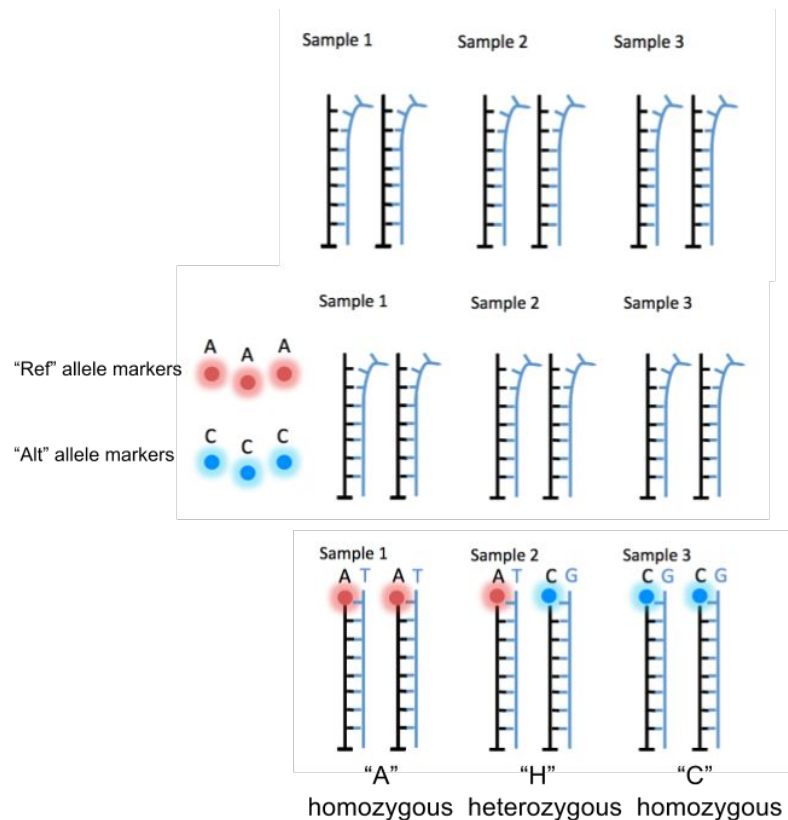
Example Genotypes

- Genotypes for a chromosome
- 1000s of probes with positions of variant
- Alleles are indicated by the nucleotide
- Rarely can a single maker resolve the founder
- Which strain would you guess?
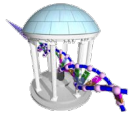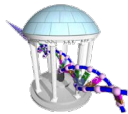
			Probe info			Founder Genotypes						Target
chromosome	positionB38	A/J	C57BL/6J	129S1/SvImJ	NOD/ShiLtJ	NZO/H1LtJ	CAST/EiJ	PWK/PhJ	WSB/EiJ	OR3199m266
2	3176721	G	T	G	T	G	G	T	T	T
2	3180256	G	G	G	G	G	A	A	G	G
2	3182308	A	G	A	G	A	G	G	A	A
2	3183784	T	G	T	G	T	G	G	T	T
2	3233750	G	G	G	G	G	A	G	G	G
2	3350920	A	A	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

Genotype Noise

- One last issue, between 1% and 5% of genotypes are simply wrong
- Source of errors
  - A probe didn't glow bright enough
  - A section of the array was damaged (fingerprints, cracks, hair, etc.)
  - Mess ups when fabricating a probe's sequence
  - DNA itself was contaminated
- Error types:
  - "No" calls (observation is uninformative)
  - A possible, but incorrect call

Reading Genotypes

```
In [1]:  fp = open("CCGenotypes.csv", 'r')
         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 range(100,110):
             print("data[%d] = %s" % (i, data[i]))
```
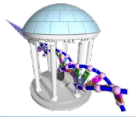
```
['Chromosome', 'Position', 'A/J', 'C57BL/6J', '129S1/SvImJ', 'NOD/ShiLtJ', 'NZO/HlLtJ', 'CAST/EiJ', 'PWK/PhJ', 'WSB/
EiJ', 'CC004/TauUnc']
Number of probes 419
data[100] = ['1', 58523233, 'T', 'T', 'T', 'T', 'C', 'C', 'C', 'T', 'T']
data[101] = ['1', 59995627, 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'C', 'C']
data[102] = ['1', 60400655, 'A', 'A', 'A', 'G', 'G', 'G', 'A', 'G', 'G']
data[103] = ['1', 60761817, 'G', 'G', 'G', 'A', 'A', 'A', 'G', 'G', 'G']
data[104] = ['1', 61312969, 'C', 'C', 'C', 'C', 'C', 'C', 'T', 'C', 'C']
data[105] = ['1', 62719241, 'A', 'G', 'A', 'A', 'A', 'A', 'G', 'A', 'A']
data[106] = ['1', 63003989, 'T', 'T', 'T', 'T', 'T', 'C', 'C', 'C', 'C']
data[107] = ['1', 64378809, 'G', 'A', 'G', 'G', 'G', 'G', 'A', 'G', 'G']
data[108] = ['1', 64700440, 'C', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
data[109] = ['1', 65504911, 'C', 'T', 'C', 'C', 'T', 'C', 'C', 'C', 'C']
```

# Emission Probabilities based on Genotypes

Each probe has its own emission probabilities

```
In [2]: i = int(input("Enter locus [0, %d] to see its Emission probability:" % len(data)))

        print(data[i])
        Nstates = 8
        ErrorRate = 0.05
        # Count expected genotypes
        count = dict([(call, data[i][2:2+Nstates].count(call)) for call in "ACGTHN"])
        print("          ", ', '.join(["%8s" % v[0:8] for v in header[2:2+Nstates]]))
        for base in count.keys():
            # Compute emission probability, assuming 5% error rate
            if (count[base] == 0):
                emission = [1.0/Nstates for j in range(2,2+Nstates)]   # unexpected
            else:
                emission = [(1.0 - ErrorRate)/count[base] if data[i][j] == base else ErrorRate/(Nstates - count[base])
                            for j in range(2,2+Nstates)]
            emission = ["%6.4f" % v for v in emission]
            print("    %s: %2d %s" % (base, count[base], emission))

        Enter locus [0, 419] to see its Emission probability:103
        ['1', 60761817, 'G', 'G', 'G', 'A', 'A', 'A', 'G', 'G', 'G']
                        A/J, C57BL/6J, 129S1/Sv, NOD/ShiL, NZO/HlLt, CAST/EiJ,  PWK/PhJ,  WSB/EiJ
            A:  3 ['0.0100', '0.0100', '0.0100', '0.3167', '0.3167', '0.3167', '0.0100', '0.0100']
            C:  0 ['0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250']
            G:  5 ['0.1900', '0.1900', '0.1900', '0.0167', '0.0167', '0.0167', '0.1900', '0.1900']
            T:  0 ['0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250']
            H:  0 ['0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250']
            N:  0 ['0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250', '0.1250']
```

Transition probabilities
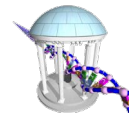
- Recombination likelihood is modeled using an exponential distribution

- Recombinations between nearby probes are unlikely

- Distant probes are more likely to be from different founders

```
In [8]:  %matplotlib inline
         import numpy
         import matplotlib.pyplot as plot

         fig = plot.figure(figsize = (8,6))
         axes = fig.add_subplot(111)

         Nstates = 8
         scale = 10000000.0
         x = numpy.arange(0,100000000.0,200000.0)
         stay = ((Nstates - 1.0) * numpy.exp(-x/scale) + 1.0) / Nstates
         switch = (1.0 - stay) / (Nstates - 1.0)

         plot.plot(x, stay, x, switch)
         plot.text(10000000, 0.7, r'$stay = P(s_{k,i} = s_{k,i-1})$', size="18")
         plot.text(40000000, 0.05, r'$switch = P(s_{k,i} = s_{j,i-1}; k \neq j)$', size="18")
         plot.xlim((0,100000000.0))
         plot.ylim((0,1.0))
         pos, labels = plot.xticks()
         result = plot.xticks(pos, ["%5.1fM" % (p/1000000) for p in pos])
```


$$stay = P(s_{k,i} = s_{k,i-1})$$

$$switch = P(s_{k,i} = s_{j,i-1}; k \neq j)$$
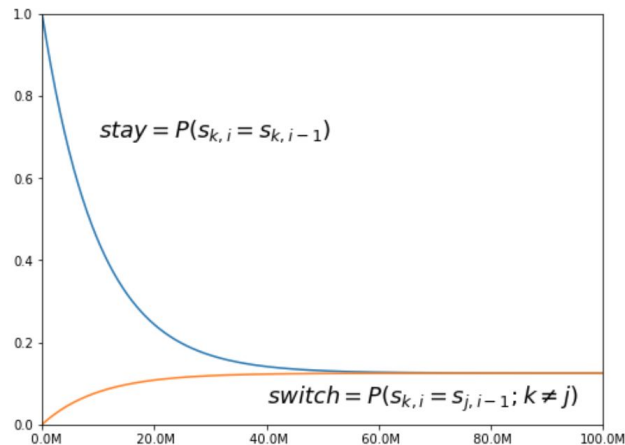
Viterbi Algorithm as a Dynamic Program

```
In [18]:  from math import exp, log10

          Nstates = 8
          prevpos = 1
          state = [[(float(len(data)),i) for i in range(Nstates)]]   # (log(p), PathToHere)
          for i in range(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/(Nstates - count[data[i][j]])
                              for j in range(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 range(Nstates):
                  choices = [(log10(emission[j])+(log10(stay) if (k==j) else log10(switch))+state[-1][k][0],k)
                              for k in range(Nstates)]
                  path.append(max(choices))   # choices is a list of tuples of (score[i], from_whence_I_arrived[i])
              state.append(path)
          print("Length of paths:", len(state))

          Length of paths: 418
```

Backtrack to find solution
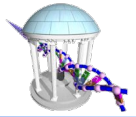
```python
# backtrack
path = state[-1]
maxi = 0
maxp = path[0][0]
for i in range(1,Nstates):
    if (path[i][0] > maxp):
        maxp = path[i][0]
        maxi = i
print(maxi, path[maxi], header[2+maxi])

for j in range(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')
prev = ''
for row in data:
    line = ','.join([str(v) for v in row])
    fp.write(line+'\n')
    if (row[-1] != prev):
        print(line)
        prev = row[-1]
print(line)
fp.close()
```

```
5 (129.58171061177885, 5) CAST/EiJ
1,3409090,C,C,A,A,C,A,A,A,A,PWK/PhJ,PWK/PhJ
1,14334166,A,G,A,A,A,G,G,G,A,129S1/SvImJ,129S1/SvImJ
1,41477940,G,A,A,A,A,G,G,G,G,A/J,A/J
1,52869070,G,G,G,A,A,G,G,A,A,WSB/EiJ,WSB/EiJ
1,67749123,A,G,A,A,A,G,G,G,A,A/J,A/J
1,132786434,C,C,C,C,C,T,C,C,C,C57BL/6J,C57BL/6J
1,172685919,A,G,A,A,A,G,G,G,A,A/J,A/J
1,176074355,A,G,G,G,A,G,A,A,G,CAST/EiJ,CAST/EiJ
1,194886567,G,G,T,G,T,T,G,T,T,CAST/EiJ,CAST/EiJ
```

A peek at the result

```
In [23]: !head result.csv; echo '...'; tail result.csv
```

```
Chromosome,Position,A/J,C57BL/6J,129S1/SvImJ,NOD/ShiLtJ,NZO/HlLtJ,CAST/EiJ,PWK/PhJ,WSB/EiJ,CC004/TauUnc,Founder
1,3409090,C,C,A,A,C,A,A,A,A,PWK/PhJ
1,3427467,A,A,A,A,A,G,G,A,G,PWK/PhJ
1,3439034,C,C,T,T,C,C,C,T,C,PWK/PhJ
1,3668628,A,G,G,G,G,A,A,G,A,PWK/PhJ
1,4504223,G,G,G,G,G,A,G,A,G,PWK/PhJ
1,4744395,T,T,T,T,T,G,T,G,T,PWK/PhJ
1,5069641,A,A,A,A,A,G,A,A,A,PWK/PhJ
1,5149169,T,G,T,G,T,G,T,T,T,PWK/PhJ
1,7698048,A,G,A,A,A,G,G,G,G,PWK/PhJ
...
1,193654902,G,A,A,G,A,G,G,G,G,CAST/EiJ
1,193673297,G,A,A,G,A,G,G,G,G,CAST/EiJ
1,193688845,A,C,C,A,C,C,A,A,C,CAST/EiJ
1,193709621,G,A,A,A,A,A,G,G,A,CAST/EiJ
1,193732571,T,C,C,C,C,C,T,C,C,CAST/EiJ
1,193928056,A,G,G,A,G,A,A,A,A,CAST/EiJ
1,194000258,C,C,C,T,C,C,C,T,C,CAST/EiJ
1,194149219,G,A,A,G,G,G,G,G,G,CAST/EiJ
1,194625219,C,T,T,T,C,T,T,C,T,CAST/EiJ
1,194886567,G,G,T,G,T,T,G,T,T,CAST/EiJ
```
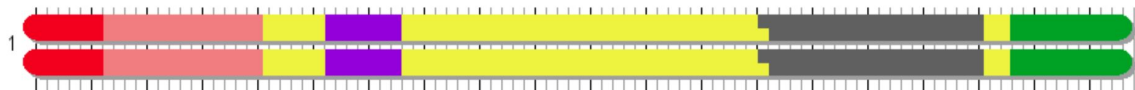

Legend:
- A. A/J
- B. C57BL/6J
- C. 129S1/SvImJ
- D. NOD/ShiLtJ
- E. NZO/HlLtJ
- F. CAST/EiJ
- G. PWK/PhJ
- H. WSB/EiJ

- The inferred Mosaic
- Repeat for every chromosome
- Most likely, but how likely?
- Other approaches
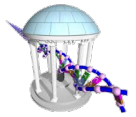
Back to the Casino with new questions

- Are there common aspects of the most likely solutions?
- Which coin was I most likely using on the 4th 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
```
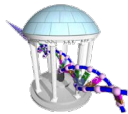
Forward-Backward Problem

Given: A sequence of coin tosses generated by an HMM.
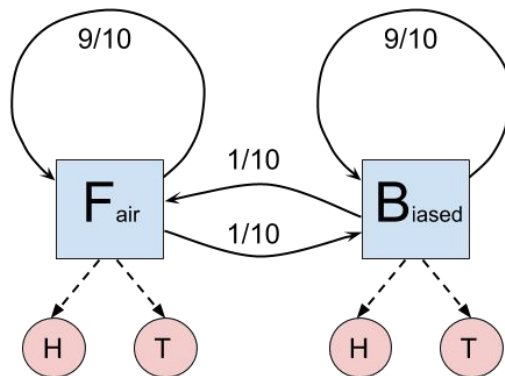
Goal: Find the most probable coin that was in use at a particular flip.

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

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

Illustrating the difference using 4 flips

Not a lot worse than the best solution

X = [T,H,H,H]

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)

Viterbi solution, the most likely sequence states.

FFBB (0.0057)
BFBB (0.0003)
FBBB (0.0085)
BBBB (0.0384)
P(x) = 0.0877

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 $x_1...x_i$ and reaching the state $\pi_i = k$.
- The recurrence for the forward algorithm is:

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

- Similar to Viterbi solution to *i*, except *all paths are multiplied together* rather than taking the Max
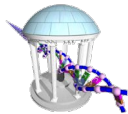
Backward Algorithm

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 probability* $b_{k,i}$ ≡ the probability of being in state $\pi_i=k$ and emitting the suffix $x_{i+1}... x_n$.
- 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

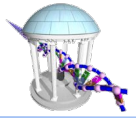- The probability that the dealer used a biased coin at any moment *i* is as follows:

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

- 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

H	H	T	T	H	H	H
$f_F(1)$	$f_F(2)$	$f_F(3)$	$f_F(4)$	$f_F(5)$	$f_F(6)$	$f_F(7)$
$f_B(1)$	$f_B(2)$	$f_B(3)$	$f_B(4)$	$f_B(5)$	$f_B(6)$	$f_B(7)$

$b_F(1)$	$b_F(2)$	$b_F(3)$	$b_F(4)$	$b_F(5)$	$b_F(6)$	$b_F(7)$
$b_B(1)$	$b_B(2)$	$b_B(3)$	$b_B(4)$	$b_B(5)$	$b_B(6)$	$b_B(7)$

Next Time

Genome Rearrangements

Nature Reviews | Molecular Cell Biology