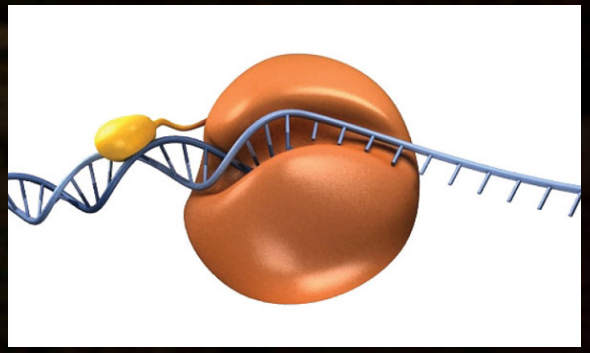
Bio/CS 123B
Spring 2021
Identification and Hidden Markov Models



Ok, so you've sequenced some DNA

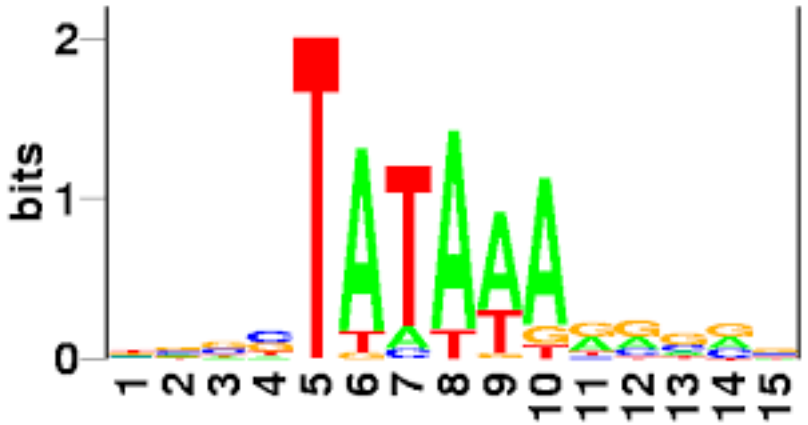
- Now you have a nucleotide string
- Or maybe an amino acid string
- But what ***is*** it?
  - What species?
    - If no known species, what genus?
    - If no known genus, what family?
    - Etc?
  - What gene?
    - If not known, can we guess its function?

Tools for identifying sequences

- BLASTn / BLASTp — *123A*
- Position Weight Matrices (PWMs) — *123A, Review here*
- Hidden Markov Models (HMMs) — *123B*

PWMs identify *Motifs*

- Short recurring sequence patterns
- Believed to have biological function
- In chromosomes, motif function often is binding site e.g. for promoters
- TATA box: helps RNA polymerase find the transcription start site (TSS) in many eukaryotic genes.



A few TATA box sequences

- TATAAAAA
- TATATAAA
- TATAAATA
- TATATATA

Sequence Motifs

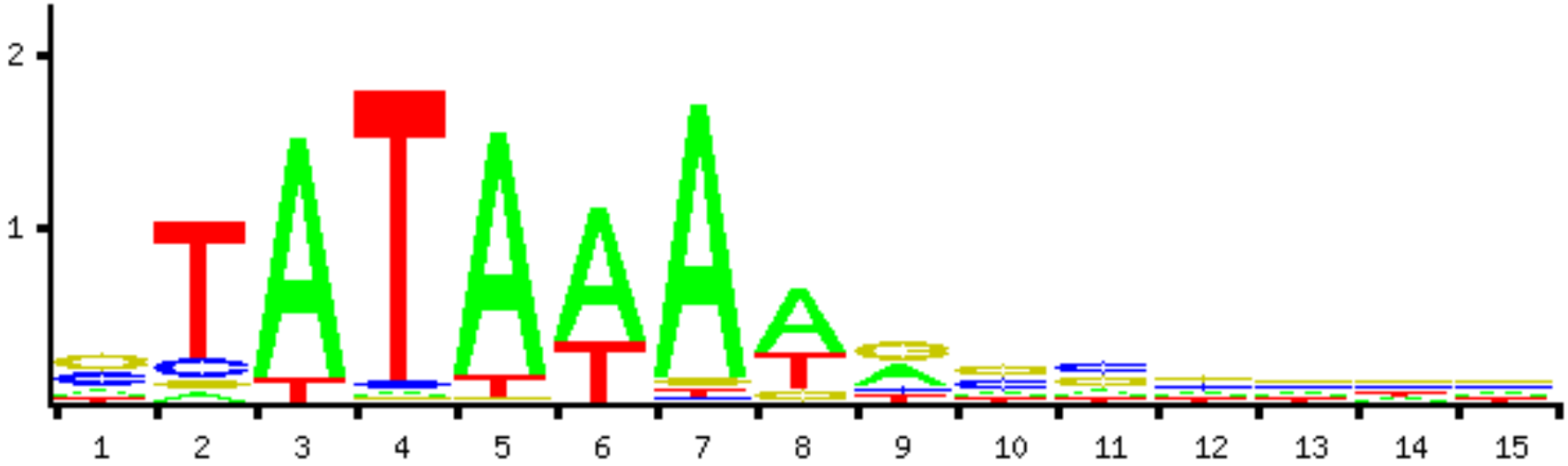


Table MM2.1 Nucleotide frequencies in 389 known TATA boxes. (*Homo sapiens*, I think)

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	61	16	352	3	354	268	360	222	155	56	83	82	82	68	77
C	145	46	0	10	0	0	3	2	44	135	147	127	118	107	101
G	152	18	2	2	5	0	10	44	157	150	128	128	128	139	140
T	31	309	35	374	30	121	6	121	33	48	31	52	61	75	71

Creating Frequency Tables

- The frequency of an A in the first position = 61/389 = 0.1568
- The frequency of a T in the second position = 309/389 = 0.7943

Table MM2.1 Nucleotide frequencies in 389 known TATA boxes.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	61	16	352	3	354	268	360	222	155	56	83	82	82	68	77
C	145	46	0	10	0	0	3	2	44	135	147	127	118	107	101
G	152	18	2	2	5	0	10	44	157	150	128	128	128	139	140
T	31	309	35	374	30	121	6	121	33	48	31	52	61	75	71

Improving on frequency tables: log-odds tables

- In previous table, Prob(A in 1st column) = 61%
- Is that high or low for this organism?
- Define odds(A in 1st column):

$$\frac{\text{Prob(A in 1st column)}}{\text{Frequency of A in this organism}}$$

- Define log-odds of A in 1st column = log2(odds of A in 1st column)
- Log-odds > 0 → base is unusually frequent
- Log-odds < 0 → base is unusually rare

Position-Weight Matrix (PWM): Frequency Table Converted to Log-Odds

A	−0.84	−2.77	1.69	−5.18	1.70	1.30	1.76	1.03	0.51
C	0.76	−0.90	−99.00	−3.10	−99.00	−99.00	−4.80	−5.42	−0.96
G	0.83	−2.25	−5.42	−5.42	−4.10	−99.00	−3.06	−0.96	0.88
T	−1.81	1.50	−1.64	1.78	−1.86	0.15	−4.14	0.15	−1.72

Frequency(C) in col 3 = 0

Log2(0) = negative infinity

Simplify … use -99

Log2($1.5*10^{-30}$) = -99

Same for C in col 5 and 6

- A sneaky workaround
- No better alternative with PWMs
- Watch for a *much* better alternative with HMMs

When you observe zero occurrences of a base in a column

- You saw zero Cs in column 3
- What does that mean?
- Maybe TATAbox never has a C in column 3:
  - ➔ C in column 3 means “this sequence can’t possibly be a TATAbox”.
- Maybe TATAbox infrequently has a C in column 3:
  - ➔ C in column 3 means “there’s a low but non-zero probability that this sequence is a TATAbox”.
- How should you decide?

Using the PWM to score a sequence

A	−0.84	−2.77	1.69	−5.18	1.70	1.30	1.76	1.03	0.51
C	0.76	−0.90	−99.00	−3.10	−99.00	−99.00	−4.80	−5.42	−0.96
G	0.83	−2.25	−5.42	−5.42	−4.10	−99.00	−3.06	−0.96	0.88
T	−1.81	1.50	−1.64	1.78	−1.86	0.15	−4.14	0.15	−1.72

- Example: A C A T A T A T A
- For each col, note log-odds of base in sequence
- Add up those log-odds
- → Score, which you compare to some threshold
- Here score = 6.00, thresh = 5.25, so with very high probability, ACATATATA is a TATAbox <u>in this organism</u>

Shortcomings of identification by PWM

- Definition of odds is based on background frequencies of A/C/G/T in some organism → can't have one matrix that works for multiple species.
- Can only classify sequences of exactly the right length.
- The zero-frequency trick was a bit shady (setting log-odds = -99 for a base that is never observed).

Hidden Markov Models (HMMs)

- Developed in late 1950s and 1960s
- 1st practical application: speech recognition
- Late 1980s: application to DNA and protein seqs

Speech Recognition Example … Imagine it's 1975

The voice on the line:

"For customer service, press 1"

Me: (presses) 1

The voice on the line:

"For customer service, press 1"

Gramma:

"I can't … umm … there's no … hello?"

And that's why the voice on the line says…

"For customer service, press OR SAY 1"

So the industry needed speech recognition software

The Challenge

- When a customer says “1”, the response is received as a sound waveform
  - Time-series of numbers
  - Air-pressure measurements, ~1 per msec
- Causes of variation:
  - Background noise in sample
  - Regional accents
  - Atmospheric conditions (temp, pressure, humidity)
  - Cold/allergies/sniffles
  - Stroke
  - Very recent use of peanut butter

The Approach

- For each sound in “zero”, “one”… “nine”
  - Collect lots of sound samples of different people under different conditions saying each number (*training sets*)
  - Use the training sets to construct 10 rigorous mathematical/statistical descriptions (*models*), 1 for each digit
  - The models assign a score to any input (digitized waveform)
- Deployment
  - Capture waveform from phone customer
  - All models evaluate waveform
  - 1 clear best-score winner ➔ success

By Analogy … DNA and protein sequences are:

- Ordered series
- Need to be identified
- Variation at all levels
  - Within an individual
  - Among individuals in a population
  - Among populations
  - Sequencing errors

Add codes

- CS: DelaCruz, Kang, Orteza
- BIOL: Rapuru, Tompkins, Torres

Pre-Test

- E-value: For query of length=L, database D, score = S. E-value = expected number of hits, with a random query of length=L, against D, with score >= S.

Some answers…1 of these is excellent

- It's the probability of getting that match by chance
- e = ld2^-S (I don't know, this is from Google)
- The E-value tells us how much S of nucleotide query of length L is similar to the Database D, is due to chance
- The E-value defines whether the query E found in D was by chance or not
- E-value is the S from similarity based on length when we are searching for D
- I don't know

HMMs are Extensions of Markov Chains

- Developed by Andrei Markov 1856-1922
- 1st publication on Markov Chains: 1906
- States and probabilities
- States
  - Usually drawn as bubbles or boxes
  - Optional START and STOP states
- Markov Process = a path from one state to another, then another, etc.
- Probabilities:
  - Initial probabilities: Prob that a process/path will start in any particular state
  - Transition probabilities: Given the process is currently in a state X, Prob that next state will be Y (for all X and Y)

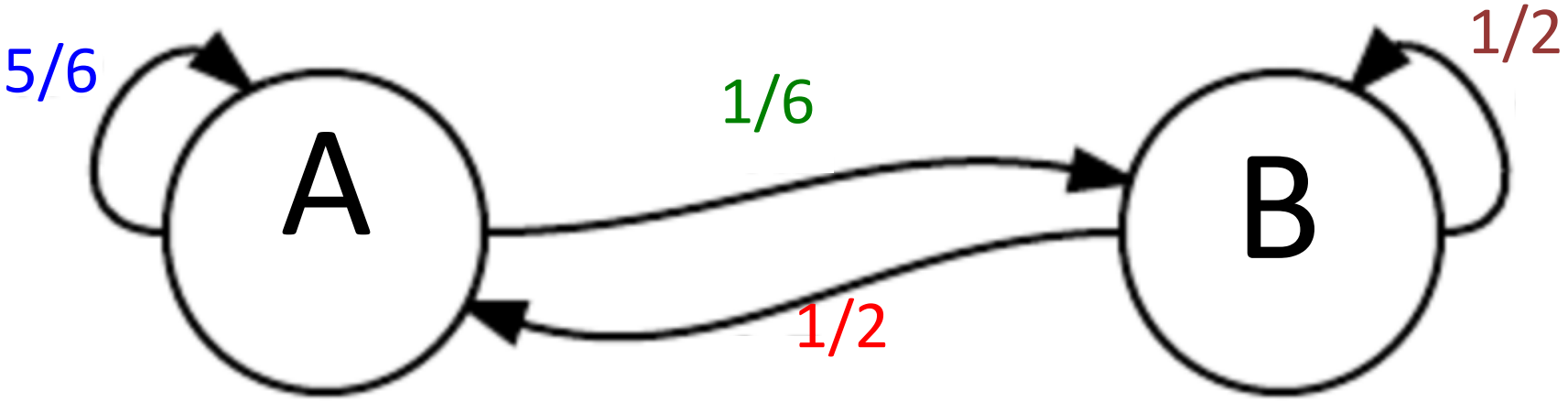
Markov Chain: States

A and B are arbitrary names. They might represent actual physical states of parts of the universe. E.g. day-to-day presence (A) or absence (B) of sunspots.

Markov Chain: Initial Probabilities

- P(Start in A) = P(Start in B) = 50% (common)
- Or maybe P(Start in A) = 30%, P(Start in B) = 70%
- Initial probabilities must sum to 1 because the process must start in exactly 1 state
- Initial probabilities are hard to draw
  - None shown → assume equiprobable

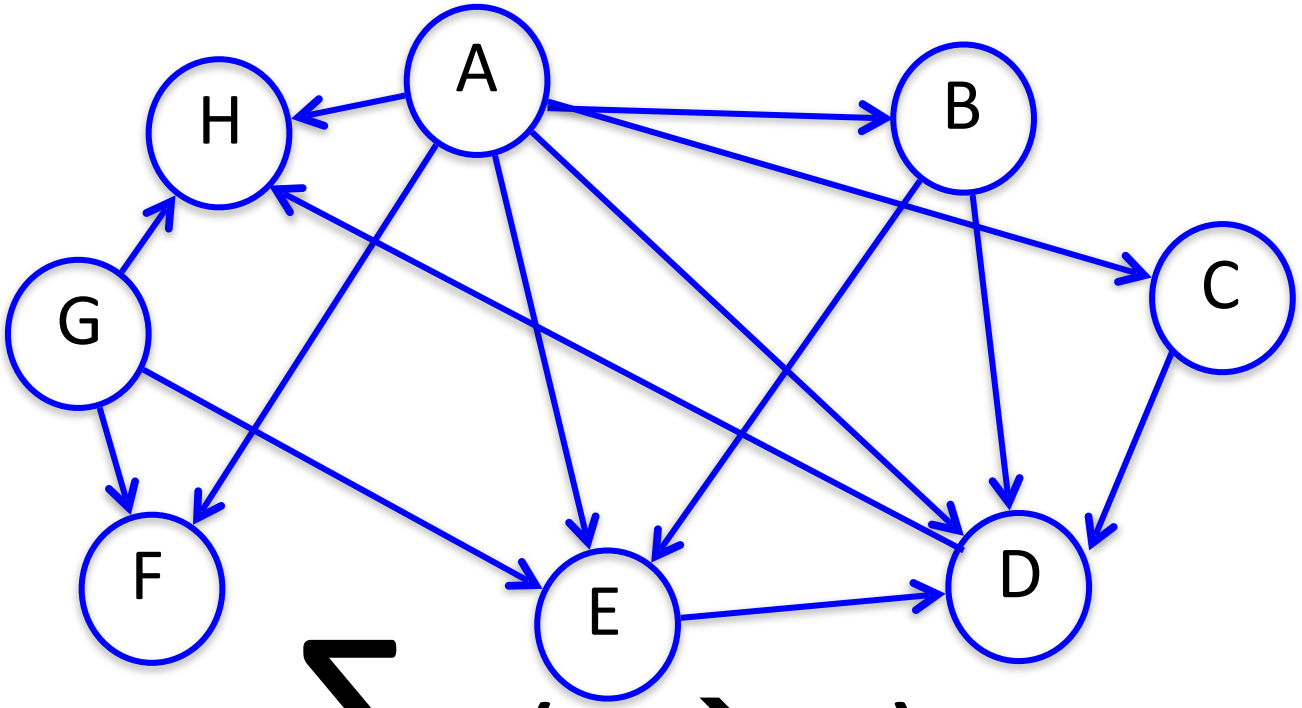
Markov Chain: Transition Probabilities



- P(A→ A) = 5/6 P(A→B) = 1/6
- P(B→ A) = 1/2 P(B→B) = 1/2

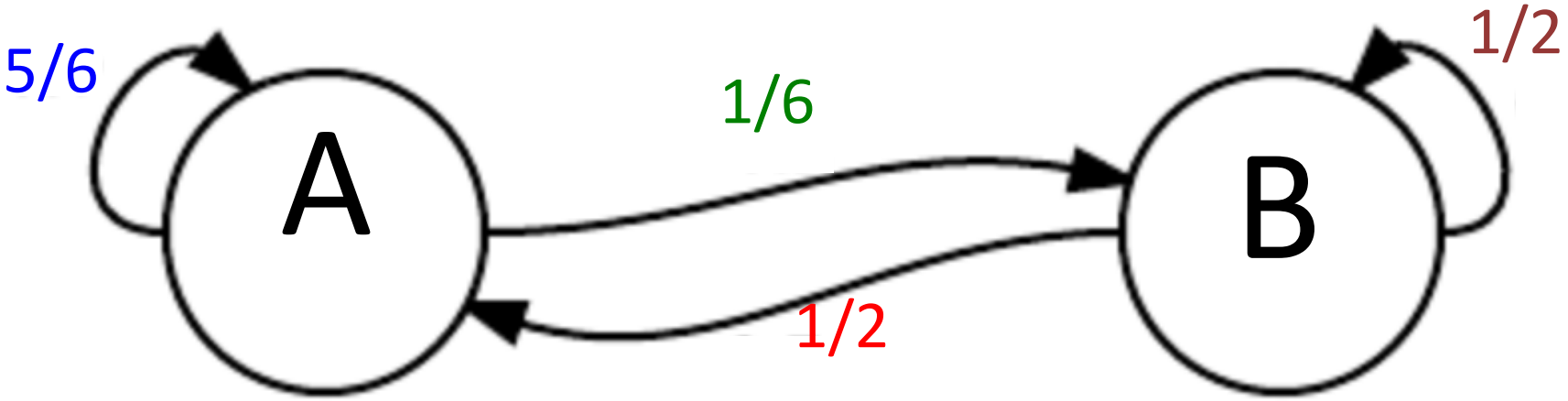
$$\sum_{\text{st in all states}} P(A \rightarrow st) = \sum_{\text{st in all states}} P(B \rightarrow st) = 1$$

Sum of exit probabilities from any state is *always* 1



$$\sum_{\text{st in all states}} P(A \rightarrow st) = 1$$

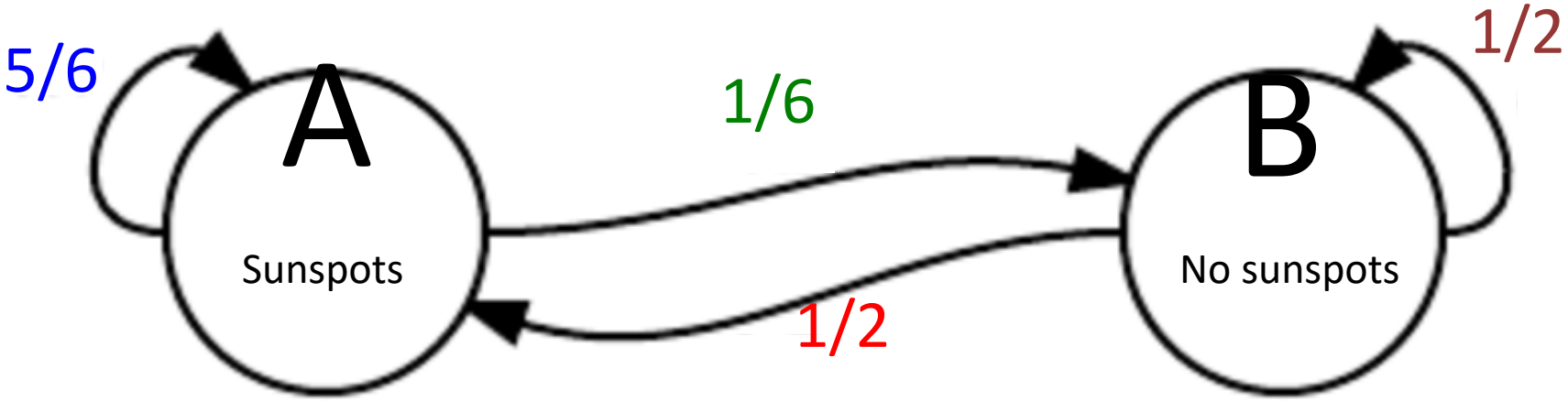
Markov Chain: Let's Play



Probabilities

- "And then..." means multiply
- P(***this*** and then ***that***) = P(***this***) * P(***that***)
- P(***roll 3 on a d6*** and then ***heads on a coin flip***) = **1/6** * **½**
- P(some state path through a Markov chain) = P(start in 1st state) * P(1st state → 2nd state) * P(2nd state → 3rd state) …

Suppose it's a sunspot model



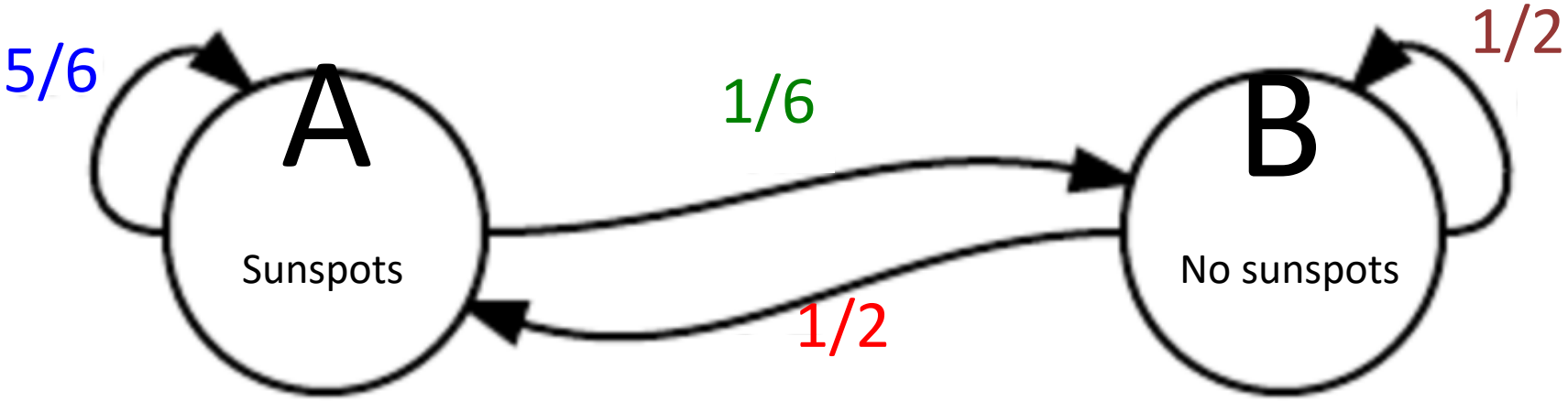
What is prob that a path generated by this model will be AAAAAA? (6 consecutive sunspot days?)

P(Start in A) * P(A→A) * P(A→A) * P(A→A) * P(A→A) * P(A→A)

1/2 5/6 5/6 5/6 5/6 5/6

= .201

Suppose it's a sunspot model



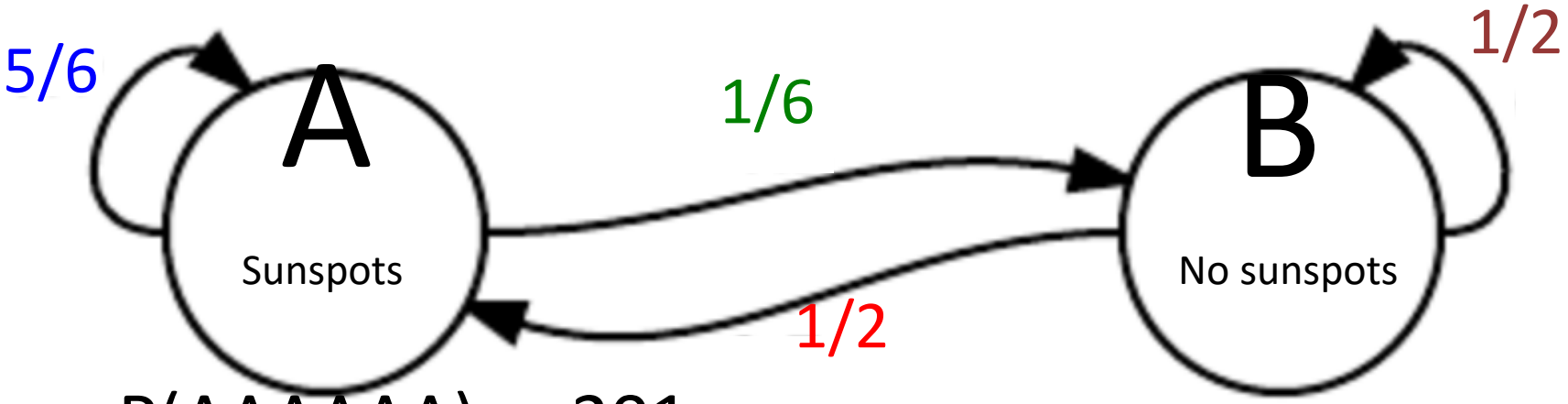
What is prob that a path generated by this model will be BBBBBB? (6 consecutive no-sunspot days?)

P(Start in B) * P(B→B) * P(B→B) * P(B→B) * P(B→B) * P(B→B)

1/2 1/2 1/2 1/2 1/2 1/2

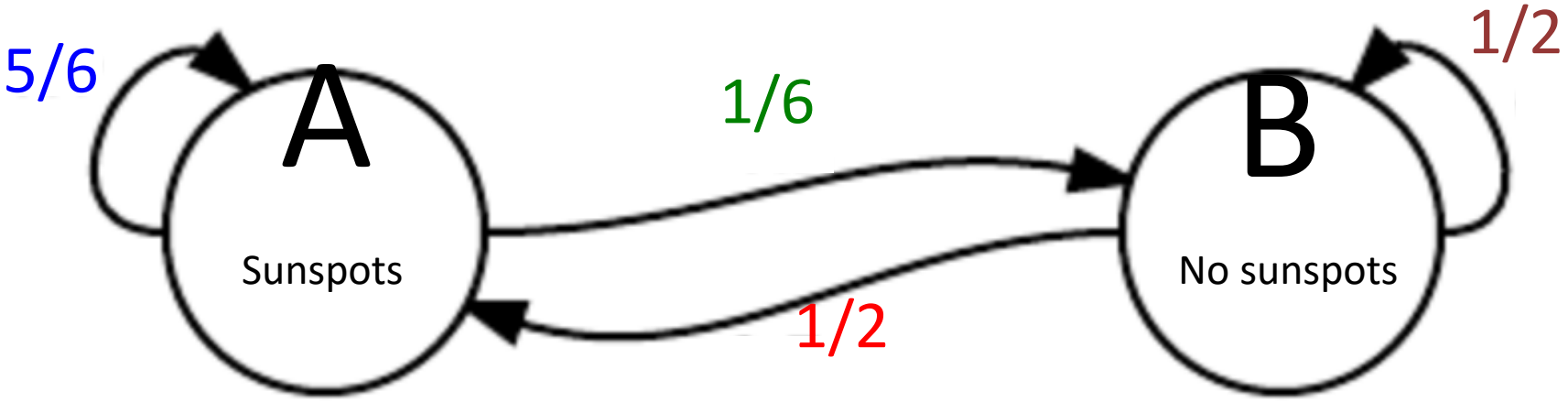
= .016

Probability of generation by the model



- P(AAAAAA) = .201
- P(BBBBBB) = .016
- Individual probabilities have no intuitive meaning
- Probabilities are meaningful in comparison: AAAAAA is 12x more probable than BBBBBB

Your turn



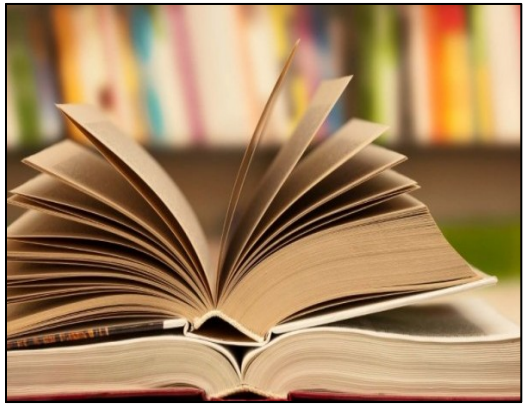
- Compute P(our sequence)
- Write it on the board
- We'll start again in 10 minutes

Competing Statistical Models

- E.g. Markov chains and (soon) HMMs
- Attempts to explain hidden aspects of reality
- Competing models can be evaluated based on relative probability of observation
- Example: *The #1 Students' Detective Agency*
  - Are you worried about how your son/daughter is doing in college?
  - Pay us, we can help!

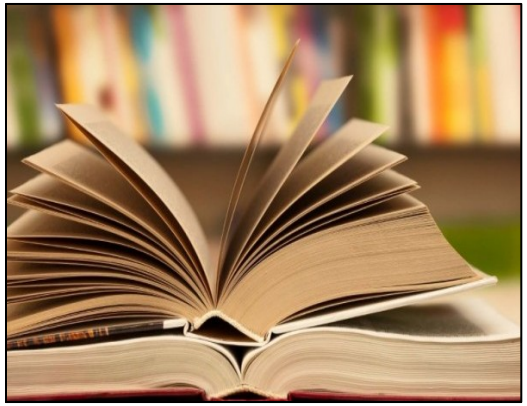
How *The #1 Students' Detective Agency* views college life

- Students are either successful or distressed
- Our business model: we follow students around and decide which they are
- When not attending classes, eating, or sleeping, students are in 1 of 3 places



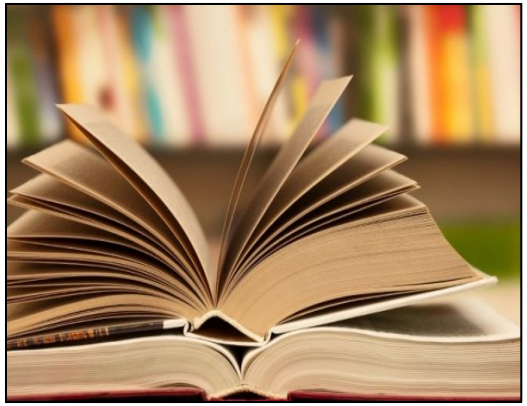
How *The #1 Students' Detective Agency* views college life

- Every waking hour outside of class, every student makes a decision
  - Stay where they are
  - Go to one of the other 2 options

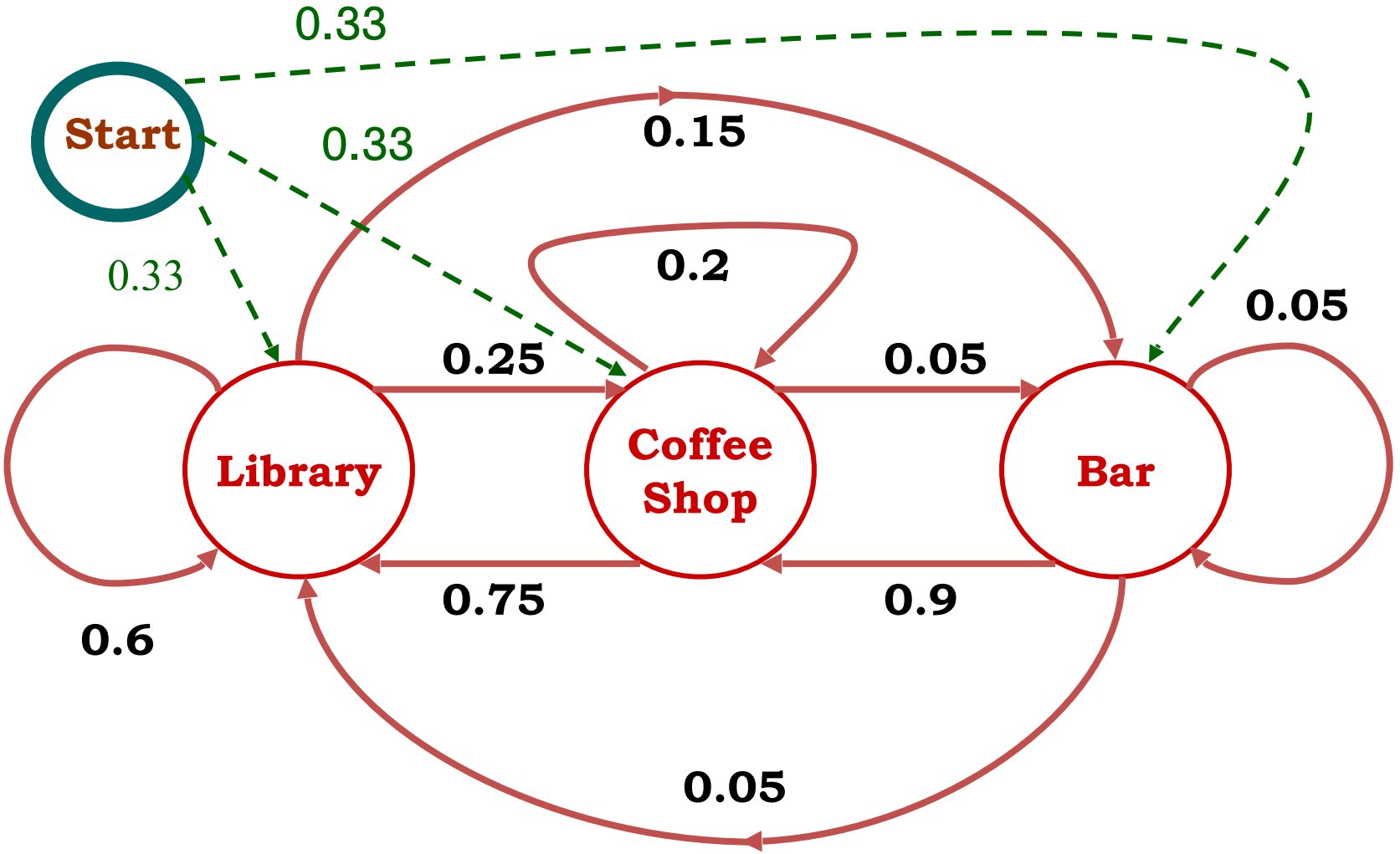


The #1 Students' Detective Agency's Technical Secret Sauce

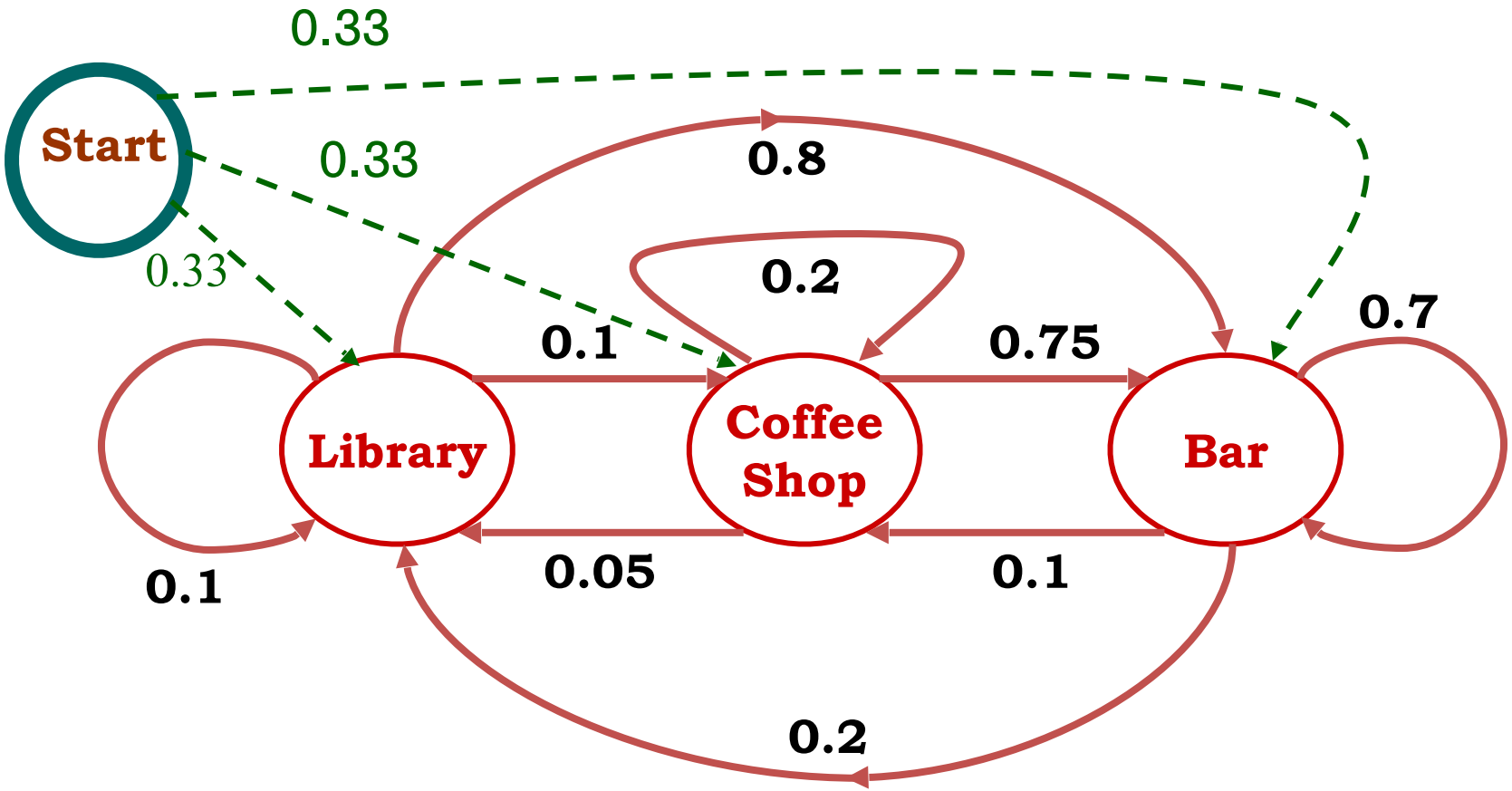
- Success & distress are represented by Markov Chains
- If *Success* model explains a student's behavior better than *Distress* model, classify the student as Successful, and vice-versa



How *The #1 Students' Detective Agency* models a successful student



How *The #1 Students' Detective Agency* models a distressed student



Client #1:
LLLLLLLLLLLL (L = Library)

- Trivial case
- Can't possibly study any more than this
- P(LLLLLLLLLLLL | successful) =

$$1/3 \; * \; \underbrace{.6 * .6 * .6 \ldots}_{\text{11 times}}$$

$$= 1/3 * .6^{11} = 9.2 * 10^{-9}$$

Client #1:
LLLLLLLLLLLL (L = Library)

- Trivial case
- Can't possibly study any more than this
- P(LLLLLLLLLLLL | distressed) =

$$1/3 \; * \; \underbrace{.1 * .1 * .1 \ldots}_{\text{11 times}}$$

$$= 1/3 * 10^{-11} = 3.3*10^{-12}$$

Client #1 according to alternative models

- P(LLLLLLLLLLL | successful) ~= 9E-9
  - Is this a lot or a little?
  - ***Individual probabilities have no intuitive meaning***
  - They are meaningful when compared to other probabilities
- P(LLLLLLLLLLL | distressed) ~= 3E-12
- P(…|successful) is 3300x greater than P(…| distressed)
- A successful student is 3300x more likely than a distressed student to be at the library 12 consecutive times
- A student who spends 12 consecutive turns in the library is 3300x more likely to be successful than to be distressed

Could Client #1 actually be distressed?

- Yes
- P(LLLLLLLLLLL|distressed) is > 0 therefore it *might* happen
- If we always predict that LLLLLLLLLLL means successful, we can expect to be wrong once every 3300 times
- Is that acceptable risk for a business? For an airline? For a spacecraft? For a pathologist? For a judge in a capital murder case?
  - Not bioinformatics questions
  - Whatever job you get, including bioinformatics, you *might* have to make the call

Client #2: BBBBBBBBBBBB (B = Bar)

- Also a trivial case
- Can't possibly study worse than this
- P(B → B | successful) = .05
- P(BBBBBBBBBBBB| successful) =

  1/3 * .05^11 = 1.6E-15
- P(B → B | distressed) = .7
- P(BBBBBBBBBBBB| distressed) =

  1/3 * .7^11 = 6.6E-3

Client #2: BBBBBBBBBBBB (B = Bar)

- Also a trivial case
- Can't possibly study any less than this
- P(B → B | *Is this a lot or a little?*
- P(BBBBBBBBBBBB| successful) =
  1/3 * .05^11 = 1.6E-15
- P(B → B | distressed) = .7
- P(BBBBBBBBBBBB| distressed) =
  1/3 * .7^11 = 6.6E-3

Client #2: BBBBBBBBBBBB

- P(BBBBBBBBBBBB| successful) = 1.6E-15
- P(BBBBBBBBBBBB| distressed) = 6.6E-3
- P(…|distressed) is 200 billion x greater than P(…|successful)
- A distressed student is 200 billion times more likely than a successful student to be at the bar 12 consecutive times

Client #3: LLLCBCLLBBLL

- Before we analyze, let's think
- Lots of L → maybe successful
- 25% of time in the bar → maybe not
- Ok, let's analyze
- We need a compact picture of all the transition probabilities


Successful
Library
Café
Bar
Library
6
.25
.15
Café
75
.2
.05
Bar
05
.9
.05
Next State
Previous State
Distressed
Library
Café
Bar
Library
1
.1
.8
Café
05
.2
.75
Bar
2
.1
.7

Client #3: LLLCBCLLBBLL | successful

Successful	L	C	B
L	.6	.25	.15
C	.75	.2	.05
B	.05	.9	.05

- P(… | successful) = 1/3 * P(L→L) * P(L→L) * P(L→C) * P(C→B) * P(B→C) * P(C→L) * P(L→L) * P(L→B) * P(B→B) * P(B→L) * P(L→L)
- = 1/3 * .6 * .6 * .25 * .05 * .9 * .75 * .6 * .15 * .05 * .05 * .6 * P(L→L) * P(L→B) * P(B→B) * P(B→L) * P(L→L)
- = 1.4E-7

Client #3: LLLCBCLLBBLL | distressed

Distressed	L	C	B
L	.1	.1	.8
C	.05	.2	.75
B	.2	.1	.7

- P(… | distressed) = 1/3 * P(L→L) * P(L→L) * P(L→C) * P(C→B) * P(B→C) * P(C→L) * P(L→L) * P(L→B) * P(B→B) * P(B→L) * P(L→L)
- = 1/3 * .1 * .1 * .1 * .1 * .8 * .75 * .7 * .05 * .2 * .1 * P(L→L) * P(L→B) * P(B→B) * P(B→L) * P(L→L)
- = 1.4E-9

Client #3: LLLCBCLLBBLL

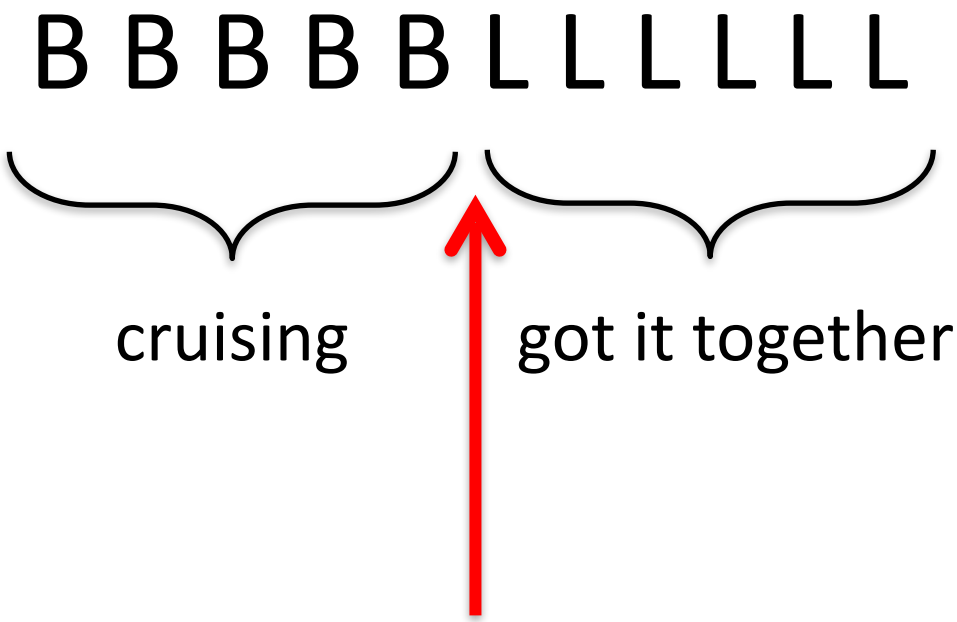
- P(… | successful) = 1.4E-7
- P(… | distressed) = 1.4E-9
- This student is 100x more likely to be successful than distressed

Client #4: BBBBBBLLLLLL

Successful	L	C	B
L	.6	.25	.15
C	.75	.2	.05
B	.05	.9	.05

Distressed	L	C	B
L	.1	.1	.8
C	.05	.2	.75
B	.2	.1	.7

- P(… | successful) = 1/3 * P(B→B)^5 * P(B→L) * P(L→L)^5 = 1/3 * .05^5 * .05 * .6^5 = 4e-10
- P(… | distressed) = 1/3 * P(B→B)^5 * P(B→L) * P(L→L)^5 = 1/3 * .7^5 * .1 * .1^5 = 5e-8
- This student is 125x more likely to be distressed than successful … does that feel right?



- This student might not get an A
- But as their professor, I'm no longer worried
- The Markov Chain can't represent attitude change

Limitations of Markov Chains

- Great for modeling certain kinds of process
- But not student behavior
- "Successful" and "Distressed" are not permanent ways of being
  - Haven't you known superstars who didn't live up to their potential?
  - Haven't you known underachievers who started achieving?
- "Successful" and "Distressed" are changeable moods/states/temperaments, each with its own characteristic library/café/bar patterns
- We need something richer than Markov Chains

Hidden Markov Models

Jack Kirby

- Many similarities to Markov Chains:
  - States
  - Initial probabilities
  - Transition probabilities
- The big difference
  - States are separate from observations
  - With the student Markov Chains, states were Library/Café/Bar, because we can observe students in those places
  - With HMMs, states generate observations according to random ("stochastic") rules
- Example: A weather god
  - Thor has moods (states)
  - Weather is created according to Thor's mood/state
  - E.g. Storms are more likely if Thor is angry

HAPPY

0.75
0.15
0.05
0.05

0.25
0.05
0.05
0.2
0.7
0.7

ANGRY

0.05
0.2
0.7
0.05

0.6

DRUNK

0.1
0.4
0.1
0.4

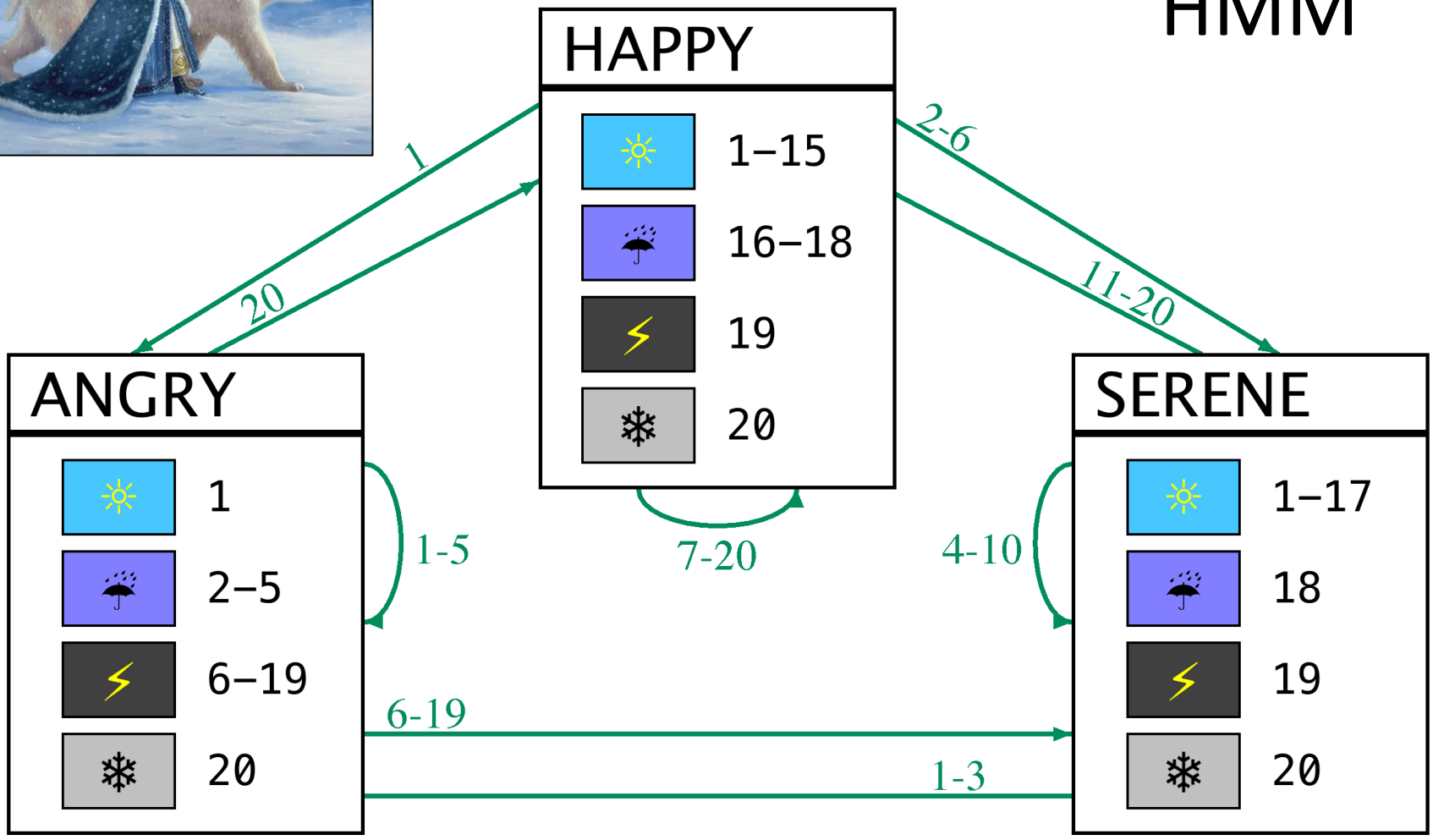
0.25
0.2

Time out to play a board game

Introducing Skadi, the Georgian weather goddess

Introducing Skadi, the Georgian weather goddess

Skadi's HMM


HAPPY
1–15
16–18
19
20
ANGRY
1
2–5
6–19
20
SERENE
1–17
18
19
20
1
20
2-6
11-20
1-5
7-20
4-10
6-19
1-3

What we know and what we guess

What we know

- Weather patterns

What we guess

- Thor exists
- His moods are Happy, Angry, and Drunk
- He rules the weather
- Skadi exists
- Her moods are Happy, Angry, and Serene
- She rules the weather

We need an HMM toolkit

- Generate patterns of any desired length

Generation

- Given a pattern, compute most likely responsible state path

Viterbi Algorithm

- Given a pattern, compute P(pattern|HMM)

Forward or Backward Algorithm

Viterbi Algorithm

- Given a pattern of emissions, compute the most likely responsible state path
- Dynamic Programming
  - It will remind you of pairwise alignment
- O(# of states^2 * pattern length)
  - Whew!
  - The more obvious “brute-force” approach is to generate all possible state paths (slow but easy), compute probability of each, and choose the most probable
  - That’s O(# of states ^ pattern length)

Review of Big-Oh

- ***Not*** the official CS146 definition
- A way to compare execution time of algorithms
  - Not particular programs which implement algorithms
  - Independent of implementation, independent of computer
- "An algorithm is O(n^2)" roughly means that execution time is proportional to n^2
  - n = input size
  - time α n^2 ➔ time = k * n^2
  - k varies across implementations and computers

Big-Oh Example

- You have 2 strings of observations.
  - Lengths = 100 and 1000.
- HMM has 5 states.
- You want to compute the most likely state path through the HMM for both strings.
- You have a free app that executes the brute-force algorithm. It took 1 second to compute the most likely path for the small string.
- You have a pay-per-use Viterbi app that charges what you pay each month in rent for inputs longer than 100. (Shorter inputs are free.) Like the free app, this one also took 1 second to compute the most likely path for the small string (for free).
- For the longer string, should you use the free app or spend your hard-earned money?
- ***Use Big-Oh to figure out how long each app will take.***

Viterbi

- On the small string (length = 100, cost = zero):
  - Algorithm is O(# of states^2 * string length)
  - Time = 1 sec = k * # of states^2 * string length
  - 1 = k * 5^2*100 ➔ k = 1/2500
- On the long string (length = 1000, cost = a lot)
  - Time = k * # of states^2 * string length
  - = 1/2500 * 25 * 1000 = 10 secs
  - A month's rent, gone in 10 seconds! Is it worth the cost?

Brute-Force (the free software)

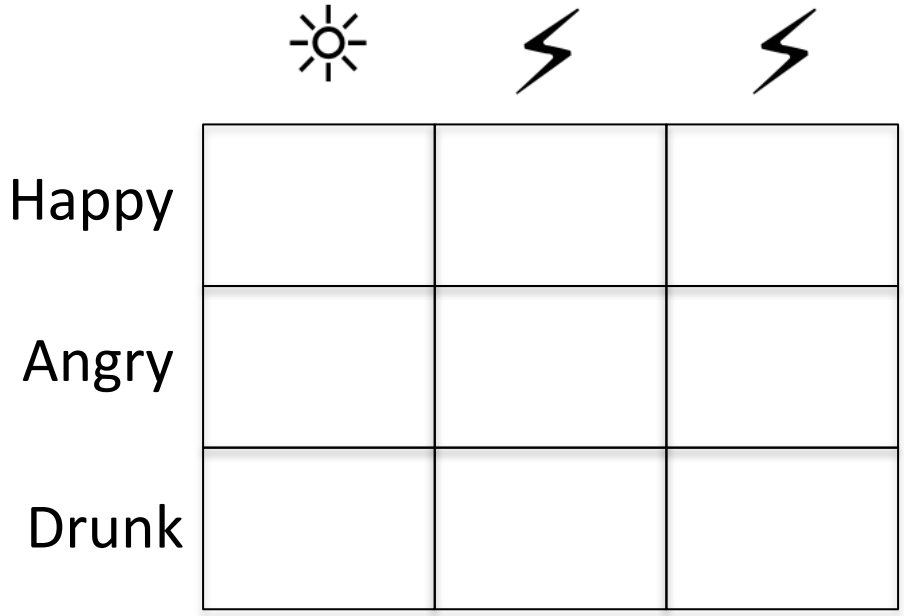
- On the small string (length = 100):
  - Algorithm = O(# of states ^ string length)
  - Time = 1 sec = k * # of states ^ string length
  - 1 = k * 5^100 ➔ k = 5 ^ -100
- On the long string (length = 1000)
  - Time = k * # of states ^ string length

    = 5^-100 * 5^1000 = 5^900 secs ~= 10^600 secs
  - Age of the universe < 10^18 secs

Viterbi Algorithm

Example: Find the Viterbi (most likely) path through the Thor HMM that generates Sun/Thunder/Thunder.

Step 1: Draw a grid. 1 row for each state of the HMM, 1 col for each member of the observation.



Viterbi Algorithm, first column:

	☼	⚡	⚡
Happy	P(start = State for row) * P(Emit weather for col from state for row)		
Angry	P(start = State for row) * P(Emit weather for col from state for row)		
Drunk	P(start = State for row) * P(Emit weather for col from state for row)		

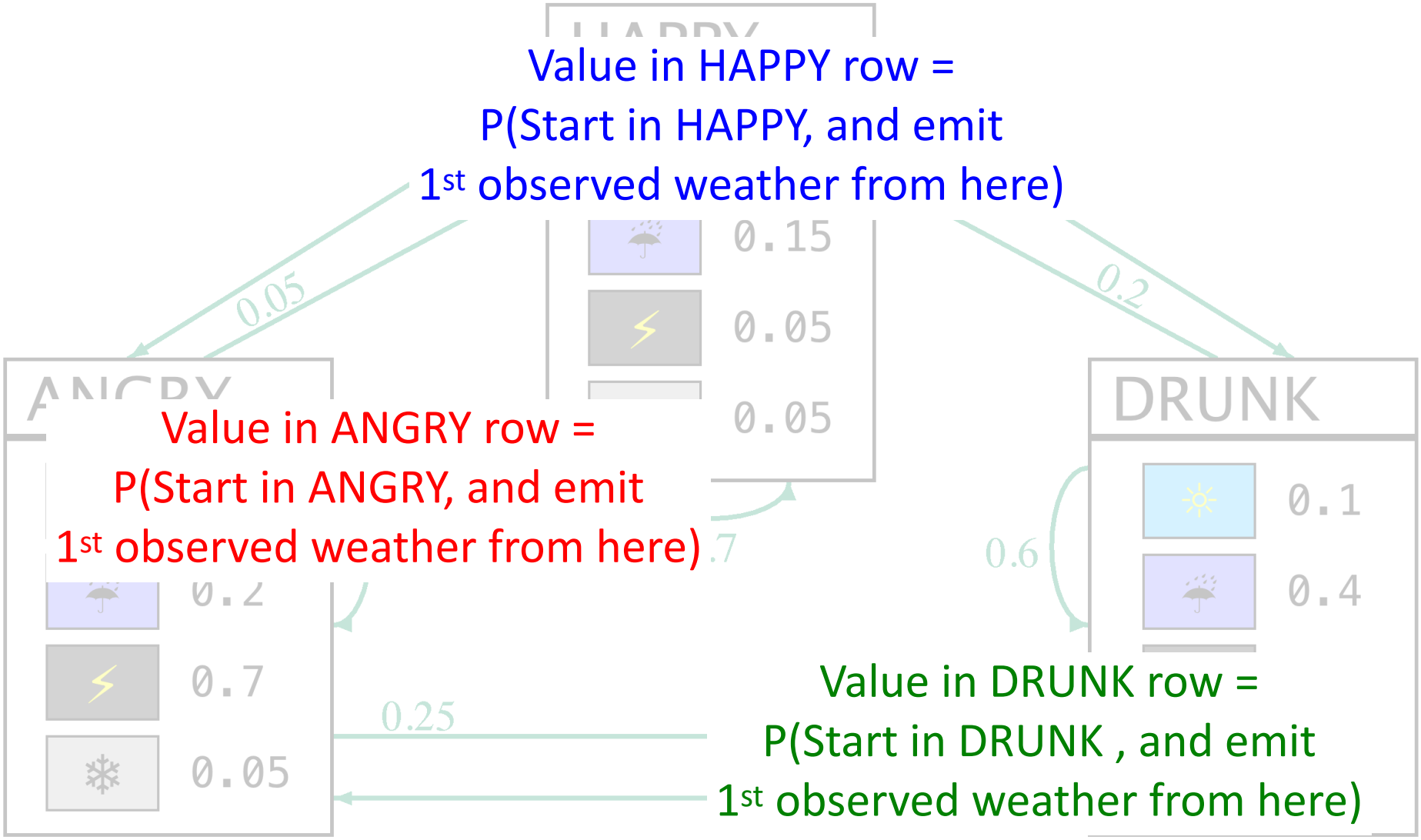
Viterbi Algorithm, first column:

			
Happy	P(start = Happy) * P(Emit  from Happy)		
Angry	P(start = Angry) * P(Emit  from Angry)		
Drunk	P(start = Drunk) * P(Emit  from Drunk)		

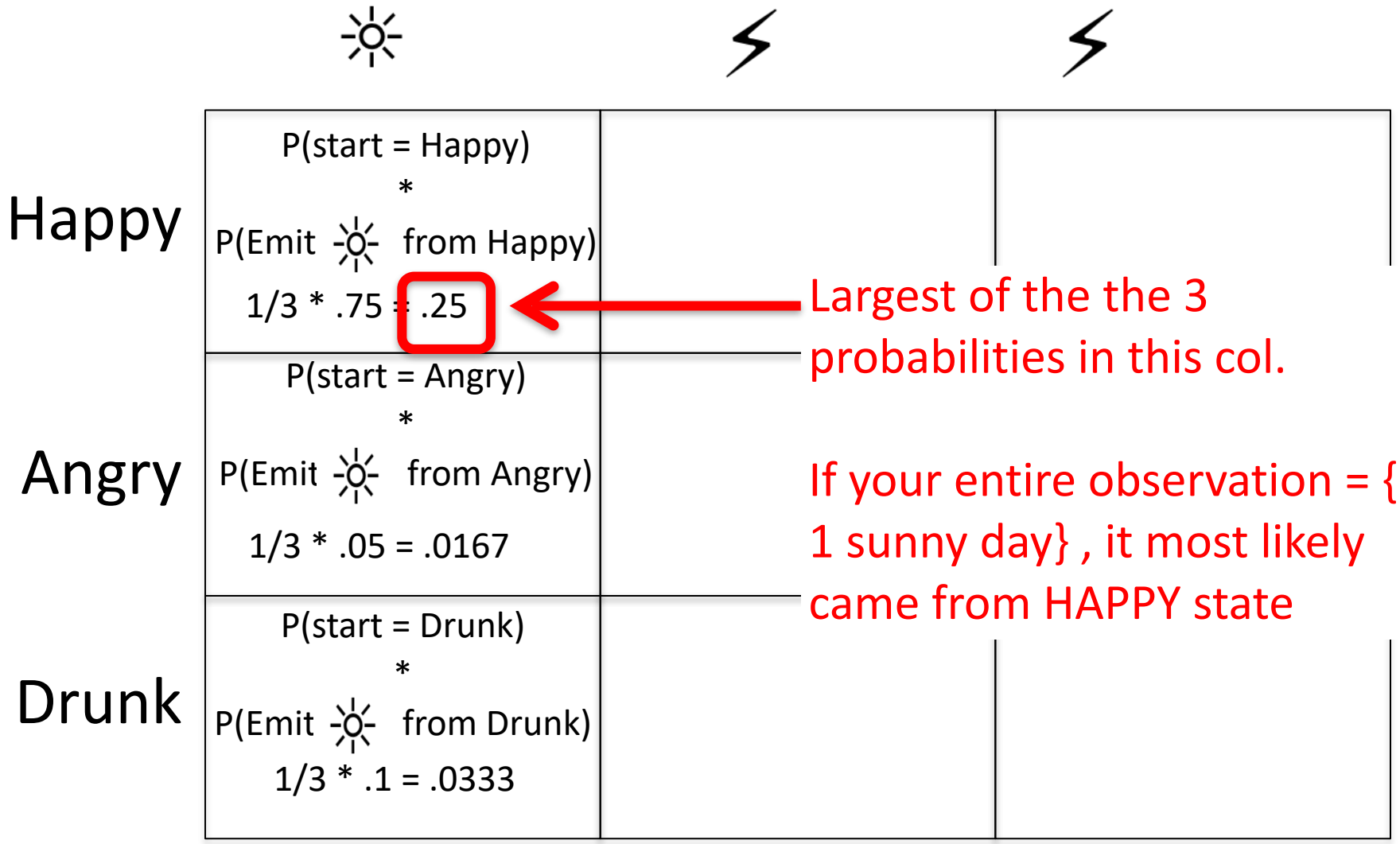
Viterbi Algorithm, first column:

			
Happy	P(start = Happy) * P(Emit  from Happy) 1/3 * .75 = .25		
Angry	P(start = Angry) * P(Emit  from Angry) 1/3 * .05 = .0167		
Drunk	P(start = Drunk) * P(Emit  from Drunk) 1/3 * .1 = .0333		

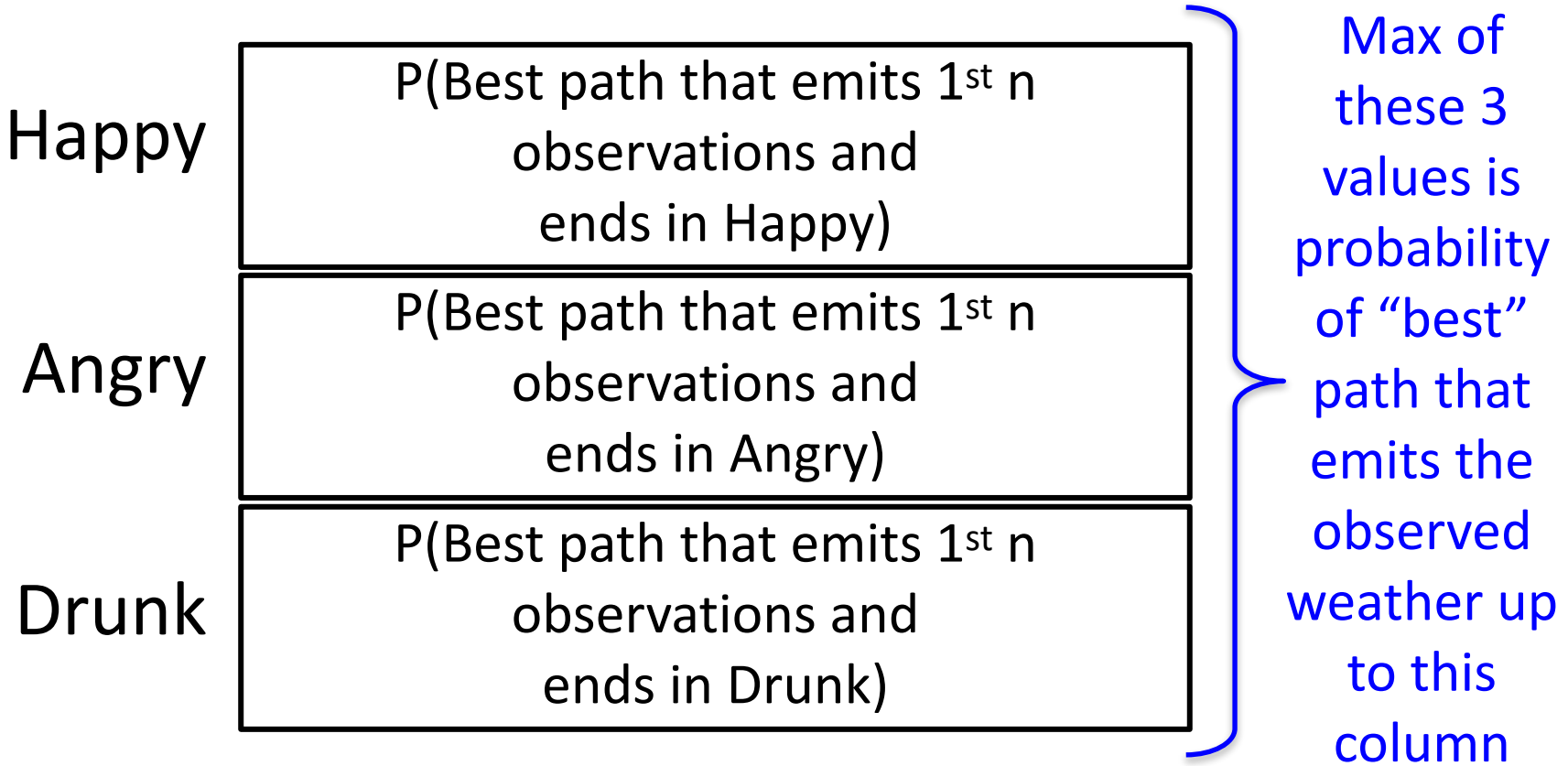
Viterbi Algorithm, first column:



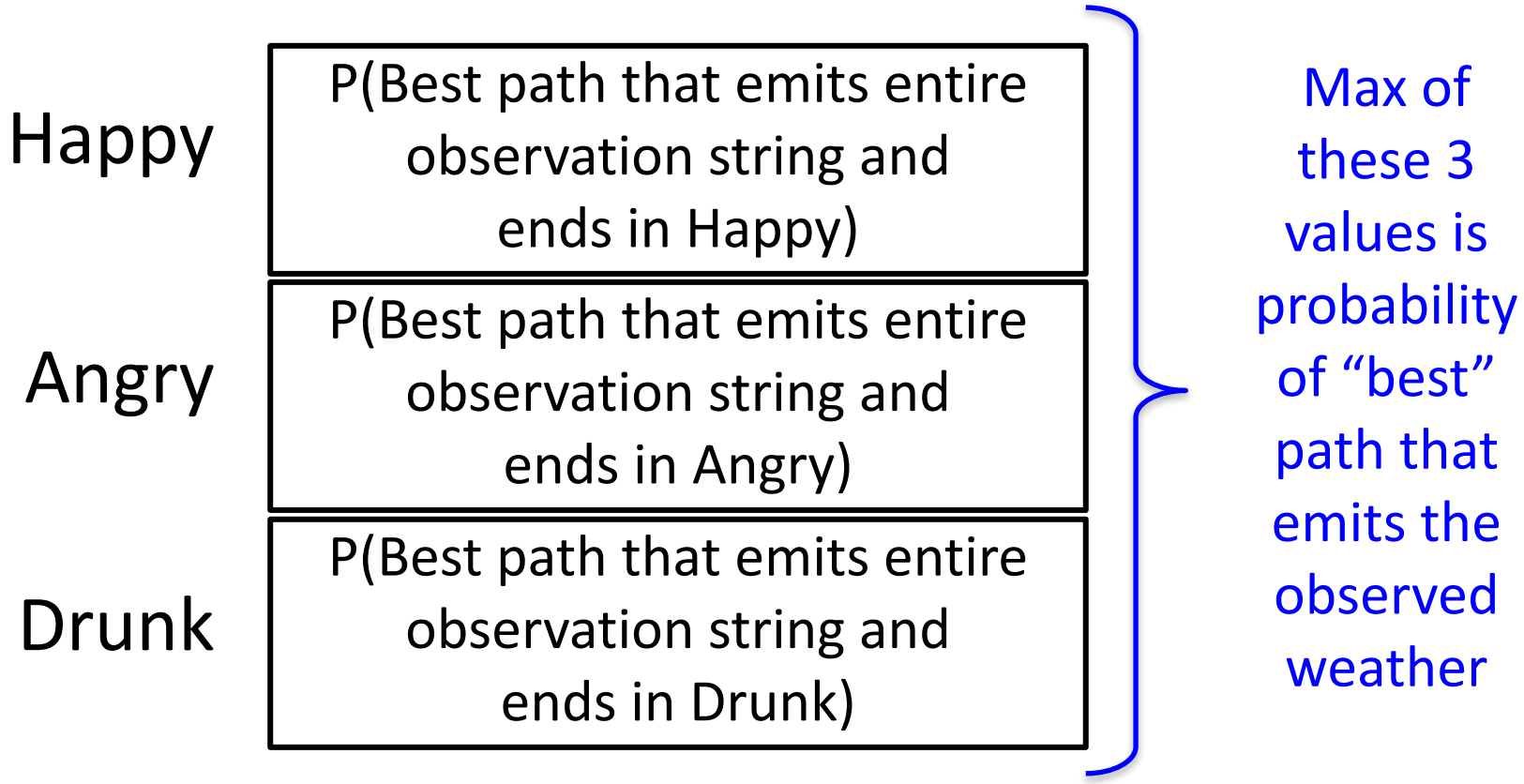
Viterbi Algorithm, first column:



Viterbi Algorithm, nth column:



Viterbi Algorithm, last column:



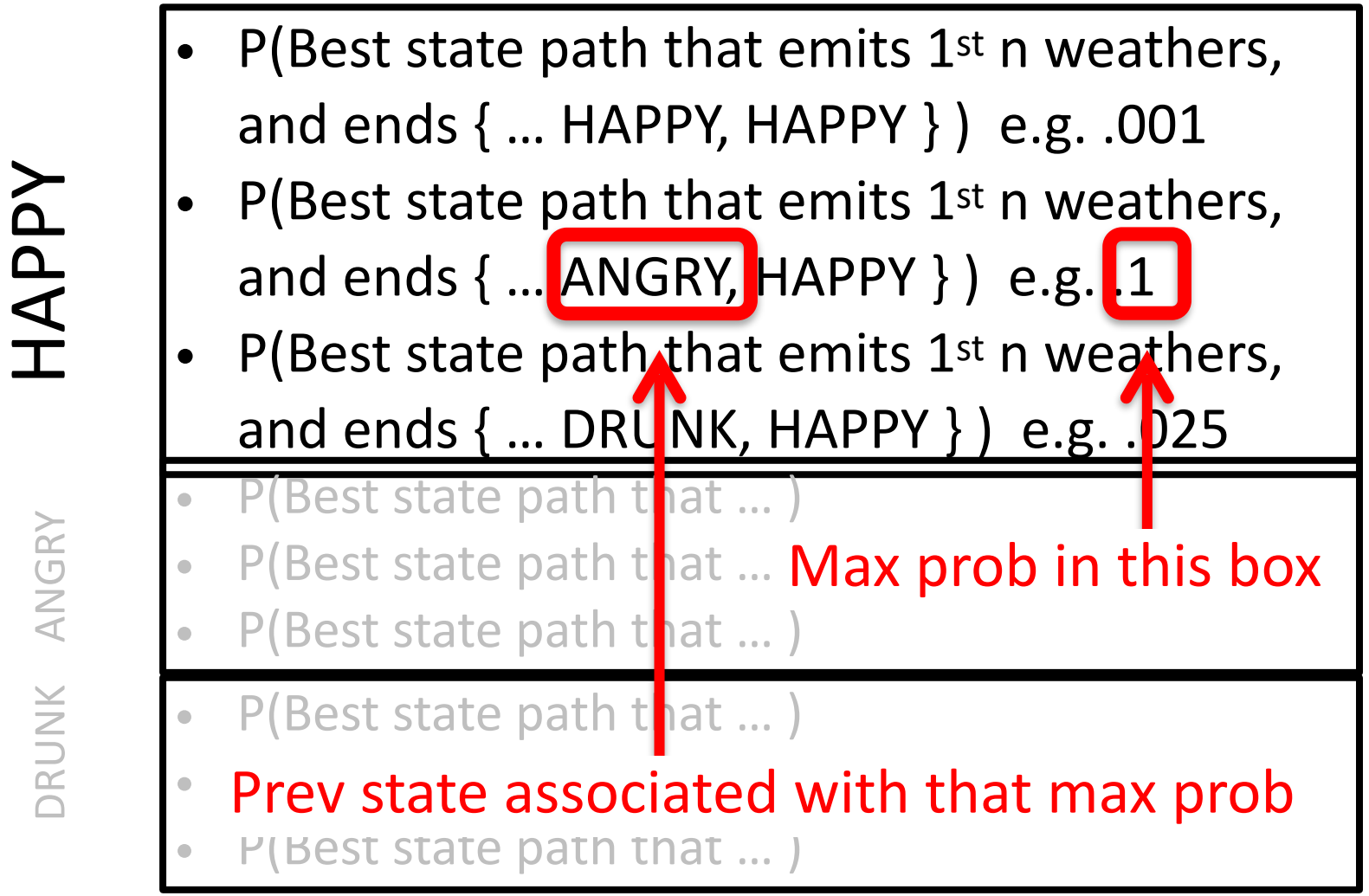
Viterbi Algorithm, any box after 1st column:

- Suppose this box is for state S, column n…
- In this box, compute one number for each possible previous state R, and select the max of these to be this box's value. Also remember which R was involved.
- Number computed for any R is the prob of the best state path that emits 1st n observations, and whose last 2 states are R and S.

Viterbi Algorithm, HAPPY box after 1st column:

HAPPY	• P(Best state path that emits 1st n weathers, and ends { … HAPPY, HAPPY }) e.g. .001 • P(Best state path that emits 1st n weathers, and ends { … ANGRY, HAPPY }) e.g. .1 • P(Best state path that emits 1st n weathers, and ends { … DRUNK, HAPPY }) e.g. .025
ANGRY	• P(Best state path that …) • P(Best state path that …) • P(Best state path that …)
DRUNK	• P(Best state path that …) • P(Best state path that …) • P(Best state path that …)

Viterbi Algorithm, HAPPY box, column n:



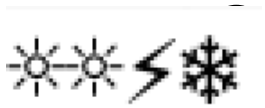
So for any box after the first column

- Suppose this column number is n
- Suppose weather for this box's column is W
- Suppose state for this box is S
- This box needs 1 number for every possible prev state R: Prob(Best state path that emits 1st n weathers, and ends with { … RS})
- = Prob(Best state path that emits 1st n-1 weathers, and ends with { … R})
  * Prob (R→S transition)
  * Prob(Emit W from S)

Box for state R in previous column

Table of transition probabilities

Table of emission probabilities

Example: What is the most likely path to emit  Step 1: Draw the Grid

	☼	☼	⚡	❄
Happy				
Angry				
Drunk				

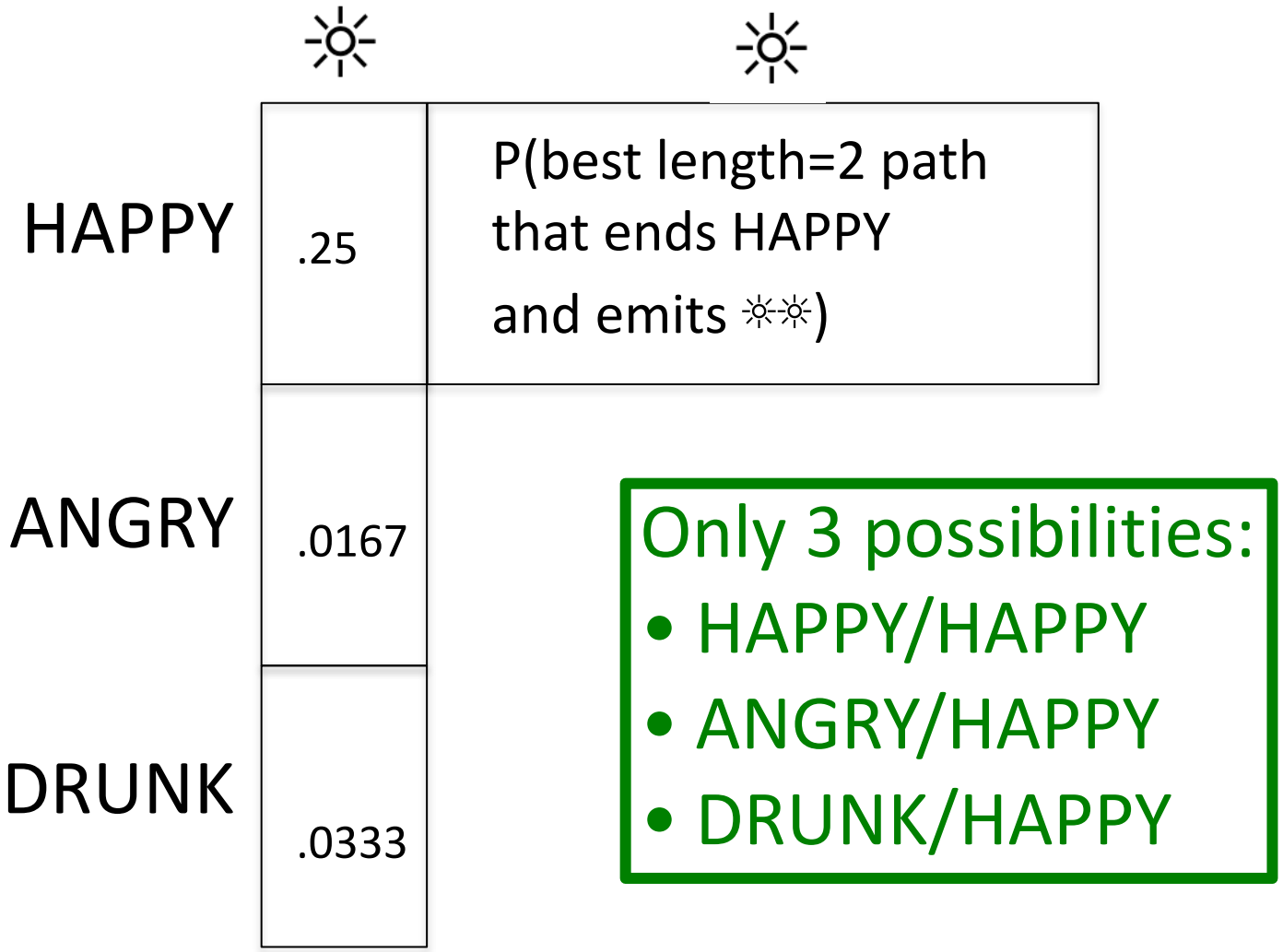
Step 2: Fill in 1st column

				
HAPPY	P(Start HAPPY and emit ) 1/3 * .75 = .25			
ANGRY	P(Start ANGRY and emit ) 1/3 * .05 = .0167			
DRUNK	P(Start DRUNK and emit ) 1/3 * .1 = .0333			

Step 3: Fill in 2nd column

	☼	☼	↯	❄
HAPPY	.25	P(best length=2 path that ends HAPPY and emits ☼☼)		
ANGRY	.0167	P(best length=2 path that ends ANGRY and emits ☼☼)		
DRUNK	.0333	P(best length=2 path that ends DRUNK and emits ☼☼)		

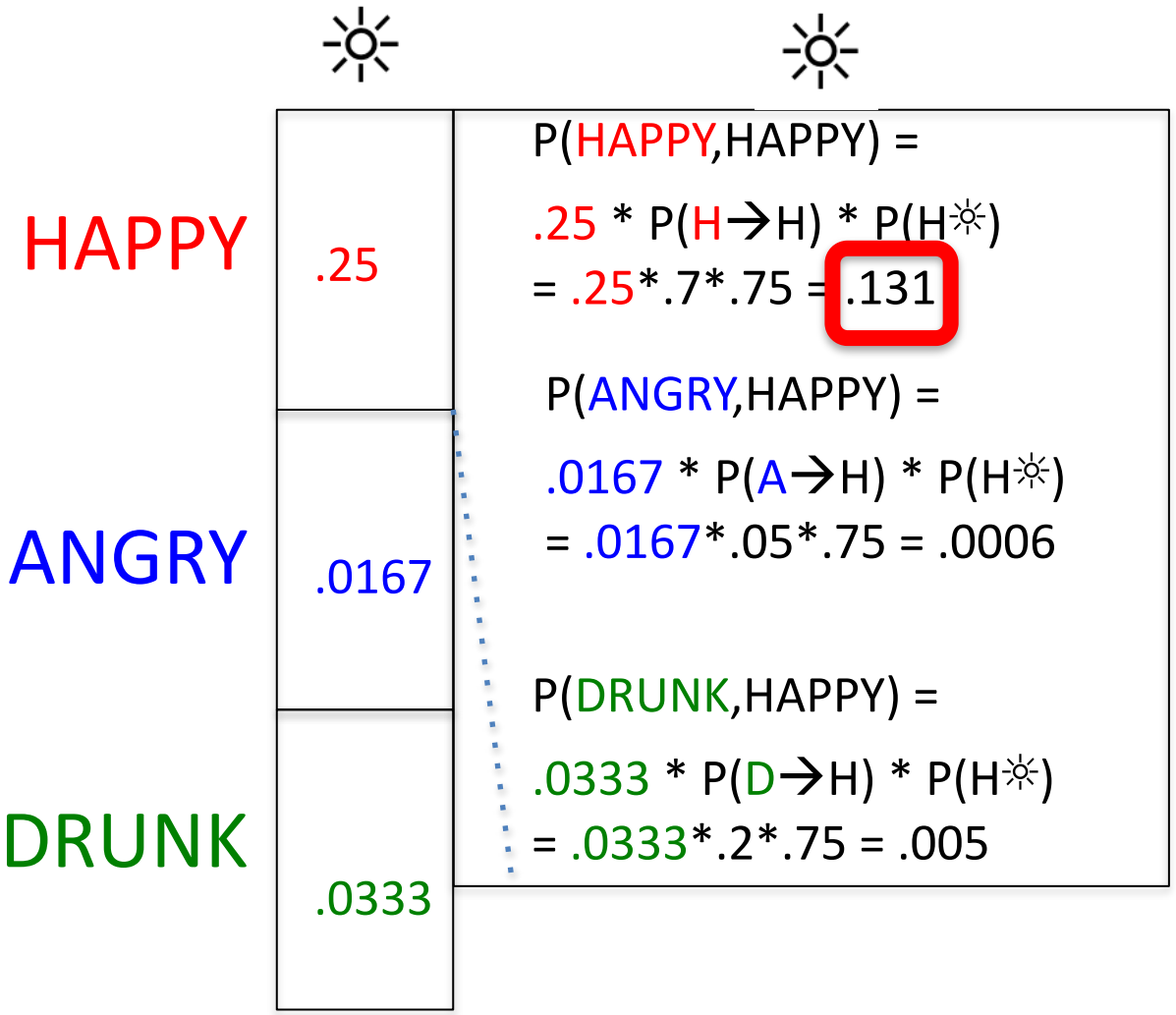
Step 3: Fill in 2nd column



Only 3 possibilities:

- HAPPY/HAPPY
- ANGRY/HAPPY
- DRUNK/HAPPY

Step 3: Fill in 2nd column, HAPPY cell



Max of the 3 probs is for path = HAPPY,HAPPY

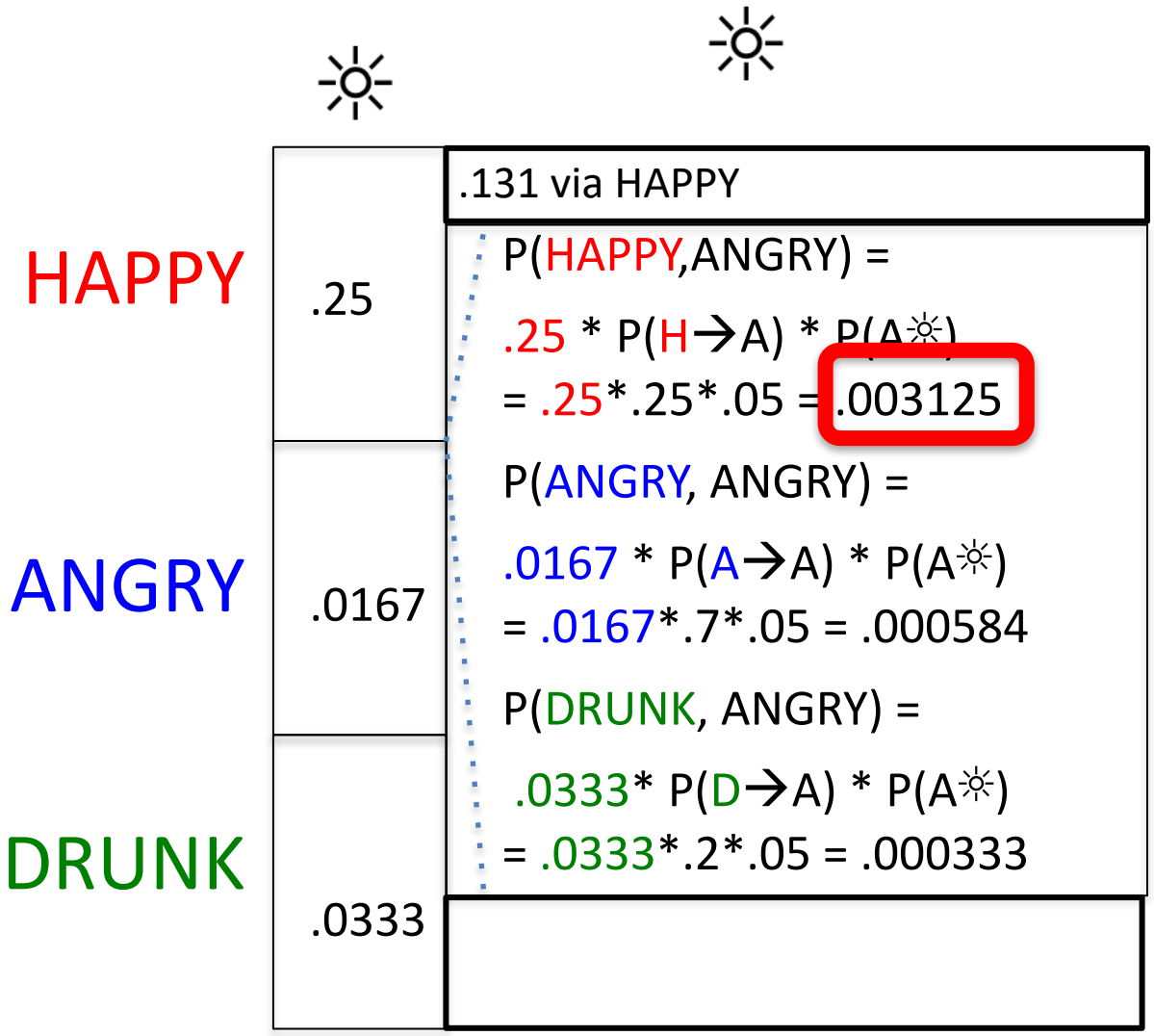
→ This is the most probable path that emits ☼☼ and ends at HAPPY. Its probability is .131

Retain .131 and remember that state from prev col was HAPPY

Step 3: Fill in 2nd column, HAPPY cell

HAPPY	.25	.131 via HAPPY
ANGRY	.0167	
DRUNK	.0333	

Step 3: Fill in 2nd column, ANGRY cell



Max of the 3 probs is for path = HAPPY,ANGRY

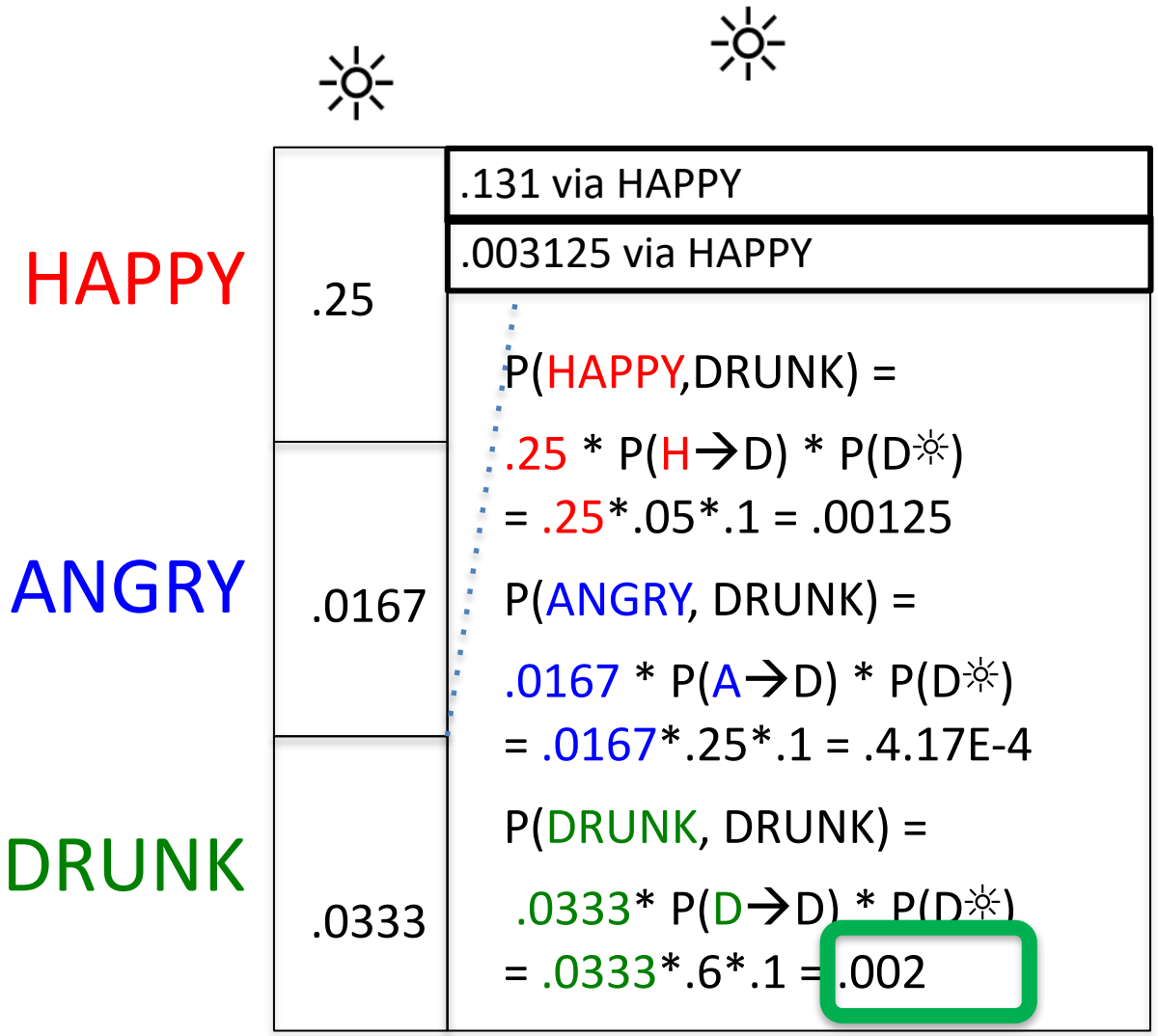
➔ This is the most probable path that emits ☼☼ and ends at ANGRY. Its probability is .003125

Retain . 003125 and remember that state from prev col was HAPPY

Step 3: Fill in 2nd column, DRUNK cell

	☼	☼
HAPPY	.25	.131 via HAPPY
ANGRY	.0167	.003125 via HAPPY
DRUNK	.0333	

Step 3: Fill in 2nd column, DRUNK cell



Max of the 3 probs is for path = DRUNK,DRUNK

→ This is the most probable path that emits ☼☼ and ends at DRUNK. Its probability is .0002

Retain .002 and remember that state from prev col was DRUNK

Step 3: Voila, column 2

HAPPY	.25	.131 via HAPPY
ANGRY	.0167	.003125 via HAPPY
DRUNK	.0333	.0002 via DRUNK

Step n, any cell, general case

- Suppose weather for this col = W, and we're computing cell for current state = S in this col
- Compute 1 prob for each just-previous state:
  - Via HAPPY: prev col(HAPPY) * Prob(HAPPY→S) * Prob(Emit W from S)
  - Via ANGRY: prev col(ANGRY) * Prob(ANGRY→S) * Prob(Emit W from S)
  - Via DRUNK: prev col(DRUNK) * Prob(DRUNK→S) * Prob(Emit W from S)
- Choose max of these 3 probs, and remember its prev state

The whole grid for ☼☼⚡❄

	☼	☼	⚡	❄
HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

How to interpret this grid:

Largest prob in last column = prob of most likely path to emit observed weather pattern

Aka the "Viterbi Path"

But what *is* that path?

☼	⚡	❄
.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

The last state in the Viterbi path is the state of the cell in the last column containing the max probability

In this example: DRUNK

Viterbi path = ? ? ? DRUNK

HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

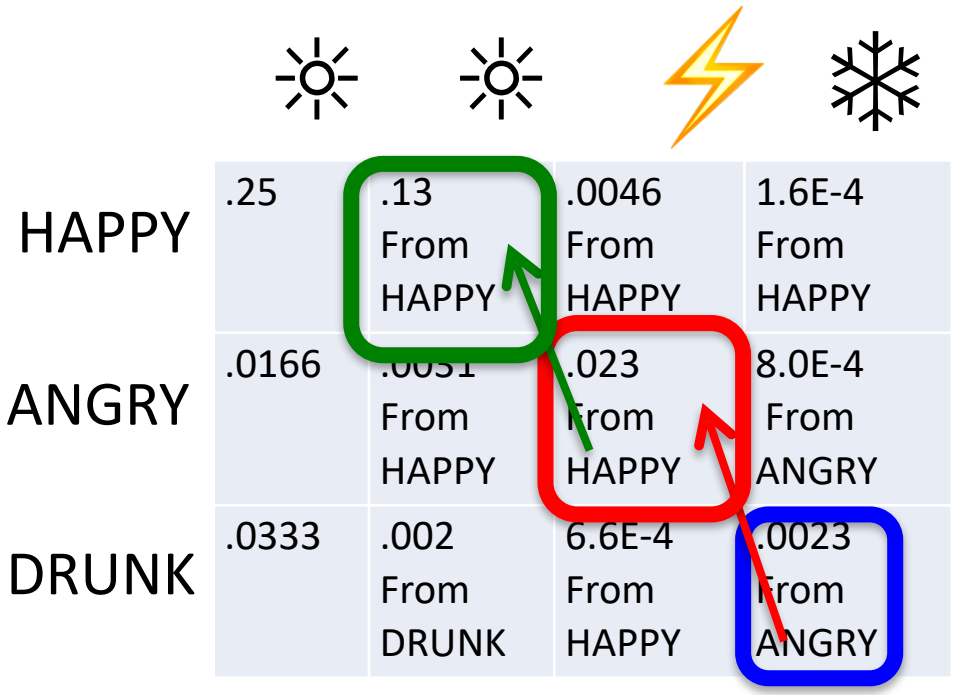
The rest of the Viterbi path is found by tracing back along the "From" states

Viterbi path = ? ? ? DRUNK

HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

The rest of the Viterbi path is found by tracing back along the "From" states

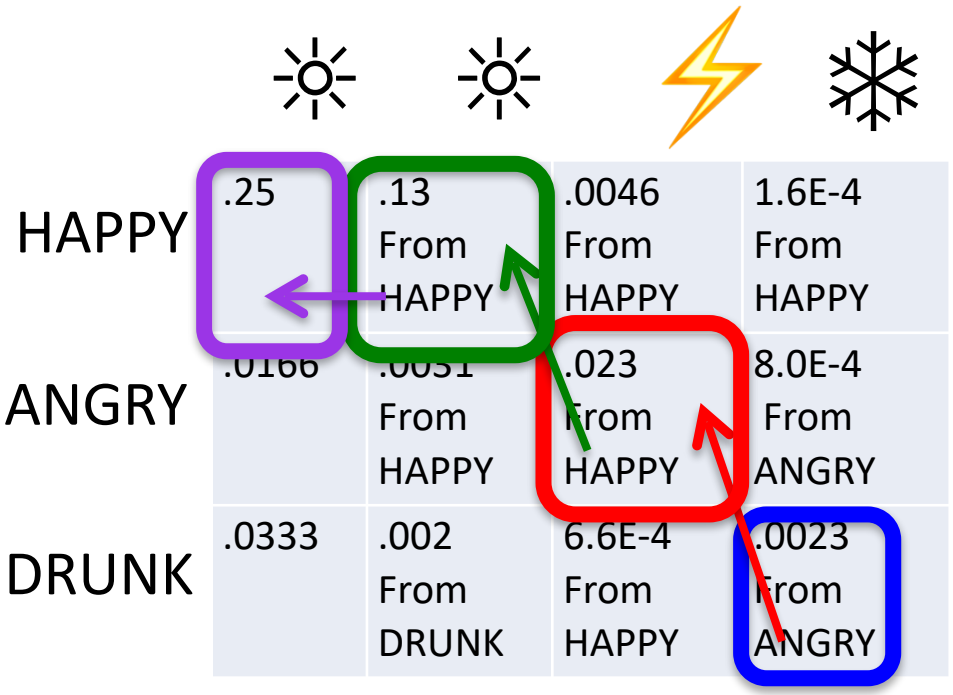
Viterbi path = DRUNK



HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

The rest of the Viterbi path is found by tracing back along the "From" states

Viterbi path = ANGRY DRUNK

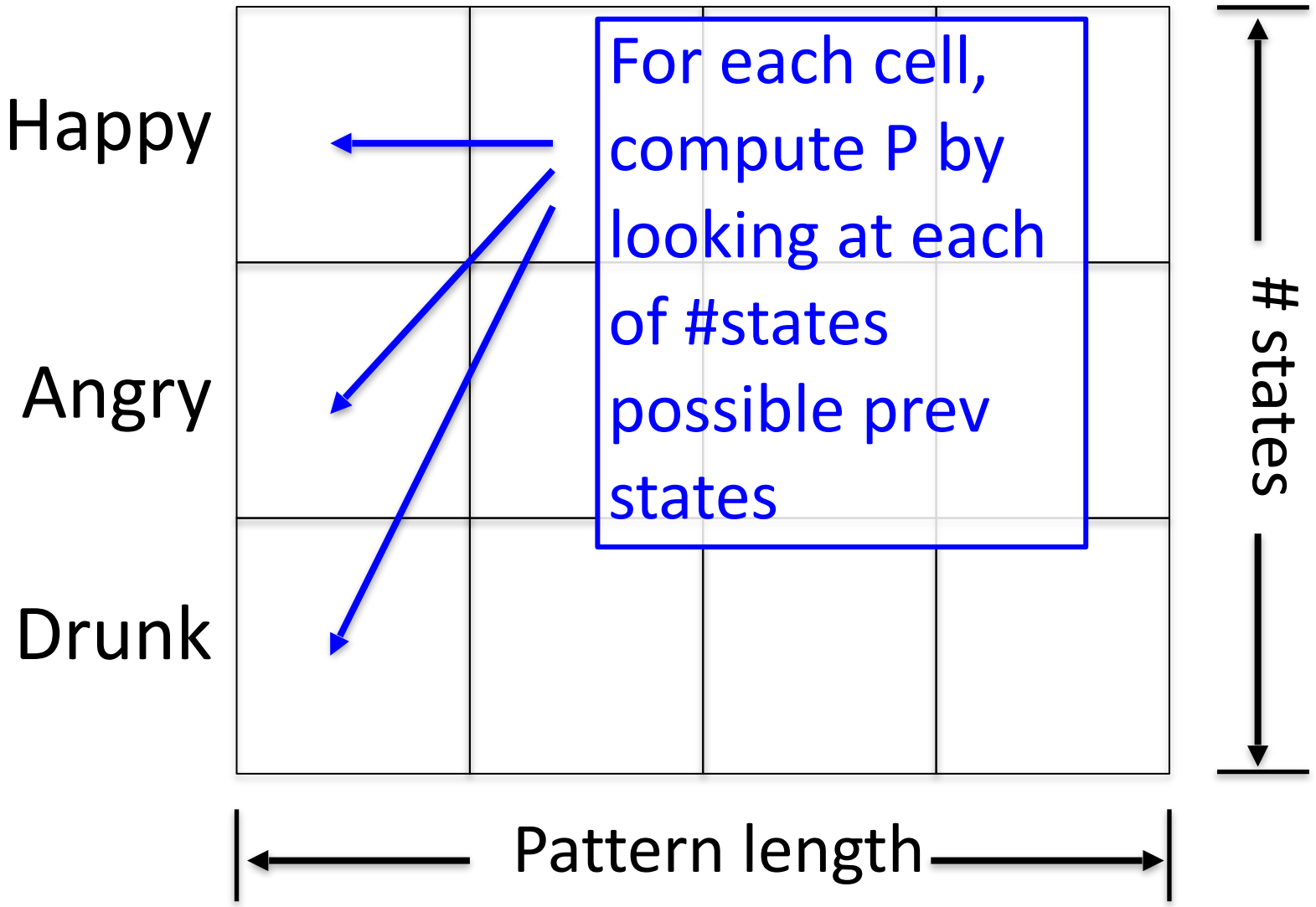


HAPPY	.25	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

The rest of the Viterbi path is found by tracing back along the "From" states

Viterbi path = HAPPY ANGRY DRUNK

Viterbi is O(# of states^2 * pattern length)



Viterbi is O(# of states^2 * pattern length)

- For each cell, compute P by doing constant-time computation for each of #states possible prev states
- # of constant-time computations for any cell = #states * #cells
- There are (#states * #cols) cells
- # of constant-time computations = #states * #states * #cells = #states^2 * #cells
- Time to execute Viterbi = (time required for 1 constant-time computation) * #states^2 * #cells
- = k * #states^2 * #cells for some k that varies for each implementation, computer speed, OS, etc.

Time for lab 1

Use 1st 30 minutes of next class if you need.
If not, it's ok to come 30 minutes late.

A Hidden Markov Model for DNA Sequences: CpG islands

- Definition: A *dinucleotide* is a consecutive pair of nucleotides in a chromosome
  - Eg T,A or C,G
- Dinucleotides are usually written XpY, where X is the 1st nucleotide and Y is the 2nd
  - Eg TpA or CpG
- CpG means *C followed by G, as we read from 5' to 3'*
- The p represents the phosphate between 2 nucleotides

Importance of CpG

- The CpG dinucleotide is rare in most vertebrates, including humans.
- The C of CpG dinucleotides can be *methylated*, meaning a methyl (CH_3) group is bonded to the C.

  Methylated genes are expressed at low rates, or not at all. This concept is central to epigenetics.
- Non-methylated CpG often appears in regions with unusually high CpG content, called “CpG Islands”, which usually are found in promoters. Computational finding of promoters in genomes helps us find genes, so it’s fundamental to bioinformatics.

The Goal

- Input: long DNA sequences, maybe entire chromosomes
- Output: software-predicted locations of all CpG islands

CpG Islands in the human genome

ISLANDS	A	C	G	T
A	.180	.274	.426	.120
C	.170	.368	.274	.188
G	.161	.339	.375	.125
T	.079	.355	.384	.182

1st nucleotide

2nd nucleotide

NOT ISLANDS	A	C	G	T
A	.300	.205	.285	.210
C	.322	.298	.078	.302
G	.248	.246	.298	.208
T	.177	.239	.292	.292

CpG Island Definition

- >= 200 bases
- GC content > 50% (Gs and Cs, not CpGs)
- # CpG * length / (#C * #G) > 0.6
  - "Observed-to-expected" ratio, not an intuitive name
  - E.g. if length = 100 and sequence = CGCG…CG, then observed-to-expected = 50 * 100 / (25*25) = 8
  - Other definitions have been proposed

Goal: Identify CpG islands in very long DNA sequences. How *not* to do it:

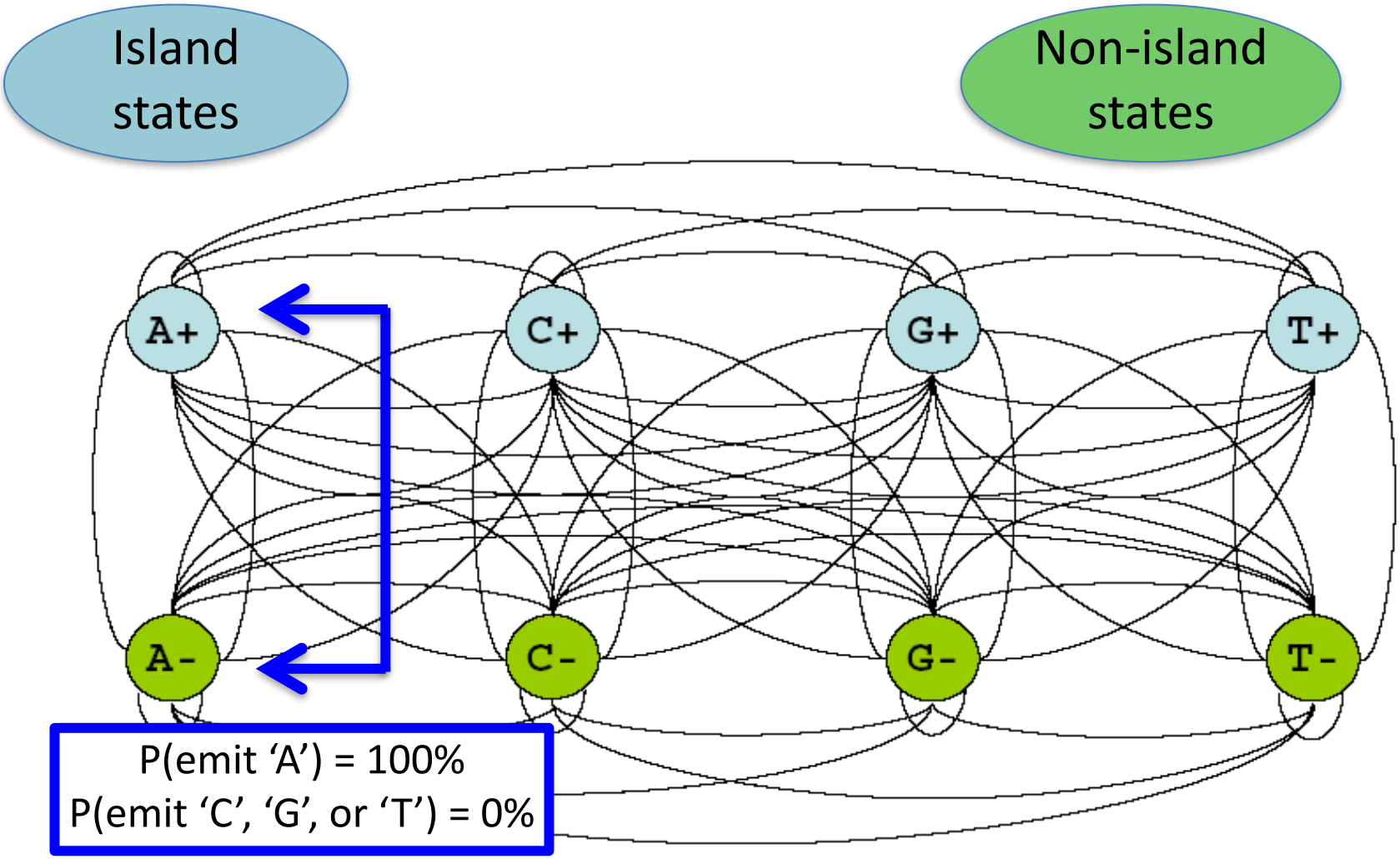
- Pick a number. 100? Ok, 100.
- Build a position weight matrix for CpG islands with 100 bases/columns, based on frequency tables.
- Now you can classify 100-mers.
- Analyze:
  - Positions 1-100
  - 101-200
  - 201-300
  - And so on
- “Sliding window” technique.

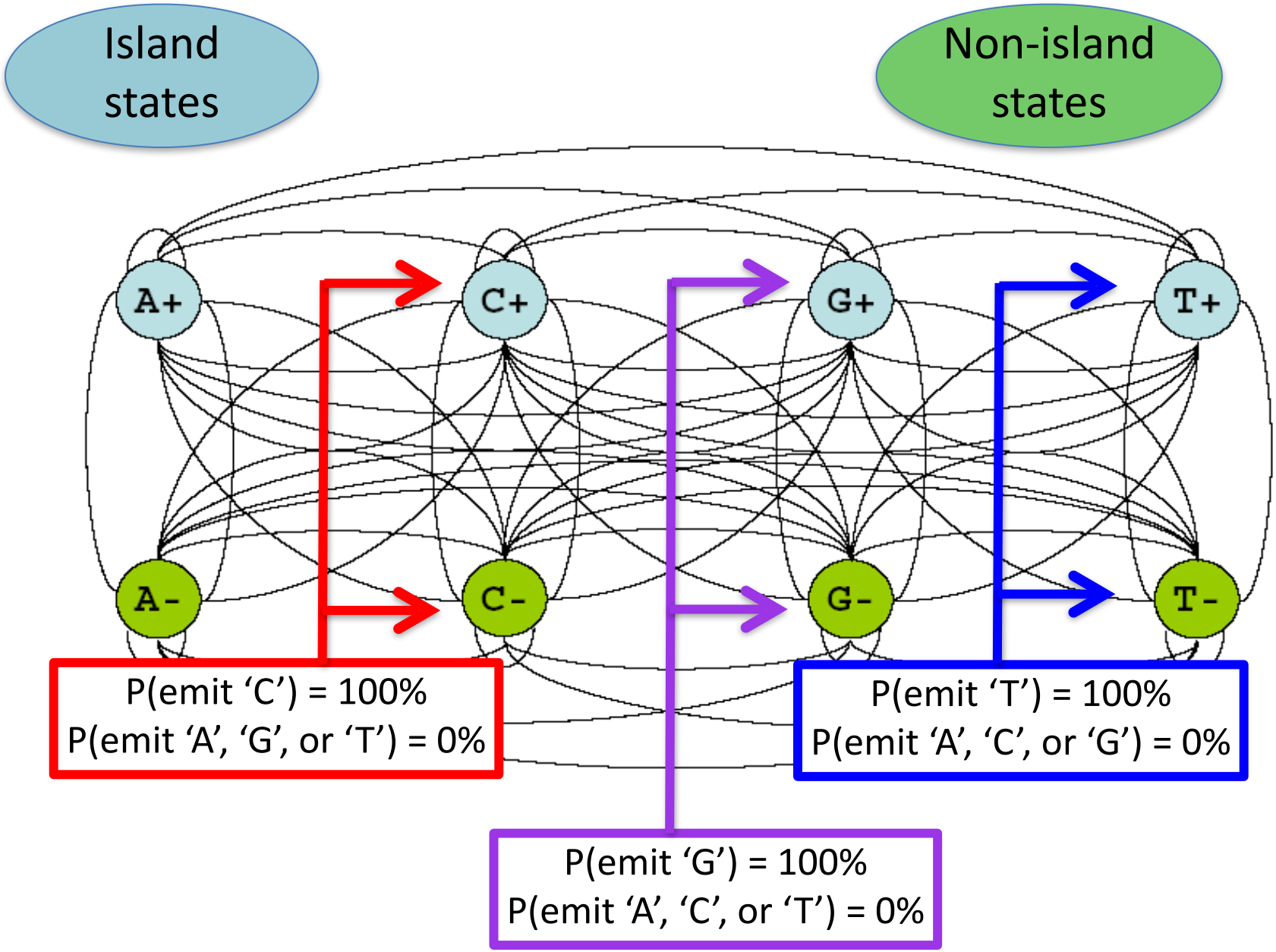
But sliding windows don't work well in practice

- Small windows are too sensitive, large windows aren't sensitive enough. What's the right size?
- Any window size will either accept too many non-islands, or reject too many islands, or some combination of both.
- HMM approaches have proven to be more successful.

The CpG HMM

- 8 states
  - A+, C+, G+, T+, A-, C-, G-, T-
  - + means an island state, - means a not-island state
- From each state, only 1 emission is possible
  - That's a bit weird for an HMM
  - Emissions are 'A', 'C', 'G', and 'T'
  - States A+ and A- can only emit 'A', etc
- State names look like emissions so don't get confused


Island states
Non-island states
A+
C+
G+
T+
A-
C-
G-
T-
P(emit 'A') = 100%
P(emit 'C', 'G', or 'T') = 0%


Island states
Non-island states
A+
C+
G+
T+
A-
C-
G-
T-
P(emit 'C') = 100%
P(emit 'A', 'G', or 'T') = 0%
P(emit 'T') = 100%
P(emit 'A', 'C', or 'G') = 0%
P(emit 'G') = 100%
P(emit 'A', 'C', or 'T') = 0%

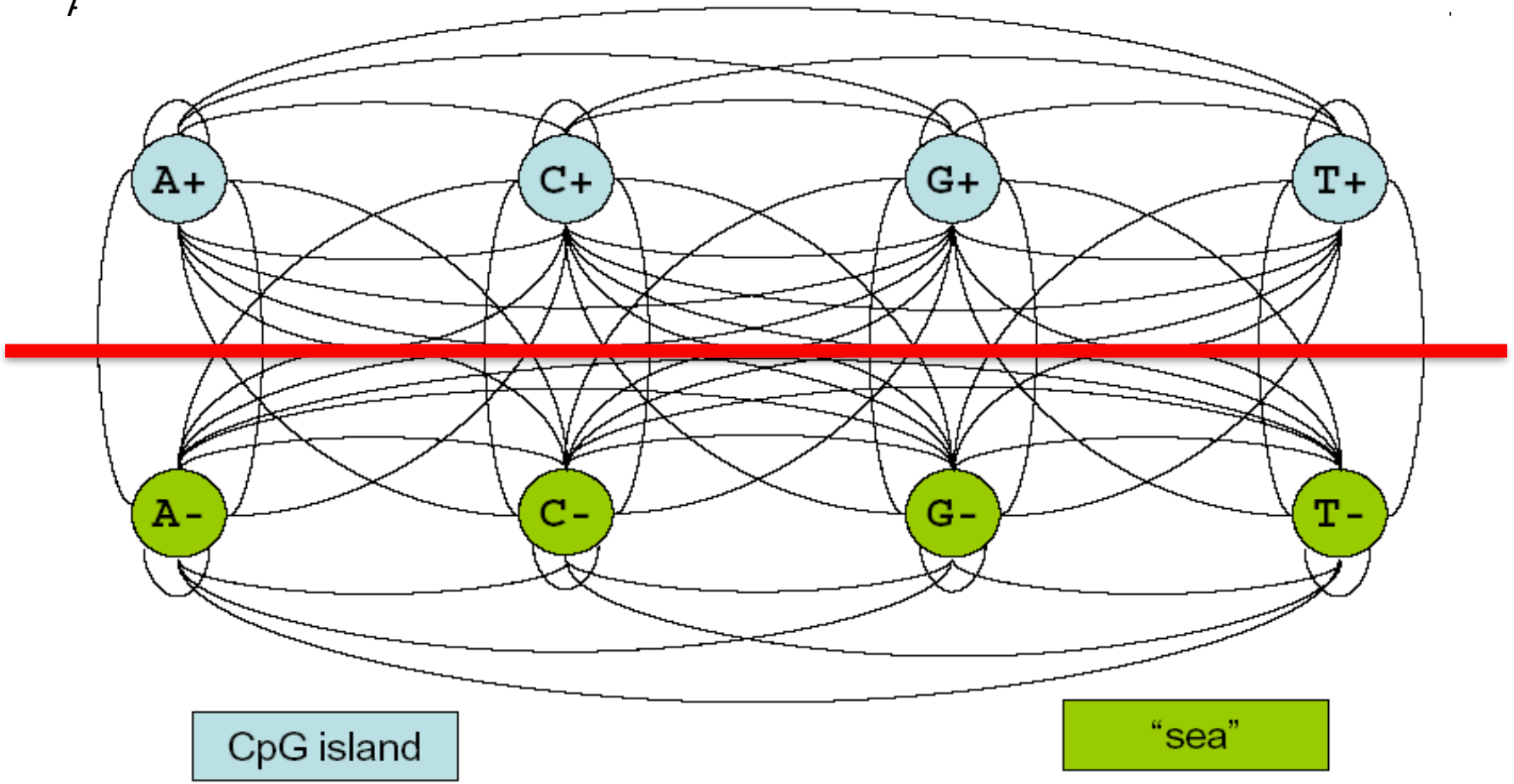
Among island states, transition probabilities are derived from the frequency tables…

ISLANDS	A	C	G	T
A	.180	.274	.426	.120
C	.170	.368	.274	.188
G	.161	.339	.375	.125
T	.079	.355	.384	.182

Prob(T+ → A+) = .079

Prob(C- → G-) = .078

NOT ISLANDS	A	C	G	T
A	.300	.205	.285	.210
C	.322	.298	.078	.302
G	.248	.246	.298	.208
T	.177	.239	.292	.292



Prob(Any "+" state → Any "–" state) = zero

This machine can never model or evaluate a transition *to* or *from* a CPG island

Fixing the transition probabilities

- Analysis shows P(Any + state → Any – state) ~= P(Any - state → Any + state) ~= 0.01% = 0.0001 = P(crossing the red line)
- Multiply all transition probabilities by 0.9999
- Now P(transition from any state) has been reduced from 1 to .9999
- Let P(A+→A-) = P(A+→C-) = P(A+ →G-) = P(A+ → T-) = .0001/4 = .000025
- Similar for all other island-to-not-island and not-island-to-island transitions

Remember The Goal

- Input: long DNA sequences, maybe entire chromosomes
- Output: software-predicted locations of all CpG islands
- How do we do that with the CpG HMM?

How to do it

- Compute the Viterbi path through the CpG HMM for your input
  - E.g. A+ C+ G+ C+ G+ A- T- G+ …
- Look at the “+” or “-” parts of the state names
  - + + + + + - - +
- Runs of “+” of length >= 200 are CpG islands

The Forward Algorithm

- Computes P(HMM emits observation) = P(obs|HMM)
  - No matter what the path might have been
- Like Viterbi, but…
- Given string of observations, Viterbi finds most likely state path, and that path's probability
- But P(obs|HMM) = sum of probabilities over all paths that can emit that observation, including the Viterbi path and many others
- P(HMM emits observation) is often what we care about for protein identification using pHMMs
- Viterbi is about max(probabilities), FA is about Σ(the same probabilities)
- Example: what is the probability that the Thor HMM emits

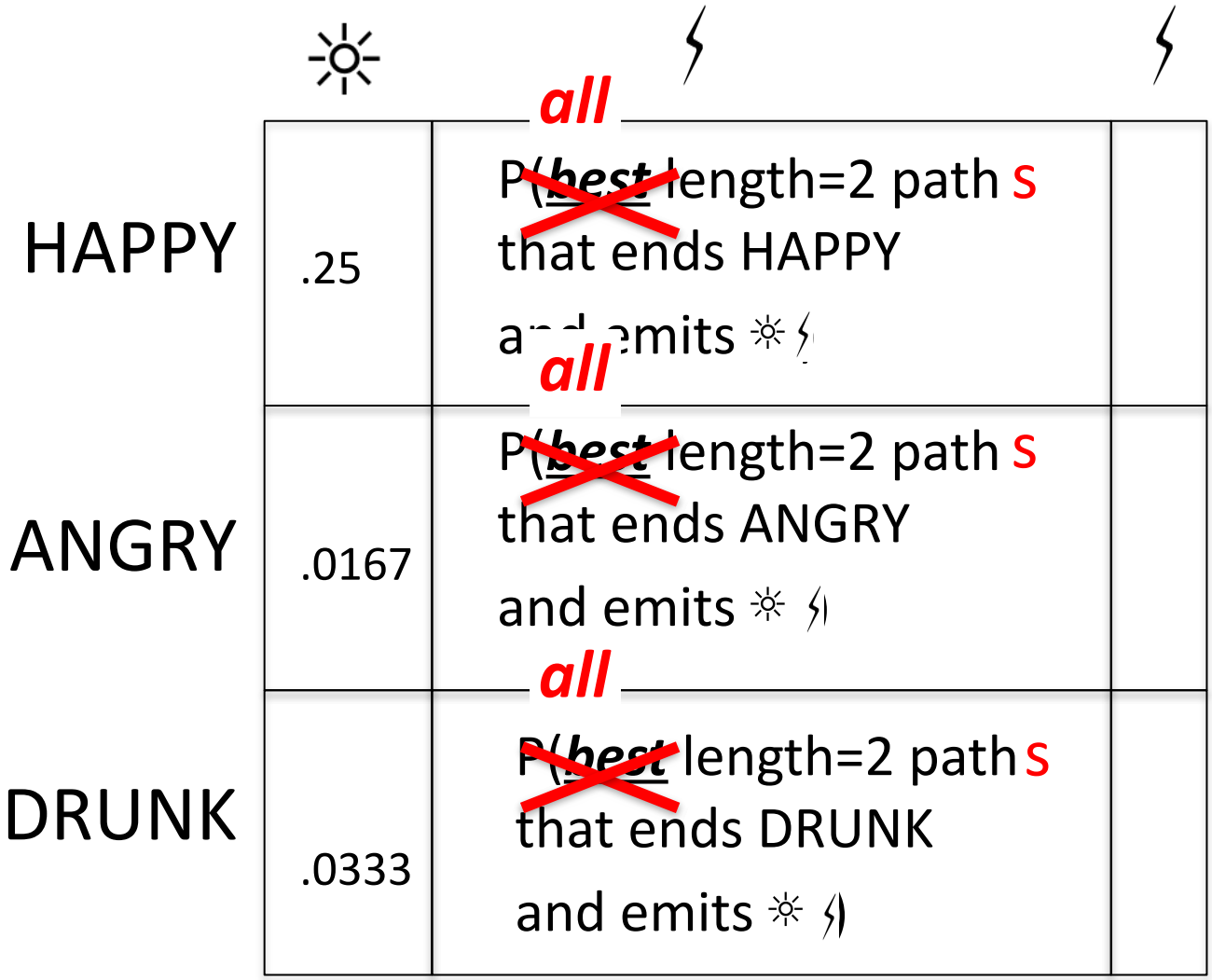
The Forward Algorithm starts like Viterbi :

	☼	⚡	⚡
Happy	P(start = State for row) * P(Emit weather for col from state for row)	P(Emit the 1st weather from this state)	
Angry	P(start = State for row) * P(Emit weather for col from state for row)	P(Emit the 1st weather from this state)	
Drunk	P(start = State for row) * P(Emit weather for col from state for row)	P(Emit the 1st weather from this state)	

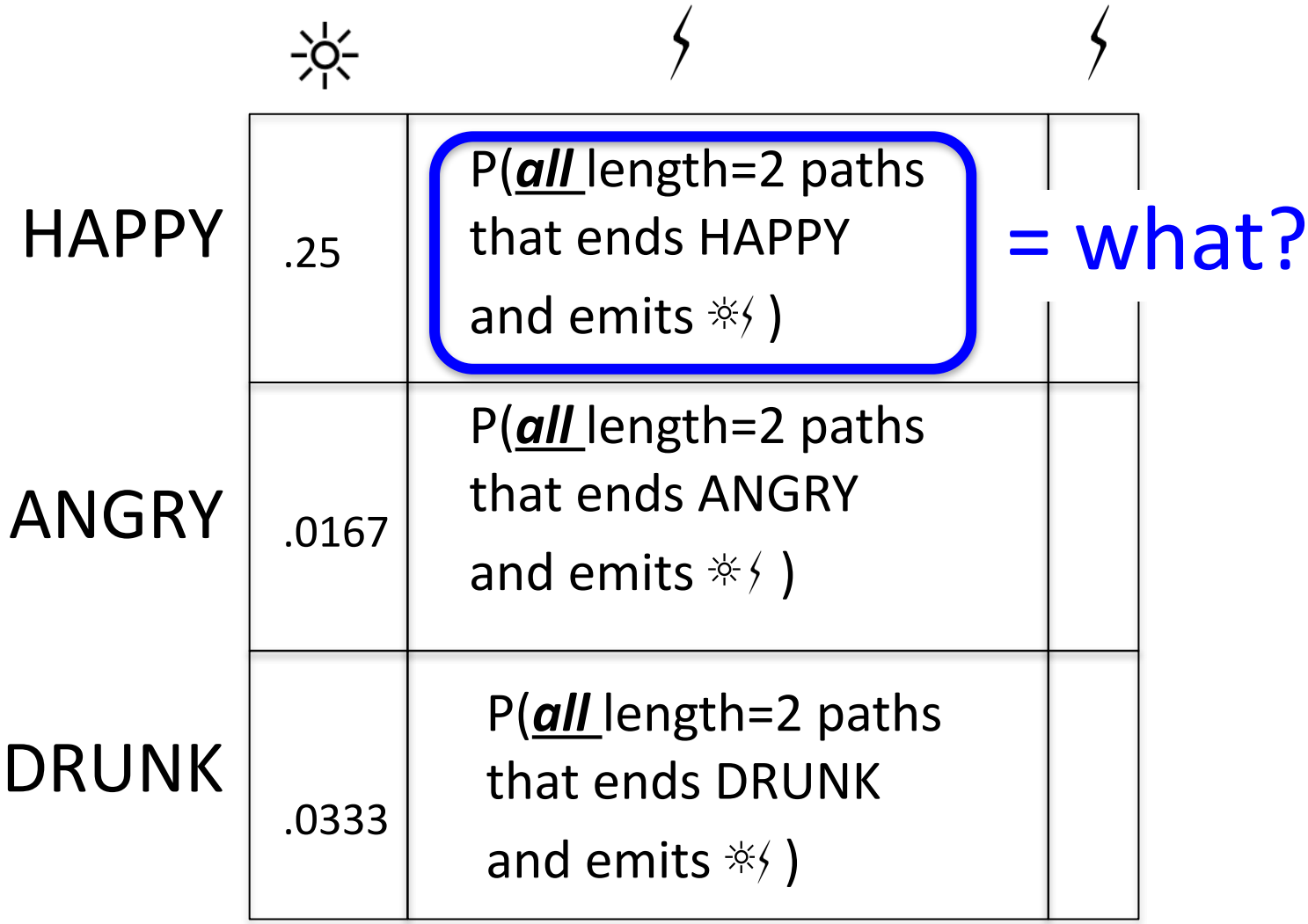
Viterbi reminder: 2nd column

	☼	⚡	⚡
HAPPY	.25	P(***best*** length=2 path that ends HAPPY and emits ☼ ⚡)	
ANGRY	.0167	P(***best*** length=2 path that ends ANGRY and emits ☼ ⚡)	
DRUNK	.0333	P(***best*** length=2 path that ends DRUNK and emits ☼ ⚡)	

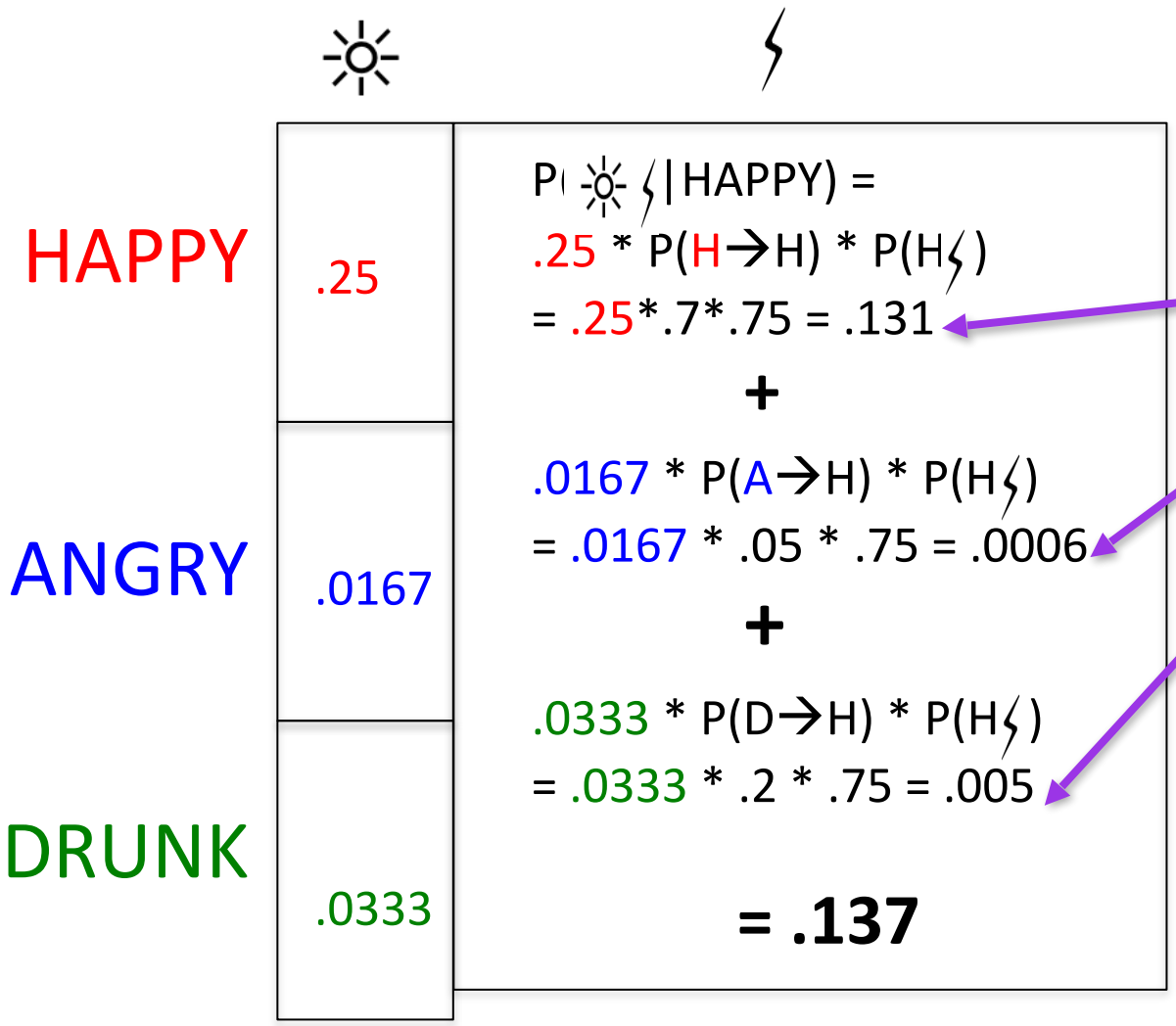
Forward algorithm: 2nd column



Forward algorithm: 2nd column



2nd column, HAPPY cell



- Before: Viterbi chose max of the 3 terms (.131, .0006, and .005)
- With the Forward Algorithm, take the sum of all 3
- = .137 (was .131)
- That's not much difference, but it grows as the algorithm progresses

For any cell after the 1st column:
Call the state for that cell “T”
Suppose corresponding emission is “x”

$$P = \sum_{S \text{ in } \{\text{all states}\}} P(S \text{ in previous column}) * P(S \rightarrow T) * P(x|T)$$

Same as Viterbi, but use summation rather than maximum

FA, Last Step

- Recall: with Viterbi, find cell in last column with maximum score, then trace back to find Viterbi path.
- With FA, compute total score of all cells in last column.
- Because we care about all possible ending states, not just the best.
- No traceback

Finishing Viterbi

HAPPY	.025	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

Max score

Trace back from here

Finishing FA
No “From” notes, no traceback

	☼	⚡	⚡
HAPPY	.025	.0091	5.0E-4
ANGRY	.0166	.057	.03
DRUNK	.0333	.0037	.0017

.0005 + .03 + .0017 = .032

➔ P(☼⚡⚡) = .032

More formally:

P(☼⚡⚡ | this HMM) = .032

Using the FA

- Suppose you have an amino acid sequence that you want to identify.
- You think it's either COI, β-globin, or nifH, and you have an HMM for all of these.
- Evaluate FA on your sequence using each HMM. If any 1 score is >> the others, you have an identification
- Sometimes better results than BLASTp vs GenBank

Wouldn’t Viterbi give the same results?

- For each gene/HMM, if P(Viterbi path) >> P(next best path), then Viterbi score ~= FA score.
- If the 2nd-best, 3rd-best, and 4th-best paths (etc.) have scores that are not much < Viterbi score, then Viterbi doesn’t accurately represent P(seq|HMM).

Example (assume 4 paths through each HMM)

	P(Viterbi Path)	P(2nd-best Path)	P(3rd-best Path)	P(4th-best Path)	FA score
COI	.5	.0005	.0005	.0005	.5015
B-globin	.25	.24	.24	.24	.97
nifH	.45	.0005	.0005	.0005	.4515

- If you look at Viterbi score, it's COI, with nifH as a close 2nd-place
- Actually, beta-globin is 2x better than the others
- For classification, prefer Forward over Viterbi

There's also a Backward Algorithm

- Computes same probability as FA
- Starts from last column of the grid and proceeds to the left
- We won't go into it
- Be aware that it exists, and produces the same results as FA

Limitation of FA and BA

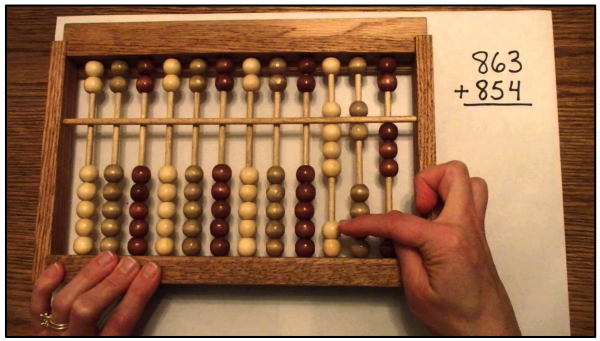
- Use a big HMM to evaluate a long sequence using FA
- Long sequences have smaller scores than short sequences
  - Score is multiplied by 1 transition probability and 1 emission probability per amino acid
- What happens when score is ~= 5x10^-324?
  - If average transition*emission at any stage ~= .1, then this happens after ~324 aas, which is a reasonable length for a sequence

5x10^-324: A special number

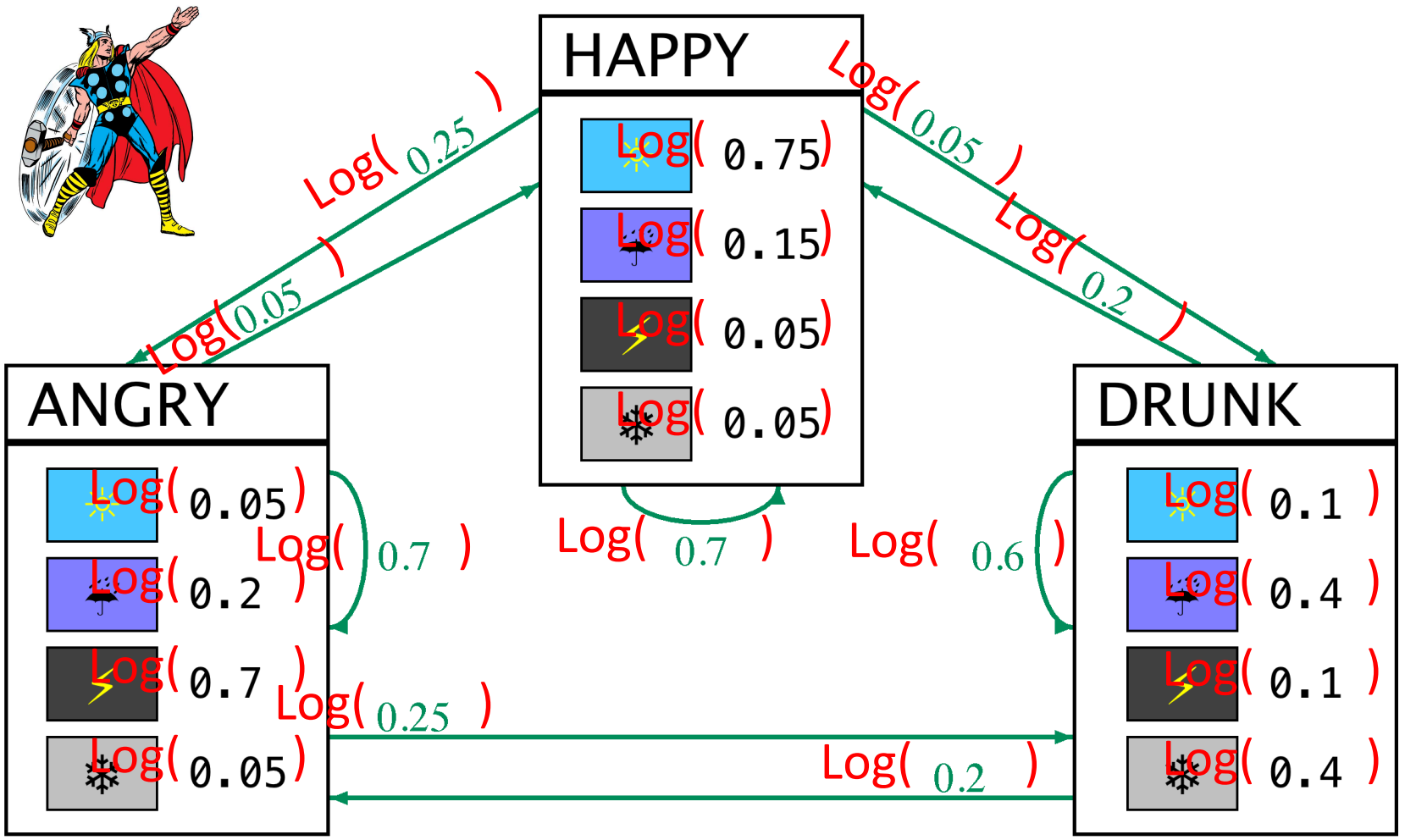
- The smallest fraction that most computers can represent
- If you multiply this by *any fraction*, the result will be rounded down to zero
- Viterbi or Forward score(any long enough sequence | any realistic HMM) will be wrongly reported as zero
- There's an easy fix for Viterbi
- ***No fix is possible for FA (or BA)***

Remember logarithms?

- log(x) = the power to which we raise 10, in order to get x
- 10 ^ log(x) = x
- log(100) = 2, log(1000) = 3, log(1/10) = -1
- log(a*b) = log(a) + log(b)
- Adding is easier than multiplying, especially on an abacus in 1614



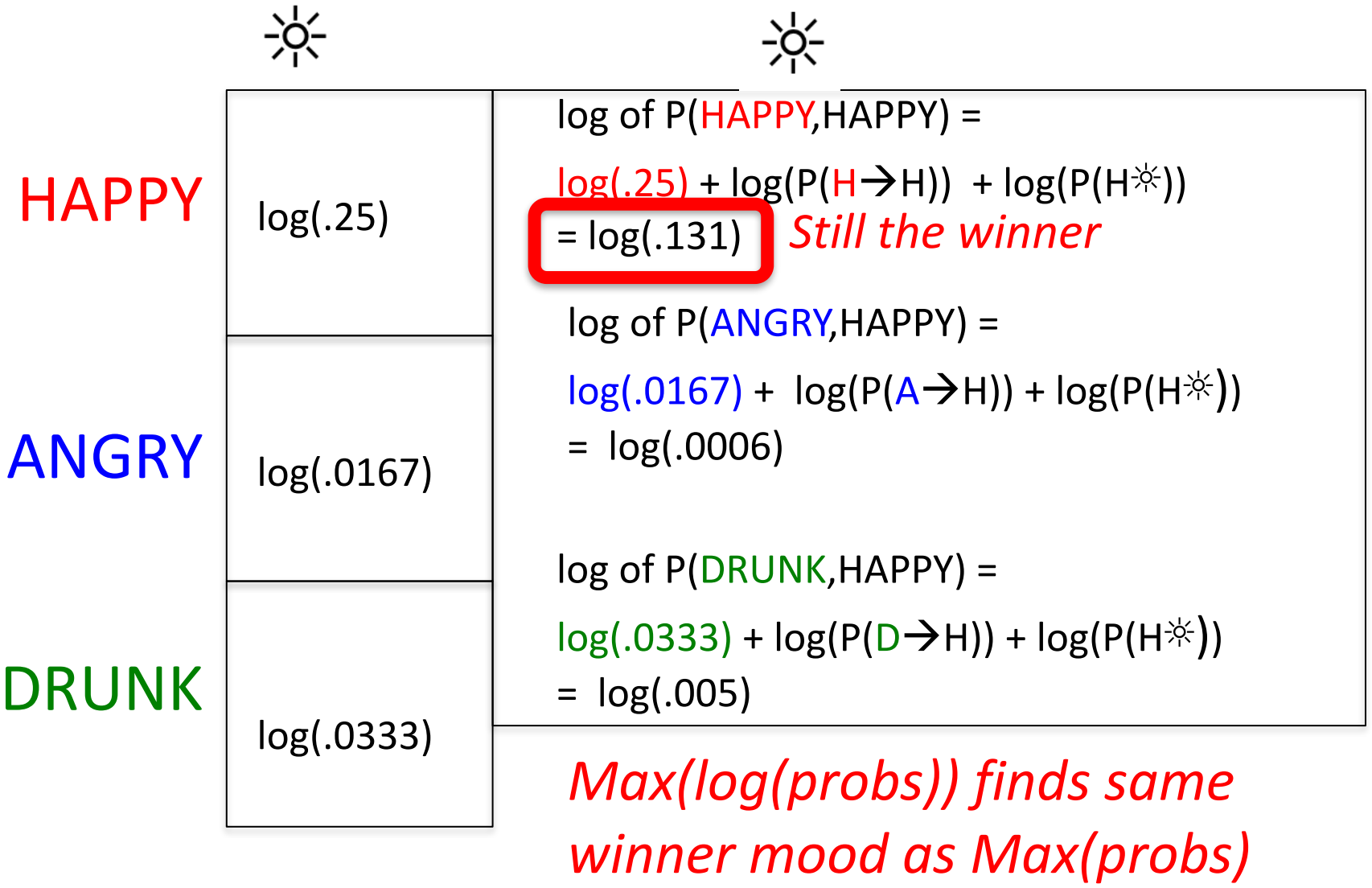
What if the HMM uses log(probability) instead of probability, for all transitions and emissions?



Viterbi would compute/store log(prob) rather than prob

	☼	☼	⚡	❄
HAPPY	Log(.025)	Log(.13) From HAPPY	Log(.0046) From HAPPY	Log(1.6E-4) From HAPPY
ANGRY	Log(.0166)	Log(.0031) From HAPPY	Log(.023) From HAPPY	Log(8.0E-4) From ANGRY
DRUNK	Log(.0333)	Log(.002) From DRUNK	Log(6.6E-4) From HAPPY	Log(.0023) From ANGRY

General Viterbi cell, with logs



Path would be the same

HAPPY	Log(.025)	Log(.13) From HAPPY	Log(.0046) From HAPPY	Log(1.6E-4) From HAPPY
ANGRY	Log(.0166)	Log(.0031) From HAPPY	Log(.023) From HAPPY	Log(8.0E-4) From ANGRY
DRUNK	Log(.0333)	Log(.002) From DRUNK	Log(6.6E-4) From HAPPY	Log(.0023) From ANGRY

Final score would be the same, after conversion

- Normal score = prob * prob * … * prob = a very small positive fraction
  - E.g. 10^-100
- Log score = log(prob) + log(prob) + … + log(prob) = a moderate size negative number
  - E.g. -100
  - Raise 10 to this power to recover original score
  - ***IF*** your computer/calculator can handle such a small fraction

General FA cell, with logs

log of P(HAPPY,HAPPY) =

log(.25) + log(P(H→H)) + log(P(H☼))
= log(.131)

log of P(ANGRY,HAPPY) =

log(.0167) + log(P(A→H)) + log(P(H☼))
= log(.0006)

log of P(DRUNK,HAPPY) =

log(.0333) + log(P(D→H)) + log(P(H☼))
= log(.005)

- Viterbi found max(.131, .0006, .005)
- Modified Viterbi found max(log(.131), log(.0006), log(.005))
- FA found .131+ .0006+ .005
- Modified FA needs log(.131+ .0006 + .005)

Modified FA needs log(.131+ .0006 + .005)

- We have log(.131), log(.0006), log(.005)
- There is no formula relating log(x), log(y), log(z), and log(x+y+z)

What we want, what we settle for

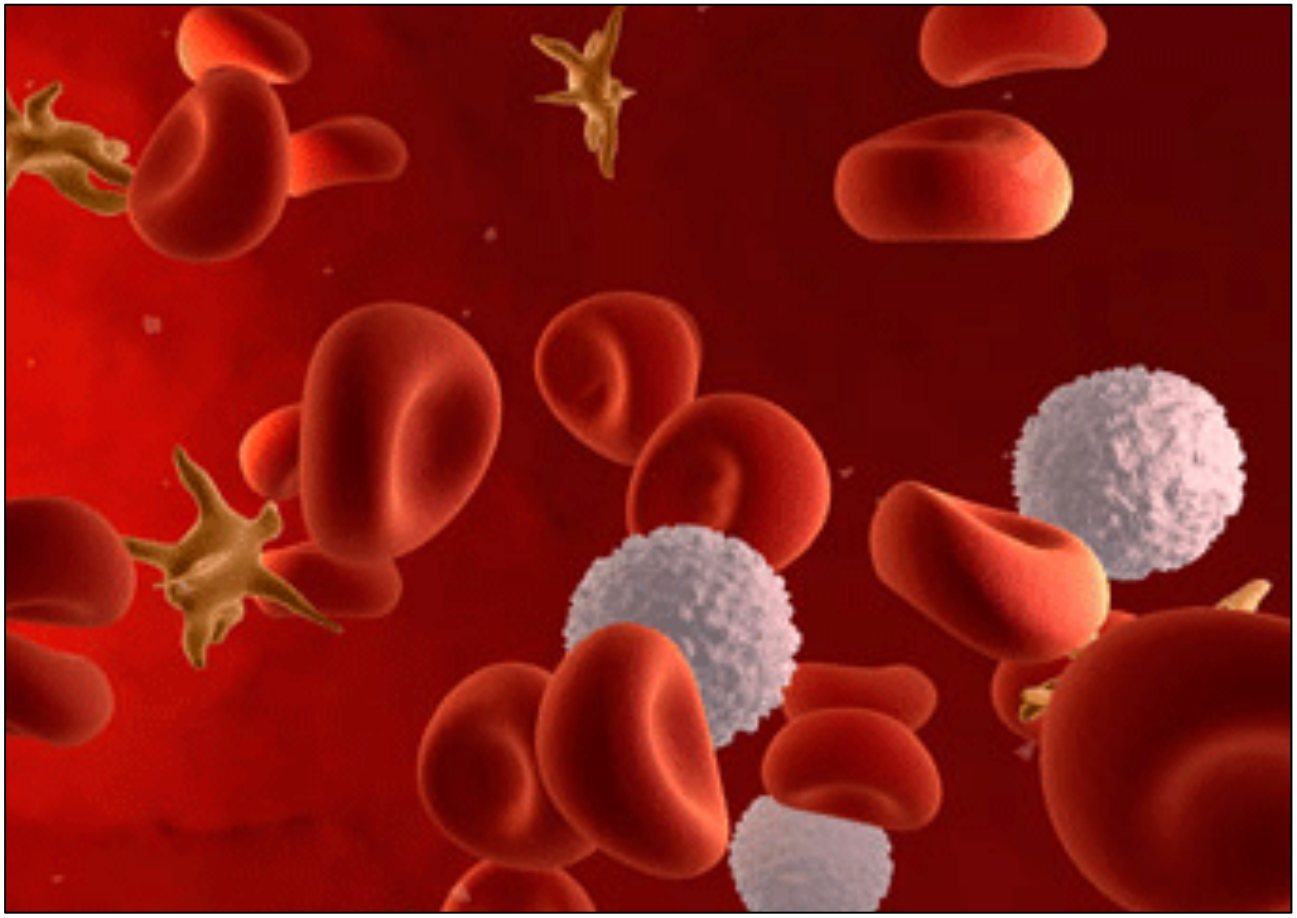
- What we want
  - Given a sequence and some HMMs, compare probabilities that the HMMs emitted the sequence
  - Requires Forward Algorithm
- What we settle for (when sequence is too long)
  - Given a sequence and some HMMs, compare Viterbi probabilities for the sequence from each HMM
  - Use Viterbi probability (probability of best path) as a "proxy" for sum of probabilities of all paths

Now you know where thunder comes from

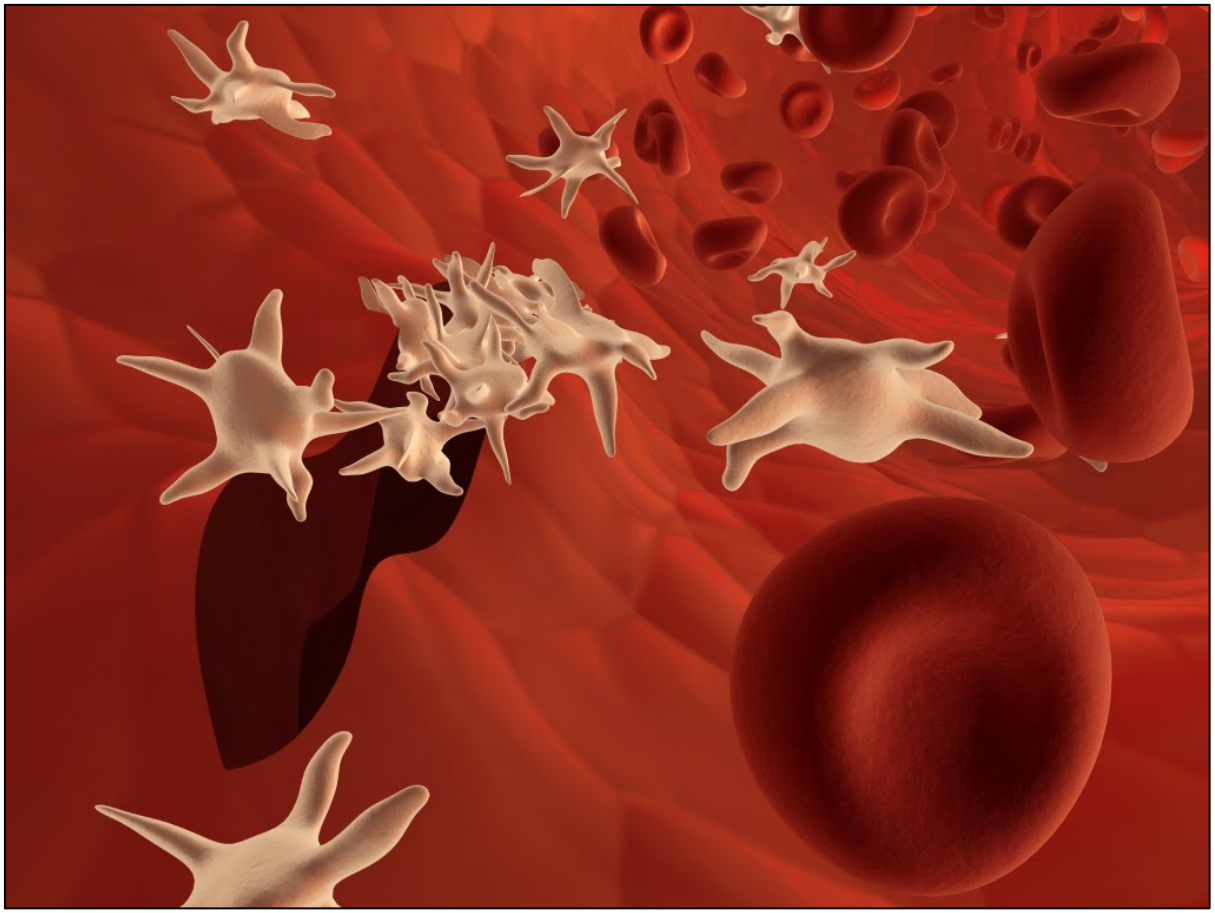
Protein HMMs

- In bioinformatics, most HMMs are about protein sequences, not nucleotide sequences.
- Model proteins using HMMs
  - "Profile" HMMs
  - Identify sequences by checking P(seq|HMM)
- Model protein families using HMMs
  - Composite HMMs

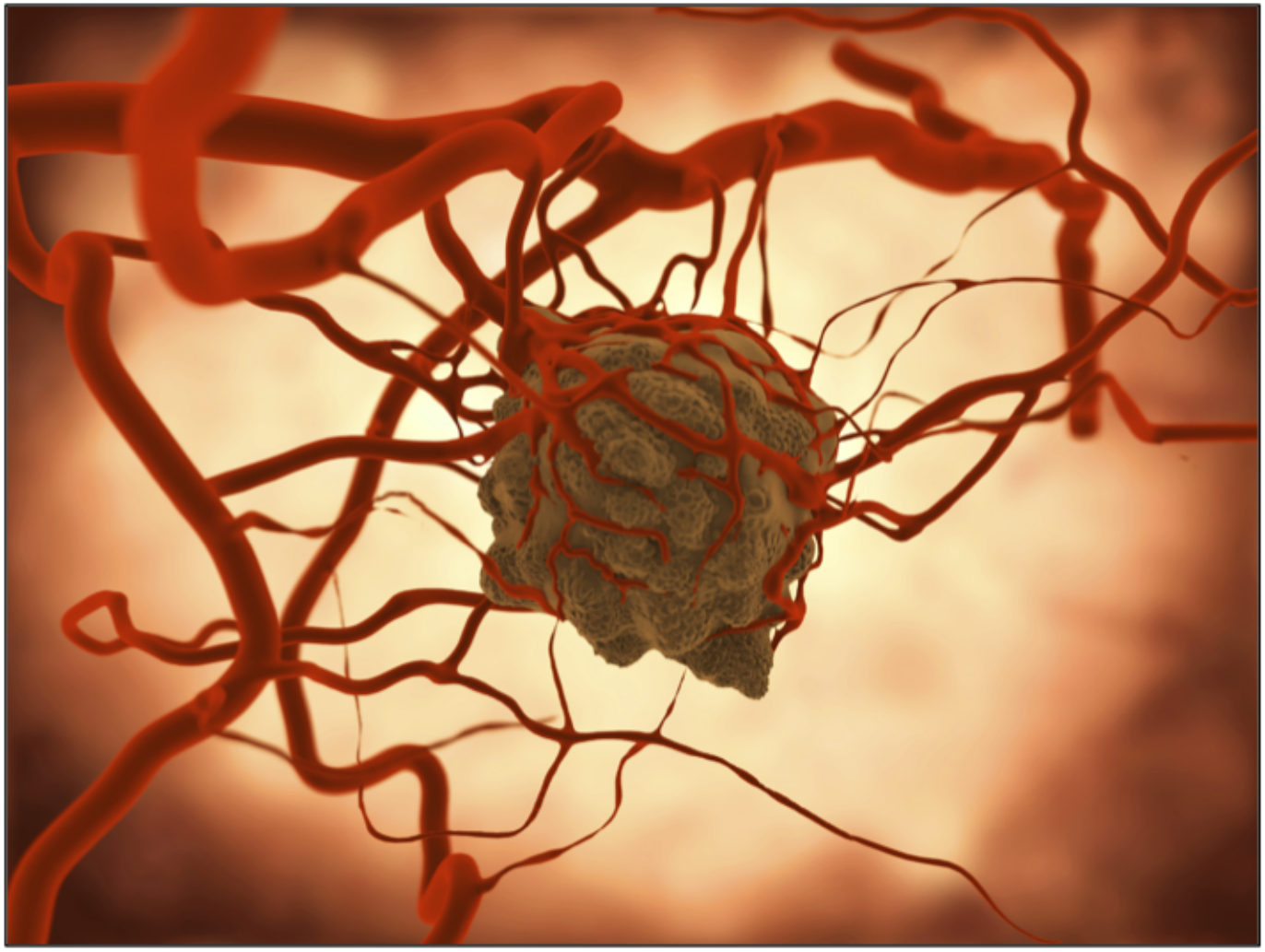
Let's start with an example: Platelet-derived growth factor (PDGF)



Platelets heal wounds by secreting a hormone that stimulates growth



Tumors subvert PDGF to recruit blood vessels and steal resources

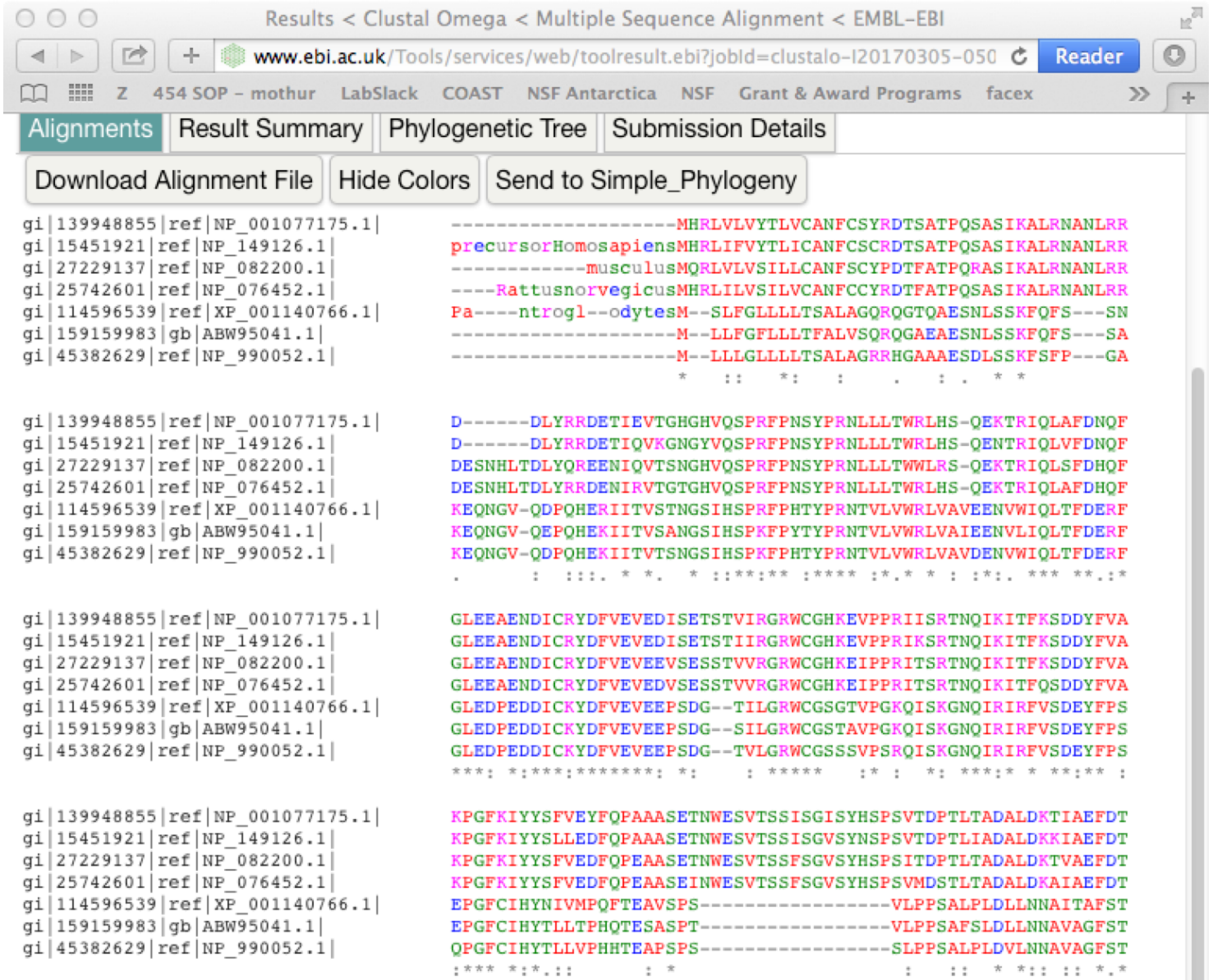


How can we identify PDGF in a tumor genome?

- Bioinformatically identify PDGF genes
- Bioinformatically translate to protein sequences
- BLAST against a reference database of known PDGF proteins?
  - Might work, might not
  - Tumor genomes mutate wildly ➔ highly variable domains might lower BLAST scores
- So build a HMM that describes PDGF conserved domains
  - If a gene evolved from PDGF but lost the ability to promote growth, it's not a threat
  - The real threat: mutated PDGF genes that can still function, because they kept their conserved domains
  - Compute FA score (if possible) else Viterbi score of candidate tumor genes
  - Investigate high-scoring sequences

Step 1: Collect a training set (search GenBank)

Step 2: Align (e.g. ClustalΩ)



Step 3: Look for conserved domains

Step 4: Build a profile Hidden Markov Model (pHMM)

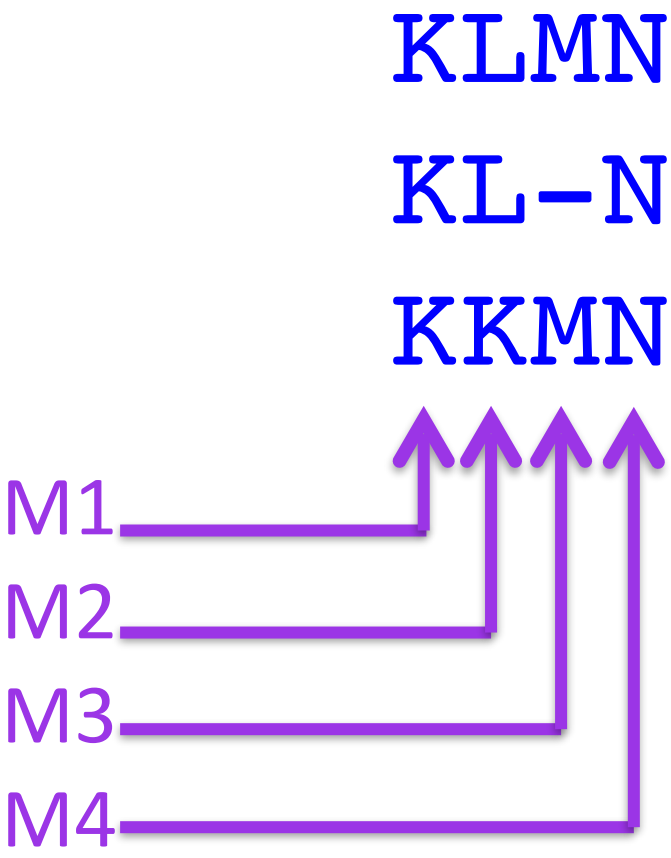
To build a protein HMM …

- Collect trusted representatives of the protein you want to model
  - "Positive training set"
- Align the positive training set

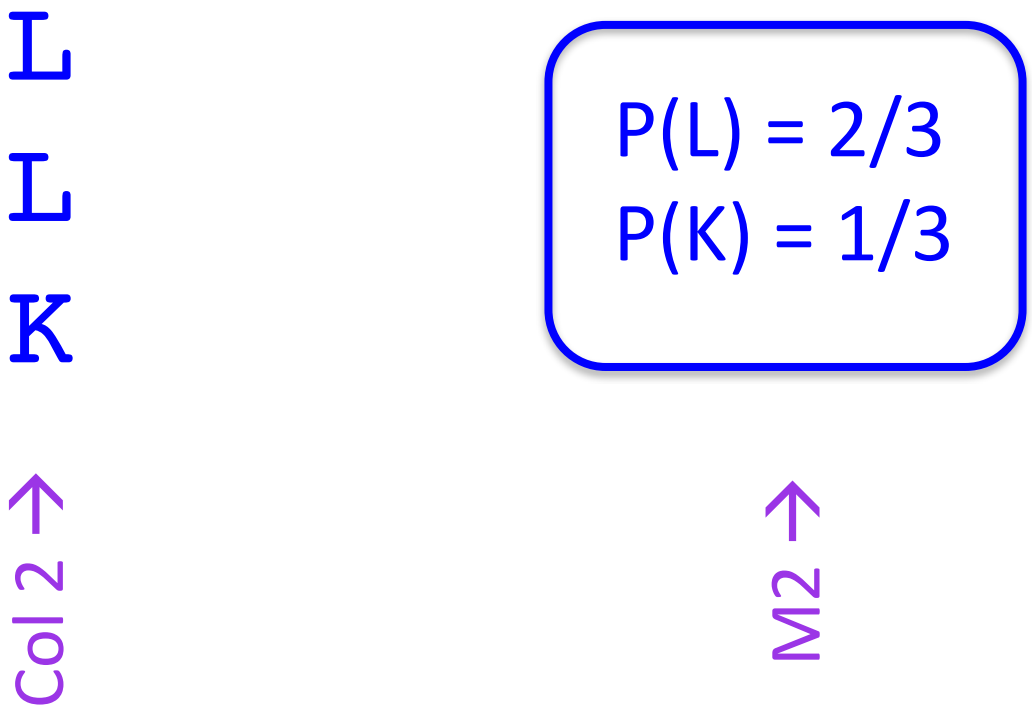
KLMN, KLN,KKMA →

```
KLMN
KL-N
KKMN
```

Each alignment column becomes a state



Frequency of char in col becomes its emission probability



Do that for every column

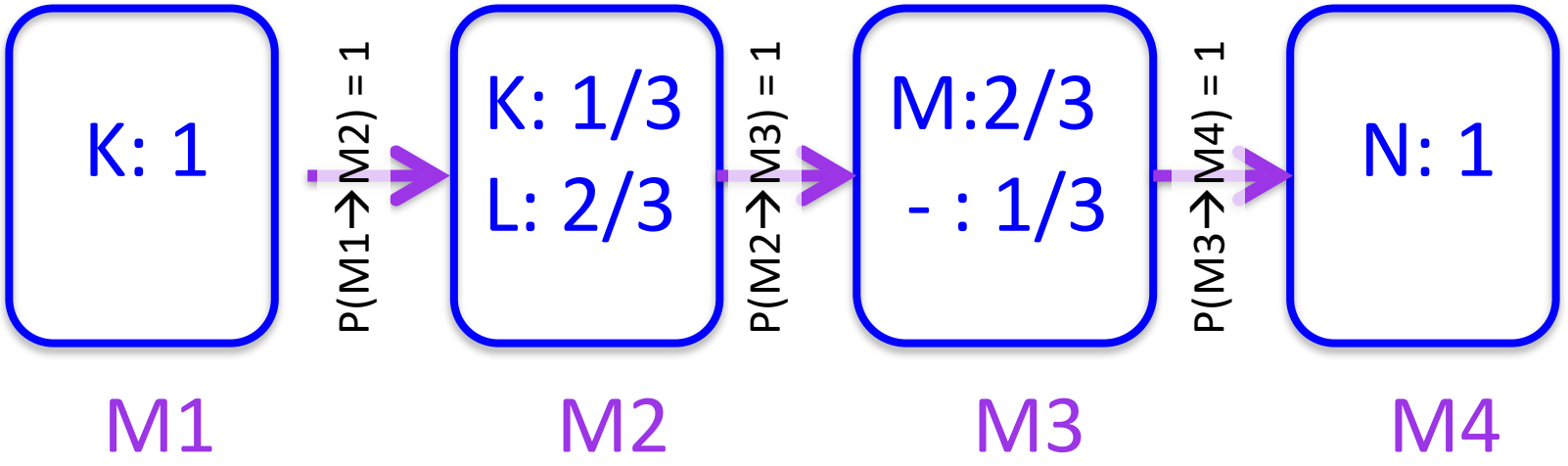
KLMN

KL-N →

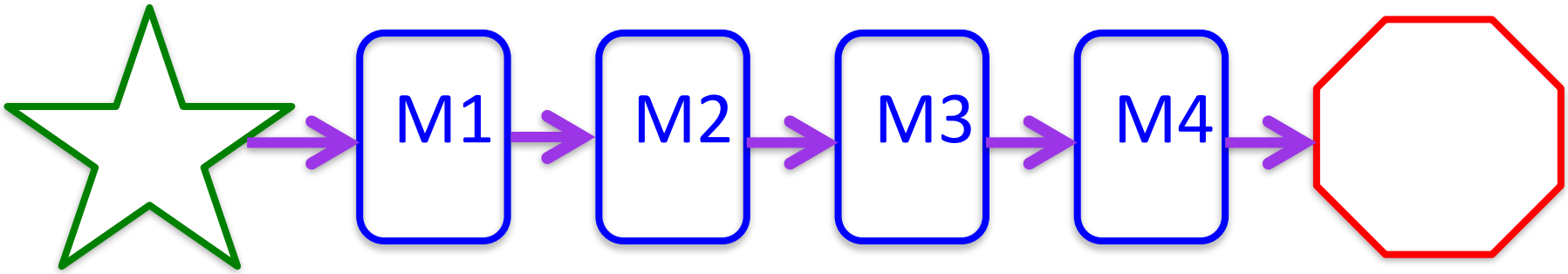
KKMN

Transitions: Only go from one column's state to the next column's state

```
KLMN
KL-N →
KKMN
```

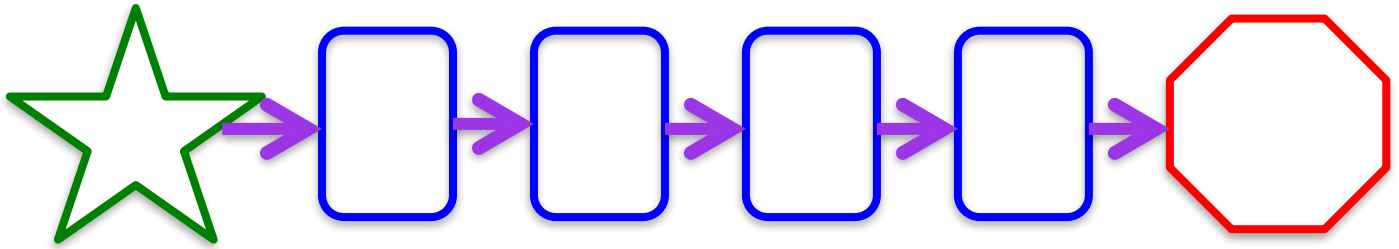


Special states: Start & Stop



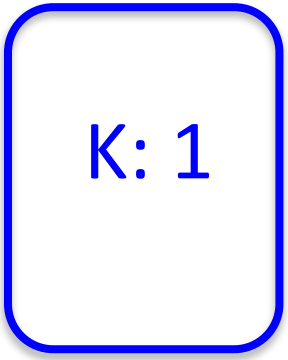
- Start:
  - All paths start here
  - No emissions
  - Transitions from here reflect initial probs
- Stop
  - All paths end here
  - No emissions

Too Simple: 3 problems



- P(Any sequence containing aas not represented in the positive training set) = 0
- Indel isn't really an emission in the same sense as the amino acids
- This HMM can only handle sequences of length=4
  - P(Any sequence of any other length) = 0

K = Lysine: Polar, hydrophilic



M1

- Similar to Arginine (R) & Histidine (H)
- P(R|S1) = P(H|S1) = 0
- This model says that in state S1, it is *impossible* for R or H to appear
  - Anywhere on the planet
  - Or any other planet
  - Ever
- Model gives high score for KLMN
- Model gives zero score for HLMN, should give low score
  - Unless you're 100% certain that H is impossible in column 1

Pseudocounts

- A more realistic column from positive training set alignment might have counts like this:
  - A=100, K=120, L=200, M=50
  - Training set size was 470
  - P(A|this state) = 100/470 = .213
  - P(C|this state) = 0 /470 = 0
- Pretend the column also contained 1 of every unrepresented amino acid
  - C = D = E = F = G = H = N = P = Q = R = S = T = V = W = Y = 1
  - P(A|this state) = 100/486 = .206
  - P(C|this state) = 1 / 486 = .00206

Like a tax on probabilities of represented aas

Pseudoprobabilities

- Like pseudocounts, but tax probabilities rather than frequencies
- For each state, decide on a small value ε for P(emit *any* unrepresented aa|that state)
  - E.g. .01 or .001
- Reduce P(every represented amino acid|that state) by same small amount, total = ε
- Set P(every unrepresented aa|that state) to same small amount, total = ε

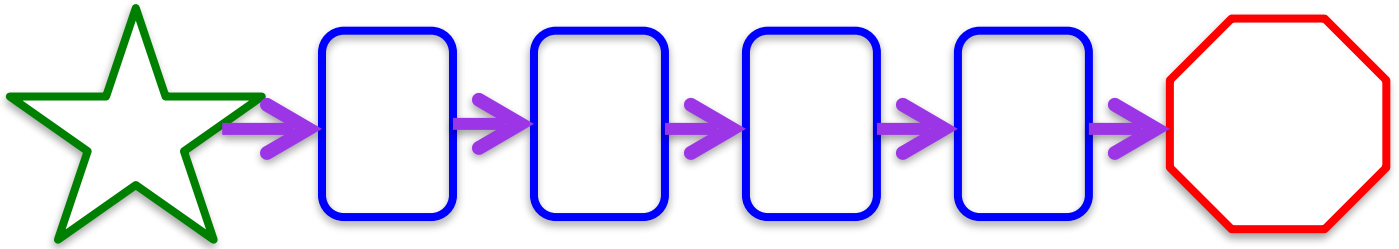
Pseudoprobabilities Example

- A=100, L=250, M=60, P=90
  - Original P(A) = 100 / (100+250+60+90) = 100/500 = .2
  - Original P(L) = 250/ (100+250+60+90) = 250/500 = .5
  - Original P(M) = 60/ (100+250+60+90) = 60/500 = .12
  - Original P(P) = 90/ (100+250+60+90) = 90/500 = .18
  - Orig P(A) + Orig P(L) + Orig P(M) + Orig P(P) = 1
- Decide that ε = .01
- 4 represented aas → reduce P(every represented aa) by ε/4 = .0025
  - P(A) = .1975, P(L) = .4975, P(M) = .1175, P(P) = .1775
- 16 unrepresented aas → Set P(every unrepresented aa) to ε/16 = .00015625
- Total emission probs from this state still = 1

With pseudocounts or pseudoprobabilities

- Any state can emit any aa, though some emission probabilities are small
- Sequences that would have zero probability now have positive probability

Too Simple: 3 problems



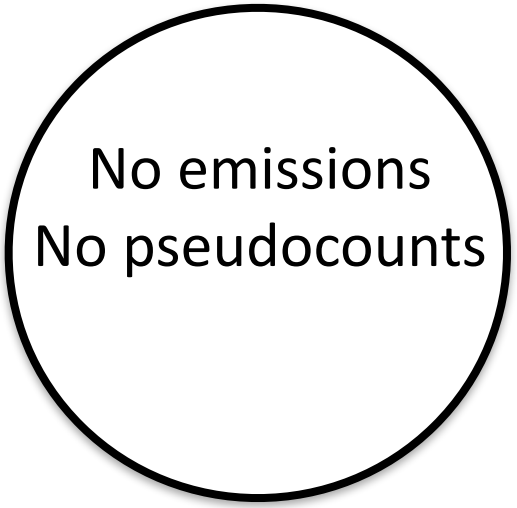
- P(Any sequence containing aas not represented in the positive training set) = 0 ✔
- Indel isn't really an emission in the same sense as the amino acids ?
- This HMM can only handle sequences of length=4
  - P(Any sequence of any other length) = 0

Gaps in the alignment

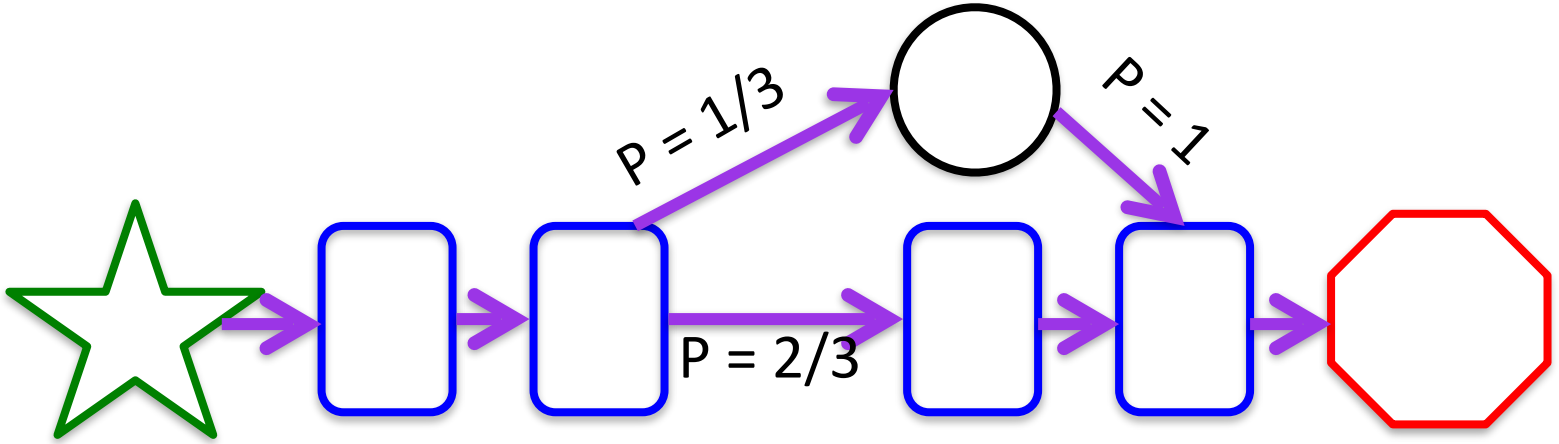
```
KLMN
KL-N
KKMN
```

- Biochemically different from amino acids.
- A gap has no physical existence.
- Don’t represent in alignment-col states (“match states”).
- Represent in special “delete states”, usually drawn as circles.

A Delete State



```
K   L     M   N
K   L     -   N
K   K     M   N
```

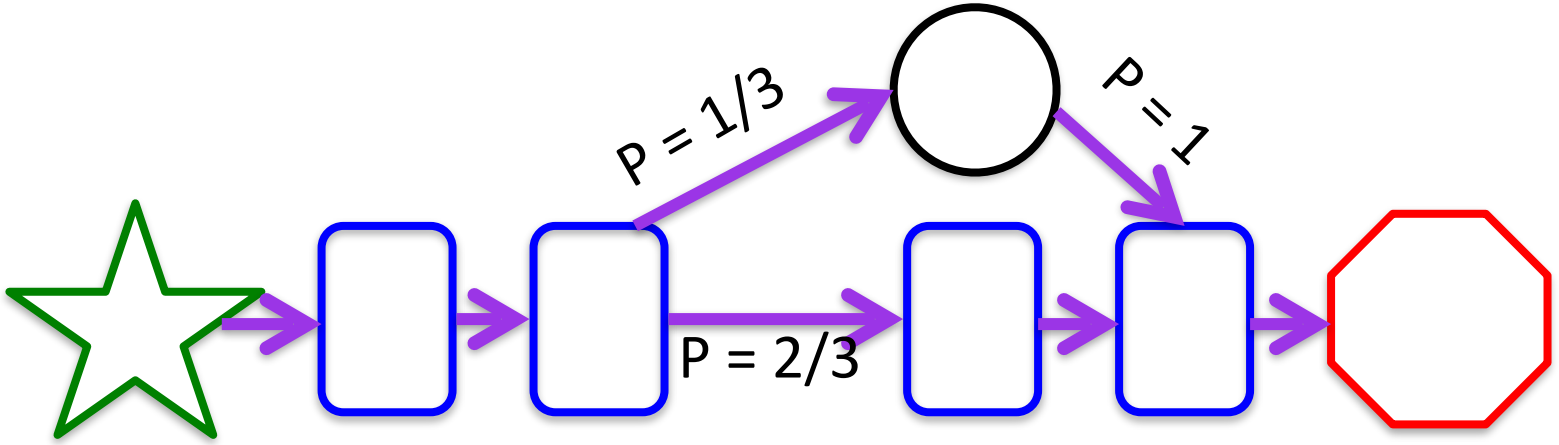


What about a more complicated alignment?

K L M N

K L – N

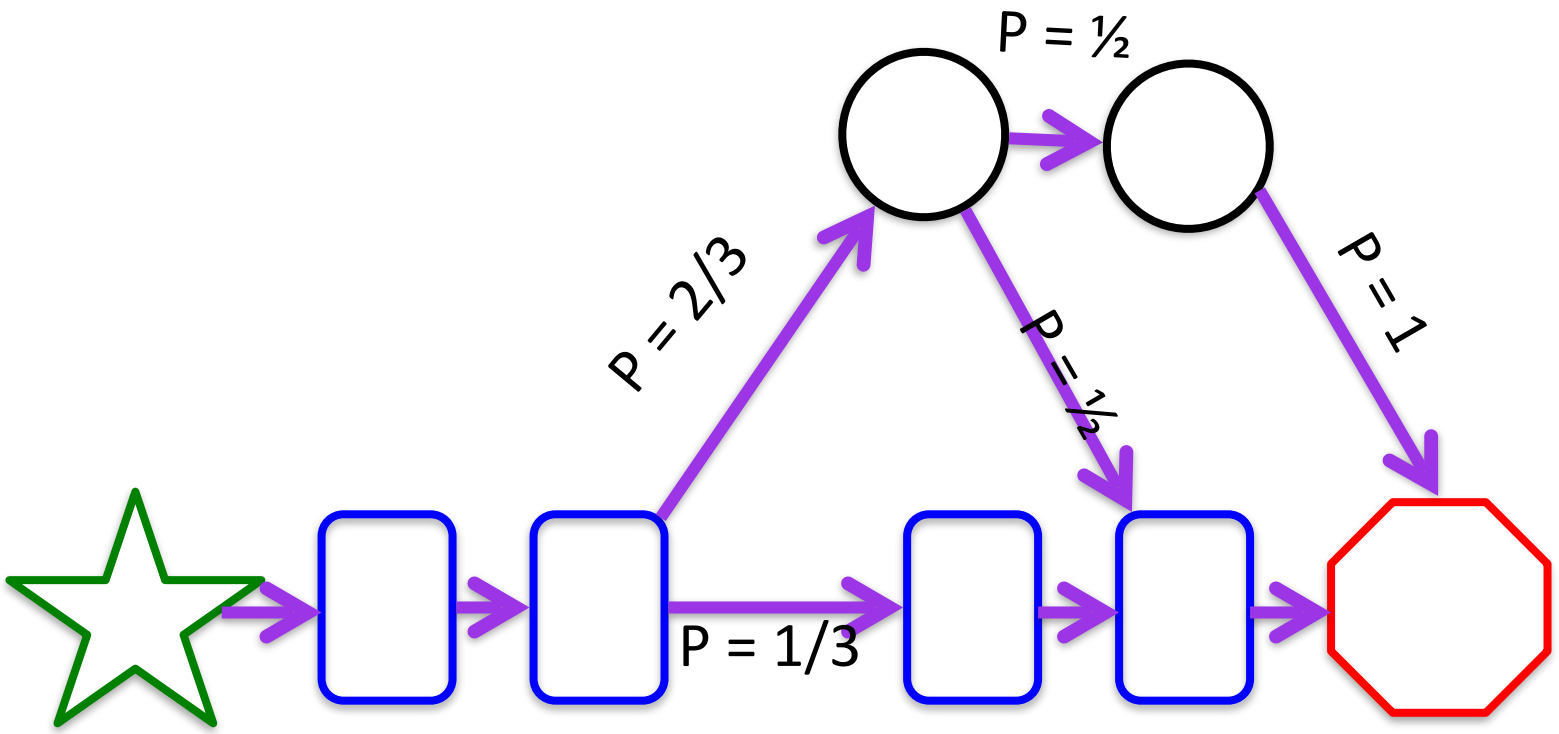
K K – – ???



After state M2:

- P(length=4) = 1/3
- P(len=3, skip M3) = 1/3
- P(len=2, skip M2&M3) = 1/3

```
K  L  M  N
K  L  –  N
K  K  –  –
```

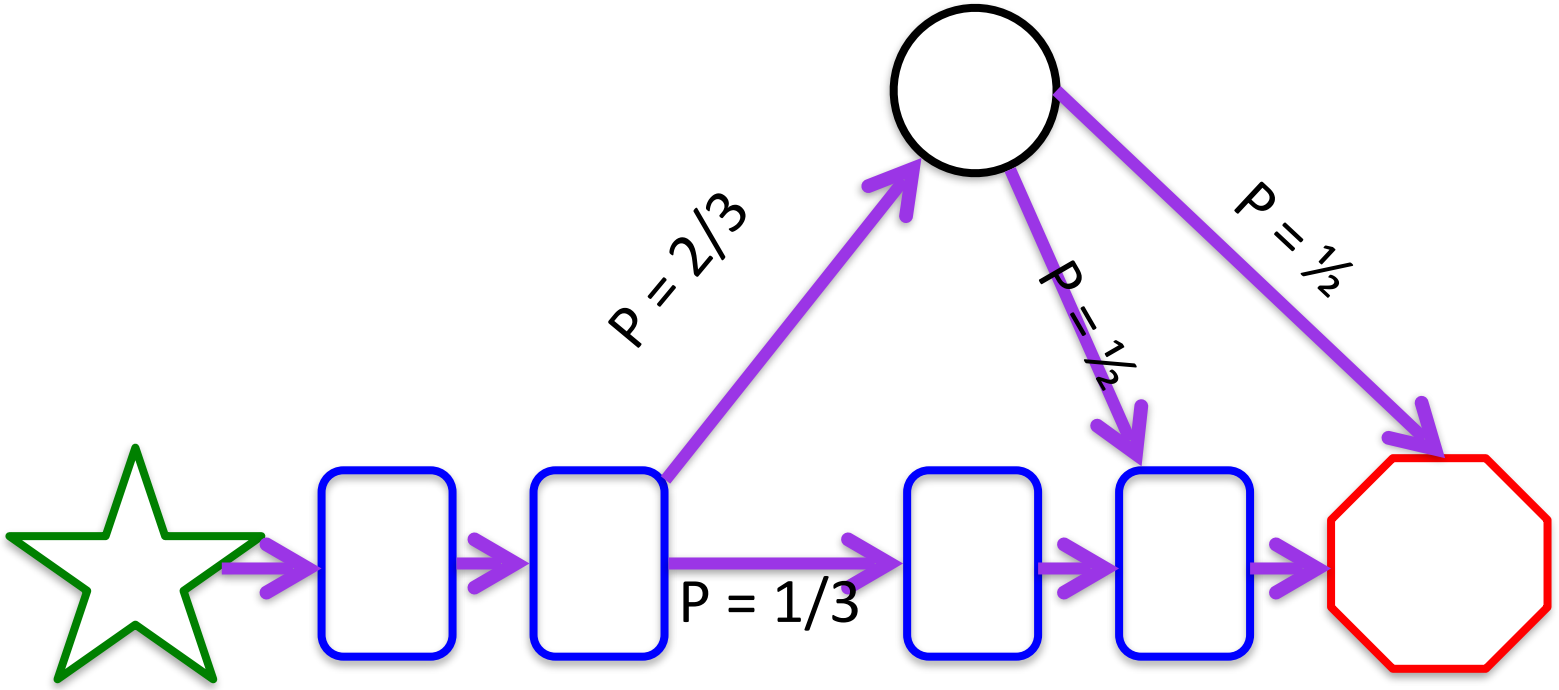


or, more simply …

After state M2:

- P(length=4) = 1/3
- P(len=3, skip M3) = 1/3
- P(len=2, skip M2&M3) = 1/3

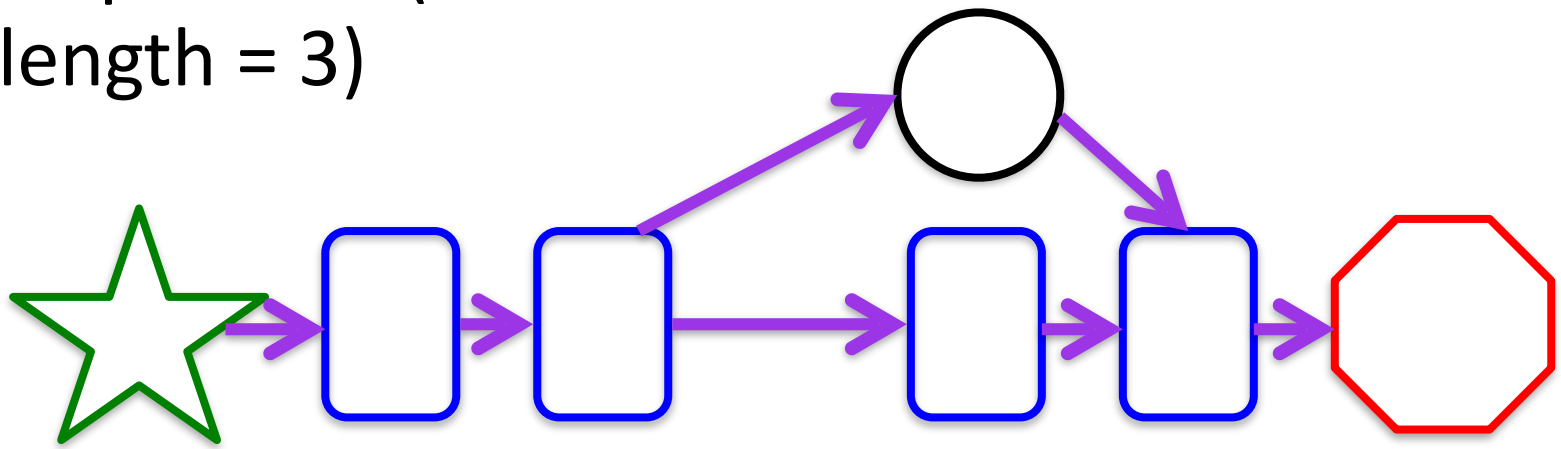
```
K  L  M  N
K  L  -  N
K  K  -  -
```



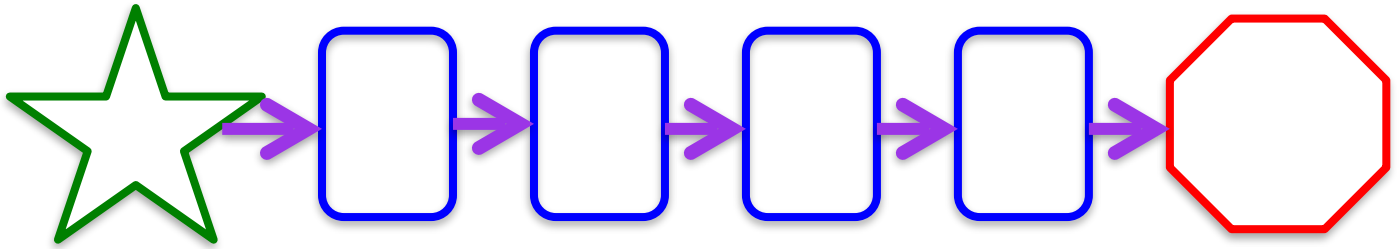
or, more simply …

- Indel indicates a sequence that is shorter than the alignment of the training set.
- Delete states let HMM deal with shorter sequences. (Here: length = 3)

```
K  L  M  N
K  L  -  N
K  K  M  N
```



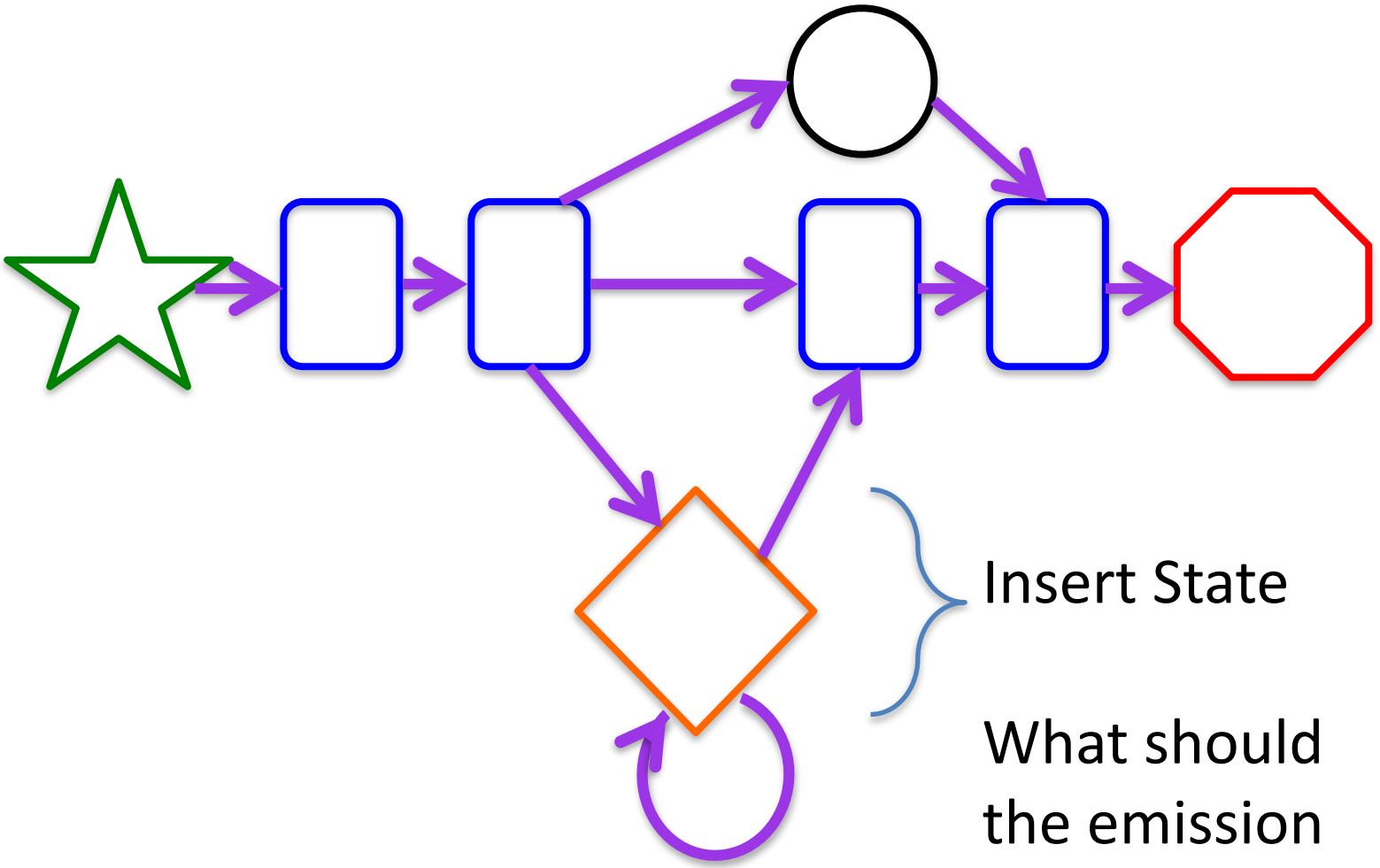
Too Simple: 3 problems



- P(Any sequence containing aas not represented in the positive training set) = 0 ✔
- Indel isn't really an emission in the same sense as the amino acids ✔
- This HMM can only handle sequences of length=4 ?
  - P(Any sequence of any other length) = 0

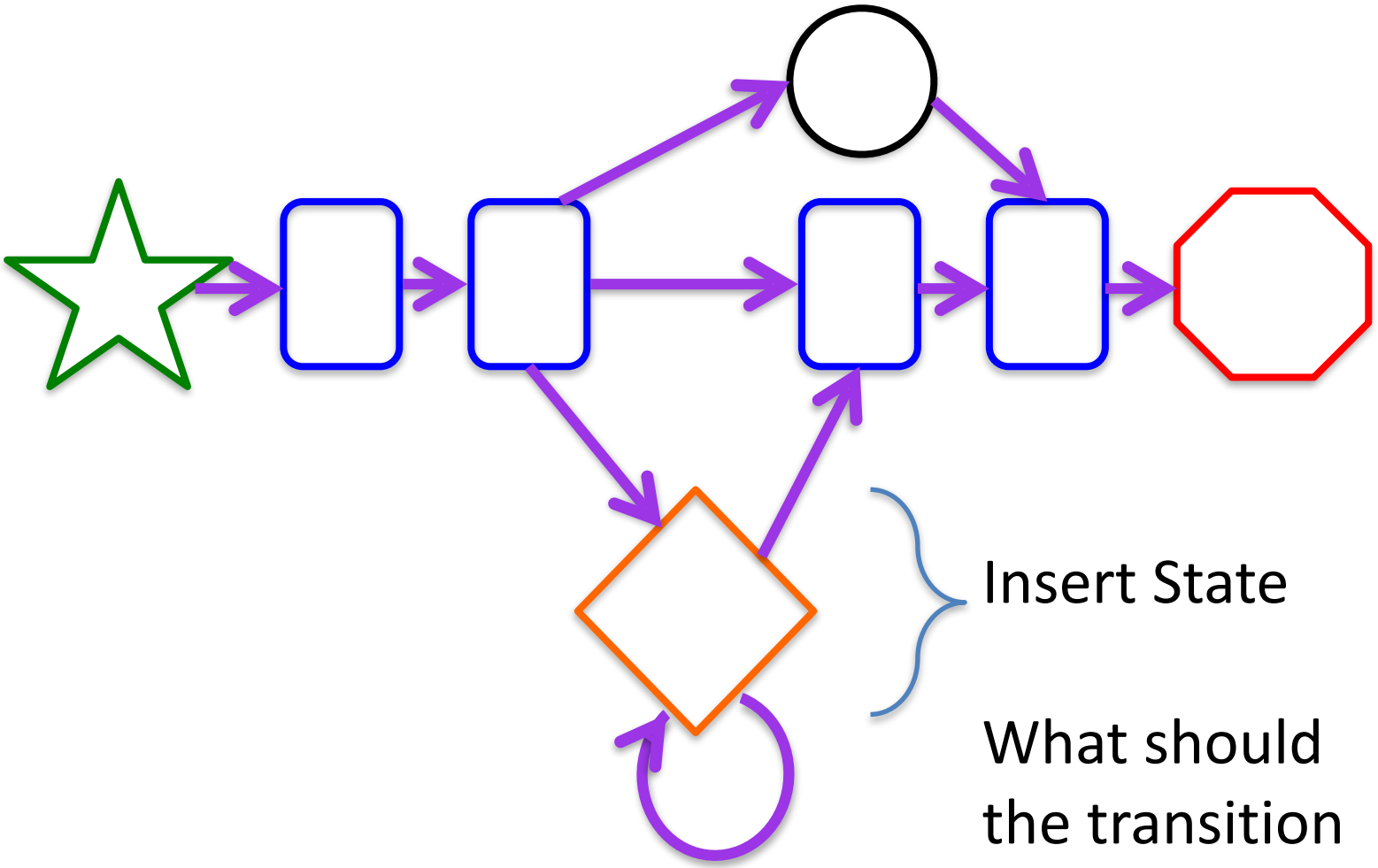
Problem: HMM can only handle sequences of same length as alignment of positive training set

- Delete states take care of shorter sequences
- Another special state takes care of longer sequences:
  - Insert States
  - Usually drawn as diamonds between/below the match states


Insert State
What should the emission probabilities be?

How do we set the emission probabilities for an insert state?

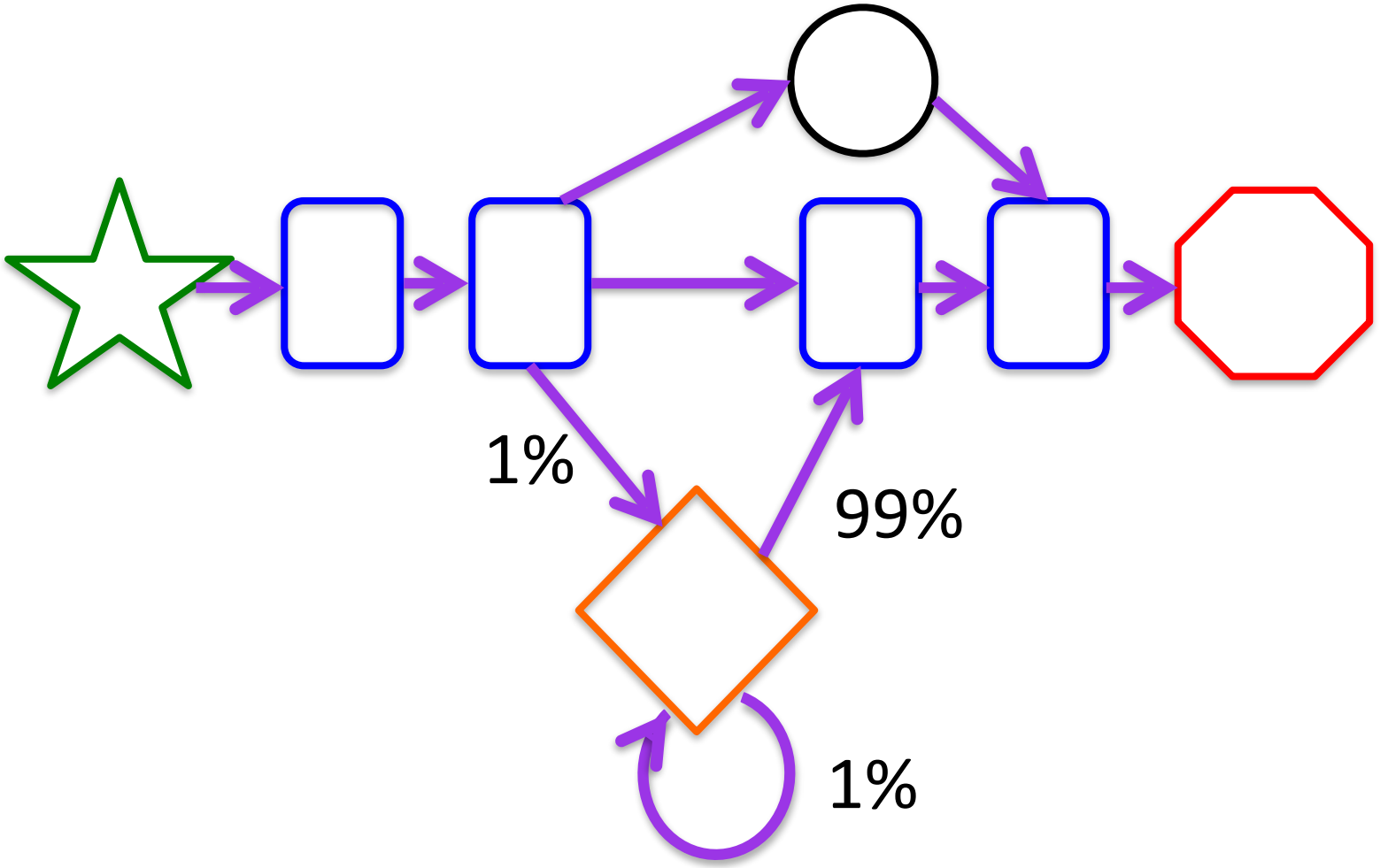
- The match states represent columns in the alignment of the positive training set.
- The insert states represent insertions that don't appear anywhere in the positive training set.
- So we don't know anything about these mysterious insertions.
  - We're just making sure that *if* they happen, the HMM can handle the extra length
- There is no reason to believe any aa is more or less likely to be emitted than any other aa.
- So set all emission probabilities to be equal (.05).
- In pHMM diagrams, such insert states are drawn as empty diamonds


Insert State
What should the transition probabilities be?

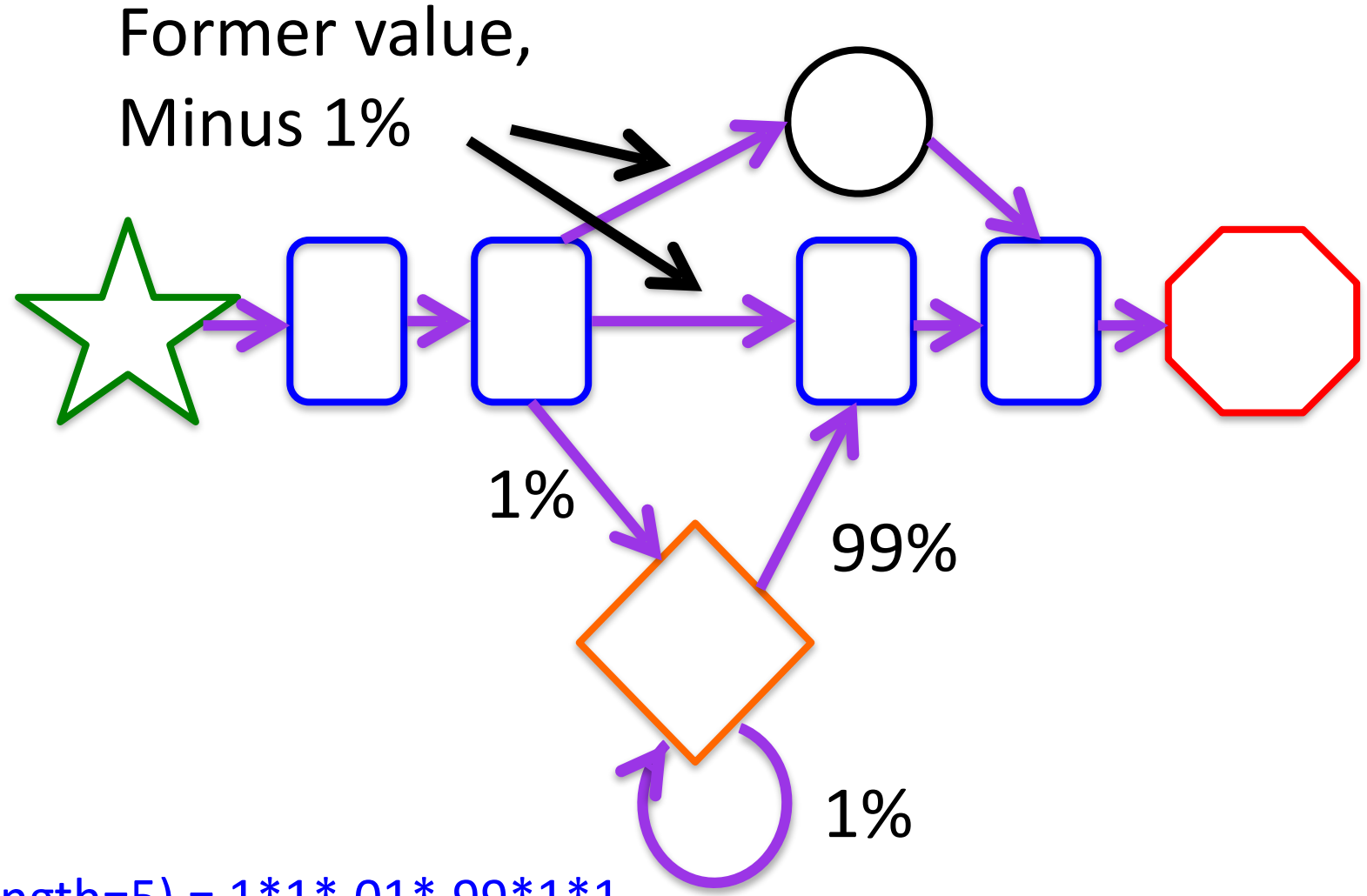
Insert state transition probabilities

- Best we can do is a good guess
- Insert states are for sequences that are longer than the training alignment
- If we had and examples, we would have included some in the training set
  - ➔ Then the alignment would be wider, and we wouldn't need to use Insert States!
- We don't know how long a mysterious "longer sequence is"
  - A good guess: single inserted aa is more likely than 2 consecutive inserted aas, which are more likely than 3, etc.

1% seems to work:



1% seems to work:

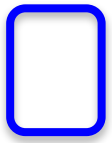


P(length=5) = 1*1*.01*.99*1*1

Profile HMM (pHMM) Algorithms

- A pHMM is just an HMM with certain kinds of states:
  - Start & Stop
  - Match (unbalanced emission probs)
  - Insert (balanced emission probs)
  - Delete (no emission probs)
  - Most transition probs = 0
  - Many paths are START → MATCH*n → STOP
  - pHMM algorithms need to be adjusted to handle these special states

Adjusting the algorithms

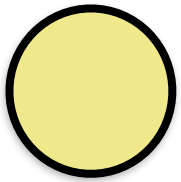


- Match states are normal

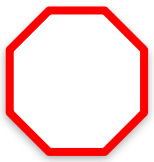
- Insert states are normal

- Start state???

- Delete states???



- Stop state???



Adjusting for start state

- Without a start state, we have a distribution of initial probabilities
  - Ex: P(start in A) = .5, P(start in B) = .4, P(start in C) = .1
- To introduce a start state
  - Initial probability distribution becomes p(start in ☆) = 1
  - Original initial probs become transition probs from
  - P(☆ → A) = .5, P(☆ → B) = .4, P(☆ → C) = .1

Adjusting for delete states

- Don’t think of indels as emissions. They aren’t *things*.
- pHMM can’t emit indels.
- Viterbi, FA, and BA algorithms need straightforward adjustments which we won’t go into.

Adjusting for Stop state: look at final column

	☼	☼	⚡	❄
HAPPY	.025	.13 From HAPPY	.0046 From HAPPY	1.6E-4 From HAPPY
ANGRY	.0166	.0031 From HAPPY	.023 From HAPPY	8.0E-4 From ANGRY
DRUNK	.0333	.002 From DRUNK	6.6E-4 From HAPPY	.0023 From ANGRY

Before you compute max or sum of scores…

HAPPY	1.6E-4 From HAPPY	* P(HAPPY →)
ANGRY	8.0E-4 From ANGRY	* P(ANGRY →)
DRUNK	.0023 From ANGRY	* P(DRUNK→)

This concludes our study of Hidden Markov Models