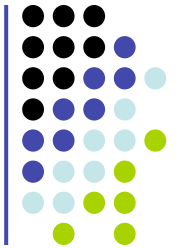


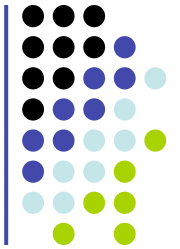
Heuristic Local Alignerers

1. The basic indexing & extension technique

2. Indexing: techniques to improve sensitivity
   Pairs of Words, Patterns

3. Systems for local alignment

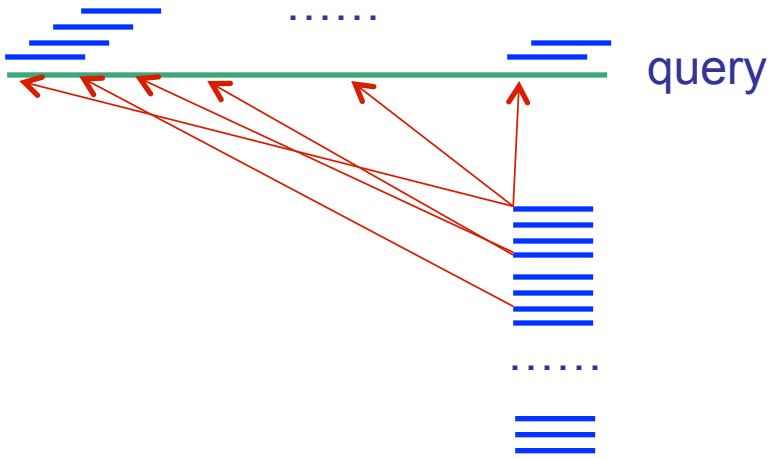
Indexing-based local alignment



Dictionary:

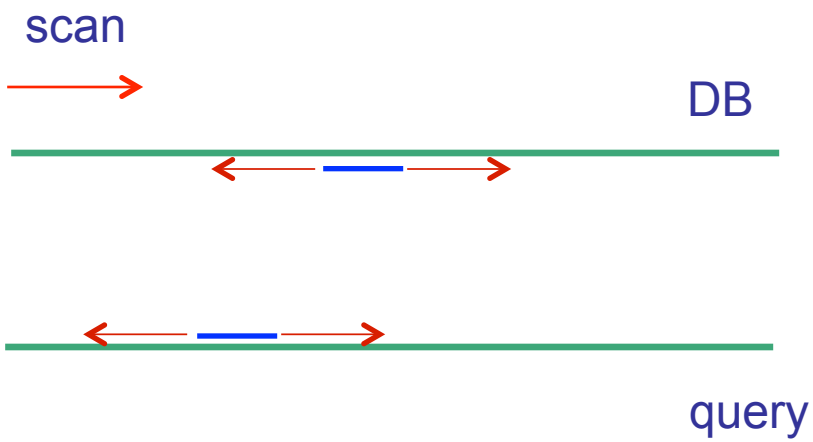
All words of length k (~10)

Alignment initiated between
words of alignment score ≥ T
(typically T = k)



Alignment:

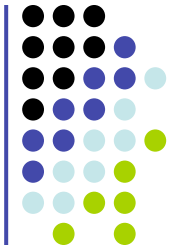
Ungapped extensions until score
below statistical threshold



Output:

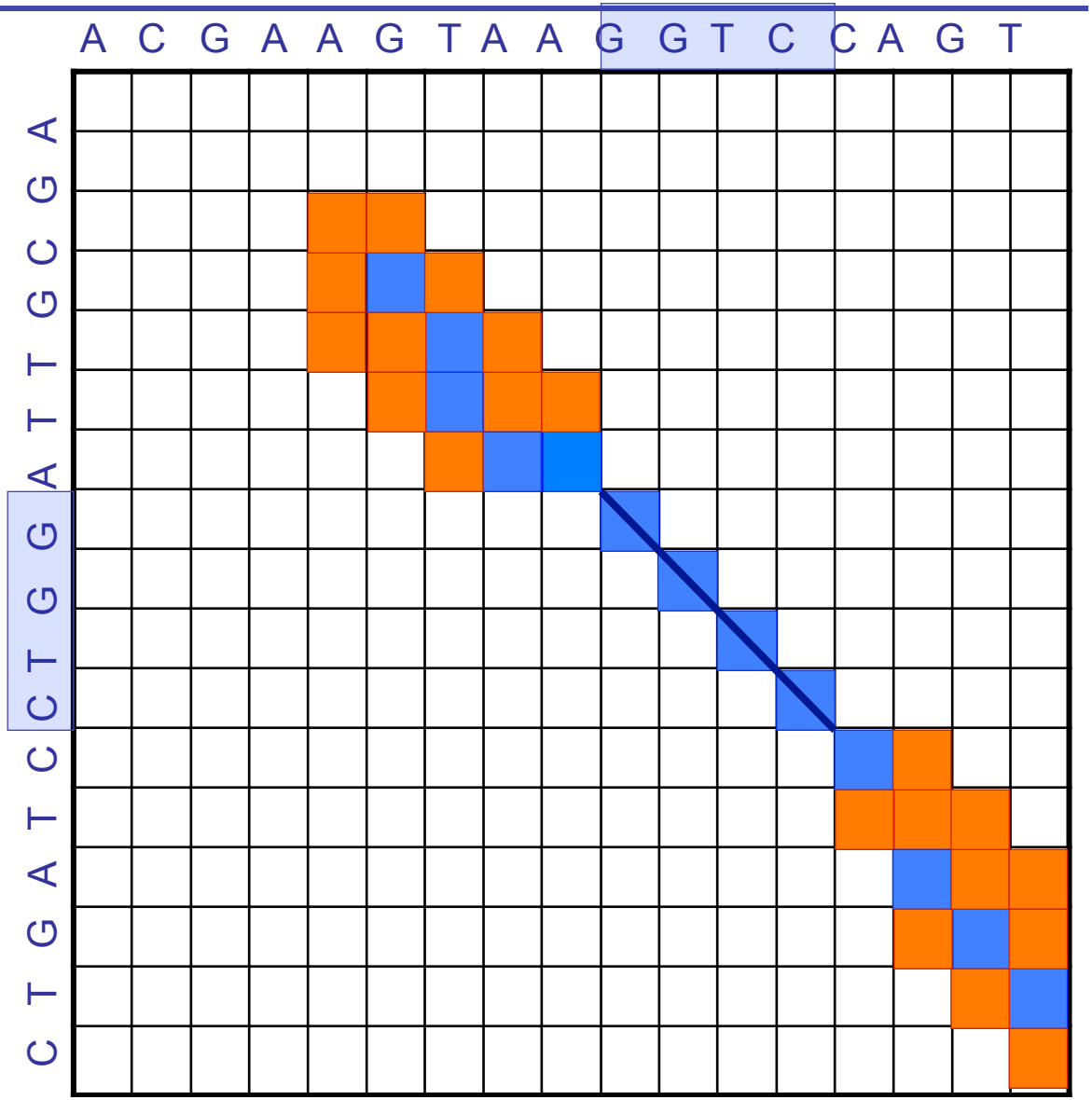
All local alignments with score
> statistical threshold

Indexing-based local alignment—Extensions



Gapped extensions until threshold

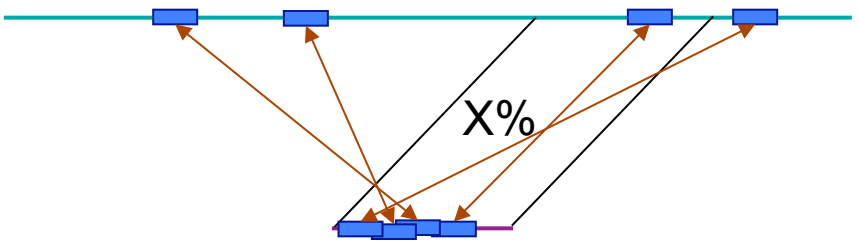
- Extensions with gaps until score < C below best score so far



Output:

```
GTAAGGTCCAGT
GTTAGGTC-AGT
```

Sensitivity-Speed Tradeoff



	long words (k = 15)	short words (k = 7)
Sensitivity		✓
Speed	✓	

Sens.

Speed

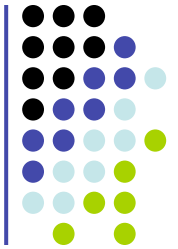
Table 3. Sensitivity and Specificity of Single Perfect Nucleotide K-mer Matches as a Search Criterion

		7	8	9	10	11	12	13	14
A.	81%	0.974	0.915	0.833	0.726	0.607	0.486	0.373	0.314
	83%	0.988	0.953	0.897	0.815	0.711	0.595	0.478	0.415
	85%	0.996	0.978	0.945	0.888	0.808	0.707	0.594	0.532
	87%	0.999	0.992	0.975	0.942	0.888	0.811	0.714	0.659
	89%	1.000	0.998	0.991	0.976	0.946	0.897	0.824	0.782
	91%	1.000	1.000	0.998	0.993	0.981	0.956	0.912	0.886
	93%	1.000	1.000	1.000	0.999	0.995	0.987	0.968	0.957
	95%	1.000	1.000	1.000	1.000	0.999	0.998	0.994	0.991
	97%	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999
B.	K	7	8	9	10	11	12	13	14
	F	1.3e+07	2.9e+06	635783	143051	32512	7451	1719	399

(*A*) Columns are for K sizes of 7–14. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated from equation 3 assuming a homologous region of 100 bases. The larger the value of K, the fewer homologies are detected.
(*B*) K represents the size of the perfect match. F shows how many perfect matches of this size expected to occur by chance according to equation 4 in a genome of 3 billion bases using a query of 500 bases.

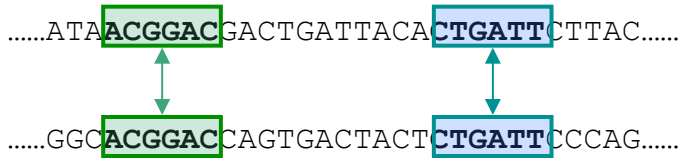
Kent WJ, Genome Research 2002

Sensitivity-Speed Tradeoff

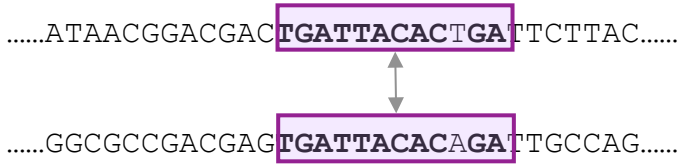


Methods to improve sensitivity/speed

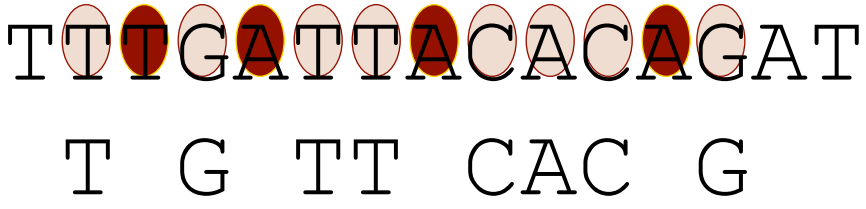
1. Using pairs of words



2. Using inexact words



3. Patterns—non consecutive positions



Measured improvement

Table 7. Sensitivity and Specificity of Multiple (2 and 3) Perfect Nucleotide K-mer Matches as a Search Criterion

	2,8	2,9	2,10	2,11	2,12	3,8	3,9	3,10	3,11	3,12
A. 81%	0.681	0.508	0.348	0.220	0.129	0.389	0.221	0.112	0.051	0.021
83%	0.790	0.638	0.475	0.326	0.208	0.529	0.339	0.193	0.099	0.045
85%	0.879	0.762	0.615	0.460	0.318	0.676	0.487	0.313	0.180	0.093
87%	0.942	0.866	0.752	0.611	0.461	0.809	0.649	0.470	0.305	0.177
89%	0.978	0.940	0.868	0.761	0.625	0.910	0.801	0.648	0.476	0.314
91%	0.994	0.980	0.947	0.884	0.787	0.969	0.914	0.815	0.673	0.505
93%	0.999	0.996	0.986	0.962	0.912	0.993	0.976	0.933	0.851	0.722
95%	1.000	1.000	0.998	0.993	0.979	0.999	0.997	0.987	0.961	0.902
97%	1.000	1.000	1.000	1.000	0.999	1.000	1.000	0.999	0.997	0.987

B. N,K	2,8	2,9	2,10	2,11	2,12	3,8	3,9	3,10	3,11	3,12
F	524	27	1.4	0.1	0.0	0.1	0.0	0.0	0.0	0.0

(*A*) Columns are for N sizes of 2 and 3 and K sizes of 8–12. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated by equation 10. (*B*) N and K represent the number and size of the near-perfect matches, respectively. F shows how many perfect clustered matches expected to occur by chance according to equation 14 in a translated genome of 3 billion bases using a query of 167 amino acids.

Table 5. Sensitivity and Specificity of Single Near-Perfect (One Mismatch Allowed) Nucleotide K-mer Matches as a Search Criterion

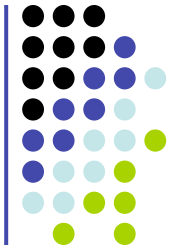
	12	13	14	15	16	17	18	19	20	21	22
A. 81%	0.945	0.880	0.831	0.721	0.657	0.526	0.465	0.408	0.356	0.255	0.218
83%	0.975	0.936	0.904	0.820	0.770	0.649	0.591	0.535	0.480	0.361	0.318
85%	0.991	0.971	0.954	0.900	0.865	0.767	0.719	0.669	0.619	0.490	0.445
87%	0.997	0.990	0.983	0.954	0.935	0.867	0.833	0.796	0.757	0.634	0.591
89%	1.000	0.997	0.995	0.984	0.976	0.939	0.920	0.897	0.872	0.775	0.741
91%	1.000	1.000	0.999	0.996	0.994	0.979	0.971	0.962	0.950	0.890	0.869
93%	1.000	1.000	1.000	0.999	0.999	0.996	0.994	0.991	0.988	0.963	0.954
95%	1.000	1.000	1.000	1.000	1.000	1.000	0.999	0.999	0.999	0.994	0.992
97%	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

B. K	12	13	14	15	16	17	18	19	20	21	22
F	275671	68775	17163	4284	1070	267	67	17	4.2	1.0	0.3

(*A*) Columns are for K sizes of 12–22. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated by equation 6 assuming a homologous region of 100 bases. (*B*) K represents the size of the near-perfect match. F shows how many perfect matches of this size expected to occur by chance according to equation 9 in a genome of 3 billion bases using a query of 500 bases.

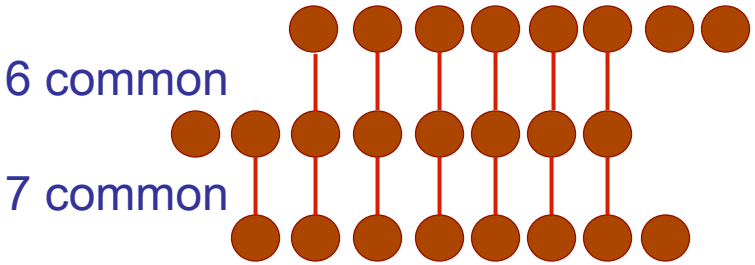
Kent WJ, Genome Research 2002

Non-consecutive words—Patterns

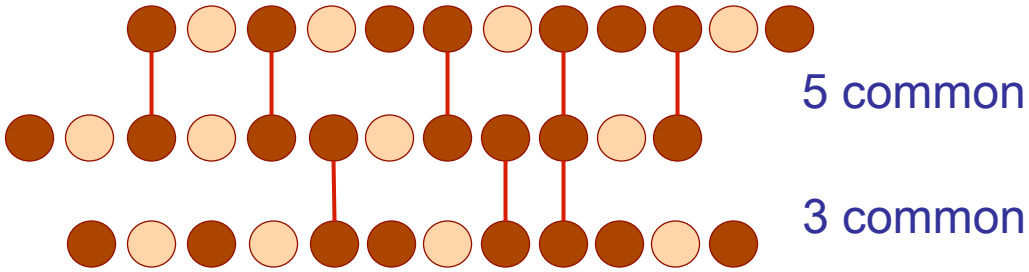


Patterns increase the likelihood of *at least one* match within a long conserved region

Consecutive Positions



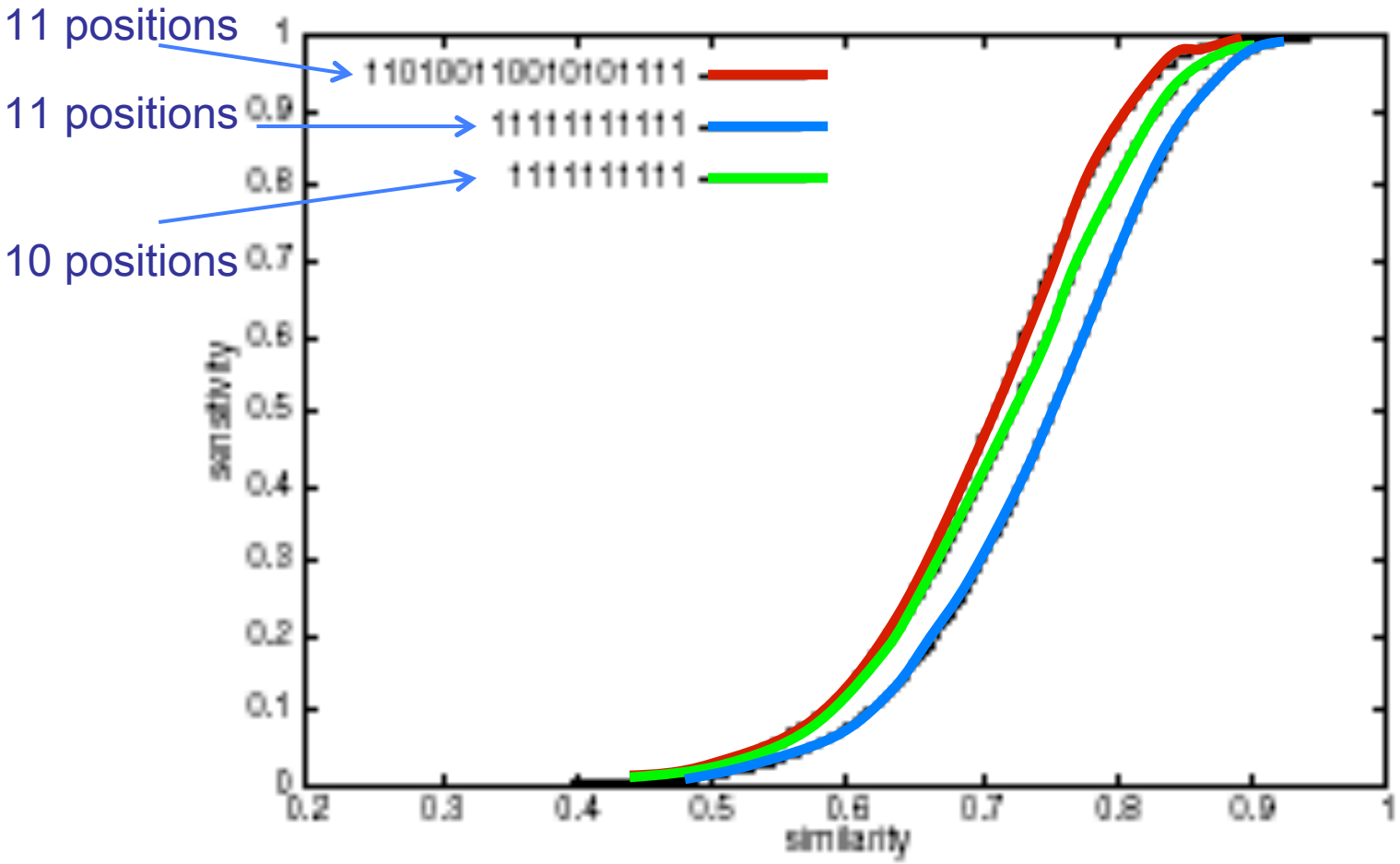
Non-Consecutive Positions



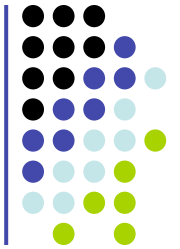
On a 100-long 70% conserved region:

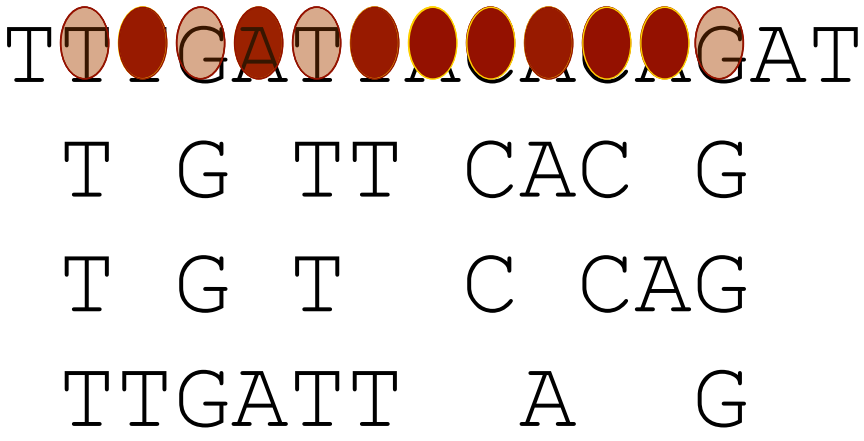
	Consecutive	Non-consecutive
Expected # hits:	1.07	0.97
Prob[at least one hit]:	0.30	0.47

Advantage of Patterns



Multiple patterns



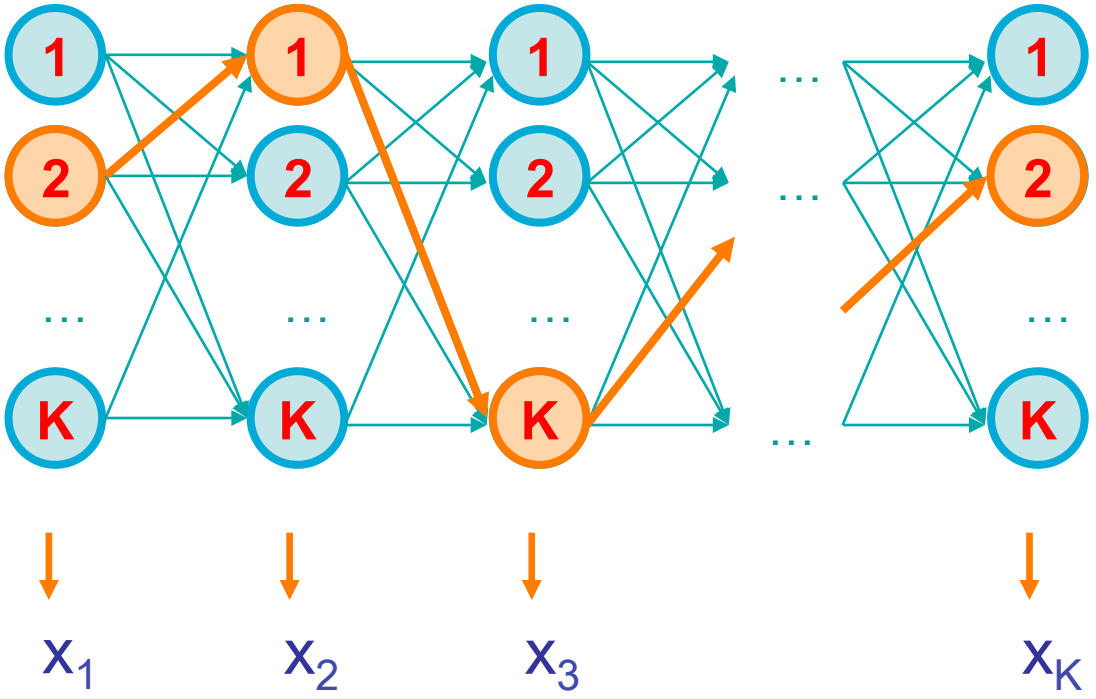


How long does it take to search the query?

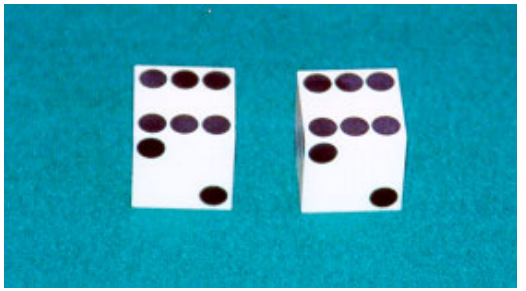
Seed	Pattern	Pr[detection]	Alignments Found	Time (s)
π_c	$\{0,1,2,3,4,5,6,7,8,9,10\}$	0.600	66419	15802
π_{c10}	$\{0,1,2,3,4,5,6,7,8,9\}$	0.707	73539	24129
π_{ph}	$\{0,1,2,4,7,9,12,13,15,16,17\}$	0.691	75518	16717
π_{N_0}	$\{0,1,2,4,7,8,11,13,16,17,18\}$	0.683	75231	16225
π_{N_5}	$\{0,1,2,3,5,6,7,10,12,13,14\}$	0.709	75547	16817
$\pi_1 + \pi_2$	$\{0,1,2,4,5,9,14,16,17,18,19,20\}+$ $\{0,1,2,3,4,6,7,8,10,11,12,13\}$	0.744	77211	22033

Buhler et al. RECOMB 2003
Sun & Buhler RECOMB 2004

Hidden Markov Models



Example: The Dishonest Casino



A casino has two dice:

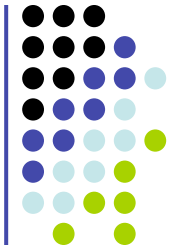
- Fair die
  P(1) = P(2) = P(3) = P(5) = P(6) = 1/6
- Loaded die
  P(1) = P(2) = P(3) = P(5) = 1/10
  P(6) = 1/2

Casino player switches back-&-forth between fair and loaded die once every 20 turns

Game:

1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2

Question # 1 – Evaluation



GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

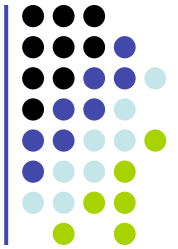
Prob = 1.3×10^{-35}

QUESTION

How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs

Question # 2 – Decoding



GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

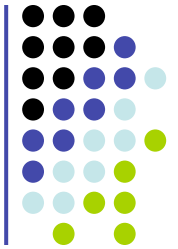
FAIR LOADED FAIR

QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

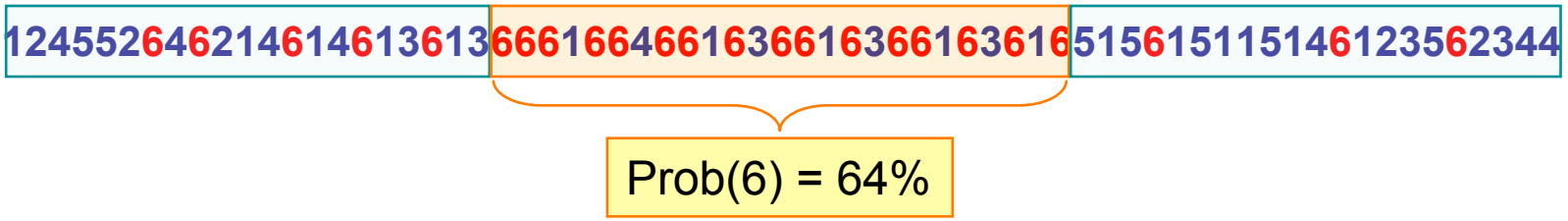
This is the **DECODING** question in HMMs

Question # 3 – Learning



GIVEN

A sequence of rolls by the casino player

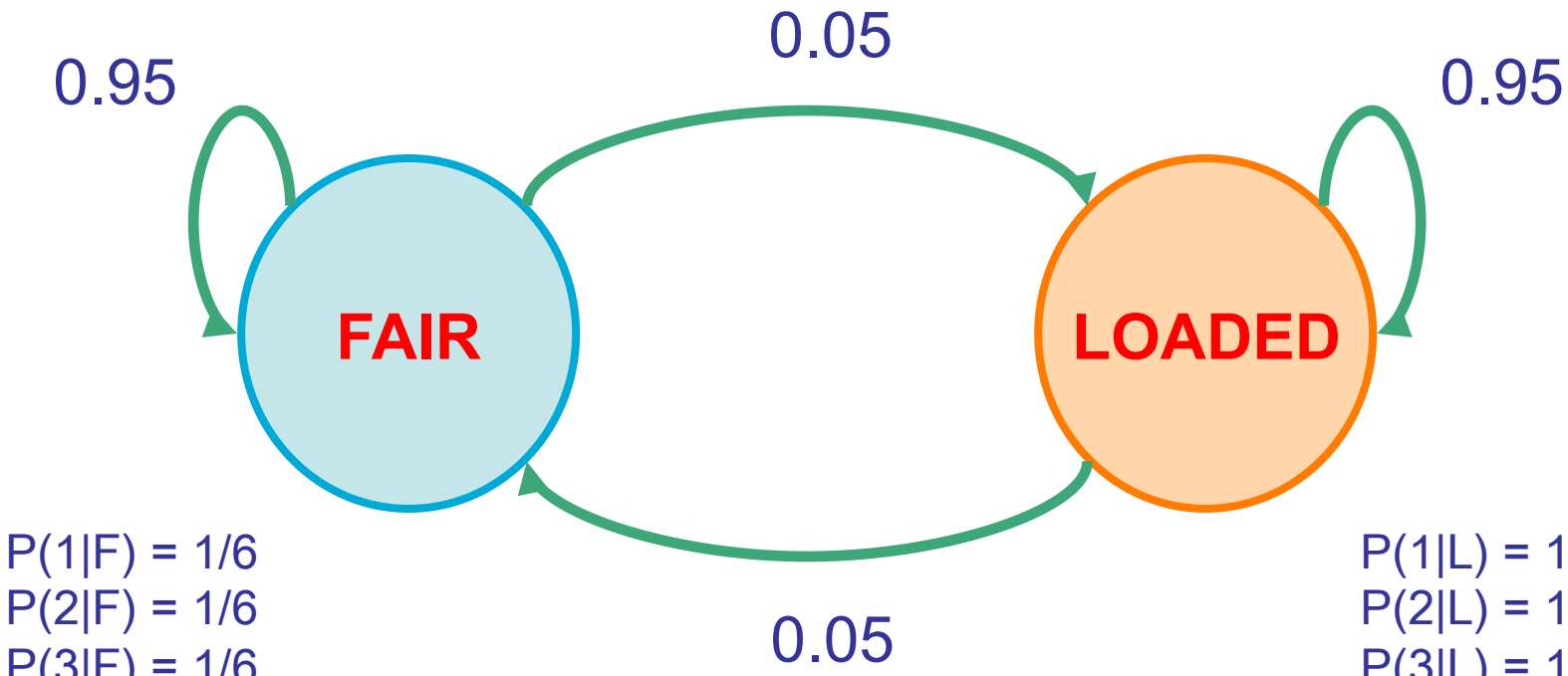


QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the **LEARNING** question in HMMs

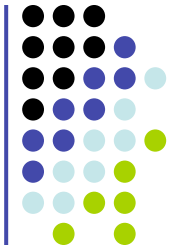
The dishonest casino model



P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2

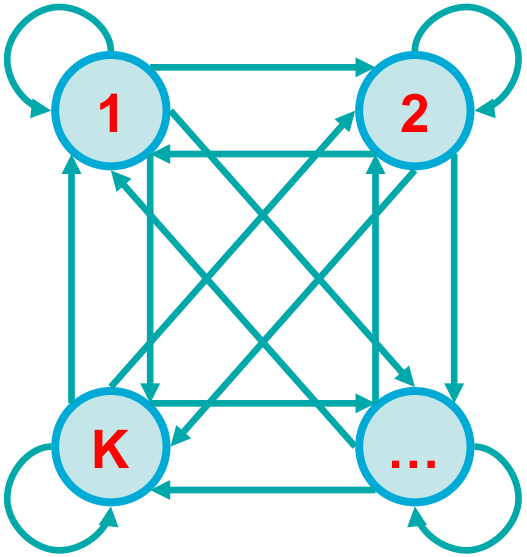
A HMM is memory-less



At each time step t,

the only thing that affects future states

is the current state π_t



Definition of a hidden Markov model

Definition: A hidden Markov model (HMM)

- Alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of states $Q = \{ 1, ..., K \}$
- Transition probabilities between any two states

a_{ij} = transition prob from state i to state j

$a_{i1} + \ldots + a_{iK} = 1$, for all states i = 1…K

- Start probabilities a_{0i}

$a_{01} + \ldots + a_{0K} = 1$

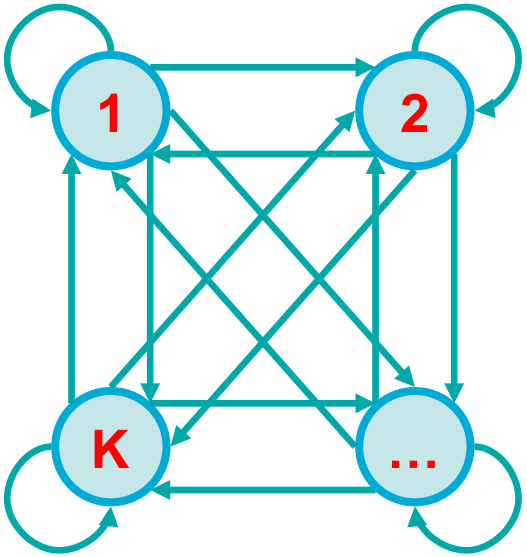
~~End Probabilities a_{i0}~~

in Durbin; not needed

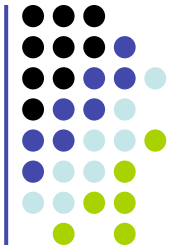
- Emission probabilities within each state

$e_i(b) = P(x_i = b \mid \pi_i = k)$

$e_i(b_1) + \ldots + e_i(b_M) = 1$, for all states i = 1…K

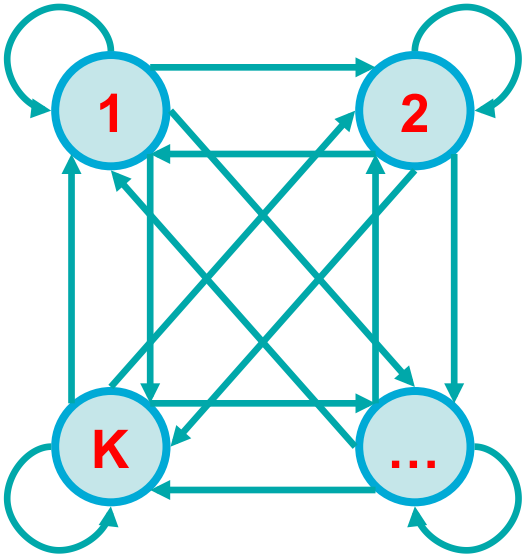


A HMM is memory-less

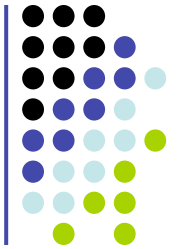


At each time step t,
the only thing that affects future states
is the current state π_t

$P(\pi_{t+1} = k \mid \text{“whatever happened so far”}) =$
$P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$
$P(\pi_{t+1} = k \mid \pi_t)$



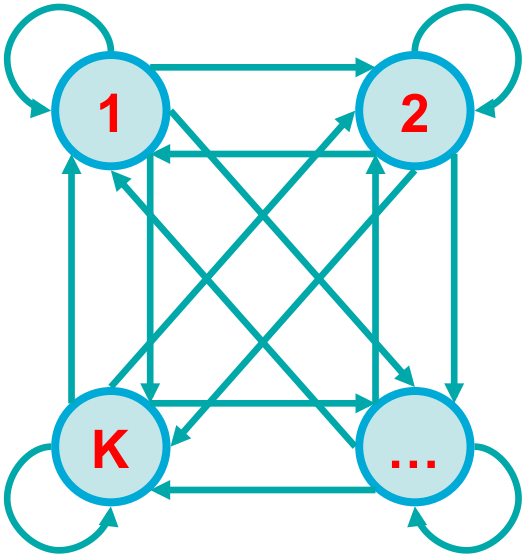
A HMM is memory-less



At each time step t,

the only thing that affects x_t

is the current state π_t

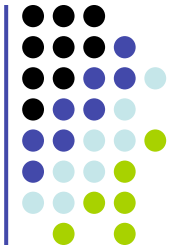


$P(x_t = b \mid \text{“whatever happened so far”}) =$

$P(x_t = b \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_{t-1}) =$

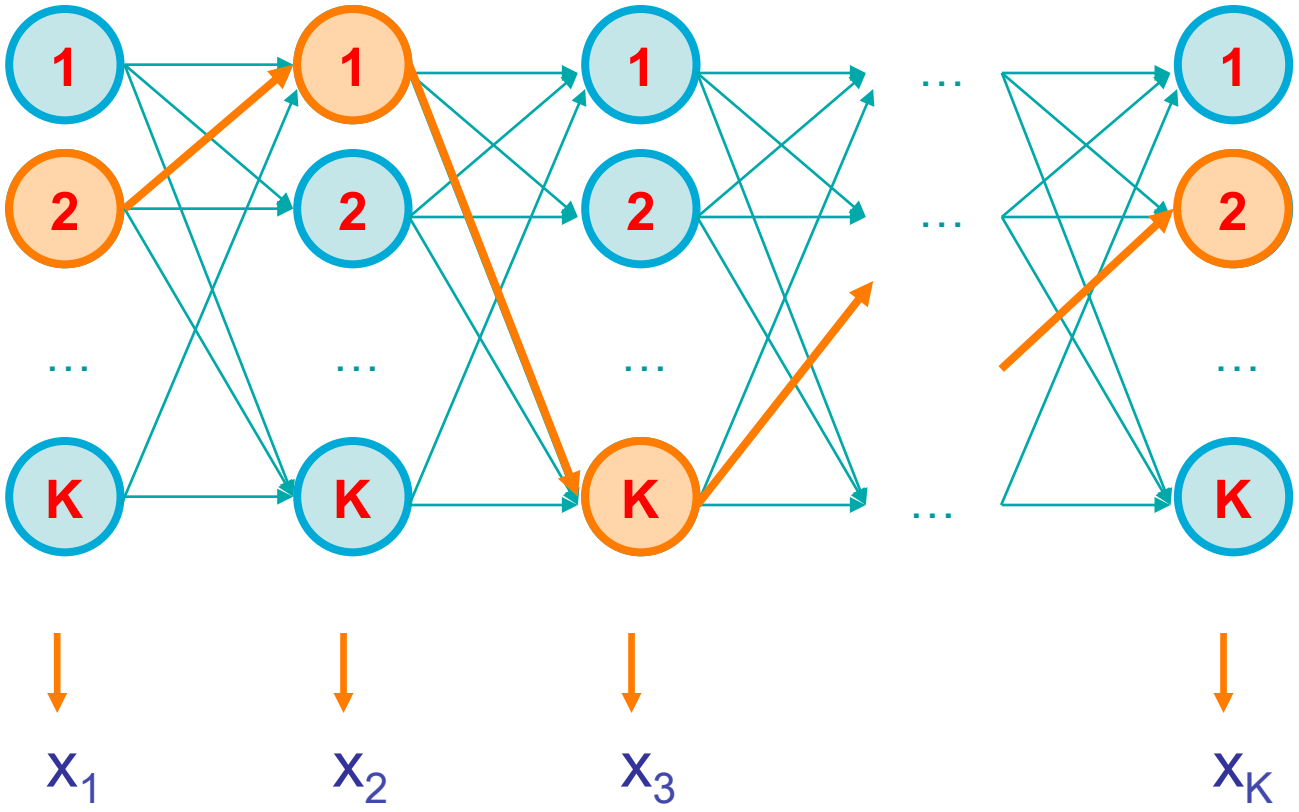
$P(x_t = b \mid \pi_t)$

A parse of a sequence

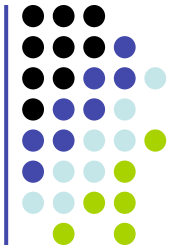


Given a sequence $x = x_1 \ldots\ldots x_N$,

A parse of x is a sequence of states $\pi = \pi_1, \ldots\ldots, \pi_N$

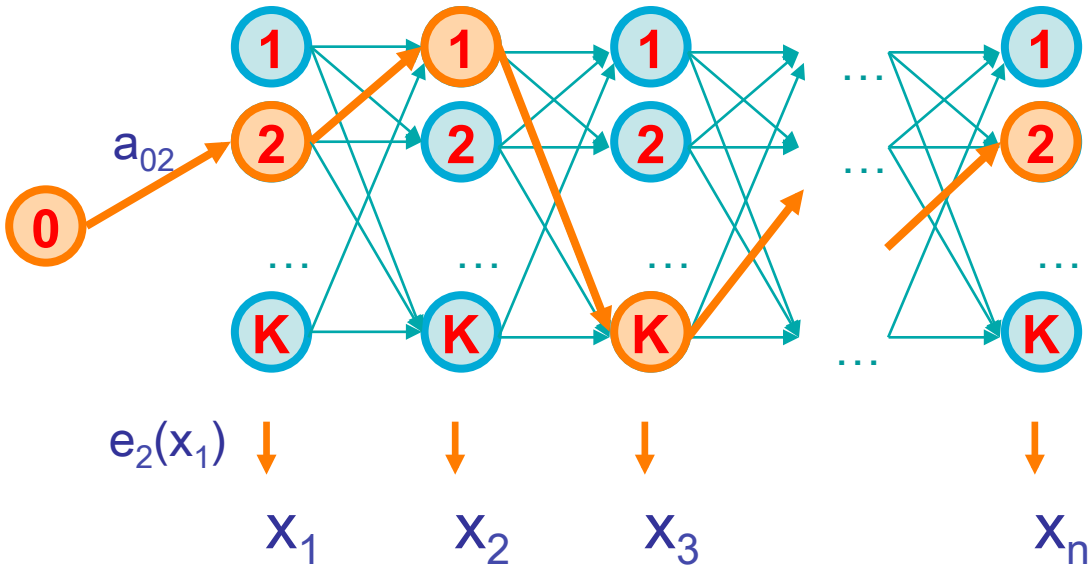


Generating a sequence by the model

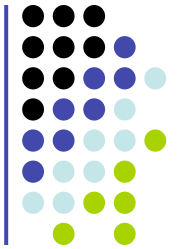


Given a HMM, we can generate a sequence of length n as follows:

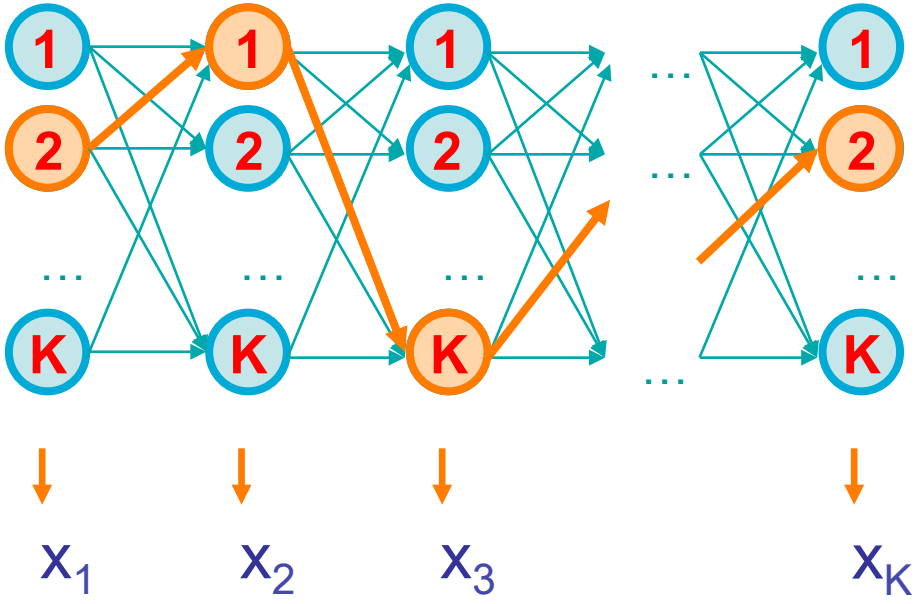
1. Start at state π_1 according to prob $a_{0\pi 1}$
2. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
3. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
4. … until emitting x_n



Likelihood of a parse



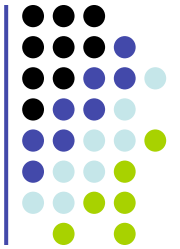
Given a sequence $x = x_1 \ldots\ldots x_N$
and a parse $\pi = \pi_1, \ldots\ldots, \pi_N$,



To find how likely this scenario is:
(given our HMM)

$$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots, \pi_N) =$$
$$P(x_N \mid \pi_N)\, P(\pi_N \mid \pi_{N-1}) \ldots\ldots P(x_2 \mid \pi_2)\, P(\pi_2 \mid \pi_1)\, P(x_1 \mid \pi_1)\, P(\pi_1) =$$
$$a_{0\pi 1}\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\, e_{\pi 1}(x_1) \ldots\ldots e_{\pi N}(x_N)$$

Likelihood of a parse



Given a sequence $x = x_1 \ldots\ldots x$

and a parse $\pi = \pi_1, \ldots\ldots, \pi_N$,

To find how likely this scenario

(given our HMM)

$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots,$

$P(x_N \mid \pi_N)\ P(\pi_N \mid \pi_{N-1})$

$a_{0\pi1}\ a_{\pi1\pi2} \ldots\ldots a_{\pi N-1\pi N}\ e_{\pi1}(x_1$

A compact way to write

$a_{0\pi1}\ a_{\pi1\pi2} \ldots\ldots a_{\pi N-1\pi N}\ e_{\pi1}(x_1) \ldots\ldots e_{\pi N}(x_N)$

Enumerate all parameters a_{ij} and $e_i(b)$; n params

Example:

$a_{0Fair} : \theta_1$; $a_{0Loaded} : \theta_2$; … $e_{Loaded}(6) = \theta_{18}$

Then, count in x and π the # of times each parameter j = 1, …, n occurs

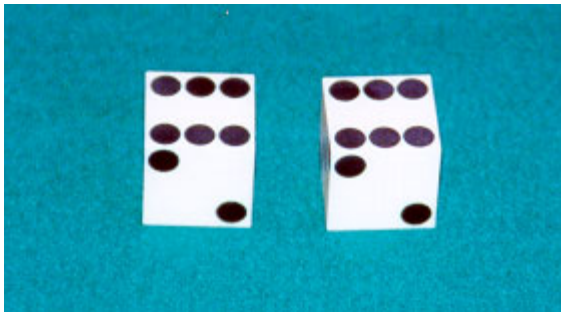
F(j, x, π) = # parameter θ_j occurs in (x, π)

*(call **F(.,.,.)** the **feature counts**)* Then,

$$P(x, \pi) = \prod_{j=1\ldots n} \theta_j^{F(j, x, \pi)} =$$

$$= \exp\left[\sum_{j=1\ldots n} \log(\theta_j) \times F(j, x, \pi)\right]$$

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 2, 1, 5, 6, 2, 1, 5, 2, 4

Then, what is the likelihood of

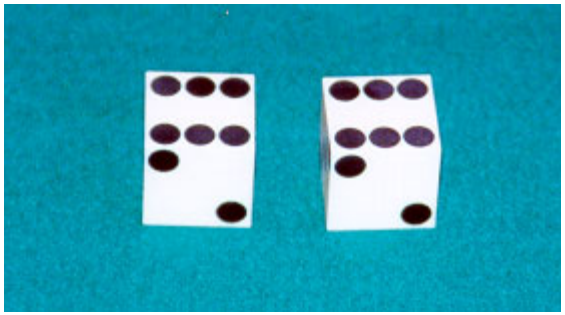
π = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs $a_{0Fair} = ½$, $a_{oLoaded} = ½$)

½ × P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$½ \times (1/6)^{10} \times (0.95)^{9} = .00000005211586472 11 \sim= 0.5 \times 10^{-9}$

Example: the dishonest casino



So, the likelihood the die is fair in this run is just 0.521×10^{-9}

What is the likelihood of

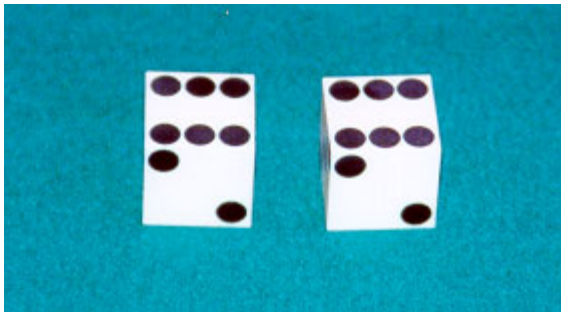
π = Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

$\frac{1}{2} \times$ P(1 | Loaded) P(Loaded, Loaded) … P(4 | Loaded) =

$\frac{1}{2} \times (1/10)^9 \times (1/2)^1 (0.95)^9 = .000000000015756235243 \sim= 0.16 \times 10^{-9}$

Therefore, it somewhat more likely that all the rolls are done with the fair die, than that they are all done with the loaded die

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

Now, what is the likelihood π = F, F, …, F?

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^{9} \sim= 0.5 \times 10^{-9}$, same as before

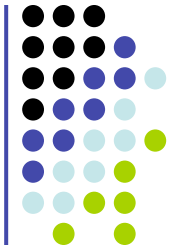
What is the likelihood

π = L, L, …, L?

$\frac{1}{2} \times (1/10)^{4} \times (1/2)^{6} (0.95)^{9} = .00000049238235134735 \sim= 0.5 \times 10^{-7}$

So, it is 100 times more likely the die is loaded

Question # 1 – Evaluation



GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

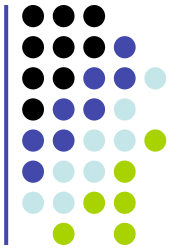
Prob = 1.3×10^{-35}

QUESTION

How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs

Question # 2 – Decoding



GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

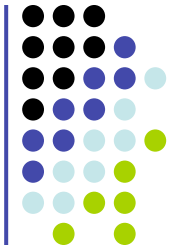
FAIR LOADED FAIR

QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

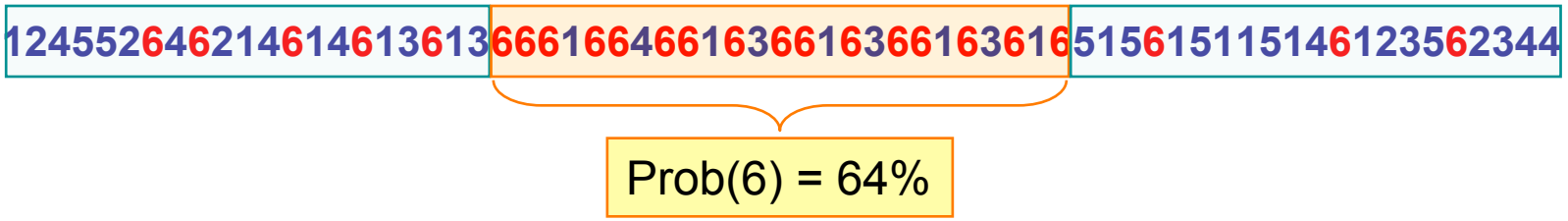
This is the **DECODING** question in HMMs

Question # 3 – Learning



GIVEN

A sequence of rolls by the casino player

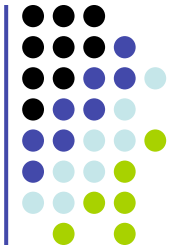


QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the **LEARNING** question in HMMs

The three main questions on HMMs



1. **Evaluation**

   GIVEN a HMM M, and a sequence x,
   FIND Prob[x | M]

2. **Decoding**

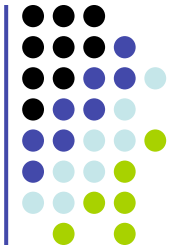
   GIVEN a HMM M, and a sequence x,
   FIND the sequence π of states that maximizes P[x, π | M]

3. **Learning**

   GIVEN a HMM M, with unspecified transition/emission probs.,
   and a sequence x,

   FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]

Let's not be confused by notation



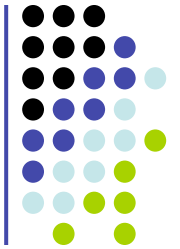
P[x | M]: The probability that sequence x was generated by the model

The model is: architecture (#states, etc)
+ parameters $\theta = a_{ij}, e_i(.)$

So, P[x | M] is the same with P[x | θ], and P[x], when the architecture, and the parameters, respectively, are implied

Similarly, P[x, π | M], P[x, π | θ] and P[x, π] are the same when the architecture, and the parameters, are implied

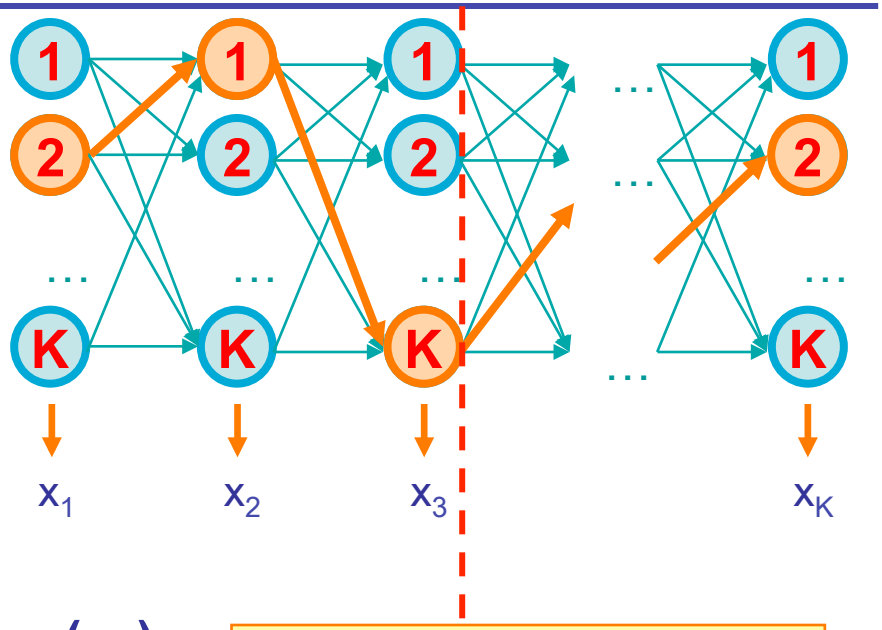
In the LEARNING problem we always write P[x | θ] to emphasize that we are seeking the θ^* that maximizes P[x | θ]



Problem 1: Decoding

Find the most likely parse of a sequence

Decoding



GIVEN $x = x_1 x_2 \ldots\ldots x_N$

Find $\pi = \pi_1, \ldots\ldots, \pi_N,$
to maximize $P[x, \pi]$

$\pi^* = \text{argmax}_\pi P[x, \pi]$

Maximizes $a_{0\pi 1}\, e_{\pi 1}(x_1)\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\, e_{\pi N}(x_N)$

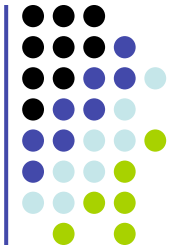
Dynamic Programming!

$V_k(i) = \max_{\{\pi 1 \ldots\, \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

= Prob. of most likely sequence of states ending at state $\pi_i = k$

Given that we end up in state k at step i, maximize product to the left and right

Decoding – main idea



Inductive assumption: Given that for all states k,
and for a fixed position i,

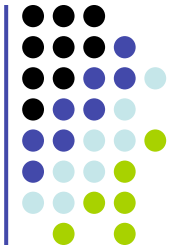
$$\mathbf{V_k(i) = \max_{\{\pi 1 \ldots \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]}$$

What is $V_l(i+1)$?

From definition,

$$V_l(i+1) = \max_{\{\pi 1 \ldots \pi i\}} P[\, x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l \,]$$
$$= \max_{\{\pi 1 \ldots \pi i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i)\, P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$$
$$= \max_{\{\pi 1 \ldots \pi i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i)\, P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$$
$$= \max_k \left[P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k)\, \mathbf{\max_{\{\pi 1 \ldots \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]} \right]$$
$$= \max_k \left[P(x_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)\, \mathbf{V_k(i)} \right]$$
$$= e_l(x_{i+1}) \max_k a_{kl}\, \mathbf{V_k(i)}$$

The Viterbi Algorithm



Input: $x = x_1 \ldots\ldots x_N$

Initialization:

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$Ptr_j(i) = \text{argmax}_k a_{kj} V_k(i-1)$

Termination:

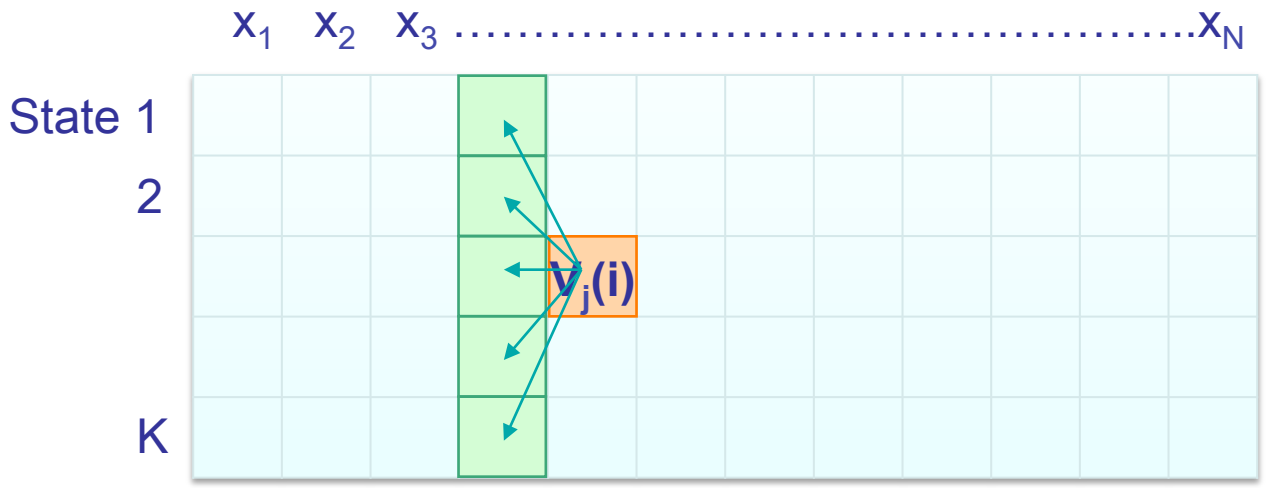
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

$\pi_N^* = \text{argmax}_k V_k(N)$

$\pi_{i-1}^* = Ptr_{\pi i}(i)$

The Viterbi Algorithm



Similar to "aligning" a set of states to a sequence

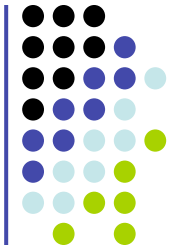
Time:

$O(K^2N)$

Space:

$O(KN)$

Viterbi Algorithm – a practical detail



Underflows are a significant problem

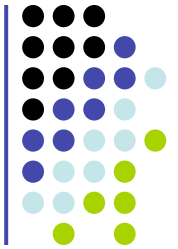
$$P[x_1,\ldots., x_i, \pi_1, \ldots, \pi_i] = a_{0\pi1} a_{\pi1\pi2}\ldots\ldots a_{\pi i} e_{\pi1}(x_1)\ldots\ldots e_{\pi i}(x_i)$$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$$V_l(i) = \log e_k(x_i) + \max_k [V_k(i-1) + \log a_{kl}]$$

Example



Let x be a long sequence with a portion of ~ 1/6 6's,
followed by a portion of ~ ½ 6's…

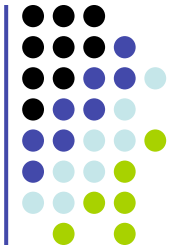
x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

FFF……..F LLL……..L

6 characters "123456" parsed as F, contribute $.95^6 \times (1/6)^6$ $= 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^1 \times (1/10)^5$ $= 0.4 \times 10^{-5}$

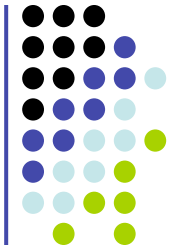
"162636" parsed as F, contribute $.95^6 \times (1/6)^6$ $= 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^3 \times (1/10)^3$ $= 9.0 \times 10^{-5}$



Problem 2: Evaluation

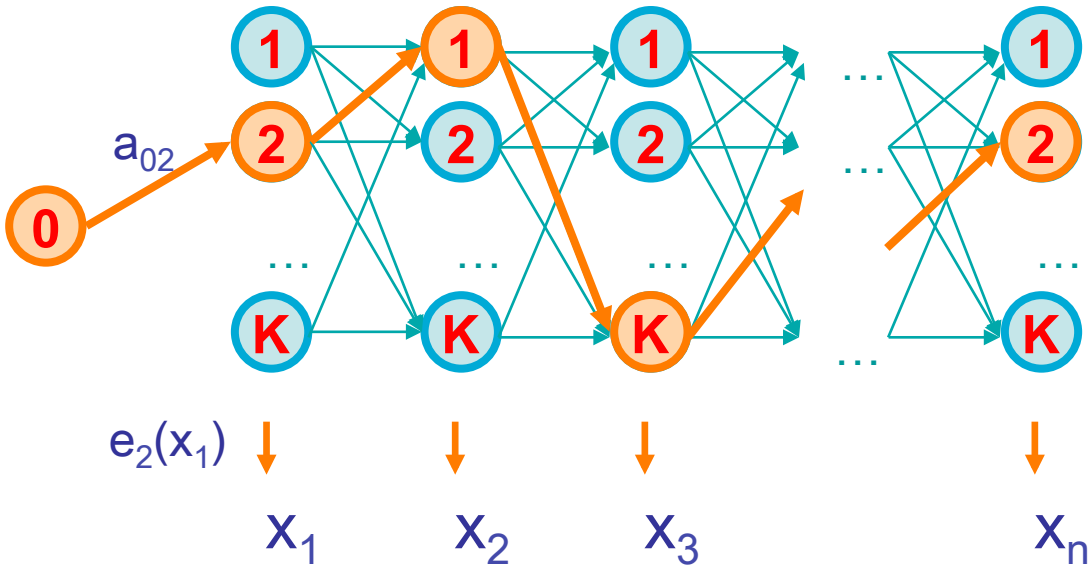
Find the likelihood a sequence is generated by the model

Generating a sequence by the model

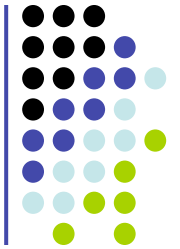


Given a HMM, we can generate a sequence of length n as follows:

1. Start at state π_1 according to prob $a_{0\pi 1}$
2. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
3. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
4. … until emitting x_n



A couple of questions



Given a sequence x,

- What is the probability that
- Given a position i, what is th

Example: the dishonest ca

P(box: FFFFFFFFFFF) =
$(1/6)^{11} * 0.95^{12}$ =
$2.76^{-9} * 0.54$ =
1.49^{-9}

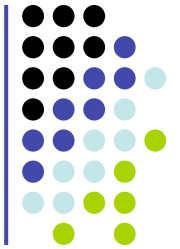
P(box: LLLLLLLLLL) =
$[(1/2)^6 * (1/10)^5] * 0.95^{10} * 0.05^2$ =
$1.56*10^{-7} * 1.5^{-3}$ =
0.23^{-9}

Say x = 12341…231626163646162341 12…21341

F F

Most likely path: π = FF……F
(too "unlikely" to transition F → L → F)
However: marked letters more likely to be L than unmarked letters

Evaluation

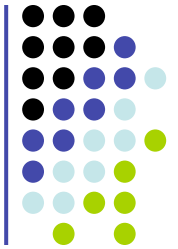


We will develop algorithms that allow us to compute:

$P(x)$	Probability of x given the model
$P(x_i \ldots x_j)$	Probability of a substring of x given the model
$P(\pi_i = k \mid x)$	"Posterior" probability that the i^{th} state is k, given x

A more refined measure of which states x may be in

The Forward Algorithm



We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

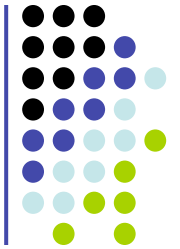
$$P(x) = \Sigma_\pi P(x, \pi) = \Sigma_\pi P(x \mid \pi) P(\pi)$$

To avoid summing over an exponential number of paths π, define

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \quad \text{(the forward probability)}$$

"generate i first characters of x and end up in state k"

The Forward Algorithm – derivation



Define the forward probability:

$f_k(i) = P(x_1 \ldots x_i, \pi_i = k)$

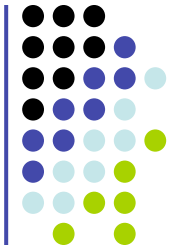
$= \sum_{\pi 1 \ldots \pi i-1} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, \pi_i = k)\, e_k(x_i)$

$= \sum_l \sum_{\pi 1 \ldots \pi i-2} \mathbf{P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = l)}\, a_{lk}\, e_k(x_i)$

$= \sum_l \mathbf{P(x_1 \ldots x_{i-1}, \pi_{i-1} = l)}\, a_{lk}\, e_k(x_i)$

$= e_k(x_i) \sum_l \mathbf{f_l(i-1)}\, a_{lk}$

The Forward Algorithm



We can compute $f_k(i)$ for all k, i, using dynamic programming!

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

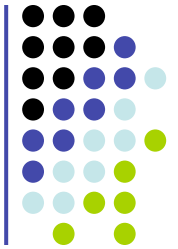
Iteration:

$$f_k(i) = e_k(x_i) \sum_l f_l(i-1)\, a_{lk}$$

Termination:

$$P(x) = \sum_k f_k(N)$$

Relation between Forward and Viterbi



VITERBI

Initialization:

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

Iteration:

$$V_j(i) = e_j(x_i) \max_k V_k(i-1)\, a_{kj}$$

Termination:

$$P(x, \pi^*) = \max_k V_k(N)$$

FORWARD

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

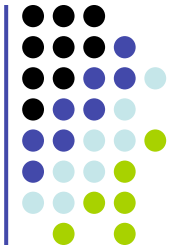
Iteration:

$$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\, a_{kl}$$

Termination:

$$P(x) = \sum_k f_k(N)$$

Motivation for the Backward Algorithm



We want to compute

$$P(\pi_i = k \mid x),$$

the probability distribution on the i^{th} position, given x

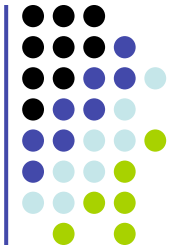
We start by computing

$$P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N)$$
$$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)$$
$$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid \pi_i = k)$$

Forward, $f_k(i)$ Backward, $b_k(i)$

Then, $P(\pi_i = k \mid x) = P(\pi_i = k, x) / P(x)$

The Backward Algorithm – derivation



Define the backward probability:

$b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k)$ *"starting from i^{th} state = k, generate rest of x"*

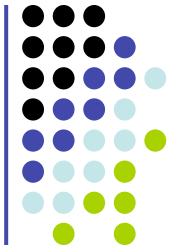
$= \sum_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$

$= \sum_l \sum_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$

$= \sum_l e_l(x_{i+1})\, a_{kl} \sum_{\pi i+1 \ldots \pi N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)$

$= \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$

The Backward Algorithm



We can compute $b_k(i)$ for all k, i, using dynamic programming

Initialization:

$b_k(N) = 1$, for all k

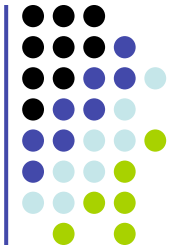
Iteration:

$$b_k(i) = \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$$

Termination:

$$P(x) = \sum_l a_{0l}\, e_l(x_1)\, b_l(1)$$

Computational Complexity



What is the running time, and space required, for Forward, and Backward?

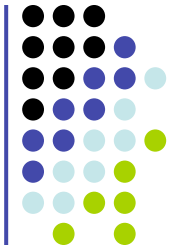
Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs

Forward/Backward: rescaling at each few positions by multiplying by a constant

Posterior Decoding



We can now calculate

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

Then, we can ask

$P(\pi_i = k \mid x) =$

$P(\pi_i = k\ , x)/P(x) =$

$P(x_1, \ldots, x_i, \pi_i = k, x_{i+1}, \ldots x_n) / P(x) =$

$P(x_1, \ldots, x_i, \pi_i = k)\, P(x_{i+1}, \ldots x_n \mid \pi_i = k) / P(x) =$

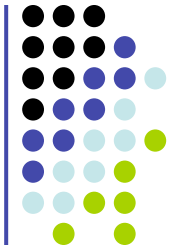
$f_k(i)\, b_k(i) / P(x)$

What is the most likely state at position i of sequence x:

Define $\hat{\pi}$ by Posterior Decoding:

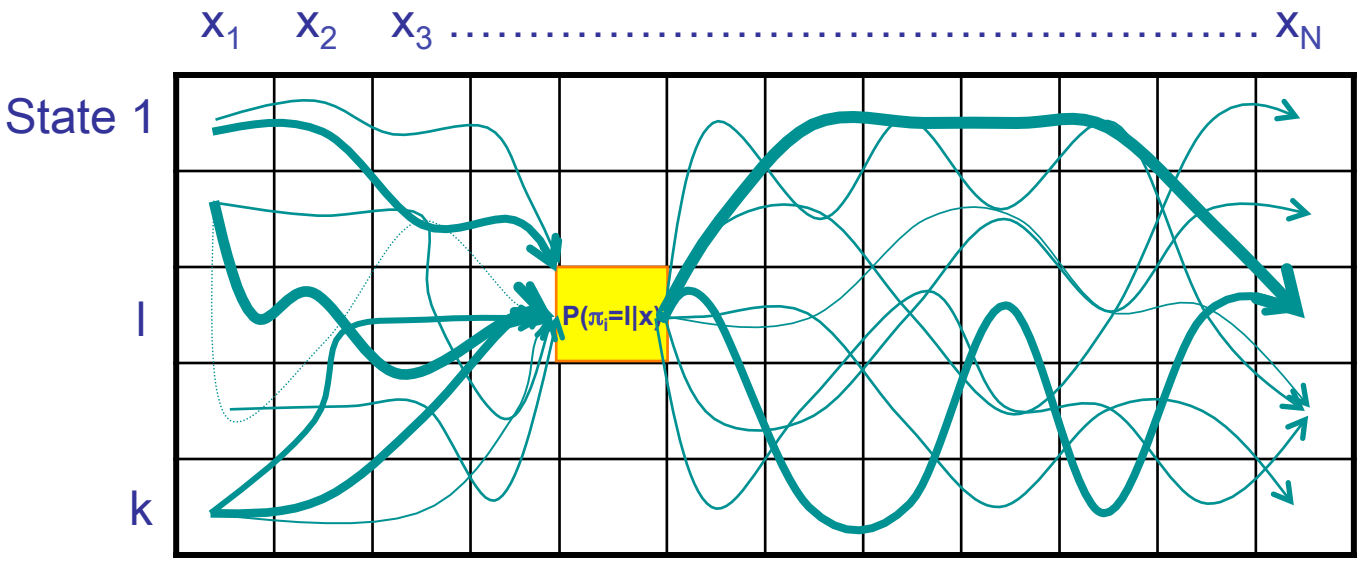
$$\hat{\pi}_i = \text{argmax}_k\, P(\pi_i = k \mid x)$$

Posterior Decoding



- For each state,
  - Posterior Decoding gives us a curve of likelihood of state for each position
  - That is sometimes more informative than Viterbi path π^*
- Posterior Decoding may give an invalid sequence of states (of prob 0)
  - Why?

Posterior Decoding

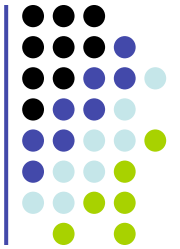


- $P(\pi_i = k \mid x) = \sum_{\pi} P(\pi \mid x)\, \mathbf{1}(\pi_i = k)$

$= \sum_{\{\pi:\pi[i] = k\}} P(\pi \mid x)$

$\mathbf{1}(\psi) = 1$, if ψ is true
0, otherwise

Viterbi, Forward, Backward



VITERBI

Initialization:

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_l(i) = e_l(x_i)\ \max_k V_k(i-1)\ a_{kl}$

Termination:

$P(x, \pi^*) = \max_k V_k(N)$

FORWARD

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\ a_{kl}$

Termination:

$P(x) = \sum_k f_k(N)$

BACKWARD

Initialization:

$b_k(N) = 1$, for all k

Iteration:

$b_l(i) = \sum_k e_l(x_i+1)\ a_{kl}\ b_k(i+1)$

Termination:

$P(x) = \sum_k a_{0k}\ e_k(x_1)\ b_k(1)$