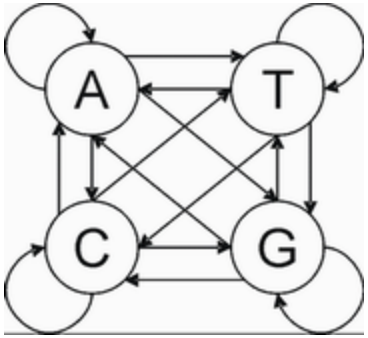


Hidden Markov Models
Part 1.

- Brief overview of discrete time finite Markov Chain
- Hidden Markov Model
- Estimation of hidden state sequence
  (with brief discussion about dynamic programming)

Discrete time finite Markov Chain

Possible states: finite discrete set S {E1, E2, … … , Es}

From time t to t+1, make stochastic movement from one state to another.

The Markov Property:

At time t, the process is at E_j,
Then at time t+1, the probability it is at E_k only depends on E_j

The temporally homogeneous transition probabilities property:

Prob(E_j -> E_k) is independent of time t.

Discrete time finite Markov Chain

The transition probability matrix:

P_{ij}=Prob(E_i -> E_j)

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \cdots & \cdots & \cdots & \cdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}$$

An N step transition: P_{ij}(N)=Prob(E_i -> ……. -> E_j)

It can be shown that $P(N) = P^N$

Discrete time finite Markov Chain

Consider the two step transition probability from Ei to Ej:

$$p_{ij}^{(2)} = \sum_k p_{ik} p_{kj}$$

This is the ijth element of P^2

So, the two-step transition matrix $P^{(2)} = P^2$

Extending this argument, we have $P^{(N)} = P^N$

- Absorbing states:

  p_{ii}=1. Once enter this state, stay in this state.

  We don't consider this.

Discrete time finite Markov Chain

The stationary state:

$$\varphi_i = \sum_k \varphi_k p_{kj}, \forall i$$

$$\varphi_i(t) = \varphi_i(t+1), \forall i$$

The probability of being at each state stays constant.

$\varphi = (\varphi_1, \varphi_2, \varphi_3, \ldots\ldots\ldots, \varphi_s)$ is the stationary state.

For **finite, aperiodic, irreducible** Markov chains, φ exists and is unique.

Periodic: if a state can only be returned to at t_0, $2t_0$,$3t_0$,…….. , $t_0>1$
Irreducible: any state can be eventually reached from any state

Discrete time finite Markov Chain

$$P^{(n)} = P^n \rightarrow \begin{bmatrix} \varphi \\ \dots \\ \varphi \end{bmatrix}$$

Note:

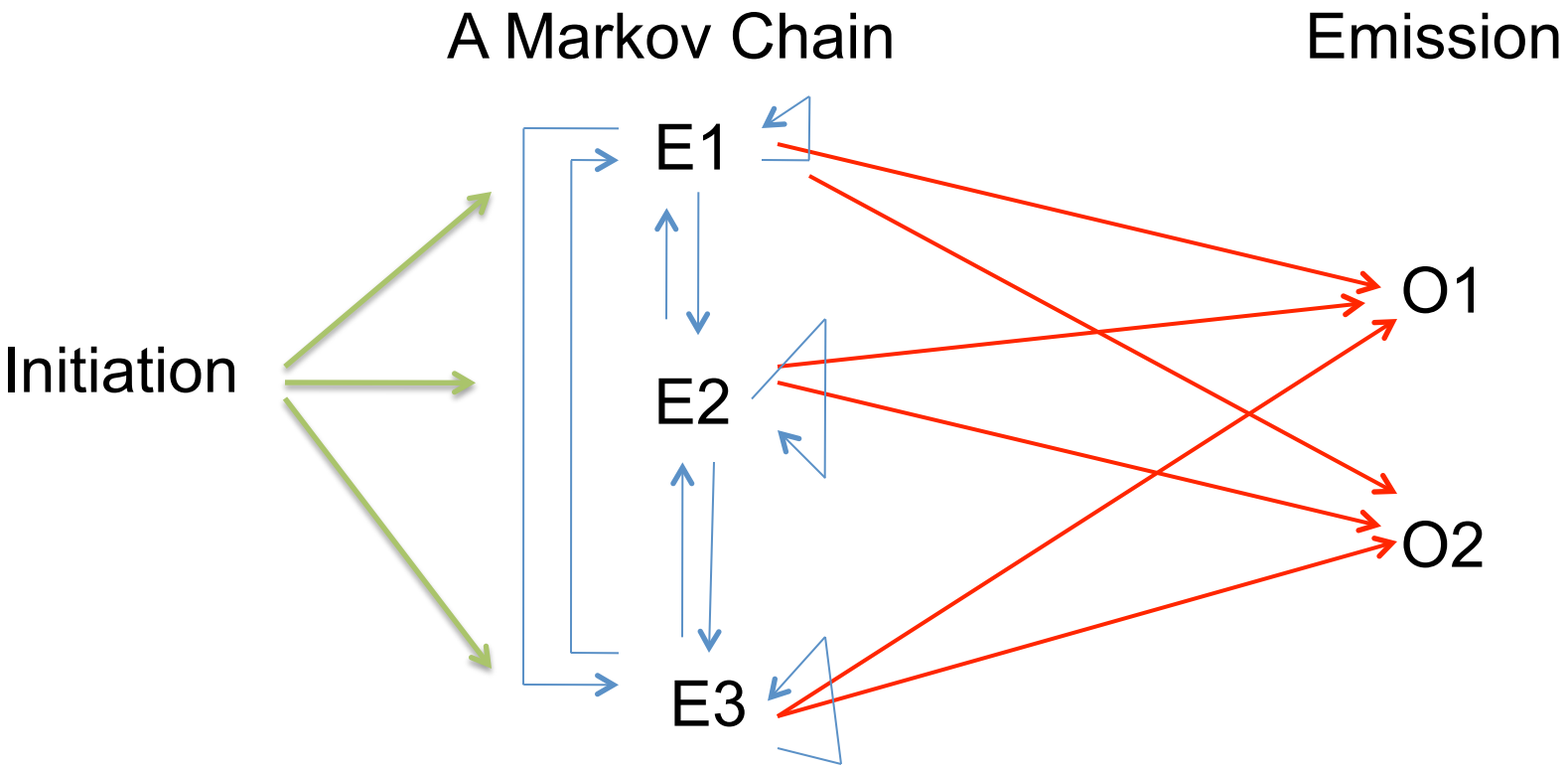
Markov chain is an elegant model in many situations in pattern recognition.

BUT the two assumptions may not hold true.

Hidden Markov Model

An extension of the Markov Model.

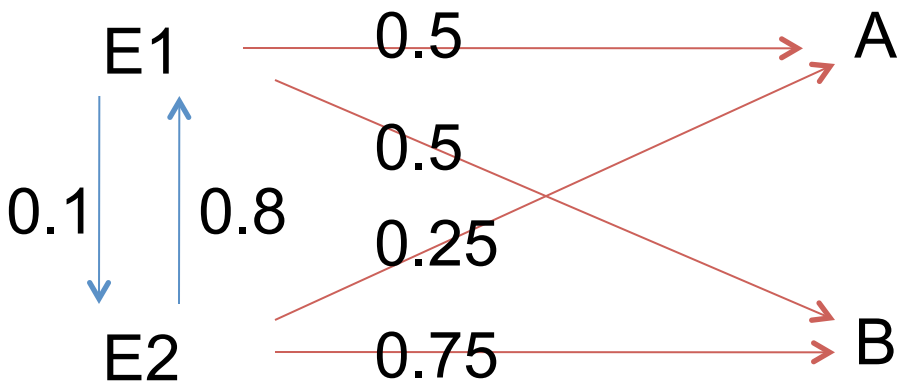
A discrete time Markov Model with extra features.



Hidden Markov Model

The emissions are **probablistic**, **state-dependent** and **time-independent**

Example:

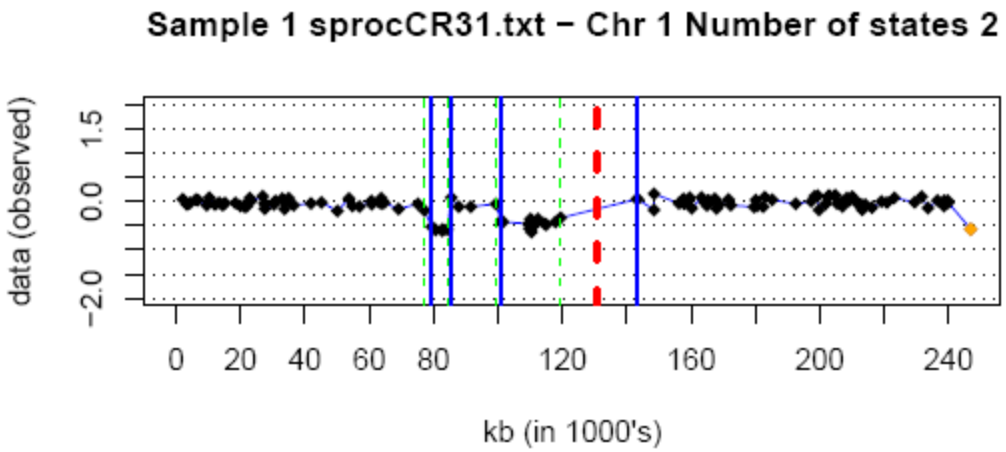


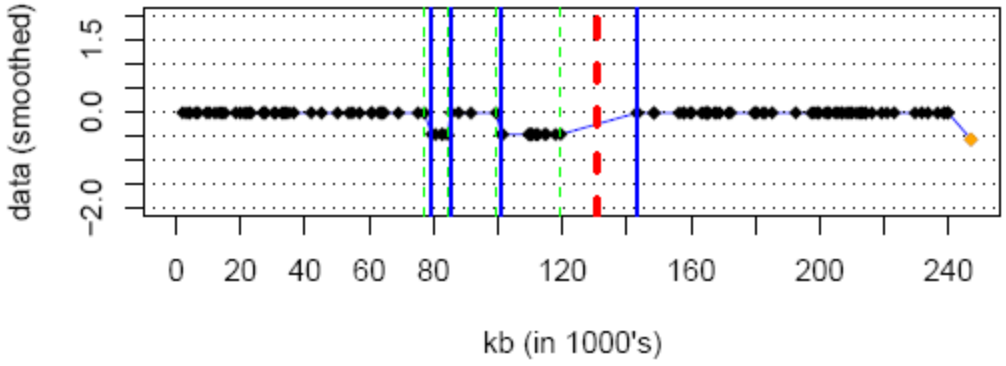
"Hidden" – we don't observe the states of the Markov Chain, but we observe the emissions.

If both E1 and E2 have the same chance to initiate the chain, and we observe sequence "BBB", what is the most likely state sequence that the chain went through?

An example

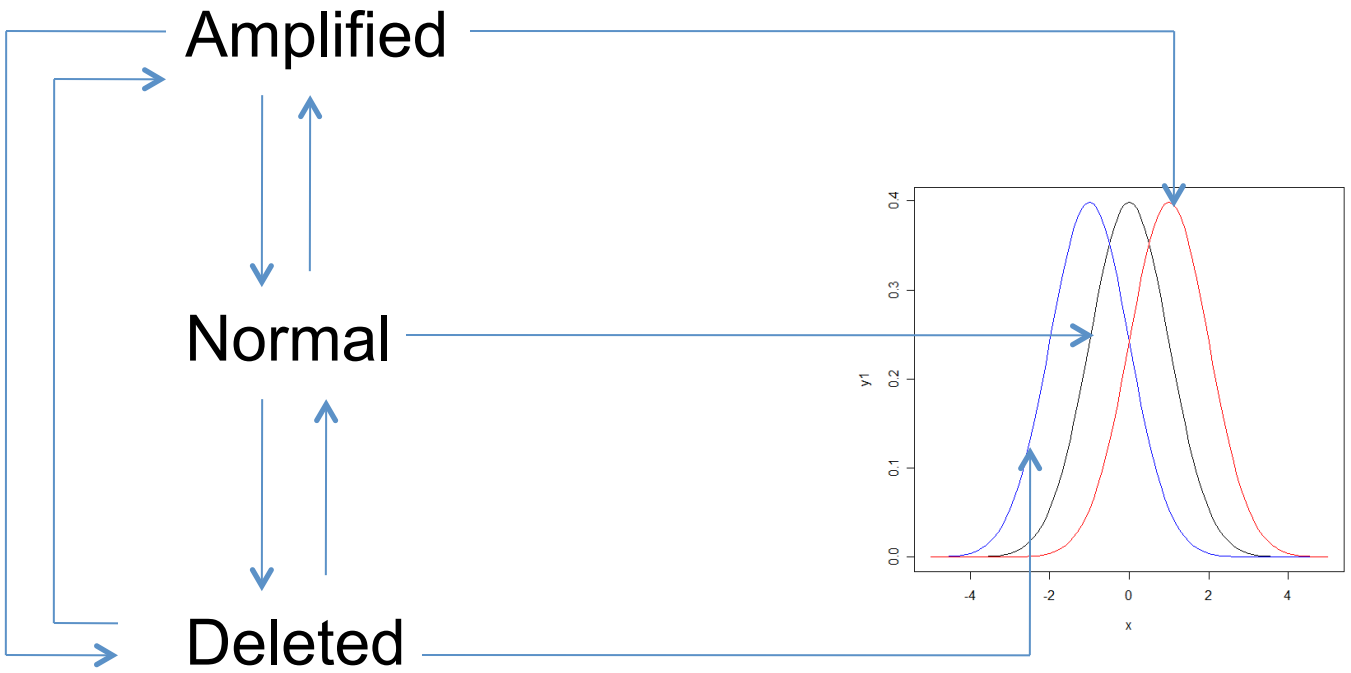
Data segmentation in array-based copy number analysis.



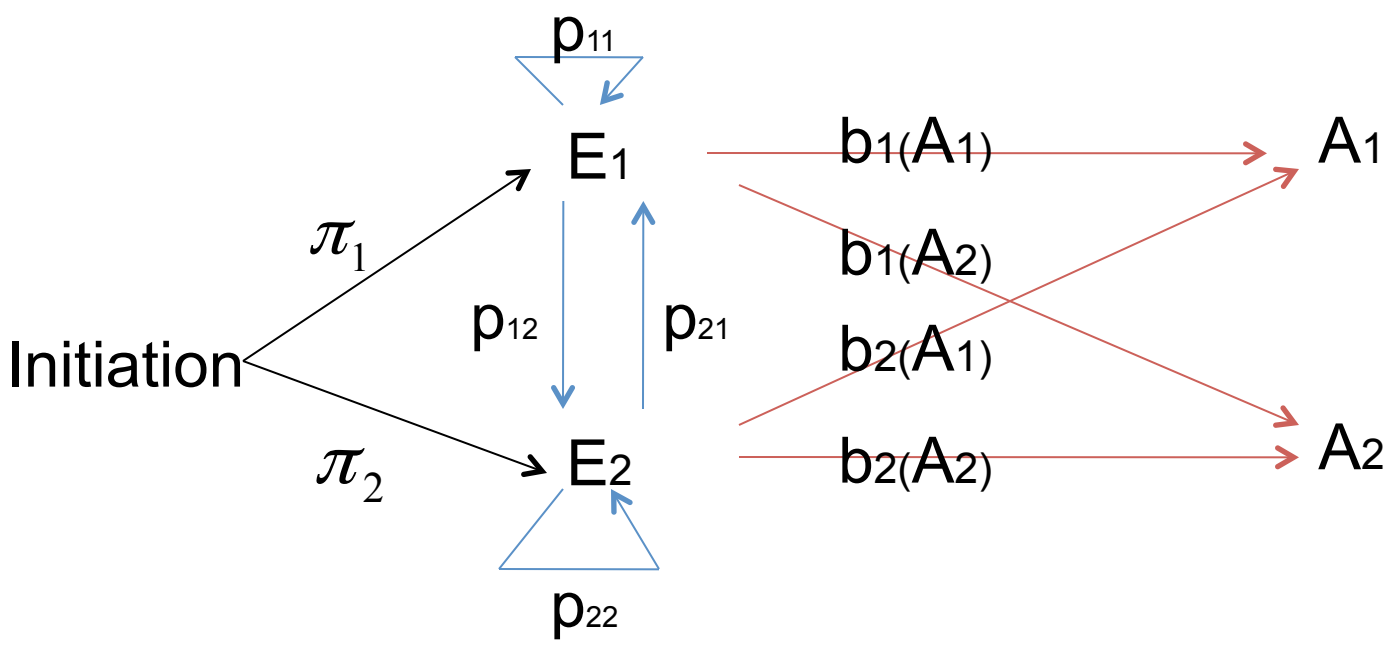


An example

An over-simplified model.



Hidden Markov Model



$$\lambda = \{\pi, P, B\}$$

initial distribution
transition probabilities
emission probabilities

Let $Q = \{q^{(1)} q^{(2)} \ldots\ldots q^{(T)}\}$

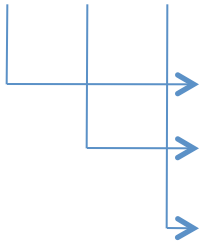
denote the state sequence, and

$$O = \{O_1, O_2, \ldots\ldots, O_T\}$$

denote the observed emission.

Hidden Markov Model

Parameters: $\lambda = \{\pi, P, B\}$



initial distribution
transition probabilities
emission probabilities

Common questions:

How to efficiently calculate emissions: $P(O \mid \lambda)$

How to find the most likely hidden state: $\arg\max_{Q} P(Q \mid O)$

How to find the most likely parameters: $\arg\max_{\lambda} P(O \mid \lambda)$

Dynamic Programming

"Two **sledgehammers** of the algorithms craft, **dynamic programming** and **linear programming**"

Dynamic programming:

Breaking the overall optimization problem into overlapping smaller problems;

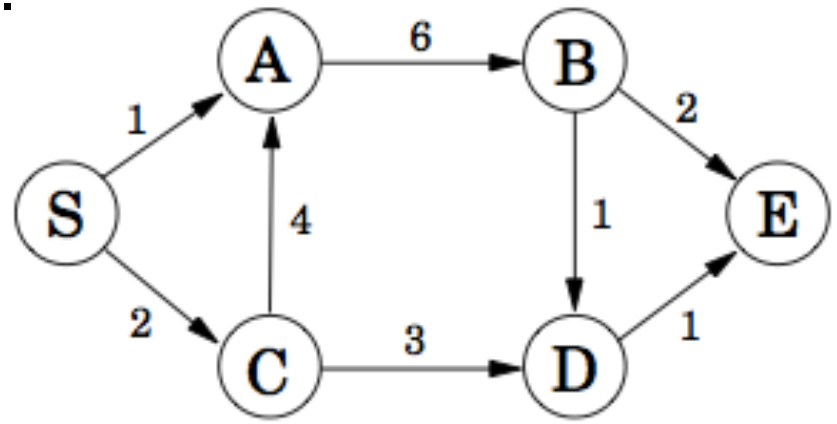
Solve each sub-problem once, and reuse the results, thus reducing the computing cost (dramatically);

Often working backward.

Dynamic Programming

A simple example:

Find the shortest path from S to E in the directed acyclic graph below.

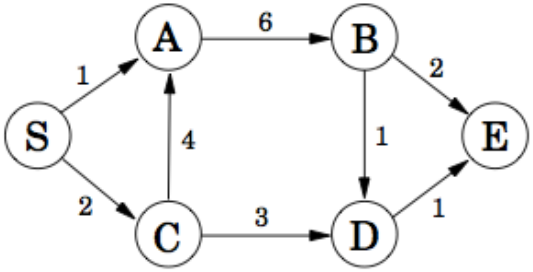


Take node D as an example. The way to get to D is through B or C. So,

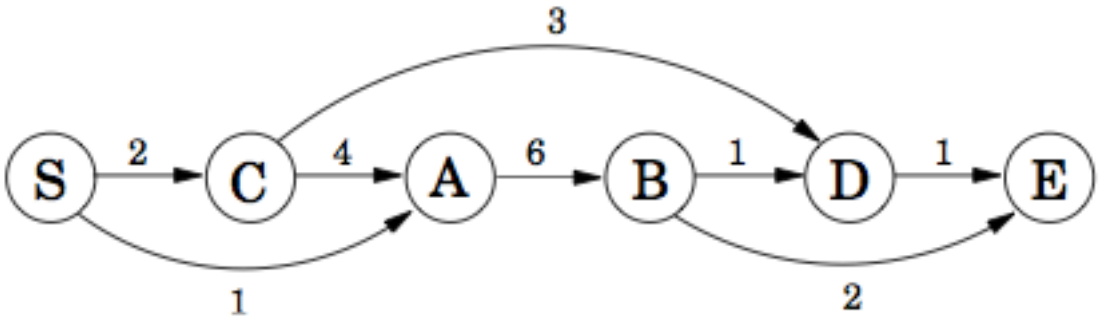
$$\texttt{dist}(D) = \min\{\texttt{dist}(B) + 1, \texttt{dist}(C) + 3\}.$$

Algorithms, Dasgupta et al.

Dynamic Programming



Linearization:

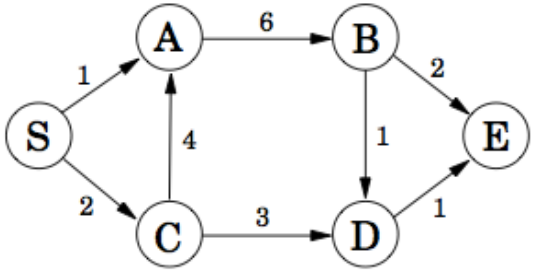


Algorithm:

```
initialize all dist(·) values to ∞
dist(s) = 0
for each v ∈ V\{s}, in linearized order:
    dist(v) = min_{(u,v)∈E} {dist(u) + l(u, v)}
```

Algorithms, Dasgupta et al.

Dynamic Programming



Compare:
(1) Exhaustive approach
SABE: 1+6+2
SCABE: 2+4+6+2
SCDE: 2+3+1
SCABDE: 2+4+6+1+1

11 additions.
Complexity grows exponentially with the size of graph

(2) DP approach
Dist(A)=min(1, 2+4)=1
Dist(C)=2
Dist(B)=min(dist(A)+6)=1+6=7
Dist(D)=min(dist(B)+1, dist(C)+3)=min(7+1, 2+3)=5
Dist(E)=min(dist(B)+2, dist(D)+1)=min(7+2, 5+1)=6

6 additions.
Complexity grows linearly with the size of graph

Dynamic Programming

Another example: A game of picking up matches.

There are 30 matches on the table. You start by picking up 1~3 matches, then your opponent picks up 1~3 matches. This goes on until the last match is picked up. The person picking up the last is the loser.

The first step: this is what you want to leave to your opponent 29x 

1, 5, 9, 13, 17, 21, 25, 29

Two steps back: this is what you want to leave to your opponent 

One step back: this is what you want to leave to your opponent 

Last step: you want to leave the last match to your opponent 

Dynamic Programming

Another example:
longest common substring

$$M_{i,j} = \begin{cases} 0, & X_i \neq Y_j \\ M_{i-1,j-1} + 1, & X_i = Y_j \end{cases}$$

		T	u	n	e	d		B	o	y	e	r	-	M	o	o	r	e		a	l	g	o	r	i	t	h	m
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	5	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	6	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10

LCS= algorithm

http://illya-keeplearning.blogspot.com/2010/04/dynamic-programming-longest-common.html

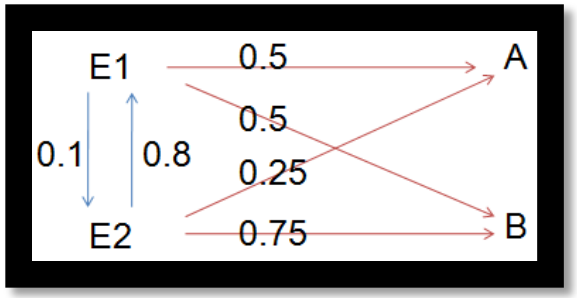
Finding most likely hidden state sequence

Let Q denote the state sequence, and O denote the observed emission. One common goal is to find:

$$\arg\max_{Q} P(Q \mid O)$$

$$P(Q \mid O) = \frac{P(O \mid Q)P(Q)}{\sum_{Q} P(O \mid Q)P(Q)}$$

O="BBB", $P(O \mid Q)P(Q)$



E1->E1->E1: 0.5*0.9*0.9*0.5*0.5*0.5= 0.050625
E1->E1->E2: 0.5*0.9*0.1*0.5*0.5*0.75= 0.0084375
E1->E2->E1: 0.5*0.1*0.8*0.5*0.75*0.5= 0.0075
E1->E2->E2: 0.5*0.1*0.2*0.5*0.75*0.75= 0.0028125
E2->E1->E1: 0.5*0.8*0.9*0.75*0.5*0.5= 0.0675
E2->E1->E2: 0.5*0.8*0.1*0.75*0.5*0.75= 0.01125
E2->E2->E1: 0.5*0.2*0.8*0.75*0.75*0.5= 0.0225
E2->E2->E2: 0.5*0.2*0.2*0.75*0.75*0.75= 0.0084375

But this approach is impractical …

The Forward and Backward Algorithm

$$P(O \mid \lambda) = \sum_{Q} P(O \mid Q, \lambda) P(Q \mid \lambda)$$

If there are a total of S states, and we study an emission from T steps of the chain, then #(all possible Q)=S^T
When either S or T is big, direct calculation is impossible.

Let $\mathrm{O}^{(\mathrm{t})} = (\mathrm{O}_1, \mathrm{O}_2, \ldots, \mathrm{O}_\mathrm{t})$

Let $q^{(t)}$ denote the hidden state at time t,

Let b_i denote the emission probabilities from state E_i
Consider the "forward variables":

$$\alpha(t,i) = P\left(O^{(t)}, q^{(t)} = E_i\right)$$

Emissions up to step t

chain at state i at step t

The Forward and Backward Algorithm

At step 1: $\alpha(1,i) = \pi_i b_i(O_1)$

At step t+1: $\alpha(t+1,i) = \sum_{k=1}^{S} \alpha(t,k)\, p_{ki}\, b_i(O_{t+1})$

… … …

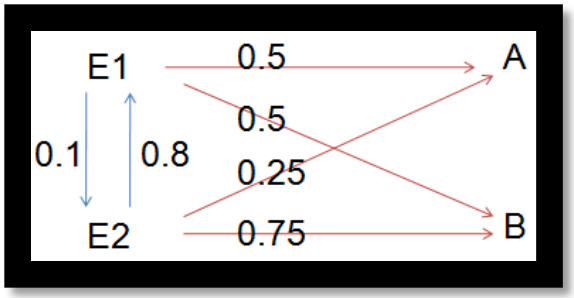
IF we know all $\alpha(T,i)$

Then P(O) can be found by $\sum_{i=1}^{S} \alpha(T,i)$

A total of $2TS^2$ computations are needed.

The Forward and Backward Algorithm

Back to our simple example, find P("BBB")



$$\alpha(1,1) = 0.5 * 0.5$$

$$\alpha(1,2) = 0.5 * 0.75$$

$$\alpha(2,1) = \alpha(1,1) * 0.9 * 0.5 + \alpha(1,2) * 0.8 * 0.5$$

$$\alpha(2,2) = \alpha(1,1) * 0.1 * 0.75 + \alpha(1,2) * 0.2 * 0.75$$

$$\alpha(3,1) = \alpha(2,1) * 0.9 * 0.5 + \alpha(2,2) * 0.8 * 0.5$$

$$\alpha(3,2) = \alpha(2,1) * 0.1 * 0.75 + \alpha(2,2) * 0.2 * 0.75$$

Now. What saved us the computing time?
It is the reusing the shared components.
And what allowed us to do that?.

The short memory of the Markov Chain.

The Forward and Backward Algorithm

The backward algorithm:

$$\beta(t,i) = P\left(O_{t+1}, O_{t+2}, \ldots\ldots, O_T \mid q_t = E_i\right)$$

Emissions after step t

chain at state i at step t

$$\beta(T, j) = 1, \forall j$$

From T-1 to 1, we can iteratively calculate:

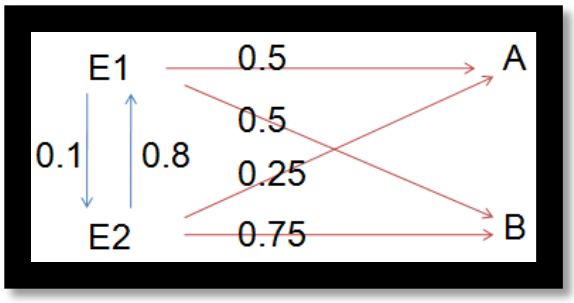
$$\beta(t-1,i) = \sum_{k=1}^{S} p_{ik} b_k(O_t) \beta(t,k)$$

The Forward and Backward Algorithm

Back to our simple example, observing “BBB”, we have

$$\beta(3,1) = 1$$

$$\beta(3,2) = 1$$



$$\beta(2,1) = P(O_3 = "B" \mid q_2 = E1)$$

$$= p_{11} b_1("B") \beta(3,1) + p_{12} b_2("B") \beta(3,2)$$

$$= 0.9 * 0.5 * 1 + 0.1 * 0.75 * 1$$

……

Posterior state probabilities

$$P(O, q^{(t)} = Ei)$$

$$= P(O_1, O_2, \ldots, O_t, q^{(t)} = Ei) P(O_{t+1}, \ldots, O_T \mid O_1, O_2, \ldots, O_t, q^{(t)} = Ei)$$

$$= P(O_1, O_2, \ldots, O_t, q^{(t)} = Ei) P(O_{t+1}, \ldots, O_T \mid q^{(t)} = Ei)$$

$$= \alpha(t,i)\beta(t,i)$$

$$P(O) = \sum_{i=1}^{S} \alpha(T,i)$$

$$P(q^{(t)} = Ei \mid O) = \frac{P(O, q^{(t)} = Ei)}{P(O)} = \frac{\alpha(t,i)\beta(t,i)}{\sum_{i=1}^{S} \alpha(T,i)}$$

The Viterbi Algorithm

To find: $\arg\max_{Q} P(Q \mid O)$

$$\max_{Q} P(Q \mid O) = \max_{Q} \frac{P(Q,O)}{P(O)}$$

$$\arg\max_{Q} P(Q \mid O) = \arg\max_{Q} \frac{P(Q,O)}{P(O)} = \arg\max_{Q} P(Q,O)$$

So, to maximize the conditional probability, we can simply maximize the joint probability.

Define:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P\left(q_1, q_2, \ldots\ldots, q_{t-1}, q_t = Ei, \text{and } O^{(t)}\right)$$

The Viterbi Algorithm

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P\left(q_1, q_2, \ldots\ldots, q_{t-1}, q_t = Ei, \text{and } O^{(t)}\right)$$

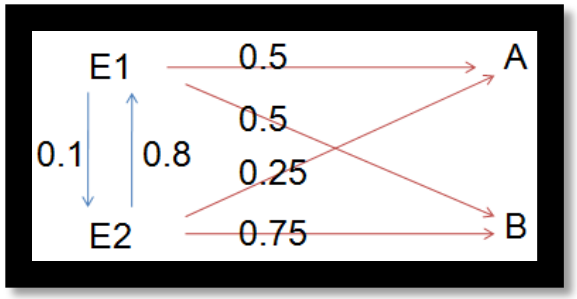
Then our goal becomes: $\max_Q P(Q, O) = \max_i \delta_T(i)$

So, this problem is solved by dynamic programming.

$$Initiation: \delta_1(i) = \pi_i b_i(O1)$$

$$Induction: \delta_t(i) = \max_k \delta_{t-1}(k) p_{ki} b_i(Ot)$$

The Viterbi Algorithm



Time:	1	2	3
States: E1	0.5*0.5	0.15	0.0675
E2	0.5*0.75	0.05625	0.01125

The calculation is from left border to right border, while in the sequence alignment, it is from upper-left corner to the right and lower borders.

Why is this efficient calculation possible?

The short memory of the Markov Chain!

References

Chapter 3, Biological Sequence Analysis,
by Richard Durbin,Sean R. Eddy,Anders Krogh,Graeme Mitchison

Chapter 6, Algorithms,
by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani.