Review: Suffix Arrays and BWT

Suffixes are sorted in the BWT matrix

Define suffix array S:

$S(i) = j$, where $X_j \ldots X_n$ is the i-th suffix lexicographically

BWT(X) constructed from S:
At each position, take the letter to the left of the one pointed by S
Review: Reconstructing BANANA

BWT matrix of string ‘BANANA’

$BANANA$
A$BANAN$
ANA$BAN$
ANANA$B$
BANANA$
NA$BAN AN$
NANA$BA$

C(a) character array:

<table>
<thead>
<tr>
<th>letter</th>
<th>occs before a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>5</td>
</tr>
<tr>
<td>N</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>$</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>

index i:

<table>
<thead>
<tr>
<th>index</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

i: indicating i-th occurrence of ‘a’ in BWT

LF() = C() + i

Reconstruct BANANA:

S := ""; r := 1; c := BWT[r];
UNTIL c = ‘$’ {
    S := cS;
    r := LF(r);
    c := BWT(r);
}
Searching for query “ANA”

Let
LFC(r, a) = C(a) + i, where i = number of a’s up to r in BWT

```plaintext
ExactMatch(W[1...k]) {
    a := W[k];
    low := C(a) +1;
    high := C(a+1);  // a+1: lexicographically next char
    i := k – 1;
    while (low <= high && i >= 1) {
        a = W[i];
        low = LFC(low – 1, a) + 1;
        high = LFC(high, a);
        i := i – 1; }
    return (low, high);
}
```

Credit: Ben Langmead thesis
BWT Index Construction

Reference Sequence Construction

BWT Construction

BWT-auxiliary Structure Construction (C & O arrays) and Compression

.CFA

Credit: Victoria Popic
BWA Inexact Match

Allow up to $n$ mismatches/gaps

**Backward search:**
Given read $W$, keep track of multiple partial alignments

Partial alignment: $(i, z, L, U)$

- $i$: current position
- $z$: remaining non-matches allowed
- $L$: current low
- $U$: current high

```
I ← ∅
I ← I ∪ InexRecur(W, i − 1, z − 1, k, l)
for each $b ∈ \{A, C, G, T\}$ do
    $k ← C(b) + O(b, k − 1) + 1$
    $l ← C(b) + O(b, l)$
    if $k ≤ l$ then
        $I ← I ∪ InexRecur(W, i, z − 1, k, l)$
        if $b = W[i]$ then
            $I ← I ∪ InexRecur(W, i − 1, z, k, l)$
        else
            $I ← I ∪ InexRecur(W, i − 1, z − 1, k, l)$
```
BWA Inexact Match

$W = \text{ACTGTGT}$

Partial alignment 4-tuple: $(i = 4, z = 3, L, U)$

Recursive step:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>gap-ref</th>
<th>gap-read</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AGT</td>
<td>CGT</td>
<td>TGT</td>
<td>GGT</td>
<td>TGTT</td>
<td>*GT</td>
</tr>
<tr>
<td>i-1</td>
<td>z-1</td>
<td>z</td>
<td>z-1</td>
<td>z-1</td>
<td>z-1</td>
<td>z-1</td>
</tr>
<tr>
<td>i-1</td>
<td>i-1</td>
<td>i-1</td>
<td>i-1</td>
<td>i-1</td>
<td>i-1</td>
<td>i</td>
</tr>
<tr>
<td>L^A</td>
<td>L^A</td>
<td>L^C</td>
<td>L^T</td>
<td>L^G</td>
<td>LU</td>
<td>L^A L^C L^T L^G</td>
</tr>
</tbody>
</table>

...GAGT ...GCGT ...GTGT ...GGGT ...G-GT ...GT[A/C/T/G]GT
...GTGT ...GTGT ...GTGT ...GTGT ...GTGT ...GT GT

$L^A = C(A) + O(A, L-1) + 1$

$U^A = C(A) + O(A, L)$

Credit: Victoria Popic
BWA Heuristics

- Lower bound array \( D \), where \( D(i) = \text{LB on number of differences} \) of exactly matching \( R[0,i] \) with the reference (can be computed in \( O(|R|) \) time → check \( n < D(i) \) instead of \( n < 0 \))

- Process best partial alignments first: use a \textit{min}-priority heap to store alignment entries (instead of recursion)

- Prune out alignments considered sub-optimal (although they might have fewer than \( n \) differences):
  \textbf{dynamically adjust} search parameters (e.g. \( n \)):
  
  (1) stop if # top hits exceeds a threshold (=30),

  (2) set \( n = n_{best} + 1 \), where \( n_{best} \) is the # of differences in top hit

- Seeding: limit the number of differences in the \textit{seed} sequence (first \( k \) bp)

- Disallow indels at the ends of the read

Li H, Durbin R.
\textit{Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics, 2009.} 7154 cites

Langmead B, Salzberg SL.

Li H
\textit{Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM}
Hidden Markov Models

\[
x_1, x_2, x_3, \ldots, x_K
\]
Example: The Dishonest Casino

A casino has two dice:
- Fair die
  \[ P(1) = P(2) = P(3) = P(5) = P(6) = \frac{1}{6} \]
- Loaded die
  \[ P(1) = P(2) = P(3) = P(5) = \frac{1}{10} \]
  \[ P(6) = \frac{1}{2} \]

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

QUESTION

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

Prob = 1.3 \times 10^{-35}

This is the EVALUATION problem in HMMs.
Question # 2 – Decoding

GIVEN

A sequence of rolls by the casino player

124552646214614613613661664661636616366163616515615115146123562344

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

12452646214614613613666166466163661636616616515615115146123562344

Prob(6) = 64%

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

- FAIR
  - 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
- LOADED
  - 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 $\pi_t$
Definition: A hidden Markov model (HMM)

- **Alphabet** \( \Sigma = \{ b_1, b_2, \ldots, b_M \} \)
- **Set of states** \( Q = \{ 1, \ldots, K \} \)
- **Transition probabilities** between any two states
  \[
  a_{ij} = \text{transition prob from state } i \text{ to state } j
  \]
  \[
  a_{i1} + \ldots + a_{iK} = 1, \quad \text{for all states } i = 1\ldots K
  \]
- **Start probabilities** \( a_{0i} \)
  \[
  a_{01} + \ldots + a_{0K} = 1
  \]
- **Emission probabilities** within each state
  \[
  e_i(b) = P( x_i = b \mid \pi_i = k)
  \]
  \[
  e_i(b_1) + \ldots + e_i(b_M) = 1, \quad \text{for all states } i = 1\ldots K
  \]
A HMM is memory-less

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

\[
P(\pi_{t+1} = k \mid "\text{whatever happened so far}" ) = \\
P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) = \\
P(\pi_{t+1} = k \mid \pi_t)
\]
A HMM is memory-less

At each time step $t$, the only thing that affects $x_t$ is the current state $\pi_t$

\[ P(x_t = b \mid \text{“whatever happened so far”}) = P(x_t = b \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_{t-1}) = P(x_t = b \mid \pi_t) \]
A parse of a sequence

Given a sequence \( x = x_1 \ldots x_N \),
A **parse** of \( x \) is a sequence of states \( \pi = \pi_1, \ldots, \pi_N \)
Given a HMM, we can generate a sequence of length n as follows:

1. Start at state $\pi_1$ according to prob $a_{0\pi_1}$
2. Emit letter $x_1$ according to prob $e_{\pi_1}(x_1)$
3. Go to state $\pi_2$ according to prob $a_{\pi_1\pi_2}$
4. ... until emitting $x_n$
Given a sequence $x = x_1 \ldots x_N$ and a parse $\pi = \pi_1, \ldots, \pi_N$, to find how likely this scenario is: (given our HMM)

$$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) = P(x_N | \pi_N) P(\pi_N | \pi_{N-1}) \ldots P(x_2 | \pi_2) P(\pi_2 | \pi_1) P(x_1 | \pi_1) P(\pi_1) = a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \ldots e_{\pi_N}(x_N)$$
Likelihood of a parse

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

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

\[
P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) = \prod_{j=1}^{N-1} P(x_j | x_{j+1}) P(\pi_j | \pi_{j-1}) a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \ldots e_{\pi_N}(x_N)
\]

A compact way to write

\[
a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \ldots e_{\pi_N}(x_N)
\]

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

**Example:**

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

Then, count in \( x \) and \( \pi \) the # of times each parameter \( j = 1, \ldots, n \) occurs

\[
F(j, x, \pi) = \# \text{ parameter } \theta_j \text{ occurs in } (x, \pi)
\]

(*call \( F(\ldots, \ldots) \) the feature counts*)

Then, count in \( x \) and \( \pi \) the # of times each parameter \( j = 1, \ldots, n \) occurs

\[
P(x, \pi) = \prod_{j=1}^{N} \theta_j^{F(j, x, \pi)} = \exp \left[ \sum_{j=1}^{n} \log(\theta_j) \times F(j, x, \pi) \right]
\]
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

\[ \pi = \text{Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair} \]

(say initial probs \( a_{0 \text{Fair}} = \frac{1}{2}, a_{0 \text{Loaded}} = \frac{1}{2} \))

\[ \frac{1}{2} \times P(1 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) P(2 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) \ldots P(4 \mid \text{Fair}) = \]

\[ \frac{1}{2} \times (\frac{1}{6})^{10} \times (0.95)^9 = .00000000521158647211 \approx 0.5 \times 10^{-9} \]
Example: the dishonest casino

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

What is the likelihood of

$$\pi = \text{Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?}$$

$$\frac{1}{2} \times P(1 \mid \text{Loaded}) \text{ } P(\text{Loaded, Loaded}) \ldots P(4 \mid \text{Loaded}) =$$

$$\frac{1}{2} \times (1/10)^9 \times (1/2)^1 (0.95)^9 = .00000000015756235243 \sim 0.16 \times 10^{-9}$$

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 \( \pi = F, F, \ldots, F \)?

\[
\frac{1}{2} \times (1/6)^{10} \times (0.95)^9 \approx 0.5 \times 10^{-9}, \text{ same as before}
\]

What is the likelihood \( \pi = L, L, \ldots, L \)?

\[
\frac{1}{2} \times (1/10)^4 \times (1/2)^6 \times (0.95)^9 = 0.00000049238235134735 \approx 0.5 \times 10^{-7}
\]

So, it is 100 times more likely the die is loaded.
A sequence of rolls by the casino player

12455264621461461361366616646616366163661636616366165156151151146123562344

**Question**

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

This is the **evaluation** problem in HMMs

\[\text{Prob} = 1.3 \times 10^{-35}\]
Question # 2 – Decoding

GIVEN

A sequence of rolls by the casino player

```
124526462146146136136661664661636616366163616515615115146123562344
```

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

![Sequence of rolls](image)

Prob(6) = 64%

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 $\text{Prob}[x | M ]$

2. Decoding

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

3. Learning

GIVEN a HMM $M$, with unspecified transition/emission probs., and a sequence $x$,
FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize $P[ x | \theta ]$
Problem 1: Decoding

Find the most likely parse of a sequence
Decoding

GIVEN $x = x_1x_2......x_N$

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

$\pi^* = \arg\max_{\pi} P[x, \pi]$

Maximizes $a_{0\pi_1} e_{\pi_1}(x_1) a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_N}(x_N)$

*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 $i$,

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

What is $V_l(i+1)$?

From definition,

$$V_l(i+1) = \max_{\pi_1 \ldots \pi_i} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l]$$

$$= \max_{\pi_1 \ldots \pi_i} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i) P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$$

$$= \max_{\pi_1 \ldots \pi_i} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i) P[x_1 \ldots x_i, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$$

$$= \max_k [P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\pi_1 \ldots \pi_{i-1}} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]]$$

$$= \max_k [P(x_{i+1} \mid \pi_{i+1} = l) \ P(\pi_{i+1} = l \mid \pi_i = k) \ V_k(i) ]$$

$$= e_l(x_{i+1}) \max_k a_{kl} \ V_k(i)$$
The Viterbi Algorithm

Input: $x = x_1 \ldots x_N$

**Initialization:**
- $V_0(0) = 1$  
  (0 is the imaginary first position)
- $V_k(0) = 0$, for all $k > 0$

**Iteration:**
- $V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$
- $\text{Ptr}_j(i) = \arg\max_k a_{kj} V_k(i-1)$

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

**Traceback:**
- $\pi_N^* = \arg\max_k V_k(N)$
- $\pi_{i-1}^* = \text{Ptr}_{\pi_i}(i)$
The Viterbi Algorithm

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

**Time:**

$O(K^2N)$

**Space:**

$O(KN)$
Underflows are a significant problem

\[ P[ x_1, \ldots, x_i, \pi_1, \ldots, \pi_i ] = a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_i} e_{\pi_1}(x_1) \ldots e_{\pi_i}(x_i) \]

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 $\pi_1$ according to prob $a_{0\pi_1}$
2. Emit letter $x_1$ according to prob $e_{\pi_1}(x_1)$
3. Go to state $\pi_2$ 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 x was generated by the model?
- Given a position i, what is the most likely state that emitted xi?

Example: the dishonest casino

Say x = 12341...23162616364616234112...21341

Most likely path: \( \pi = FF......F \)

(too “unlikely” to transition F \( \rightarrow \) L \( \rightarrow \) F)

However: marked letters more likely to be L than unmarked letters

\[
P(\text{box: FFFFFFFFFFFF}) = (1/6)^{11} * 0.95^{12} = 2.76^{-9} * 0.54 = 1.49^{-9}
\]

\[
P(\text{box: LLLLLLLLLLL}) = [(1/2)^6 * (1/10)^5] * 0.95^{10} * 0.05^2 = 1.56*10^{-7} * 1.5^{-3} = 0.23^{-9}
\]
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
We want to calculate

\[ P(x) = \text{probability of } x, \text{ given the HMM} \]

Sum over all possible ways of generating \( x \):

\[
P(x) = \sum_\pi P(x, \pi) = \sum_\pi P(x | \pi) P(\pi)
\]

To avoid summing over an exponential number of paths \( \pi \), 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:

\[ f_k(i) = \mathbb{P}(x_1 \ldots x_i, \pi_i = k) \]

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

\[ = \sum_{l} \sum_{\pi_1 \ldots \pi_{i-2}} \mathbb{P}(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = l) \ a_{lk} \ e_k(x_i) \]

\[ = \sum_{l} \mathbb{P}(x_1 \ldots x_{i-1}, \pi_{i-1} = l) \ a_{lk} \ e_k(x_i) \]

\[ = e_k(x_i) \sum_{l} f_l(i - 1) \ a_{lk} \]
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

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

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

\[
= P(x_1 \ldots x_i, \pi_i = k) b_{k}(i)
\]

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:**

$$b_k(N) = 1, \text{ for all } k$$

**Iteration:**

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

**Termination:**

$$P(x) = \sum_i a_{0i} e_i(x_1) b_i(1)$$
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

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

Define \( \pi^\wedge \) by Posterior Decoding:

\[ \pi^\wedge_i = \text{argmax}_k P(\pi_i = k \mid x) \]
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 $\pi^*$
  - Posterior Decoding may give an invalid sequence of states (of prob 0)
    - Why?
Posterior Decoding

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

\[= \sum \{\pi : \pi[i] = k\} \, P(\pi \mid x)\]
Viterbi, Forward, Backward

<table>
<thead>
<tr>
<th>VITERBI</th>
<th>FORWARD</th>
<th>BACKWARD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization:</strong></td>
<td><strong>Initialization:</strong></td>
<td><strong>Initialization:</strong></td>
</tr>
<tr>
<td>$V_0(0) = 1$</td>
<td>$f_0(0) = 1$</td>
<td>$b_k(N) = 1$, for all $k$</td>
</tr>
<tr>
<td>$V_k(0) = 0$, for all $k &gt; 0$</td>
<td>$f_k(0) = 0$, for all $k &gt; 0$</td>
<td></td>
</tr>
<tr>
<td><strong>Iteration:</strong></td>
<td><strong>Iteration:</strong></td>
<td><strong>Iteration:</strong></td>
</tr>
<tr>
<td>$V_l(i) = e_l(x_i) \max_k V_k(i-1) a_{kl}$</td>
<td>$f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl}$</td>
<td>$b_l(i) = \sum_k e_l(x_{i+1}) a_{kl} b_k(i+1)$</td>
</tr>
<tr>
<td><strong>Termination:</strong></td>
<td><strong>Termination:</strong></td>
<td><strong>Termination:</strong></td>
</tr>
<tr>
<td>$P(x, \pi^*) = \max_k V_k(N)$</td>
<td>$P(x) = \sum_k f_k(N)$</td>
<td>$P(x) = \sum_k a_{0k} e_k(x_1) b_k(1)$</td>
</tr>
</tbody>
</table>