CS697 Class Notes

Steve Revilak

Sept. 2006 - Dec. 2006

This are Steve Revilak's course notes from CS697, Algorithmic Aspects of Molecular Biology. This course was taught by Professor Dan Simovici at UMass Boston, during the Fall 2006 Semester.

Part 1

Dynamic Programming and Sequence Alignments

1.1 Lecture 1 – 9/7/2006

The course page is `http://www.cs.umb.edu/cs697`.

1.1.1 Overview of DNA

DNA stands for *deoxyribonucleic acid*. A DNA word is a very long sequence of 4 symbols: A, C, G, T. In a human, the length of such a word is rougly 3 billion letters

A, C, G, T are known as *nucleotides*.

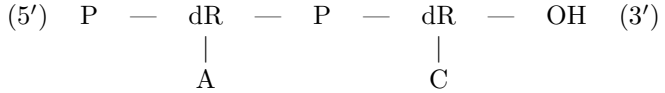
A	adenine
C	cytosine
G	guanine
T	Thymidine

A DNA molecule is an alternating sequence of desoxyribone (dR) and phosphate (P) pieces:

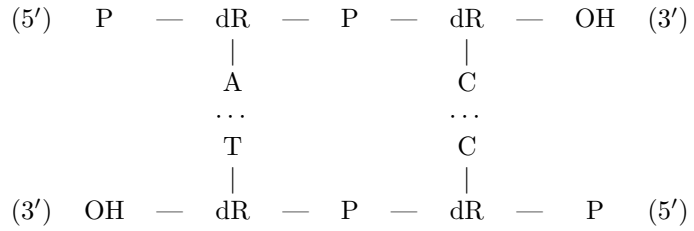
dR — P — dR — P — ...

These constitute the spine or "ladder rails" of the DNA molecule.

The A, C, G, T nucleotides hang from the dR molecules. The chain has a direction. The beginning of the chain is the 5′ (five prime) side and the end is the 3′ (three prime) side. DNA always starts with a P and ends with an OH.



The figure above shows half of a DNA chain. Because DNA is double-stranded, there is a second chain of dR — P molecules, and the second chain goes in the opposite direction. In the middle (the ladder rungs) nucleotides will be aligned in certain combinations. Not every alignment is possible, only certain ones.



The top and bottom halves of the chain can be separated fairly easily. This splitting occurs when DNA replicates.

Cell fabrication uses RNA (ribonucleic acid). RNA is single-stranded. Where DNA is composed of the letters A, C, G, T, RNA is composed of the letters A, C, G, U. U is *uracil*. When RNA is made from DNA, Thymidine molecules will become uracil molecules.

The basic sequence of DNA replication

- We start with DNA
- DNA is used by a cell to produce RNA
- RNA is used to produce protiens

A *protien* is a chain of amino acids. There are 20 such chains.

Sequences of three nucleotides are used as *codons* in the making of amino acids. Because we have three-letter sequences and four possible letters, there are 64 combinations. Note that there are more codons than amino acids.

An *exon* is the part of DNA that is used for protien production.

An *intron* is the part of DNA that is not used for protien production.

A *stop codon* indicates where the transcription of a particular protien should end. There are three stop codons: UAG, UGA, and UAA.

The three symbols in a stop codon are not equally significant. Most of the transcription choice is determined by the first two codons. In many cases, the third codon doesn't matter.

1.1.2 Similarity of Protiens

In biology, protiens have one-letter representations. Let's suppose we had a pair of protien sequences:

D E K A C N P Q ...
E E C T G P ...

How closely do these two sequences match, and how does one measure their degree of similarity? Note that these sequences have different lengths – this is not uncommon for protien sequences.

A common approach to judging similarity is to compute a score that represents the quality of the match. This is called an *alignment score*, which we'll denote as $s(a, b)$.

The *Blosum 50 matrix* is a square matrix indexed by amino acid. The rows and columns are amino acids. Each matrix cell contains a score indicating how well the two protiens match. We can find an overall score by adding up scores for individual pairs. However, we may need to insert gaps in order to get an alignment. Gaps will produce a penalty.

Given two protein sequences, how many different alignments are possible? As it turns out, this reduces to the question "how many different ways can elements of the two sequences be shuffled?".

There is a bijection between shuffle and sequence alignment (one of our homework problems will address this).

Let

$$\begin{aligned} x &= x_1 \dots x_n \\ y &= y_1 \dots y_m \end{aligned}$$

Shuffled, these give a string of length $m + n$. The number of different ways to shuffle (arrange the elements of x and y is: $\binom{m+n}{n}$. If the sequences have the same length ($m = n$) this is

$$\begin{aligned} \binom{m+n}{n} &= \binom{2n}{n} \\ &= \frac{(2n)!}{n! \cdot n!} \\ &= \frac{(2n)!}{n!^2} \end{aligned}$$

1.1.3 Dynamic Programming

Let's consider the problem of finding the longest common subsequence between two sequences.

Give

$$x = x_1 \dots x_n$$

we say that z is a subsequence of x if

$$z = x_{i_1} x_{i_2} \dots x_{i_3} \dots$$

if $i_1 \leq i_2 \leq i_3 \leq \dots$

Note that a subsequence is <u>not</u> the same thing as an infix. In a subsequence, symbols do not have to be consecutive. They only have to appear in the proper order.

A sequence of length n has 2^n possible subsequences.

Given

$$\begin{aligned} x &= x_1 \dots x_n \\ y &= y_1 \dots y_m \end{aligned}$$

how can we find the longest length of a longest common subsequence of x and y. Note that the longest length of a common subsequence <u>is</u> unique, while the subsequence itself <u>is not</u> unique.

Example:

$$\begin{aligned} X &= GTCAGA \\ Y &= AGCGTAG \end{aligned}$$

The longest common subsequence has length 4. There are two such subsequences: $GTAG$ and $GCAG$.

Let $c(m, n)$ denote the length of the longest common subsequence between m and n. This problem can be broken into two cases:

Case 1 If $x_m = y_n$ then

$$c(m, n) = c(m - 1, n - 1) + 1$$

Case 2 If $x_m \neq y_n$ then

$$c(m, n) = \max(\ c(m - 1, n),\ c(m, n - 1)\ )$$

Dynamic programming is similar to divide and conquer in that it decomposes a problem into smaller subproblems. Unlike divide and conquer, dynamic programming saves all of the intermediate results and re-uses them as necessary.

1.1.4 Combinatorial Refresher

Formula for selection of r elements from a set of n elements:

- r permutations, without repetition

$$\frac{n!}{(n-r)!}$$

- r permutations, with repetition

$$n^r$$

- r combinations, without repetition

$$\frac{n!}{r!(n-r)!}$$

- r combinations, with repetition

$$\frac{(n+r-1)!}{r!(n-1)!}$$

1.2 Chapter 1 notes

1.2.1 Conditional Probability

Given one die, the probability of rolling an i on die #1 is $P(i|D_1)$. We call this a *conditional probability*. In general, think of this as the probability of i under the parameter D_1.

1.2.2 Joint Probability

The formula for *joint probability* is

$$P(X, Y) = P(X|Y) \cdot P(Y)$$

$P(X, Y)$ is the probability of Y occurring and the probability of X occurring in the presence of Y.

Consider: at a casino, 1% of the die are loaded. On these loaded die, a 6 will come up 50% of the time. The other 99% of the dice are normal. If you pick up a die, what is the probability of rolling a 6.

There are two things to consider: (1) the probability of rolling a 6 on an unloaded die and (2) the probability of rolling a 6 on a loaded die.

The probability of rolling a 6 on a loaded die is 50%. The probability of picking up a loaded die is 1%.

$$\begin{aligned} P(6, loaded) &= P(6|D_{loaded}) \cdot P(loaded) \\ &= 0.5 \cdot 0.01 \\ &= 0.005 \end{aligned}$$

$$\begin{aligned} P(6, unloaded) &= P(6|D_{fair}) \cdot P(fair) \\ &= 1/6 \cdot 0.99 \\ &= 0.165 \end{aligned}$$

$$\begin{aligned} P(6) &= P(6, loaded) + P(6, fair) \\ &= 0.005 + 0.165 \\ &= 0.17 \end{aligned}$$

1.2.3 Bayes Theorem

Bayes theorem is:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}$$

Example: Using the casino from the previous example, we pick up a die and roll it three times; each roll is a 6. What is the probability that the die we picked up is loaded?

$$\begin{aligned} P(loaded|3 \text{ sixes}) &= \frac{P(3 \text{ sixes}|loaded) \cdot P(loaded)}{P(3 \text{ sixes})} \\ &= \frac{(0.5)^3 \cdot 0.01}{(0.5)^3 \cdot 0.01 + (1/6)^3 \cdot 0.99} \\ &= 0.21 \end{aligned}$$

It's still more likely that we picked up a fair die.

1.2.4 Marginal Probability

Given a conditional or joint probability, we can calculate a *marginal probability* that removes one of the variables:

$$P(X) = \sum_{Y} P(X, Y) = \sum_{Y} P(X|Y) \cdot P(Y)$$

$P(X)$ is the summation over all possible events Y.

Bayes theorem is also known as *posterior probability*.

1.3 Lecture - 9/11/2006

Logistics

- Prof. Simovici will be away from 9/24/2006 - 9/27/2006.
- Make up call on Saturday 9/23/2006 at 10:00 am
- Data mining seminars will be held on Tuesdays from 13:00 - 14:00 in the web lab (3rd floor science center). The seminars are open to anyone who is interested.
- See first paper in dsim's list of publications.

1.3.1 Longest Common Subsequences

We return to the problem of finding the length of a longest common subsequence between to words. We have

$$\begin{aligned} x &= (x_1 \dots x_m) \\ y &= (y_1 \dots y_n) \end{aligned}$$

$$\begin{aligned} x_{1,k} &= (x_1 \dots x_k) \text{ a prefix of } x \\ y_{1,j} &= (y_1 \dots y_j) \text{ a prefix of } y \end{aligned}$$

In finding a longest common subsequence, there are two cases to consider

- $x_n = y_m$ – the last symbols match
- $x_n \neq y_m$ – the last symbols do not match

We denote Longest common subsequence as LCS and the length of a longest common subsequence as LLCS.

Last Symbols Match

If $x_m = y_n$ then

$$\mathsf{LLCS}(x, y) = LLCS(x_{1,m-1}, y_{1,n-1}) + 1$$

If t is a LCS of $x_1 \dots x_{m-1}$ and $y_1 \dots y_{n-1}$, then ta is an LCS of $x_1 \dots x_{m-1}$ and $y_1 \dots y_{n-1}$ where $a = x_m = y_n$

If $x_m = y_n$, then we have reduced the problem to one sub-problem.

Last Symbols Don't Match

If $x_m \neq y_n$, then we have two cases to consider: $(x_{1,m-1}, y)$ and $(x, y_{1,n-1})$. Therefore

$$\mathsf{LLCS}(x, y) = \max(\mathsf{LLCS}(x_{1,m-1}, y), \mathsf{LLCS}(x, y_{1,n-1})$$

This reduces the problem to two sub-problems.

1.3.2 Dynamic Programming

Dynamic programming minimizes the re-computation of sub-problems. We present an algorithm that uses a $(m+1) \times (n+1)$ array, $M[i, j]$. (i is row, j is column). In $M[i, j]$ we will have the length of the $\mathsf{LLCS}(x_{1,i}, y_{1,j})$.

The first row and first column of the matrix are special cases. By definition,

$$\begin{aligned} x_{1,i} &= \lambda \text{ if } i = 0 \\ y_{1,j} &= \lambda \text{ if } j = 0 \end{aligned}$$

Aside from the first row an column, we fill in the array from left to right, from top to bottom as follows.

$$M[i,j] = \begin{cases} M[i-1,j-1] & \text{if } x_i = y_j \\ \max(M[i-1,j], M[i,j-1]) & \text{if } x_i \neq y_j \end{cases}$$

Along with building up LLCS values, we will also maintain pointers (arrows) to show where each value was derived. These arrows allow us to reconstruct (one of) the least common subsequences that produced the LLCS.

In the case where $M[i-1,j] = M[i,j-1]$, we'll prefer $M[i-1,j]$ (the up arrow). This means that there are more than one possible LCS. To recover all of the strings, we'd need to record two pointers when a tie occurs, and work backwards along each branch during the reconstruction.

Example: $x = GTCAGA$, $y = AGCGTAG$.

		A	G	C	G	T	A	G
	0	0	0	0	0	0	0	0
G	0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
T	0	$\uparrow$ 0	$\uparrow$ 1	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2
C	0	$\uparrow$ 0	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 2
A	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
G	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
A	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Figure 1.1: Example of Dynamic LLCS algorithm

The LLCS value resides in the lower right-hand corner.

To find an LCS that produced the LLCS, we simply need to follow the arrows from the lower-right corner. A $\nwarrow$ denotes that two symbols matched – these will be the symbols in the LCS.

Above, the LLCS = 4 and the word is $GTAG$.

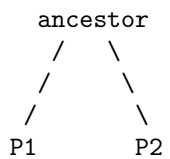
1.3.3 LCS and matching protiens

The dynamic programming technique used in Figure 1.1 can be applied to the problem of aligning amino acids or nucleotides. Let's consider the problem of finding an alignment between two strands of amino acids.

Suppose we have two protiens (using single-letter amino acid designations)

- $DENCTY$
- $DGACY$

We may ask, "Did these protiens come from a common ancestor?". Put another way, can we identify long protien sequences P_1 an P_2, that have evolved from the same ancestry?



We can answer these questions by aligning the protiens. The alignment will tell us what the long sequences are. We should expect our alignment to contain gaps, but we don't necessarily know where or how long these gaps will be.

For $DENCTY$ and $DGACY$, one possible alignment is

$$\begin{array}{cccccccc} D & E & N & C & T & - & Y \\ D & - & G & A & - & C & Y \end{array}$$

Ideally, we'd like our alignment to have (1) perfect matches or (2) amino acids that line up.

Given amino acids a, b it is possible to establish the probability that a, b have evolved from a common ancestor. Call that probability $p_{a,b}$. These probabilities come from biological observations.

Let q_a be the probability of amino acid a occurring independently. If alignment probabilites were independent, the probability of a and b occurring together would be $q_a \cdot q_b$. However, this is not the case in reality.

1.3.4 Probability and Biology

Definition: Odds ratio. The *odds ratio* is

$$\frac{p_{a,b}}{q_a \cdot q_b}$$

Some properties:

- If $\mathcal{A}$ is the set of all amino acids
$$\sum q_a = 1$$
- on $\mathcal{A} \times \mathcal{A}$
$$\sum_a \sum_b q_a \cdot q_b = 1$$
- on $\mathcal{A} \times \mathcal{A}$
$$\sum_{a \in A, b \in A} p_{a,b} = 1$$

We will score according to the log of the odds ratio.

$$s(a,b) = \log \frac{p_{a,b}}{q_a \cdot q_b} \tag{1.3.1}$$

$s(a,b)$ is the score for a particular match. The BLOSUM50 matrix is one such table of scores.

Suppose we have $r_1 \ldots r_n$, $s_1 \ldots s_n$ as a probability distribution.

$$\begin{aligned} \sum_{i=1}^{n} r_i &= 1 \\ \sum_{j=1}^{n} s_j &= 1 \end{aligned}$$

We will introduce a measure of dissimilarity between these distributions (the Kullach-Leibler dissimilarity). This measure is denoted as $KL(r,s)$ or $\mathcal{H}(r|s)$.

$$\mathsf{KL}(r,s) = \sum r_i \cdot \log \frac{s_i}{r_i}$$

Claim: $KL(r,s) \leq 0$ in most cases. If $\mathsf{KL}(r,s) = 0$ then $r = s$.

Claim:

$$\sum q_a \cdot q_b \cdot \log \frac{p_{a,b}}{q_a \cdot q_b} \leq 0$$

Consider: $f(x) = \ln(x)$. The tangent to this line will be $y = x - 1$. Therefore $ln(x) \leq (x - 1)$.

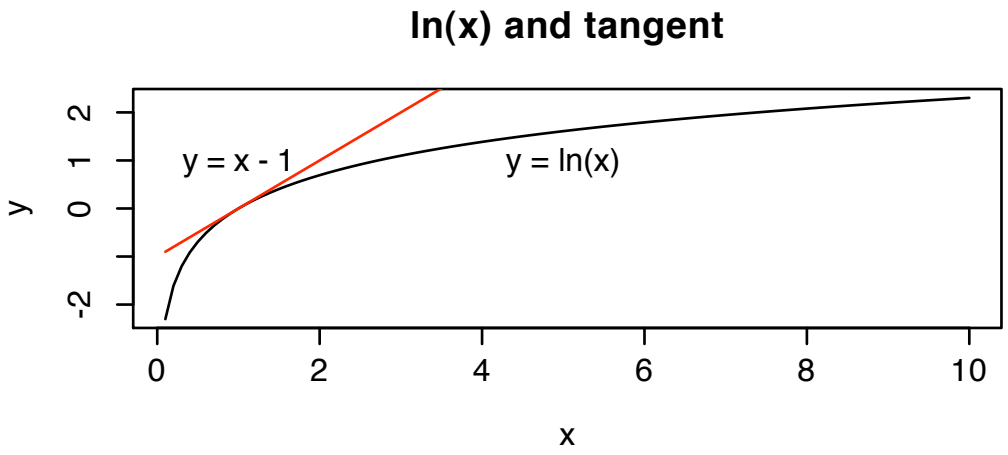


Figure 1.2: Graph of $f(x) = \ln(x)$ and tangent line

From this we see that

$$\begin{array}{rcll} \log(\frac{s_i}{r_i}) & \leq & \frac{s_i}{r_i} - 1 & \\ r_i \cdot \log \frac{s_i}{r_i} & \leq & s_i - r_i & \text{multiply by } r_i \\ \sum r_i \cdot \log \frac{s_i}{r_i} & \leq & \sum s_i - \sum r_i = 0 & \end{array}$$

Equivalently,

$$\sum r_i \cdot \log \frac{r_i}{s_i} \geq 0 \tag{1.3.2}$$

(Note: formula above is the same as KL but with the fraction flipped).

The average score of amino acid alignments is **non-positive**.

1.4 Lecture Notes - 9/13/2006

1.4.1 Alignment Scores

High values of the alignment score $s(a, b)$ suggest that a and b have a common ancestor. $s(a, a)$ should produce the highest possible score (perfect match).

Recall that we are dealing in alignments with gaps: there are three combinations to consider

$$\begin{array}{cccccc} A & \cdots & N & \cdots & - \\ C & \cdots & - & \cdots & Y \end{array}$$

Above, the first pair shows two protiens. The second pair shows a protein in x aligned with a gap in y. The third pair shows a gap in x aligned with a protein in y. It is not legal to align a gap with a gap.

The goal of obtaining the best possible global score will influence our decisions for gap alignments.

As before we assume two sequences, $x = (x_1 \dots x_n)$ and $y = (y_1 \dots y_n)$. Let i be the index in x and let j be the index into y. Let u be a sequence with from x and let v be a sequence with gaps from y.

If (u_i, v_j) are pairs, then the total score will be $\sum(u_i, v_j)$. Note that $u_i, v_j \in (\mathcal{A} \cup \{-\})$.

When gaps are used, we will assign a *gap penalty*. The gap penalty will typically be a constant, and we denote it by d.

For two sequences with gaps, the total gap penalty will be proportional to the number of gaps. If there are k gaps, then the total penalty will be $d \cdot k$.

1.4.2 Needleman-Wunsch Algorithm

The Needleman-Wunsch Algorithm attempts to match full sequences, allowing for gaps.

Given

$$\begin{aligned} x &= x_1 \dots x_m \\ y &= y_1 \dots y_n \end{aligned}$$

we wish to align $x_{1,i}$ and $y_{1,j}$ (prefixes). Let $F(i, j)$ be the score of a prefix of x, y – our goal will be to maximize $F(i, j)$ globally.

There are three possibilities to consider

$$F(i,j) = \max \begin{cases} F(i-i, j-i) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

- In the first case, we match without a gap. In the algorithm given later, we denote these cases with $\nwarrow$
- In the second case, we add a gap to y. We denote these cases with $\leftarrow$.
- In the third case, we add a gap to x. We denote these cases with $\uparrow$.

This assumes we are working with a matrix where (1) x goes across the columns, (2) y goes down the rows, (3) $M[i, j]$ is finding column i and row j.

Example: $x = HEAGAWGHEE$, $y = PAWHEAE$.

Below is a matrix based on these sequences where each cell contains the BLOSUM50 score of the matches.

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	16	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

Note that the highest scores are those with an exact match.

The matching technique is similar to the one we used to find longest common subsequences of strings. Here we are using scores instead of lengths and we are inserting gap penalties.

Note that we fill the $[0, 0]$ cell with 0, and use that to derive gap penalties of increasing lengths across the first row and down the first column.

For ties, our order of preference is $\nwarrow$, $\uparrow$, and $\leftarrow$.

		H	E	A	G	A	W	G	H	E	E
	0	$\leftarrow$ -8	$\leftarrow$ -16	$\leftarrow$ -24	$\leftarrow$ -32	$\leftarrow$ -40	$\leftarrow$ -48	$\leftarrow$ -56	$\leftarrow$ -64	$\leftarrow$ -72	$\leftarrow$ -80
P	$\uparrow$ -8	$\nwarrow$ -2	$\nwarrow$ -9	$\nwarrow$ -17	$\leftarrow$ -25	$\nwarrow$ -33	$\leftarrow$ -41	$\leftarrow$ -49	$\leftarrow$ -57	$\nwarrow$ -65	$\nwarrow$ -73
A	$\uparrow$ -16	$\nwarrow$ -10	$\nwarrow$ -3	$\nwarrow$ -4	$\leftarrow$ -12	$\nwarrow$ -20	$\leftarrow$ -28	$\leftarrow$ -36	$\leftarrow$ -44	$\leftarrow$ -52	$\leftarrow$ -60
W	$\uparrow$ -24	$\uparrow$ -18	$\uparrow$ -11	$\nwarrow$ -6	$\nwarrow$ -7	$\nwarrow$ -15	$\nwarrow$ -5	$\leftarrow$ -13	$\leftarrow$ -21	$\leftarrow$ -29	$\leftarrow$ -37
H	$\uparrow$ -32	$\nwarrow$ -14	$\nwarrow$ -18	$\nwarrow$ -13	$\nwarrow$ -8	$\nwarrow$ -9	$\uparrow$ -13	$\nwarrow$ -7	$\nwarrow$ -3	$\leftarrow$ -11	$\leftarrow$ -19
E	$\uparrow$ -40	$\uparrow$ -22	$\nwarrow$ -8	$\leftarrow$ -16	$\nwarrow$ -16	$\nwarrow$ -9	$\nwarrow$ -12	$\uparrow$ -15	$\nwarrow$ -7	$\nwarrow$ 3	$\nwarrow$ -5
A	$\uparrow$ -48	$\nwarrow$ -30	$\uparrow$ -16	$\nwarrow$ -3	$\leftarrow$ -11	$\nwarrow$ -11	$\nwarrow$ -12	$\nwarrow$ -12	$\uparrow$ -15	$\uparrow$ -5	$\nwarrow$ -2
E	$\uparrow$ -56	$\uparrow$ -38	$\uparrow$ -24	$\uparrow$ -11	$\nwarrow$ -6	$\nwarrow$ -12	$\nwarrow$ -14	$\nwarrow$ -15	$\nwarrow$ -12	$\nwarrow$ -9	$\nwarrow$ 1

The maximum score is 1, with a sequence

H	E	A	G	A	W	G	H	E	–	E
–	–	P	–	A	W	–	H	E	A	E

Note that this sequence has length 11, and that we started with sequences of lengths 10 and 7.

Finally, note how $\uparrow$ and $\leftarrow$ "point" to the sequence where the gap should be inserted.

As noted above, this algorithm is intended to match full sequences while allowing for gaps (aka - a "gapped alignment of a full sequence"). Other algorithms have different goals.

1.4.3 Waterman-Smith Algorithm

The Waterman-Smith algorithm attempts to find alignments with infixes. The algorithm seeks to compute $F(i, j) \geq 0$. Because we are only matching infixes (not entire sequences), we don't care about gaps.

The decomposition technique is similar to the Needleman-Wunsch algorithm, but there are four cases to consider:

$$F(i,j) = \max \begin{cases} 0 \\ F(i-i, j-i) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

1.4.4 Resources

- ClustalX – a program for doing gene manipulation and analysis. Download from `ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalX/`.
- Biological databases `http://www.ncbi.nlm.nih.gov/`. This is the National Center for Biotechnology Information, courtesty of the National Library of Medicine, courtesy of the National Institute of Health.

1.5 Lecture - 9/18/2006

Along with a few notes from the course text.

1.5.1 Smith-Waterman Algorithm

The Smith Waterman algorithm is designed to find the best alignment between subsequences of x and y. The highest-scoring subsequence alignment is called the best *local alignment*.

The Smith-Waterman algorithm is based on the following recurrence relation:

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

Where $s(x_i, y_j)$ is a scoring function and d is the gap penalty.

Note that the effect of the first case in this formula is to discard any scores with negative values – our matrix will contain only scores that are ≥ 0. Taking the option of zero corresponds to starting a new subsequence alignment.

The matrix used to compute this recurrence relation will have

- zeros across the top row (as opposed to $-id$)
- zeros down the first column (as opposed to $-jd$)

When using traceback to reconstruct an alignment, we will start from the cell that contains the maximum score. This cell can be anywhere in the matrix – it need not be the one in the lower right hand corner.

Traceback stops when we reach a cell with a value of zero. (We don't include the zero-valued pair in the alignment).

This algorithm asumes that the score of a random match is negative (it won't work otherwise).

Example: $HEAGAWGHEE$ and $PAWHEAE$.

For references, below are the BLOSUM50 alignment scores for these two protien sequences.

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	16	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

The matrix produced by running the Waterman-Smith algorithm on these sequences (with $d = 8$):

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	**0**	0	0	0	0	0	0
A	0	0	0	$\nwarrow$ 5	0	$\nwarrow$ **5**	0	0	0	0	0
W	0	0	0	0	$\nwarrow$ 2	0	$\nwarrow$ **20**	$\leftarrow$ **12**	$\leftarrow$ 4	0	0
H	0	$\nwarrow$ 10	$\leftarrow$ 2	0	0	$\nwarrow$ 0	$\uparrow$ 12	$\nwarrow$ 18	$\nwarrow$ **22**	$\leftarrow$ 14	$\leftarrow$ 6
E	0	$\uparrow$ 2	$\nwarrow$ 16	$\leftarrow$ 8	$\leftarrow$ 0	0	$\uparrow$ 4	$\uparrow$ 10	$\nwarrow$ 18	$\nwarrow$ **28**	$\nwarrow$ 20
A	0	0	$\uparrow$ 8	$\nwarrow$ 21	$\leftarrow$ 13	$\nwarrow$ 5	0	$\nwarrow$ 4	$\uparrow$ 10	$\uparrow$ 20	$\nwarrow$ 27
E	0	0	$\nwarrow$ 6	$\uparrow$ 13	$\nwarrow$ 18	$\nwarrow$ 12	$\leftarrow$ 4	0	$\nwarrow$ 4	$\nwarrow$ 16	$\nwarrow$ 26

The numbers in bold show the best local alignment (we start from 28, because 28 is the largest value in the matrix). Note that we do not include the pair $(E, A) = 27$ in the last column. Including that pair would yield a longer subsequence, but the score would be less (27 vs. 28).

The alignment is

$$\begin{array}{cccccc} A & W & G & H & E \\ A & W & - & H & E \end{array}$$

Again, we omit the zero-valued pair from the last cell in the traceback.

Note that another subsequence is

$$\begin{array}{ccc} H & E & A \\ H & E & A \end{array}$$

but this has a smaller score (21 vs. 28).

1.5.2 Repeated Matches – Multiple Local Alignments

Another variation on the problem asks us to find all high-scoring local alignments that do not overlap.

This approach assumes that we are only interested in local alignments whose score is above a threshold T.

We define two recurrence relationships: one for the first row ($F(i, 0)$), and another for the rest of the matrix.

$$\begin{aligned} F(0,0) &= 0 \\ F(i,0) &= \max \begin{cases} F(i-1,0) \\ F(i-1,j) - T \end{cases} \quad \text{for } 1 \leq j \leq m \end{aligned}$$

For the rest of the matrix:

$$F(i,j) = \max \begin{cases} F(i,0) \\ F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

Note carefully how this relation requires the matrix be computed. To compute $F(i, 0)$, we need to know $F(i-1, j)$ for all j in the previous column. In other words, we must fill down columns (as opposed to filling across rows).

Finally, we add one more cell: $F(n+1, 0)$ computed as $F(i, 0)$ are computed. This final cell will tell us where to begin traceback.

The final alignment will be partitioned into two types of regions: (1) regions containing alignments with gaps and (2) unmatched regions.

- $F(i, j)$, $j \geq 1$ holds the best score for a local match of $x_{1,i}$.
- $F(i, 0)$ contains the best scores of matches so far (less the threshold)
- $F(i, 0)$ cells will have pointers to a cell in the previous column, if the $F(i, 0)$ value was derived from a previous-column cell. ($j \geq 1$).
- When traceback hits a cell in row zero, this corresponds to a break between local subsequences.

1.5.3 What is an Alignment

The following is a fairly precise definition of what an alignment is.

An alignment is a set of pairs $p_1 \dots p_l$ such that

$$p_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix}$$

Where

$$\begin{aligned} u_i &\in x_1 \dots x_n \cup \{-\} \\ v_i &\in y_1 \dots y_m \cup \{-\} \end{aligned}$$

To convert from alignments with gaps to the orignal sequences, we define a morphism

$$h(t) = \begin{cases} t & \text{if } t \in (\{x_1 \dots x_n\} \cup \{y_1 \dots y_n\}) \\ \lambda & \text{if } t = - \end{cases}$$

$h(t)$ maps alignments with gaps back to the original symbols.

1.5.4 Notes on Homework #1

Problem 1a contains a small flaw. Suppose we have an alignment that includes

$$\begin{matrix} x_1 & - \\ - & y_1 \end{matrix}$$

The w produced by these pairs would be $w = x_1 \ - \ - \ y_1 = x_1 y_1$ We cannot differentiate between the original alignment (above) and

$$\begin{matrix} x_1 \\ y_1 \end{matrix}$$

Therefore, we will modify problem #1 to prohibit alignments such that w would contain two consecutive gap symbols.

For part b, this implies that the number of possible alignments is *at least* $\binom{m+n}{m}$. Because of the ambiguity noted above, the number of possible alignments could actually be greater.

1.6 Lecture – 9/20/2006 (part 1)

1.6.1 Multiple Local Alignments (cont'd)

Picking up from the last lecture, we are looking for a way to find multiple local alignments between a pair of sequences. More precisely, given

$$x = x_1 \dots x_n$$
$$y = y_1 \dots y_m$$

we are seeking to align fragments of x with fragments of y, and we are looking for fragments whose score exceeds a specified threshold T. The algorithm uses dynamic programming. All cells in the $M[i, j]$ matrix will have non-negative values.

Our recurrence relations are

$$F(0,0) = 0$$
$$F(i,0) = \max \begin{cases} F(i,0) \\ F(i-1,j) - T \quad \text{for } 1 \le j \le m \end{cases}$$
$$F(i,j) = \max \begin{cases} F(i,0) \\ F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

A few things to note about these relations

- $F(i, 0)$ depends on all cells in column $i - 1$. As a result, we have to fill the array by going down each column. We can't work across rows.
- Once we find a value that exceeds T, that value will become the basis for the next $F(i, 0)$ (using $\max(F(i - 1, j) - T)$). $F(i, 0)$ cells computed in this way will have pointers to the $F(i - 1, j)$ cell from which they were derived.

Below is an example of this algorithm. For convenience, we repeat the set relevant BLOSUM50 scores.

	H	*E*	*A*	*G*	*A*	*W*	*G*	*H*	*E*	*E*
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	5	0	5	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	15	-3	-3	-3	-3
H	16	0	-2	-2	-2	-3	-2	10	0	0
E	0	6	-1	-3	-1	-3	-3	0	6	6
A	-2	-1	5	0	5	-3	0	-2	-1	-1
E	0	6	-1	-3	-1	-3	-3	0	6	6

This matrix is computed with $T = 20$ and $d = 8$.

		H	E	A	G	A	W	G	H	E	E	
	0	0	0	0	**1**	1	1	1	1	3	**9** ←	**9**
P	**0**	0	0	0	**1**	1	1	1	1	3	9	
A	**0**	0	0	↖ 5	1	↖ **6**	1	1	1	3	9	
W	**0**	0	0	0	1	1	↖ **21**	← **13**	← 5	3	9	
H	0	↖ **10**	← 2	0	1	1	↑ 13	↖ 19	↖ **23**	← 15	9	
E	0	↑ 2	↖ **16**	← 8	1	1	↑ 5	↑ 11	↖ 19	↖ **29**	← 21	
A	0	0	↑ 8	↖ **21**	← 13	↖ 6	1	↖ 5	↑ 11	↑ 21	↖ 28	
E	0	0	↖ 6	↑ 13	↖ 18	↖ 12	← 4	1	↖ 5	↖ 17	↖ 27	

The traceback produces the alignment

H	E	A	G	A	W	G	H	E	E
H	E	A	.	A	W	–	H	E	.

The "." characters denote breaks in the partial matches.

Part 2

Hidden Markov Models

2.1 Lecture - 9/20/2006 (part 2)

2.1.1 Hidden Markov Models

We'll use the terms "Markov Chain", "Markov Process", and "Markov Model" interchangeably. See course web site for introductory paper on Markov Models.

In order to understand markov models, we'll need to background in probability. Recall that independent probabilities are calculated as

$$P(x = a \ \wedge \ Y = b) = P(x = a) \cdot P(y = b)$$

Suppose that we have two functions f, g

$$f, g \colon \mathbb{R} \to \mathbb{R}$$

Let x, y be two independant variables where $x, y \in \mathbb{R}$. How can we prove that $f(x)$ and $g(y)$ are also independant? More precisely, we want to prove that

$$P(f(x) = c \ \wedge \ g(y) = d) = P(f(x) = c) \cdot P(g(y) = d)$$

Because f, g are functions we'd like to think that because each x and y map to a single c and d the independence is obvious. However, it's not quite that simple. Given $f(x) = c$, c could be the image under several values of x. Similarly, for $g(y) = d$, d could be the image under several values of y.

Let a_p represent some possible value of c and let b_q represent some possible value for d. We can think of the probabilities as

$$P(f(x) = c \wedge g(y) = d)$$
$$P(\bigcup_p (x = a_p) \wedge \bigcup_q (y = b_q))$$

The second line treats the ranges of the functions as sets.

Next, we take the second line and pull the unions to the front.

$$P(\bigcup_p \bigcup_q (x = a_p \wedge y = b_q))$$

And replace the unions by sums

$$\Sigma_p \Sigma_q P(x = a_p \wedge y = b_q)$$

Now we're clearly dealing with independent events, so

$$\begin{aligned}&\Sigma_p\Sigma_q\left(P(x=a_p)\cdot P(y=b_q)\right)\\&\Sigma_pP(x=a_p)\cdot\Sigma_qP(y=b_q)\\\therefore&P(f(x)=c)\cdot P(g(y)=d)\end{aligned}$$

Thus, $f(x)=c$ and $g(y)=d$ can be considered as independent events.

2.1.2 Stochastic Processes

At its most basic form, a stochastic process is simple a sequence of random variables – $x_0, x_i, \ldots, x_n$

Consider the following: we have a closed vessel with two halves separated by a porous partition. There are a total of N molecules in the vessel, x_t in side A and $N-x_t$ in side B. At regular time intervals we will randomly move a molecule from one side to the other. In this problem, the state of our vessel changes at discrete moments in time.

Let x_t be the number of molecules in side A at time t. For two successive values of t, the difference in x_t will be $+1$ or -1. The state at $t+1$ is determined by

$$x_{t+1}=x_t+Z_{t+1}$$

each Z value will be selected from the domain $\{-1,1\}$. If $x_{t-1}=i$, then

$$Z_n=\begin{pmatrix}1 & -1\\ \frac{N-i}{N} & \frac{i}{N}\end{pmatrix}$$

[srevilak – I'm not quite sure what that notation for Z_n means]

Suppose we have the sequence

$$x_n=i_n, x_{n-1}=i_{n-1}, \ldots, x_0=i_0$$

what is the probability that the next discrete state will be

$$x_{n+1}=i_{n+1}$$

Recall the definition of conditional probability

$$P(A|B)=\frac{P(A\wedge B)}{P(B)}$$

The events A and B must both be satisifed, and B must occur. Note that for $P(A|B)$ to be useful, B must not be impossible.

Returning to our sequence, we are interested in the probability of

$$P(x_{n+1}=i_{n+1}|x_n=i_n, x_{n-1}=i_{n-1}, \ldots x_0=i_0)$$

If you work out the iterations down to x_0, many of the terms cancel out leaving

$$P(x_{n+1}=i_{n+1}|x_n=i_n)$$

This is called the *Markov Equation*. The key point to note is that the $(n+1)^{th}$ step depends only on the n^{th} step. The events that happened prior to n don't matter.

2.1.3 White Noise Driven Markov Processes

Suppose we have a sequence $Z_1 \dots Z_n$, where each Z_i is independent and identically distributed (e.g. - like a random number generator?) Let us also have an x_0 whose value is independent of any Z_i. Let

$$Z_i \in \mathcal{U}$$
$$x_0 \in \mathcal{S}$$

and let f be a function

$$f : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$$

Let us have a sequence of x_i values, determined as follow:

$$\begin{aligned} x_0 \\ x_1 &= f(x_0, Z_1) \\ x_2 &= f(x_1, Z_2) \\ &\dots \\ x_{n+1} &= f(x_n, Z_{n+1}) \end{aligned}$$

The Z_i are called *white noise*. x_{n+1} is based on x_n, plus white noise from a Z value.

The values $x_0, x_1, \dots, x_n$ satisify the markov equality, because x_0 is independent of any Z value.

Let $x_0 = i_0$, $x_1 = i_1$, $\dots x_n = i_n$. We have

$$P(f(i_n, Z_{n+1}) = i_{n+1}) = P(x_{n+1} = i_{n+1} | x_n = i_n)$$

2.1.4 Markov Models and Genetics

Recall that a DNA sequence is a word written with the symbols A, C, G, T.

C has a special role: together C, T, U form the pyrimidines.

C can mutate into a T through a process called methylation. C is vulnerable to this mutation when paired with a G. The nucleotide pair C–G is frequently written CpG (to distinguish it from the C-G pair across base strands. This leads to CG pairings to occur more often than one would expect given their base probabilities of occurrence.

For biologically important reasons, the methylation process is supressed in certain parts of the DNA sequence – typically the start of a gene region. In these regions we see more CpG dinucleotide pairs than in other parts of the gene. Such regions are called *CpG islands*.

How can we recognize CpG islands, vs normal areas of the sequence? This problem can be solved using Markov processes.

We assume that in a DNA sequence, the nucleotide in position k is determined by the nucleotide in position $k - 1$. This is precisely the set of conditions where Markov models are applicable.

2.2 Lecture - 9/23/2006

We'll return the the discussion of CpG islands. CpG islands are portions of the DNA sequence where cytosine is not mutated (methylated).

2.2.1 Markov Property

The essential property of Markov Models is that their memory goes back only 1 step. Stated formally

$$P(x_{n+1} = q_{q+1} | x_n = q_n)$$

Note that $P(x_{n+1} = q_{q+1} | x_n = q_n)$ is a function that depends entirely on n. If

$$P(x_{n+1} = q | x_n = q')$$

then the transiation to q depends only on q', and we say that the Markov Model is *homogenous*.

Let $P(x_0 = q)$ be the probability that the initial state of the model is q. Also assume that the set of model states is finite. The means that x_0 is a random variable which can take on a finite number of values. We can represent the initial distribution of the starting states as a vector

$$x_0 = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \end{pmatrix}$$

Because each initial state π_i is independant and mutually exclusive, the sum of the probabilities of the initial states will be one:

$$\sum_{i=1}^{n} \pi_i = 1$$

Now consider x_1. Let's say that the set of possible states for x_1

$$x_1 = \{S_1, S_2, \ldots, S_n\}$$

The probability function $P(x_{t+1} = S_j | x_t = S_i)$ is a function of i and j. If we are in state S_i at time t, then at time $(t+1)$ we will from state S_i to state S_j, where $S_j \in \{S_1 \ldots S_n\}$.

Let a_{ij} denote the probability of moving from S_i to S_j. For a given S_i, the sum of the probabilities of the available S_j must sum to one:

$$\sum_{j=1}^{n} a_{ij} = 1$$

This will hold for all states S_i.

The transition probabilities for a Markov Model can be specified by an $(n \times n)$ matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

The sum of each row must be one. Also, because each a_{ij} is a probability value, that value must be non-negative. We call a matrix that has these properties a *Stochastic matrix*.

Suppose that $\mathbf{v} \in \mathbb{R}^n$. $\mathbf{v}$ is a vector of n real numbers. If we multiply $A \cdot \mathbf{v}$, then we have

$$A \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$$

- λ is an eigenvalue
- $\mathbf{v}$ is an eigenvector

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

This gives us equations

$$\begin{aligned} a_{11}x + a_{12}y &= \lambda x \\ a_{21}x + a_{22}y &= \lambda y \end{aligned}$$

$$\begin{aligned} (a_{11} - \lambda)x + a_{12}y &= 0 \\ a_{21}x + (a_{22} - \lambda)y &= 0 \end{aligned}$$

And the determinant

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = 0$$

(the vertical bars denote determinant).

The characteristic equations of this matrix are

$$\begin{aligned} (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} &= 0 \\ \lambda^2 - \lambda(a_{11} + a_{12}) + a_{11}a_{22} - a_{12}a_{21} &= 0 \end{aligned}$$

There will be two solutions, λ_1 and λ_2. These are the eigenvalues.

For a stochastic matrix

$$\begin{aligned} a_{11} + a_{12} &= 1 \\ a_{21} + a_{22} &= 1 \end{aligned}$$

$$\begin{vmatrix} a_{11} - \lambda & 1 - \lambda \\ a_{21} - \lambda & 1 - \lambda \end{vmatrix} = 0$$

(srevilak – I'm not sure I got that last determinant right).

A summary of properties of stochastic matricies

- They are square
- all values are non-negative
- the sum of the values in each row is one
- a stochastic matrix raised to any power is still a stochastic matrix.

Assume we have an initial distribution of starting states

$$\mathbf{v} = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \end{pmatrix}$$

The probability $P(x_0 = i)$ comes from the initial data in the Markov model. For the next state,

$$\begin{aligned} P(x_1 = j) &= \sum_{i=1}^{n} P(x_1 = j | x_0 = i) \cdot P(x_0 = i) \\ &= \sum_{i=1}^{n} \pi_i a_{ij} \\ &= (\pi^t \cdot A)_j \end{aligned}$$

(above, the superscript t means "transposed")

We can also view this as a transition vector:

$$\begin{aligned}\pi_1^t &= \pi_0^t \cdot A \\ \pi_{n+1}^t &= \pi_n^t \cdot A \\ \pi_{n+1}^t &= \pi_n^t \cdot A \cdot \pi_{n-1}^t \cdot A \\ \pi_n^t &= \pi_0^t \cdot A^n\end{aligned}$$

2.2.2 Using Markov Models for Detecting CpG Islands

The premise for using Hidden Markov Models to detect CpG islands in a DNA sequence is that nucleotides are placed in a random (but disciplined) fashion. We assume that nucleotide n depends only on nucleotide $(n-1)$. This may be a dubious assumption, but it is an accepted assumption.

We can regard a nucleotide as a random variable which comes from the alphabet $\{A, C, G, T\}$.

$$P(x_{n+1} = S_j | x_n = S_i) = a_{ij}$$

we can view this as a graph, where nodes are states, edges are transitions, and each edge is labeled with its associated probability.

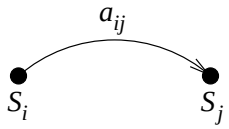


Figure 2.1: Transition from S_i to S_i with probability a_{ij}

In this case, where each state produces a single output, we can represent the model as a strongly connected graph: All possible edges are present in Figure 2.2 – any nucleotide is allowed to follow any

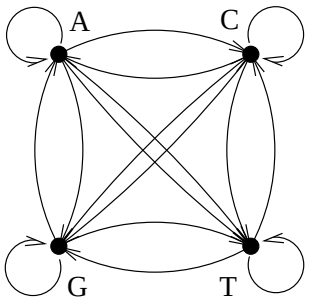


Figure 2.2: Markov Model for ACGT

other nucleotide.

The transition probabilities for Figure 2.2 are be determined empirically. People have looked at many long strands of DNA and counted the frequency of dinucleotides to get a_{CG}, a_{GC}, a_{AG}, etc.

- Let A_{island} be a 4×4 matrix for showing probability in CpG islands.
- Let $A_{non-island}$ be another 4×4 matrix for showing probability outside of CPG islands.

A model to detect CPG islands would make use of both matrix (different a_{ij} values inside an outside of CpG islands). Additionally, there will be transitions from each 4×4 matrix to the other.

When analyzing data, we would like a way to tell which of the matricies the data most closely resembles – if the data is closer to A_{island} we are probably in a CpG island. If the data is closer to $A_{non-island}$ the we are probably outside of a CpG island.

2.2.3 Hidden Markov Models

Hidden Markov Models (HMMs) are similar to Markov Models (aka Markov Chains). In both cases, there is a set of states and a set of probabilities giving the probability of transitions between pairs of states.

With hidden Markov Models:

- Each state q is capable of producing some set of output symbols. Symbols are produced by q in accordance with a probability distribution.
- Outputs (symbols produced by) a hidden markov model are the observable part of the process.
- State transitions are not observable. The state transitions are the "hidden" part.
- A useful question to ask – given a set of output, how can we recover the state transitions that (most probably) generated that output?

Logistics

- No class next Monday (9/25) or Wednesday (9/27).
- The next class will be 10/2/2006.

2.3 Hidden Markov Models - Durbin

Hidden Markov models are characterized by a series of states.

For each state transition (eg - for each path between a connected pair of states) there is a probability associated with the transition. We denote the probability of transitioning from state k to state l as

$$a_{kl} = P(\pi_i = l | \pi_{i-i} = k)$$

Above π_i is the i^{th} element of the path π. i prepresents discreet time units.

Thus, a_{kl} gives us the probability of being at state l at time i under the premise that we were in state k at time $(i-1)$.

Each state is also associated with a set of emission probabilities. A symbol is emitted each time we reach a state. Emission properties are denoted

$$e_k(b) = P(x_i = b | \pi_i = k)$$

Here x_i is the i^{th} symbol in the output string x. $e_k(b)$ is the probability that the i^{th} symbol of x is b, under the premise that the i^{th} member of the state path π is k. (In other words, we were in state k at time i).

Finally, there is also a set of probabilities that are used to choose the initial state. We denote these as a_{0k}. If the model has a unique starting state, the initial probablity of that state will be one; the initial probability of all other states will be zero.

2.4 More Probability Notes

Basic probability formula

$$P(A) = \frac{\#\text{ of successful outcomes}}{\text{number of possible outcomes}}$$

Probability of Independent Events

$$P(AB) = P(A) \cdot P(B)$$

Conditional Probability

$$P(A|B) = \frac{P(AB)}{P(B)}$$

This is the probability of A occurring under the condition B.

Independent Events: outcomes are not affected by other outcomes

Dependent Events: outcomes are affected by other outcomes.

Multiplication Rule

To compute *joint probability* (the probability of two ore more independent events occuring), multiply the individual probabilities of each event.

Addition Rule

Given mutually exclusive events, to find the probability of at least one occurring, add probabilities.

Probability of Non-Mutually Exclusive Events

$$P(A + B) = P(A) + P(B) - P(AB)$$

Example: we draw a card from a standard deck. What is the probability of drawing an ace or a spade?

$$\begin{aligned} P(\text{ace}) &= \frac{4}{52} \\ P(\text{spade}) &= \frac{13}{52} \\ P(\text{ace of spades}) &= \frac{1}{52} \end{aligned}$$

$$\begin{aligned} P(\text{ace} + \text{spade}) &= \frac{4}{52} + \frac{13}{52} - \frac{1}{52} \\ &= \frac{16}{52} \\ &= \frac{4}{13} \end{aligned}$$

2.5 Notes on Markov Models

These are notes taken from the first half of the IEEE paper *A Tutorial on Hidden Markov Models and Selection Applications in Speech Recognition*, Lawrence R. Rabiner, Proceedings of the IEEE, Vol 77 No. 2, Feb. 1989.

A Markov Chain consists of a series of states. At regularly spaced discrete intervals of time the system undergoes a state change (changing from state s_t to s_{t+1}).

A probability is associated with every path between a pair of states, a_{ij}. This denotes the probability of moving from state s_i to state s_j. The probability of transitioning to a state s_i depends *only* upon the state s_{i-1}. (Stochastic process).

2.5.1 HMM Characteristics

Hidden Markov Models are defined by five characteristics.

1. The set of states, $S = (s_1, s_2, \ldots, s_n)$.
2. The set of observable outputs $V = (v_1, v_2, \ldots, v_m)$.
3. The transition probability distribution a_{ij}. This distribution gives the probability of moving from state s_i to state s_j.
4. The observation symbol probability distribution, $b_j(k)$. This is the probability that symbol k is produced by state s_j. Each state can have a different symbol probability distribution.
5. The initial state distribution probability, π. This is used to determine the starting state of the process. If the model has a single starting state, then we will have $\pi_i = 1$ for some s_i and $\pi_j = 0$ for all $s_j \neq s_i$.

When the model runs, the output is a series of observations, $o_1, o_2 \ldots o_T$, where each $o_i \in V$. The series of states that produced this output is "hidden".

How an HMM Runs

A hidden markov model runs according to the following procedure.

1. Choose the initial state $q_1 = s_i$, according to the initial state distribution π. (Note s_i denotes a single state in the model. q_1 denotes the first state in the path of state transitions).
2. Set $t = 1$. t is the time interval counter.
3. Choose o_t, according to the observable symbol distribution property for the current state s_i. (i.e. - $o_t = b_i(k)$)
4. Transition to a new state $q_{t+1} = s_j$, according to the state transition probability a_{ij}.
5. Set $t = t + 1$. If $t < T$, repeat from step (3). Otherwise, halt. Here, T represents the number of discrete time intervals for which the model is to be run.

We can denote models as a 5 tuple

$H = (S, O, A, B, \pi)$

Note: There is a small difference between the paper's notation, and the notation we've used in class. We use O for the set of output symbols (instead of V), and we denote the model with H instead of λ.

2.5.2 Three Basic Problems For Hidden Markov Models

1. Given a sequence of observations $O = o_1, o_2, \ldots o_T$, how do we compute the probability that this sequence of outputs was produced by our model. This problem can also be viewed as the question "how well does our model match a particular observed sequence?".
2. Given a sequence of observations $O = o_1 \ldots o_T$ and model H, what sequence of states 'best explains' the set of observations. There is no single correct answer to this problem – there are potentially many state sequences capable of producing a given set of observations.

   Often, we'll ask this question with some notion of an 'optimal' answer – that criteria will provide some basis for determining the set of states.
3. How do we adjust model parameters to maximize the probability of seeing a specific output sequence: $P(O|H)$.

   This problem is analogous to training the model, or creating the best model of a real set of observations.

2.5.3 HMM Designs

So far, we have looked at fully connected hidden markov models. These are called *ergodic*. Ergodic models allow every state to be reached from every other state. In the set of matrix of transition probabilities, every a_{ij} element will be positive.

A *left-right*, also known as a *bakis* model allows you to move from s_i to s_j only if $j \geq i$. You can only move forwards, or stay in the same place. Backwards movement is not possible. We have $a_{ij} = 0$ if $j < i$

Left right models typically have a unique starting state.

Left-right models can impose additional constraints. For example, transitions from s_i to s_j may be limited to $i \leq j \leq j + \Delta$. Delta represents a limit on forward motion in a single step.

These are just two examples. Many other variations are possible.

2.5.4 Implementation issues for HMMs

For large models (many states), underflow can be a problem – we keep multiplying small probabilities together. A common way of handling this situation is to scale the probabilities, and carry along a separate exponent.

Choosing the initial model parameters can also be a difficult task. In some cases, one will assign a set of uniform probability distributions. We then solve problems 1 and 3 iteratively in order to train the model until it becomes acceptably accurate.

2.6 Lecture Notes - 10/2/2006

2.6.1 HMM Example

Suppose we have two die, one fair and one loaded. Let D_1 be the fair die and D_2 be the loaded one. Let the output probabilities of these two die be

$$D_1 = \left\{ \begin{array}{c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{array} \right\}$$

$$D_2 = \left\{ \begin{array}{c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 0.05 & 0.05 & 0.05 & 0.05 & 0.05 & 0.75 \end{array} \right\}$$

We watch someone roll the dice. This person switches back and fourth between D_1 and D_2 – we can't tell which is being used, we only see the numbers that come up. Any number could have been produced by a fair or a loaded die; we don't know which. We assume that the die is chosen at random with some given probability.

In principle, we have two sources of output, D_1 and D_2. An HMM for this process will have two states. Each state has a different set of emission probabilities, and there is some set of probabilities associated with going back and fourth between the two die:

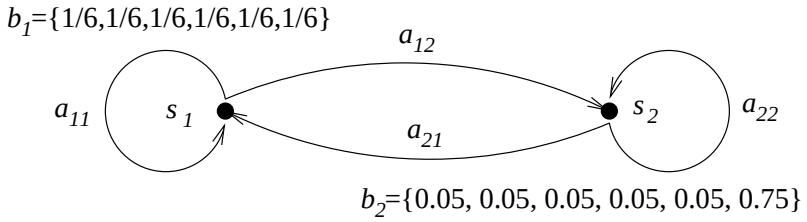


Figure 2.3: Hidden Markov Model for Two Die System

Recall that this model must have

$$a_{11} + a_{12} = 1$$
$$a_{21} + a_{22} = 1$$

The *emission probability*, $b_i(o_i)$ is the probability that the system produces the output o_i while in the state s_i.

We can directly observe the outputs of the model. The state transitions are hidden from us. In general, there will be many state paths that can produce a given sequence of output symbols.

2.6.2 Forward Probability

Forward probability calculations can provide an exact answer to problem 1 (page 31): what is the probability that a HMM generated a given output sequence.

We denote the forward probability as

$$\alpha_t(i) = P(o_1 \ldots o_t,\ q_t = s_i \mid H)$$

$\alpha_t(i)$ represents the probability of seeing the observation sequence $o_1 \ldots o_t$ (until time t, not necessarily the whole sequence), and being in state s_i at time t, using the model H.

$\alpha_t(i)$ can be solved inductively.

Before we get into the details of $\alpha_t(i)$ let's look at the brute force approach for solving the problem of determining the probability with which a given sequence was generated.

The probability of a state path is

$$P(S|H) = \pi_{i_1} \cdot a_{i_1 i_2} \cdot a_{i_2 i_3} \ldots a_{i_{t-1} i_t}$$

The probability of a series of observations, given a particular state path is

$$P(O|H) = \sum_{s=s_1 \ldots s_t} P(S|H) \cdot P(O|S,H)$$
$$P(O|S,H) = b_{i_1}(o_1) \cdot b_{i_2}(o_2) \cdot \ldots b_{i_t}(o_t)$$

(note – in the first formula above, $P(S|H)$ is not a conditional probability)

Putting these together, we have

$$P(O|S,H) = (\pi_{i_1} b_{i_1}(o_1)) \cdot (a_{i_1 i_2} b_{i_2}(o_2)) \cdot a_{i_2 i_3} \cdot \cdots \cdot b_{i_t}(o_{t-1}) \cdot (a_{i_{t-1} i_t} b_{i_t}(o_t))$$

To do the computation with this approach, we'd consider every possible state transition and every possible output symbol in each state. This is *very* expensive. There are n^T different sequences of states, each of which requires $2t$ computations – a total of $2t \cdot n^t$.

Given $n = 10$ and $t = 100$, there would be $20 \cdot 10^{100}$ computations. This is not feasable.

The probability of obtaining the prefix $o_1 \ldots o_k$ and winding up in state s_i is:

$$\alpha_k(i) = P(\,(o_1 \ldots o_k) q_k = s_i | H)$$

q_k denotes the state at moment k.

$$\sum_{i=0}^{n} \alpha_t(i) = P(o_1 \ldots o_t | H)$$

The inductive computation for $\alpha_k(i)$ is

- initialization:

$$\alpha_1(i) = \pi_i \cdot b_i(o_1) \quad 1 \le i \le n$$

- Inductive step

$$\begin{aligned}
\alpha_{k+1}(j) &= P((o_1, o_2, \ldots, o_k, o_{k+1}), q_{k+1} = s_j | H) \\
&= \left[\sum_{s_i \in S} P((o_1, \ldots o_k), q_k = s_i) \cdot a_{ij} \right] \cdot b_j(o_{k+1}) \\
&= \left[\sum_{i=1}^{n} \alpha_k(i) \cdot a_{ij} \right] \cdot b_j(o_{k+1})
\end{aligned}$$

$$\alpha_{k+1}(j) = \left[\sum_{s_i \in S} \alpha_k(i) a_{ij} \right] \cdot b_j(o_{k+1})$$

In summary,

$$\alpha_k(i) = P((o_1 \ldots o_k, q_k = s_i | H)$$

2.6.3 Backward Probability

Analogous to forward probability, there is backward probability, denoted $\beta_k(i)$.

$$\beta_k(i) = P((o_{k+1} \dots o_t)|q_k = s_i, H)$$

Where $\alpha_k(i)$ worked on prefixes of sequences, $\beta_k(i)$ works on suffixes of sequences. $\beta_k(i)$ is computed from $o_t \dots o_1$.

$$\begin{aligned} \beta_t(i) &= 1 && \text{for every } i \\ \beta_{k+1}(j) &= P((o_{k+2} \dots o_t|q_{k+1} = s_j) \\ \beta_k(i) &= \sum_j \beta_{k+1}(j)\, a_{ij}\, b_j(o_{k+1}) \end{aligned}$$

Note: the Rabiner paper gives this recurrence for β

$$\begin{aligned} \beta_n(i) &= 1 \\ \beta_t(i) &= \sum_{j=1}^{n} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \end{aligned}$$

The essence of the inductive part of the equation is as follows: in order to have been in state s_i at time t and to account for observations $o_{t+1} \dots o_n$, we have to consider all possible states s_j at time $t+1$, accounting for transitions from states s_i to s_j (the a_{ij} term) and then accounting for the remaining partial observation sequence from state s_j (the $\beta_{t+1}(j)$ term).

2.6.4 Finding Optimal State Sequences

To find the set of states that most likely produced a set of observations, we'll look at

$$P(S|O, H)$$

Read this as the probability that the state sequence was S, given the observation sequence O and the hidden markov model H.

Let

$$\gamma_k(i) = P(q_k = s_i \mid O, H)$$

$\gamma_k(i)$ denotes the probability that s_i was the k^{th} state used, given the observation sequence O and model H.

Our goal is to find the state s_i with the highest probability of producing the output state o_k.

$$\frac{P(q_k = s_i, O|H)}{P(O|H)}$$

Let us denote the observation sequence as

$$O = o_1, o_2, \dots, o_k, o_{k+1}, \dots, o_t$$

The probability that s_i generates o_k is represented by $\alpha_k(i)$. The probability that $o_{k+1} \dots o_t$ is the suffix is given by $\beta_k(i)$.

Think of this as cutting the observation into two halves. $\alpha_k(i)$ gives the probability of the prefix, and $\beta_k(i)$ gives the probability of the suffix.

$\alpha_k(i)$ and $\beta_k(i)$ can be used to derive $\gamma_k(i)$:

$$\begin{aligned}\gamma_k(i) &= \frac{\alpha_k(i) \cdot \beta_k(i)}{P(O|H)} \\ &= \frac{\alpha_k(i) \cdot \beta_k(i)}{\sum_{i=1}^{n} \alpha_k(i) \cdot \beta_k(i)}\end{aligned} \tag{2.6.1}$$

$\gamma_k(i)$ is a probability measure, so that

$$\sum_{i=1}^{n} \gamma_t(i) = 1$$

If all $a_{ij} > 0$, then we can use (2.6.1). However, if some $a_{ij} = 0$, we have a problem – this algorithm could deduce a set of states that are not possible to traverse. (It always looks for locally optimal emissions values – it could pick a succession of states that is not traversible because some $a_{ij} = 0$ along the path).

$$\delta_k(i) = \max_{q_1 \ldots q_{k-1}} P(q_1 \ldots q_{k-1}, q_k = s_i, o_1 \ldots o_k|H)$$

$\delta_k(i)$ contemplates different paths that allow us to reach s_i while generating the output sequence $o_1 \ldots o_k$.

$\delta_t(i)$ is the largest probability of the sequences that allow us to reach q_i.

$\delta_t(i)$ gives us the probability of a state sequence, but not the state sequence itself. To recover the state sequence, we'll use the Viterbi algorithm.

2.7 Lecture - 10/4/2006

2.7.1 Review: Forward/Backward Probabilities

The forward probability $\alpha_k(i)$ is

$$\alpha_k(i) = P(O, q_k = s_j|H)$$

Our convention for notation:

- i is an index that refers to a state
- k is an index that refers to a time interval

Backward probability is

$$\beta_k(i) = P((o_{k+1} \dots o_t)|q_k = s_i, H)$$

If $k = t$ the $\beta_k(i)$ represents the probability of the null suffix (always 1). k counts down from $t \dots 1$. t is the number of time intervals (and also the length of the sequence of observations).

The backward probability recurrence:

$$\begin{aligned}
\beta_t(i) &= 1 && \text{for every } i \\
\beta_k(i) &= P((o_{k+1} \dots o_t)|q_k = s_i) \\
&= P(o_{k+1}, (o_{k+2} \dots o_t)|q_k = s_i) \\
&= \sum_j \beta_{k+1}(j) a_{ij} b_k(o_k + 1)
\end{aligned}$$

In the last line above, note that k counts down from k to 1.

2.7.2 Viterbi Algorithm

Given a set of outputs, our goal is to recover the sequence of states that most likely produced the set of outputs.

$$\delta_k(i) = \max_{q_1 \dots q_{k-1}} P(q_1 \dots q_{k-1}, q_k = s_i, o_1 \dots o_k|H)$$

Again k represents time and i represents a state number.

δ is the probability of a conjunction of events.

δ eventually gives us the probability that a given sequence was produced.

Recurrence for Viterbi Algorithm

Initialization:

$$\begin{aligned}
\delta_1(i) &= P(q_1 = s_i, o_1|H) \\
&= \pi_i b_i(o_1) \qquad 1 \le i \le n
\end{aligned}$$

Recurrence:

$$\delta_{k+1}(j) = \max_{q_1 \dots q_k} P((q_1 \dots q_{k-1} q_k), q_{k+1} = s_j, (o_1 \dots o_k o_{k+1})|H)$$

$$\delta_{k+1}(j) = \left[\max_i \delta_k(i) \cdot a_{ij}\right] \cdot b_j(o_{k+1})$$

For a model with a moderate number of states (say 10 or 15), this recurrence will probably produce arithmetic underflow. But, we can adapt the computation using logarithms (assume $\log_2$).

$$\log \delta_{k+1}(j) = \max_i(\delta_k(j) + \log a_{ij}) + \log b_j(o_{k+1})$$

δ gives us a probability. To get the actual sequence, we need an auxillary data structure $I[j,k]$. j denotes a state, and k denotes a position. $I[j,k]$ tells us what state s_j most likely produced the symbol o_k.

$$I[j,k] = \text{argmax}_{1 \le i \le n}\, \delta_{k-1}(i) \cdot a_{ij}$$

argmax means "argument that maximizes" the expression.

The termination of the sequence is

$$p^* = \max_{1 \le i \le n} \delta_t(i)$$

Using the I array, we can trace back through the state paths.

Viterbi Example

Consider a simple HMM – we have two coins. One is fair, the other is crooked. Our HMM will have two states, and a two-symbol observation alphabet.

$O = \{H, T\}$	heads or tails
$S = \{G, K\}$	good or crooked

Of course we also need distributions:

$$\begin{aligned}
\pi = \pi_G = \pi_K &= 1/2 \\
b_G(H) = b_G(T) &= 1/2 \\
b_K(H) &= 1/4 \\
b_K(T) &= 3/4 \\
A &= \begin{pmatrix} 0.5 & 0.5 \\ 0.75 & 0.25 \end{pmatrix} \\
\log A &= \begin{pmatrix} -1 & -1 \\ \log 3 - 2 & -2 \end{pmatrix}
\end{aligned}$$

Under this model, what series of states are most likely to have produced the sequence $HHHTHT$.

When working this out in class, we drew a graph like

$H - H - H - T - H - T$

$G \quad G \quad G \quad G \quad G \quad G$

$K \quad K \quad K \quad K \quad K \quad K$

The top row are observations, the second row is the G state and the third row is the K state. We drew lines to represent the most probable state transitions, and filled in $\delta_k(i)$ below the state.

Working it out here, I'm just going to use tables. Although we'd want to use logs on a computer, I'm going to work it out with rational numbers.

k	o_k	$\max \delta_{k-1}(i)a_{ij}$	$\delta_k(G)$	$\max \delta_{k-1}(i)a_{ij}$	$\delta_k(K)$
1	H	1/2	1/4	1/2	1/8
2	H	1/8	1/16	1/8	1/32
3	H	1/32	1/64	1/32	1/128
4	T	1/128	1/256	1/128	3/512
5	H	9/2048	9/4096	1/512	1/2048
6	T	9/8192	9/16384	9/8192	27/32768

The $I[j,k]$ matrix:

o_k	G	K
H	–	–
H	G	G
H	G	G
T	G	G
H	K	G
T	G	G

2.7.3 Improving the Quality of Markov Models

There is an iterative process for improving the quality of a Markov model (i.e. - training).

$$\xi_k(i,j) = P(q_k = s_i, q_{k+1} = s_j | O, H)$$

$\xi_k(i,j)$ is similar to a_{ij}, but it is conditioned by output.

$$\begin{aligned}
\xi_k(i,j) &= P(q_k = s_i, s_{k+1} = s_j, (o_1 \dots o_k o_{k+1} \dots o_t)|H) \\
&= \frac{\alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j)}{P(O|H)} \\
&= \frac{\alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j)}
\end{aligned}$$

The training process can be conducted using the Baum-Welsch Algorithm.

2.8 Lecture – 10/11/2006

Hidden Markov Models are described by a 5-tuple: $H = (S, O, A, b, \pi)$. Let N be the number of states and let M be the number of alphabet symbols.

This entire system can be described by a series of numbers p_{ijl}

$$p_{ijl} = P(q_k = s_i, q_{k+1} = s_j, \text{ output at time } k+1 = o_l)$$

Recall that a_{ij} is

$$\begin{aligned} a_{ij} &= P(q_{t+1} = s_i | q_t = s_j) \\ &= \frac{P(q_{t+1} = s_i,\, q_t = s_j)}{P(q_t = s_i)} \end{aligned}$$

If we sum p_{ijl} on l, we get something similar (but not exactly the same as) a_{ij}

$$\begin{aligned} a \sum_l p_{ijl} &= P(q_k = s_i, q_{k+1} = s_j) \\ &= a_{ij} \cdot P(q_t = s_i) \\ &= a_{ij} \cdot P(q_1 = s_i) \qquad \text{(check this with someone)} \end{aligned}$$

The combination of p_{ijl} values determines the entire markov chain, assuming you have an initial distribution.

If we sum p_{ijl} on i we have

$$\begin{aligned} \sum_i p_{ijl} &= P(q_{k+1} = s_j, \text{ output at time } k+1 = o_l) \\ &= b_j(o_l) \end{aligned}$$

A chain is a combination of $N^2 * M$ parameters, where N is the number of states and M is the number of alphabet symbols.

Recall that

$$\xi_k(i,j) = P(q_k = s_i, q_{k+1} = s_j | O, H)$$

where O is a vector: $O = o_1 o_2 \dots o_t$.

Finally, recall that

$$\begin{aligned} \gamma_k(i) &= \sum_{j=1}^{N} \xi_k(i,j) \\ &= P(q_k = s_i | O, H) \end{aligned}$$

Let's examine ξ a little more.

$$\xi_k(i,j) = P(q_k = s_i, q_{k+1} = s_j | o_1 \dots o_k o_{k+1} \dots o_t, H)$$

In words,

$$\begin{aligned}
\xi_k(i,j) = & P(q_k = s_i, o_i \dots o_k | H) && (2.8.1)\\
& \wedge\ P(\text{we switch from } s_i \text{ to } s_j \text{ at moment } k) && (2.8.2)\\
& \wedge\ P(\text{we have output } o_{k+1} \text{ at moment } k+1 \text{ is } s_j) && (2.8.3)\\
& \wedge\ P(o_{k+2} \dots o_t \text{ assuming that at moment } k+1 \text{ we have output } o_{k+1}) && (2.8.4)
\end{aligned}$$

Line (2.8.1) is equivalent to $\alpha_k(i)$.

Line (2.8.2) is equivalent to q_{ij}.

Line (2.8.3) is equivalent to $b_k(o_{k+1})$.

Line (2.8.4) is equivalent to $\beta_{k+1}(s_j)$.

Therefore

$$\begin{aligned}
\xi_k(i,j) &= \frac{\alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j)}{P(O|H)} \\
&= \frac{\alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j)}{\sum_i^N \sum_j^N (\alpha_k(i) \cdot a_{ij} \cdot b_j(o_{k+1}) \cdot \beta_{k+1}(j))}
\end{aligned}$$

The initial distribution $\pi(i) = \gamma_1(i)$. We can estimate $\pi(i)$ by using probabilities from $\gamma_1(i)$.

2.8.1 The Baum-Welch Algorithm

Suppose we wanted to estimate a_{ij}:

$$\sum_{k=1}^{t-1} \xi_k(i,j)$$

$\sum_{k=1}^{t-1} \gamma_k(i)$ will be proportional to the number of times we pass from s_i to s_j.

Let us denote the estimate of a_{ij} by a_{ij}^* ($a_{ij} \approx a_{ij}^*$).

$$a_{ij}^* = \frac{\sum_{k=1}^{t-1} \xi_k(i,j)}{\sum_{k=1}^{t-1} \gamma_k(i)}$$

We can also estimate $b_j(o_l)$. Let's denote the estimate as $b_j^*(o_l)$.

$$b_j^*(o_l) = \frac{\sum_{k=1}^{t} \gamma_k(i) \text{ when output is } o_l}{\sum_{k=1}^{t} \gamma_k(i)}$$

The numerator is the number of $\gamma_k(i)$ at all times when the output is o_l.

This is the essence behind the Baum-Welch estimation algorithm. This algorithm is used for iteratively improving probability estimates. The general procedure is as follows:

1. We start with an arbitrary A and b_i. (The b_i are vectors, one for each state)
2. we compute a_{ij}^* and $b_j^*(o_l)$
3. we replace a_{ij} with a_{ij}^*, and replace $b_j(o_l)$ with $b_j^*(o_l)$
4. If more accuracy is desired, repeat the steps above.

This process makes local optimizations – we improve the model for a specific set of output. Each iteration will improve the model for a specific set of output.

$\sum_j p_{ijl} = 1$ for a fixed set of output

2.8.2 Comparing Sets of Discrete Random Variables

Suppose we have two sets of discrete random variables X and Y. Let a be an element of X and let b be an element of Y.

Let's also suppose that we have two models for these variables, P and P'.

Claim 2.8.2.1: If

$$\sum_a P(X = a|Y = b) \cdot \log \frac{P'(X = a, Y = b)}{P(X = a, Y = b)} > 0 \text{ for every } a$$

then

$$P'(Y = b) > P(Y = b) \text{ for every } b$$

This is called the *EM Theorem* or the *Expectation Maximization Theorem.*

Let's say we fix the b value – $P(X = a|Y = b)$ for a fixed b

$$\sum_a P(X = a|Y = \text{ a fixed } b) = 1$$

In Section 1.3.4 (page 12) we showed that

$$\sum (p_i \cdot \log \frac{q_i}{p_i}) > 0$$

for probability distributions p_i and q_i. Here, we will show that

$$\log P'(Y = b) - \log P(Y = b) > 0$$

$$\sum_a P(X = a|Y = b) \cdot \log \frac{P'(X = a, Y = b)}{P(X = a, Y = b)} \tag{2.8.5}$$

$$= \sum_a P(X = a|Y = b) \cdot \log P'(Y = b) - \sum_a P(X = a|Y = b) \cdot \log P(Y = b) \tag{2.8.6}$$

$$= \sum_a P(X = a|Y = b) \cdot \log \frac{P'(X = a, Y = b)}{P'(X = a|Y = b)} - \sum_a P(X = a|Y = b) \cdot \log \frac{P(X = a, Y = b)}{P(X = a|Y = b)} \tag{2.8.7}$$

$$= \sum_a P(X = a|Y = b) \cdot \log \frac{P'(X = a, Y = b)}{P'(X = a|Y = b)} + \sum_a P(X = a|Y = b) \cdot \log \frac{P(X = a|Y = b)}{P(X = a, Y = b)} \tag{2.8.8}$$

$$= \sum_a P(X = a|Y = b) \cdot \log \left(\frac{P'(X = a, Y = b)}{P(X = a, Y = b)} \cdot \frac{P(X = a|Y = b)}{P'(X = a|Y = b)} \right) \tag{2.8.9}$$

$$= \sum_a P(X = a|Y = b) \cdot \log \frac{P'(X = a, Y = b)}{P(X = a, Y = b)} + \sum_a P(X = a|Y = b) \cdot \log \frac{P(X = a|Y = b)}{P'(X = a|Y = b)} \tag{2.8.10}$$

In Equation (2.8.10) note that each of the expressions being added will be > 0

$\therefore P'(Y = b) > P(Y = b)$

Given the derivations above, we can say that

$$\sum_a P(X = a|Y = b) \cdot \log P'(X = a, Y = b) > \sum_a P(X = a|Y = b) \cdot \log P(X = a, Y = b) \quad (2.8.11)$$

This is sufficient to ensure that P' is a better model than P.

If we can arrange for inequality (2.8.11) then P' will be a better model.

If we can maximize

$$\sum P(X|O) \cdot \log P'(X|O) \quad (2.8.12)$$

Where the summation in (2.8.12) occurs over all values in X, then chances are that P' is better than P in explaining O.

P is defined by p_{ijl}.

P' is defined by p'_{ijl}

2.8.3 Gradients

Let f be a function $f\colon \mathbb{R}^n \to \mathbb{R}$

The set of partial deriviatives

$$\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n}$$

exists as a point $\overline{x} \in \mathbb{R}^n$.

This vector of partial derivatives

$$\left(\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n} \right)$$

is called the *gradient* of f. It is typically written

$$\nabla f = \left(\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n} \right)$$

Example 2.8.3.1: Let

$f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$

In this case, f denotes a sphere.

$$\begin{aligned} \nabla f &= (2x_1, 2x_2, 2x_3) \\ &= 2(x_1, x_2, x_3) \end{aligned}$$

2.9 Lecture – 10/16/2006

First, some background material leading up to the Baum-Welch algorithm.

2.9.1 Extremes with Constraints

Imagine a surface in $\mathbb{R}^3$, for example $x^2 + y^2 + z^2 = 0$. This example represents a sphere of radius 1. How would we maximize a function like

$$f(x, y, z) = x + 2y + 3z$$

such that $(x, y, z) \in$ the sphere?

A simpler example in two dimensions: given a circle $x^2 + y^2 = 1$, we like to find the extremes for $v = x + 2y$.

If $y = 0$ then $x = v$. If $x = 0$, then $y = v/2$. These equations give us a series of parallel lines

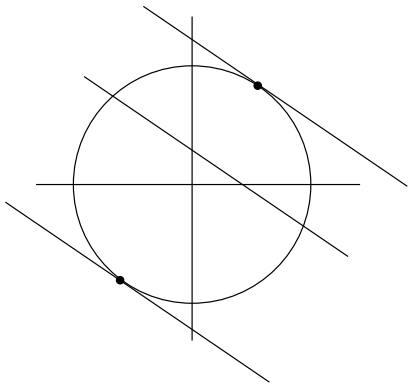


Figure 2.4: $x^2 + y^2 = 1$. Dots show extreme points (roughly)

Complicated functions can have many maximums and minimums. The Baum-Welch algorithm seeks *locally* extreme points (local maximum and local minimum).

General Surfaces

In $\mathbb{R}^n$ we'll have functions of the form

$$f(x_1 \dots x_n) = 0$$

This function represents a surface $\mathcal{S}$. Let's take a point $p(x_0 \dots x_n)$ such that $p \in \mathcal{S}$.

$$\begin{aligned} x_1 &= x_1(s) \\ x_2 &= x_2(s) \\ &\dots \\ x_n &= x_n(s) \end{aligned}$$

When s varies, one of these points describes a curve $\mathcal{C}$.

The tangent vectors to this curve have components

$$t = \left(\frac{\partial x_1}{\partial s}, \cdots, \frac{\partial x_n}{\partial s} \right)$$

Given a piece of $\mathcal{S}$ take a curve $\mathcal{C}$ such that $\mathcal{C}$ is located on the surface of $\mathcal{S}$, and there is a point p_0 such that $p_0 \in \mathcal{C}$.

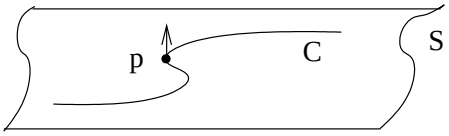


Figure 2.5: Tangent line to a point on a curve in a surface

If the equation of $\mathcal{S}$ is

$$g(x_1 \dots x_n) = 0$$

then

$$g(x_1(s) \dots x_n(s)) = 0$$

The first derivative will be

$$g'(s) = \left(\frac{\partial g}{\partial x_1} \cdot \frac{\partial x_1}{\partial s} + \frac{\partial g}{\partial x_1} \cdot \frac{\partial x_1}{\partial s} + \cdots \right) = 0$$

for every curve $\mathcal{C}$ on the surface of $\mathcal{S}$.

We also have

$$\left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \cdots, \frac{\partial g}{\partial x_n}, \right) \cdot \left(\frac{\partial x_1}{\partial s}, \frac{\partial x_2}{\partial s}, \cdots, \frac{\partial x_n}{\partial s}, \right)^T = 0 \tag{2.9.1}$$

In equation (2.9.1) the first factor represents the gradient; the second factor represents the tangent to the curve.

The gradient is perpendicular to the tangent at the point p_0.

The combination of all tangents to the curve that pass through p_0 form the *tangent plane*.

Thus, the gradient is perpendicular to the tangent plane.

Parital derivatives are usually denoted with the symbol ∇ (nabla).

Let $y = f(x_1 \dots x_n)$. Let us pick a curve in space

$$y(s) = f(x_1(s) \dots x_n(s))$$

To find maximums and minimums, we need to find points where $y'(s) = 0$ (first derivative). To tell whether such points are maximums or minimums we need the concavity test of the second derivative, $y''(s) = 0$.

Let us denote the tangent vector by $\bar{t}$.

$$y'(s) = (\nabla f)_{p_0} \cdot \bar{t}_{p_o}^T$$

To achieve an extrema (maximum or minimum) we need

$$(\nabla f)_{p_0} \perp \bar{t}_{p_0}^T$$

The gradient of a constraint must be perpendicular to the tangent.

$$\nabla f = \lambda \nabla g$$

where λ is a constant, g is a constraint and f is the function we are optimizing on the surface $g = 0$.

The combination of constraints and the original function give us a system of $n+1$ equations:

$$\begin{aligned}
\frac{\partial f}{\partial x_1} &= \lambda \cdot \frac{\partial g}{\partial x_1} \\
\frac{\partial f}{\partial x_2} &= \lambda \cdot \frac{\partial g}{\partial x_2} \\
&\dots \\
\frac{\partial f}{\partial x_n} &= \lambda \cdot \frac{\partial g}{\partial x_n} \\
g(x_1 \dots x_n) &= 0
\end{aligned}$$

Above, λ is called a *LaGrange multiplier*.

By solving this system, we will find the maximum and minimum.

Example 2.9.1.1: Let our equation be $f(x,y) = x + 2y$, and let our constraint be c: $x^2 + y^2 - 1 = 0$. We seek extremes of f, constrained by c.

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \lambda \cdot \frac{\partial g}{\partial x} = \lambda \cdot 2x \\
\frac{\partial f}{\partial y} &= \lambda \cdot \frac{\partial g}{\partial y} = \lambda \cdot 2y \\
x^2 + y^2 &= 1 = x^2 + y^2
\end{aligned}$$

$$\begin{aligned}
x &= \frac{1}{2\lambda} \\
y &= \frac{1}{\lambda}
\end{aligned}$$

$$\frac{1}{4\lambda^2} + \frac{1}{\lambda^2} = 1$$

$$\begin{aligned}
\lambda^2 &= \frac{5}{4} \\
\lambda &= \pm\frac{\sqrt{5}}{2}
\end{aligned}$$

$$\begin{aligned}
x_1 &= \frac{\sqrt{5}}{4} \\
x_2 &= -\frac{\sqrt{5}}{4} \\
y_1 &= \frac{\sqrt{5}}{2} \\
y_2 &= -\frac{\sqrt{5}}{2}
\end{aligned}$$

2.9.2 The Baum-Welch Algorithm

Assume that we have two hidden markov models

- H': the old model

- H: the new model (the improved model).

We want a process to take us from H' to H such that H is a better fit for an output (i.e. training) sequence $\omega = (\omega_1, \ldots, \omega_t)$.

Let $q = (q_1 \ldots q_t)$ represent a set of states.

Our goal is to have

$$P(\omega) > P'(\omega)$$

whereby the new model is better.

We can say

$$\log P(\omega) - \log P'(\omega) > 0$$

Observe that

$$\sum_q P'(q|\omega) = 1$$

This is a summation on the sequence of states q.

Below is a series of equations similar to the ones on page 41, but tailored to q and ω.

$$\sum_q P'(q|\omega) \cdot \log P(\omega) - \sum_q P'(q|\omega) \cdot \log P'(\omega) \tag{2.9.2}$$

$$= \sum_q P'(q|\omega) \cdot \log \frac{P(\omega, q)}{P(q|\omega)} - \sum_q P'(q|\omega) \cdot \log \frac{P'(\omega, q)}{P'(q|\omega)} \tag{2.9.3}$$

$$= \sum_q P'(q|\omega) \cdot \log \frac{P(\omega, q)}{P(q|\omega)} + \sum_q P'(q|\omega) \cdot \log \frac{P'(q|\omega)}{P'(q, \omega)} \tag{2.9.4}$$

$$= \sum_q P'(q|\omega) \cdot \log \frac{P(\omega, q)}{P(q|\omega)} \cdot \frac{P'(q|\omega)}{P'(q, \omega)} \tag{2.9.5}$$

$$= \sum_q P'(q|\omega) \cdot \log \frac{P'(q|\omega)}{P(q|\omega)} + \sum_q P'(q|\omega) \cdot \log \frac{P(\omega, q)}{P'(\omega, q)} \tag{2.9.6}$$

In equation (2.9.6) the first term being added will be > 0. If the second term is > 0, then

$$\log P(\omega) - \log P'(\omega) > 0$$

will hold.

Above, note that

$$\frac{P(\omega, q)}{P(q|\omega)} = \frac{P(\omega, q)}{\frac{P(q,\omega)}{P(\omega)} = P(\omega)}$$

We insist that

$$\sum_q P'(q|\omega) \cdot \log \frac{P(\omega, q)}{P'(\omega, q)} \geq 0 \tag{2.9.7}$$

Equation (2.9.7) is called the *Kullbach-Leiber Inequality*. It implies that

$$\log P(\omega) - \log P'(\omega) > 0$$

If we optimize the left-hand side of

$$\sum_q P'(q|\omega) \cdot \log P(\omega, q) \geq \sum_q P'(q|\omega) \cdot \log P'(\omega, q)$$

then the inequality is guaranteed to hold.

If we maximize

$$\sum_q P'(q|\omega) \cdot P(\omega, q) \tag{2.9.8}$$

this ensures that ω will be more probable under the new model H'.

Equation (2.9.8) is the Baum-Welch function.

Equivalently, we can maximize

$$\sum_q P'(q, \omega) \cdot \log P(q, \omega)$$

P' represents the probability under the new model H', and P represents the probability under the old model H.

Under the new model $P(q, \omega)$ is

$$P(q, \omega) = \pi_{q_0} \cdot \prod_{k=1}^{t} a_{q_{k-1} q_k} \cdot b_{q_k}(\omega_k) \tag{2.9.9}$$

In order to maximize equation (2.9.9) we need to take

$$\sum_q P'(q, \omega) \cdot \log \left[\pi_{q_o} \prod_{k=1}^{t} a_{q_{k-1} q_k} \cdot b_{q_k}(\omega_k) \right] \tag{2.9.10}$$

In equation (2.9.10) we can sum the log products to get

$$\sum_q P'(q, \omega) \log \pi_{q_o} + \sum_q P'(q, \omega) \sum_{k=1}^{t} \log a_{q_{k-1} q_k} + \sum_q P'(q, \omega) \sum_{k=1}^{t} \log b_{q_k}(\omega_k) \tag{2.9.11}$$

The three added terms in equation (2.9.11) call all be computed independently. Therefore, we seek to maximize each of these sums individually, including constraints.

Let $q \in \{s_1 \dots s_n\}$ denote states.
Let $\omega \in \{o_1 \dots o_m\}$ denote output symbols.

Optimizing π

To optimize π we use

$$\sum_q P'(q, \omega) \log \pi_{q_o}$$
$$= \sum_{i=1}^{n} P'(\omega, q_0 = s_i) \log \pi_i$$

Our constraint is

$$c\colon \sum_{i=1}^{n} \pi_i = 1$$

Think of this as

$$f - \lambda g = 0$$

where f is the function to optimize and g is a constant.

$$\begin{aligned}
\Phi &= \sum_{i=1}^{n} P'(\omega, q_0 = s_i) \log \pi_i - \lambda \left(\sum_{i=1}^{n} \pi_i - 1 \right) \\
0 &= \frac{\partial \Phi}{\partial \pi_i} \\
0 &= \frac{1}{\pi_1} \cdot P'(\omega, q_0 = s_i) - \lambda \\
\pi_1 &= \frac{P'(\omega, q_0 = s_i)}{\lambda} \\
\pi_1 &= \frac{P'(\omega, q_0 = s_i)}{\sum_{i=1}^{n} P'(\omega, q_0 = s_i)} \\
\pi_1 &= \frac{P'(\omega, q_0 = s_i)}{P'(\omega)}
\end{aligned}$$

2.10 Lecture – 10/18/2006

2.10.1 Baum-Welch Algorithm

Note: see also handout from
`http://www.cs.umb.edu/cs697/bw.pdf`

Recall that we started with a Hidden Markov Model that we wanted to improve. Let H' be the old HMM and H be the new HMM.

The idea was to maximize the function

$$Q(H, H') = \sum_m P'(\omega, q) \log P(\omega, q) \tag{2.10.1}$$

Equation (2.10.1) decomposes into three sums that vary independently.

$$P(\omega, q) = \pi_{q_0} \prod_{k=1}^{T} a_{q_{k-1} q_k} b_{q_k}(\omega_k)$$

In our last lecture, we found that the first of these sums allowed us to maximize π_i:

$$\pi_1 = \frac{P'(\omega, q_0 = s_i)}{P'(\omega)}$$

Estimating a_{ij}

The portion that maximizes a_{ij} is

$$\sum_q \sum_{k=1}^{T} P'(\omega, q) \log a_{q_{k-1} q_k}$$

We need to maximize

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} P'(\omega, q_{k-1} = s_i, q_k = s_j) \cdot \log a_{ij}$$

within the constraint

$$\sum_{j=1}^{N} a_{ij} = 1$$

The constraint is that each row must sum to one – there are really N constraints, one for each state.

Because we have N constraints, we also have N Lagrange Multipliers.

$$\Psi = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{T} P'(\omega, q_{k-1} = s_i, q_k = s_j) \cdot \log a_{ij} - \sum_{i=1}^{N} \lambda i \left(\sum_{j=1}^{N} a_{ij} - 1 \right)$$

We have

$$\frac{\partial \Psi}{\partial a_{ij}} = 0$$

$$\frac{\partial \Psi}{\partial a_{ij}} = \frac{1}{a_{ij}} \sum_{k=1}^{t} P'(\omega, q_{k-1} = s_i, q_k = s_j) - \lambda i = 0$$

$$a_{ij} = \frac{\sum_{k=1}^{T} P'(\omega, q_{k-1} = s_i, q_k = s_j)}{\lambda i}$$

$$= \frac{\sum_{j=1}^{N} \sum_{i=1}^{N} P'(\omega, q_{k-1} = s_i, q_k = s_j)}{\lambda i}$$

The final form is

$$a_{ij} = \frac{\sum_{k=1}^{T} P'(\omega, q_{k-1} = s_i, q_k = s_j)}{\sum_{k=1}^{T} P'(\omega, q_{k-1} = s_i)} \tag{2.10.2}$$

Estimating b

The portion that deals with b is

$$\sum_{q} \sum_{k=1}^{T} P'(\omega, q) \cdot \log b_{q_k}(\omega_k) \tag{2.10.3}$$

constrained by

$$\sum_{l=1}^{M} b_{il} = 1$$

So we have

$$\sum_{i=1}^{N} \sum_{k=1}^{T} P'(\omega, q_k = s_i) \cdot \log b_i(\omega_k)$$

Again, because each b row must sum to one, there are N constraints and we will have N Lagrange Multiplers – one for each state.

Let

$$\Theta = \sum_{i=1}^{N} \sum_{k=1}^{T} P'(\omega, q_k = s_i) \cdot \log b_i(\omega_k) = \sum_{i=1}^{N} \lambda i \left(\sum_{l=1}^{M} b_{il} - 1 \right) \tag{2.10.4}$$

The partial derivative of our constraint is

$$\frac{\partial \Theta}{\partial b_{il}} = 0$$

One complication here: $\log b_{il}(\omega_k)$ only matters when $\omega_k = o_l$ – the generated symbol has to match the one in the output sequence. To better handle this, we'll define a function

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x = y \end{cases}$$

NOTE: this δ has nothing to do with δ from the viterbi algorithm.

As before, our constraint is

$$\sum_{i=1}^{M} b_{il} = 1$$

So

$$\frac{\partial \Theta}{\partial b_{ij}} = \frac{1}{b_{il}} \cdot \sum_{k=1}^{T} P'(\omega, q_k = s_i) \cdot \delta(\omega_k, o_l) - \lambda i = 0$$

$$b_{il} = \frac{\sum_{k=1}^{T} P'(\omega, q_k = s_i) \cdot \delta(\omega_k, o_l)}{\lambda i}$$

$$\lambda i = \sum_{l=1}^{M} \sum_{k=1}^{T} P'(\omega, q_k = s_i) \delta(\omega_k, o_l)$$

$$\lambda i = \sum_{k=1}^{T} P'(\omega, q_k = s_i)$$

The final equation is

$$b_{il} = \frac{\sum_{k=1}^{T} P'(\omega, q_k = s_i) \cdot \delta(\omega_k, o_l)}{\sum_{k=1}^{T} P'(\omega, q_k = s_i)} \tag{2.10.5}$$

Note: all three of these formulas are expressed in terms of P'. These are the probabilities of the old model.

2.10.2 Pair Hidden Markov Models

A normal markov model generates one symbol per state transitions. Pair markov models generate two symbols per transition (eg - a pair).

One use of pair HMMs is to employ an affine (linear) gap penalty. Instead of using $-d$ as a penalty, we'll use $-d - e(g-1)$ where d, and d are constants, and g is the size of the gap.

Example: Consider three types of sequence alignments (each illustrates a different use of gaps).

I	G	A	x_i	|	A	I	G	A	x_i	|	G	A	x_i	–	–
L	G	V	y_j	|	G	V	y_j	–	–	|	S	L	G	V	y_j

Let's defined three functions:

- $M(i,j)$ – the best score assuming that x_i is aligned with y_j
- $I_x(i,j)$ – the best score assuming that x_i is aligned with a gap
- $I_y(i,j)$ – the score assuming that y_j is aligned with a gap.

These functions are defined in terms of each other

$$M(i,j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j) \\ I_x(i-1, j-1) + s(x_i, y_j) \\ I_y(i-1, j-1) + s(x_i, y_j) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1, j) - d \\ I_x(i-1, j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i, j-1) - d \\ I_y(i, j-1) - e \end{cases}$$

This combination can be easily translated into an automaton.

1. There will be one state for each of M, I_x and I_y.
2. Each state will be associated with a set of (i, j) increments (corresponding to where gaps are used).
3. edges will be labeled with scores and penalties.
4. We can consider an edge as consuming and x and a y, (s, s'); an x and a gap $(s, -)$; or a gap and a y, $(-, s')$.

A diagram of this automaton:

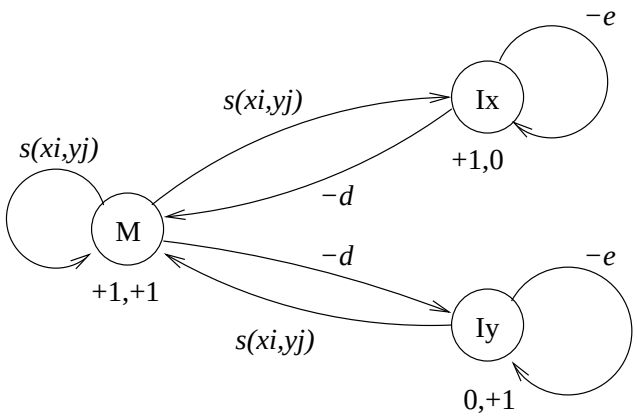


Figure 2.6: State machine for M, I_x, and I_y

To get a better idea of how this works, lets look at an example.

Example 2.10.2.1: Say we have the sequences

$$x = VLSPADK$$
$$y = HLAESK$$

and the alignment with gaps

V	L	S	P	A	D	–	K
H	L	–	–	A	E	S	K
M	$M \to M$	$M \to I_x$	$I_x \to I_x$	$I_x \to M$	$M \to M$	$M \to I_y$	$I_y \to M$

The bottom line isn't part of the alignment, it just shows the state transitions that will be used.

We can apply this alignment to our state machine as pairs of symbols

$$M(1,1) = s(V,H)$$
$$M(2,2) = M(1,1) + s(L,L)$$
$$I_x(3,2) = M(2,2) - d$$
$$I_x(4,2) = I_x(3,2) - e$$
$$M(5,3) = I_x(4,2) + s(A,A)$$
$$M(6,4) = M(5,3) + s(D,E)$$
$$I_y(6,5) = M(6,4) - d$$
$$I_y(7,6) = I_y(6,5) + s(K,K)$$

In this example, the system worked as a recognizer. However it is possible to re-write it as a generator, whereby we can use it to generate alignments.

2.11 Lecture – 10/23/2006 (part I)

2.11.1 Pairwise HMMs

Note: see chapter 4 in Durbin for more on Pairwise HMMs

During our last lecture, we saw how a finite state automaton (FSA) could be used to check alignments. The FSA we looked at had three states, M, I_x, and I_y corresponding to an alignment between two symbols, an alignment between x and a gap, and an alignment between y and a gap.

During our last lecture, we also stated that such an FSA was a *recognizer*, but that this FSA could be turned into a *generator*. To make the FSA a generator, we turn it into an HMM.

We'll start with an HMM that has three states: M, I_x, and I_y. Later, we'll look at variations.

The emission probabilities of this HMM will look a little odd. Rather than emitting single symbols, this HMM will emit pairs of symbols.

- M has emission probabilities $P_{x_i y_j}$
- I_x has emission probabilities $P_{x_i -}$
- I_y has emission probabilities $P_{- y_j}$

For now, let's say that the probability of going from M to I_x (or I_y) is δ, the probability of transitioning from I_x to I_x is ϵ, and the probability of transitioning from I_y to I_y is ϵ.

Our first cut at this HMM is shown in figure 2.7.

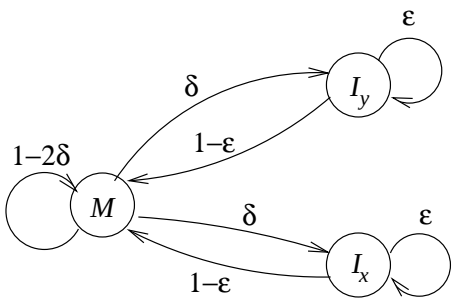


Figure 2.7: Pairwise HMM – first version

This HMM is not complete – we haven't specified any initial probabilities. Durbin shows an example where 'dummy' start and end states have been added, and the probability of moving to the end state is τ. Also, the probability of moving to I_x or I_y from the start state is δ. Figure 2.8 shows this version of the HMM. The start state has been labeled S and the end state has been labeled E.

In Figure 2.8, notice how the transition probabilites have been adjusted so that they will sum to one. If we make the decision that all alignments will start with a pair of symbols (i.e. - not with a gap in x or y), then we can simplify the model by removing the start state. Because the first pair in the alignment consists of a pair of symbols, the starting state will be M.

In Figure 2.9 shows the final version, with the start states removed.

The model shown in Figure 2.9 is really based on three parameters, δ, ϵ and τ. Having fewer parameters makes the estimation process easier.

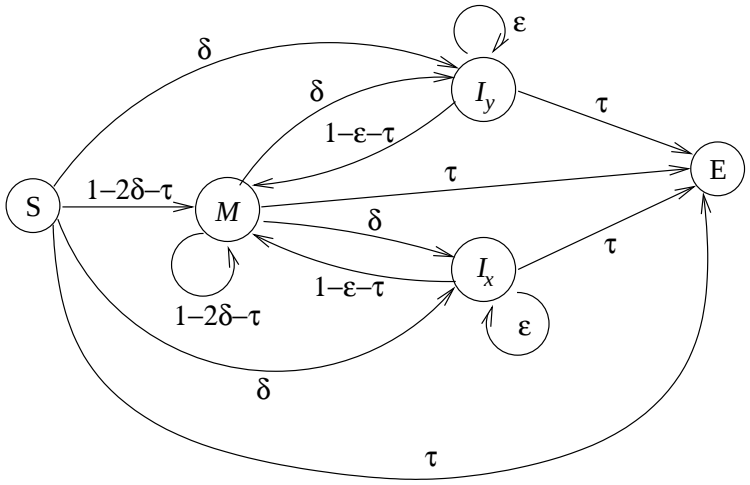


Figure 2.8: Pairwise HMM – Dummy start and end states

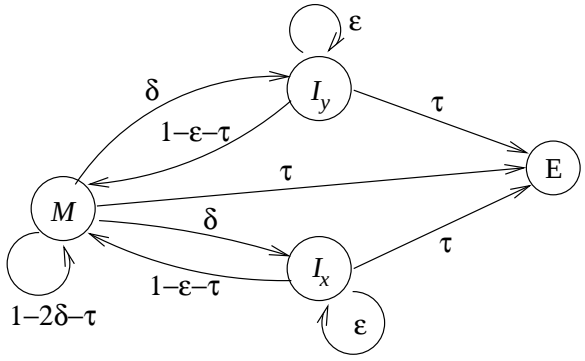


Figure 2.9: Pairwise HMM – Dummy start state removed

Now, let's give the viterbi algorithm in terms of this new model.

- $v^M(i,j)$ denotes a match between x_i and y_j
- $v^M(0,0) = 1$
- $v^?(0,j) = v^?(i,0) = 0$, where $v^?$ is either of v^{I_x} or v^{I_y}.

This much says that we always start from state M – never from I_x or I_y.

$$v^M(i,j) = P(x_i, y_j) \cdot \max \begin{cases} (1-2\delta-\tau) \cdot v^M(i-1,j-1) \\ (1-\epsilon-\tau) \cdot v^{I_x}(i-1,j-1) \\ (1-\epsilon-\tau) \cdot v^{I_y}(i-1,j-1) \end{cases} \tag{2.11.1}$$

$$v^{I_x}(i,j) = q_{x_i} \cdot \max \begin{cases} \delta \cdot v^M(i-1,j) \\ \epsilon \cdot v^{I_x}(i-1,j) \end{cases} \tag{2.11.2}$$

$$v^{I_y}(i,j) = q_{y_j} \cdot \max \begin{cases} \delta \cdot v^M(i,j-1) \\ \epsilon \cdot v^{I_y}(i,j-1) \end{cases} \tag{2.11.3}$$

The overall probability of the path is

$$v^E(x,y) = \max \begin{cases} v^M(n,m) \\ v^{I_x}(n,m) \\ v^{I_y}(n,m) \end{cases} \tag{2.11.4}$$

As with the "standard" viterbi algorithm, it is preferable to work with logarithms.

Another approach would be to consider all paths from M to the end state.

Let $f^k(i,j)$ denote the *sum* of the probabilities of paths that end in state k using the prefixes $x_1 \ldots x_i$ and $y_1 \ldots y_j$.

Because $f^k(i,j)$ is the sum of probabilties, $f^k(i,j)$ is <u>not</u> a probability itself. It is a *measure*.

We can find $f^k(i,j)$ by taking Equations (2.11.1) through (2.11.4) and replacing max with $\sum$.

Part 3

Phylogenetic Trees

3.1 Lecture – 10/23/2006 (part II)

3.1.1 Trees

Definition 3.1.1.1 (Tree): A Tree is a graph that is (1) connected and (2) acyclic. In a tree, the number of edges will be $|E| = |V| - 1$.

Suppose we have a graph $G = (V, E)$ and we wish to find the shortest path between two nodes x and y. Let us denote distance between x and y by $d(x, y)$.

We can observe three properties about d:

$$d(x, x) = 0 \tag{3.1.1}$$
$$d(x, y) = d(y, x) \tag{3.1.2}$$
$$d(x, y) \leq d(x, z) + d(z, y) \tag{3.1.3}$$

The combination of the three properties (3.1.1), (3.1.2), and (3.1.3) is referred to as a *metric*.

Suppose we are given a metric $\mathcal{S}$ and a function

$$d\colon \mathcal{S} \times \mathcal{S} \to \mathbb{N}$$

Is there a graph $G = (V, E)$ such that $\mathcal{S} \subseteq V$? The answer is yes.

Definition 3.1.1.2 (Weighted Graph): A weighted graph is a pair (G, W) where G is a graph, $G = (V, E)$ and W is a function $W\colon E \to \mathbb{R}$.

W is just a function that assigns weights to edges.

In a weighted graph, we denote distance as

$$d(x, y) = \min \sum w(e)$$

The distance is the smallest sum of weights among all paths that join x and y.

Definition 3.1.1.3 (Four-Point Condition): Also known as *Buneman's Condition*

$$d(x, y) + d(u, v) \leq \max \begin{cases} d(x, u) + d(y, v) \\ d(x, v) + d(y, u) \end{cases}$$

If $u = v$, then this degenerates into a three-point condition (the Triangulation Theorem).

In our next lecture, we'll talk more about trees and their use in finding affiliation between species.

3.2 Lecture – 10/25/2006

3.2.1 Phylogenetic Trees

Suppose that we are looking at protiens from different organisms, and that we have a way of comparing the protiens. Our method of comparison involves measuring the "distance" between the protien sequences. We denote the distance between protiens P_i and P_j as $d(P_i, P_j)$.

Knowing distances, phylogenetic trees allow us to rebuild the evolutionary process.

We start with a series of leaves $x_1 \dots x_n$, where each leaf is a protein. We'd like to construct a tree that shows their evolution from a common ancestor. We'll construct this tree by grouping data points with a clustering process.

Grouping Points in a Hierarchy

A grouping process is a recursive one. It it called *hierarchical clustering*. To get an intuitive sense of how the algorithm works, let's work through the process with five hypothetical points, $x_1 \dots x_5$.

- we begin with five points and a 5×5 matrix of d_{ij} distances. Effectively, we have five clusters with one point each.
- we find the pair with the smallest distance (say, (x_2, x_3)).
- our next iteration uses a 4×4 matrix and 4 clusters: x_1, (x_2, x_3), x_4, x_5. Suppose the pair with the smallest distance is (x_4, x_5)
- Our third iteration will use a 3×3 matrix and three clusters. This time, let's say with pair x_1 with (x_2, x_3). This gives clusters: $(x_1, (x_2, x_3))$ and (x_4, x_5).
- In the final iteration, we merge the two remaining clusters, giving: $((x_1, (x_2, x_3)), (x_4, x_5))$.

In this example, I've used parenthesis to indicate the tree structure. The tree representation is shown in figure 3.1.

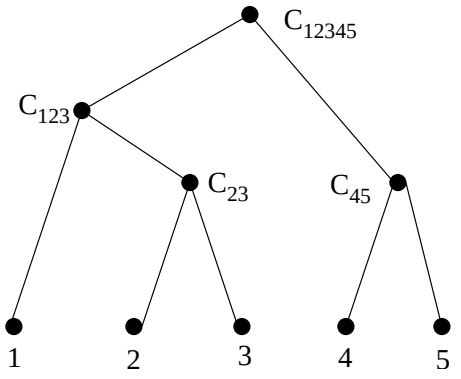


Figure 3.1: The tree $((x_1, (x_2, x_3)), (x_4, x_5))$

3.2.2 Computing Distances Between Clusters

There are several ways in which we could compute the distance between the clusters C and C':

1. $d(C, C') = \min d_{ij} | i \in C, j \in C'$
2. $d(C, C') = \max d_{ij} | i \in C, j \in C'$

3. $$d(C, C') = \frac{\sum_{i \in C, j \in C'} d_{ij}}{|C| \cdot |C'|}$$

(1) is called the single-linked method.
(2) is the called the complete-linked method.
(3) is the unweighted pair group arithmetic average method, or UPGMA.

UPGMA is the method most used by biologists, and the method we will use here.

3.2.3 UPGMA Clustering Algorithm

- **Input**
  - an $n \times n$ matrix of distances.
- **Initialization**
  - Assign each leaf node (sequence) to its own cluster C_i. For leaves x_i, we have clusters $C_i = \{x_i\}$, $1 \leq i \leq n$
  - Place each leaf at height zero.
- **Iteration**
  - Let C_i, C_j be the clusters such that d_{ij} is minimal
  - Eliminate C_i and C_j from the current clustering. Replace these with a new cluster $C_k = C_i \cup C_j$.
  - Place C_k at height $\frac{d_{ij}}{2}$.
  - Update (rebuild) the d_{ij} matrix
- **Termination**
  - When there are only two clusters left, combine those (again at height $\frac{d_{ij}}{2}$) and stop.

Updating the d_{ij} Matrix

With each iteration of the UPGMA algorithm, the dimensions of the d_{ij} matrix decrease by one: we remove rows (and columns) for C_i and C_j, replacing them with a row (column) for C_k. This will require the recomputation of each element in the matrix.

For the new cluster C_k we need to find d_{kl} for every other cluster l, where $l \neq i$ and $l \neq j$.

Between two clusters, the distance d_{ij} is given by

$$d_{ij} = \frac{\sum_{p \in C_i, q \in C_j} d_{pg}}{|C_i| \cdot |C_j|}$$

If $C_k = C_i \cup C_j$ is the union of two clusters, then

$$d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}$$

The second equation can be derived from the first as follows:

$$d_{ij} = \frac{\sum_{p \in C_i, q \in C_j} d_{pg}}{|C_i| \cdot |C_j|}$$

$$\sum_{p \in C_i, q \in C_j} d_{pq} = d_{ij} \cdot |C_i| \cdot |C_j|$$

$$\begin{aligned} d_{kl} &= \frac{\sum d_{il} + \sum d_{jl}}{|C_i + C_j| \cdot C_l} \\ &= \frac{|C_i| \cdot |C_l| \cdot d_{il} + |C_j| \cdot |C_l| \cdot d_{jl}}{|C_i + C_j| \cdot C_l} \\ &= \frac{|C_l|(|C_i| \cdot d_{il} + |C_j| \cdot d_{jl})}{|C_i + C_j| \cdot C_l} \\ &= \frac{|C_i| \cdot d_{il} + |C_j| \cdot d_{jl}}{|C_i + C_j|} \end{aligned}$$

Height of C_k

In each iteration, we place the new cluster C_k at height $\frac{d_{ij}}{2}$

The immediate ancestor of a pair of nodes is always at a greater height than the child nodes. Height is significant because it suggests the evolutionary time needed between two species.

We always join the pairs that are closest in height. In building the tree, we will have

$d_{ij} \leq \max(d_{ih}, d_{jh})$ for any h

Ultrametric Property of Distances

The distances d_{ij} are said to be ultrametric if, for any triplet of sequences x_i, x_j, x_k, the distances d_{ij}, d_{jk}, d_{ik} are either all equal, or two are equal and the remaining one is smaller.

This condition will hold for distances derived from a tree with a molecular clock (the UPGMA algorithm generates this kind of tree).

3.3 Lecture – 10/30/2006

Logistics

- Our first exam will be a take home test. The exam will be given on 11/8/2006. It will be due the following Monday.

3.3.1 Additive Trees

- A tree is a graph $T = (V, E)$ with two properties. (1) T must be connected and (2) T must be acyclic. T need not have a root.
- From the two properties given above, it follows that there is a *unique* path between any pair of nodes in T.
- We denote a rooted tree by (T, v_0). Here, we have simply chosen a vertex v_0 to act as the root of the tree.

Weighting Functions on Trees

A weighing function on a tree is a function $w\colon E \to \mathbb{R}$. w assigns weights to edges.

We assume that weights are non-negative.

Distance Defined by a Graph

The distance defined by a graph, $d(x, y)$ is the length of the shorted path that links x to y.

Note for a tree, $d(x, y)$ represents a unique path.

Distance Defined for Weighted (rooted) Trees

- Trees have unique paths between any two vertices x and y. This is called the *additive distance*.
- Trees with weights are sometimes called *additive trees*. Note that in this context "additive" refers to the weights, not to the trees.

Let x, y be two nodes in the tree and let $e_1 \dots e_n$ be the unique path from x to y. The weight of this path will be $\sum_{i=1}^{n} w(e_i)$.

We denote the *additive distance* as

$$d_w(x, y) = w(e_1) + \ldots + w(e_n)$$

If we have a set S and a distance $d\colon S \times S \to \mathbb{R}$ (a metric), is this distance induced by a tree? (Is it an additive distance?)

3.3.2 The 4-Point Condition (Tree Metrics)

Peter Buneman proved that a distance d on set S is induced by a tree IFF the 4 point condition is satisifed. The four point condition is given in equation (3.3.1).

$$d(x,y) + d(u,v) \leq \max \begin{cases} d(x,u) + d(y,v) \\ d(x,v) + d(y,u) \end{cases} \tag{3.3.1}$$

This condition is also called a *tree metric*

Let's say d is induced by T with weight w, and look at the 4-point condition with respect to the tree in Figure 3.2. Dotted lines in this figure represent paths (which may pass through an arbitrary number of vertices).

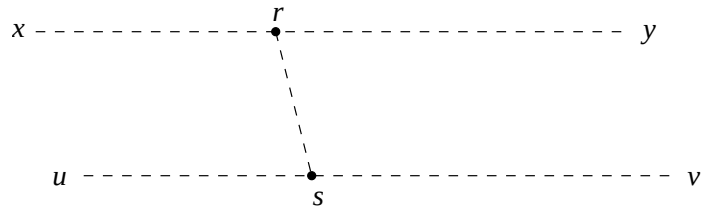


Figure 3.2: Tree for 4-point condition

Because this is a tree, there is a path between any pair of vertices. Therefore there is a path from x to u. In figure 3.2 we've shown this path as going through r and s. Note that r could be x and s could be u.

We can immediately see several distance inequalities:

$$\begin{aligned} d(x,v) &\leq d(x,r) + d(r,s) + d(s,v) \\ d(x,u) &\leq d(x,r) + d(r,s) + d(s,u) \\ d(y,u) &\leq d(y,r) + d(r,s) + d(s,u) \\ d(y,v) &\leq d(y,r) + d(r,s) + d(s,v) \end{aligned}$$

Because paths in a tree are unique, we could actually rewrite these as equalities:

$$d(x,v) = d(x,r) + d(r,s) + d(s,v) \tag{3.3.2}$$
$$d(x,u) = d(x,r) + d(r,s) + d(s,u) \tag{3.3.3}$$
$$d(y,u) = d(y,r) + d(r,s) + d(s,u) \tag{3.3.4}$$
$$d(y,v) = d(y,r) + d(r,s) + d(s,v) \tag{3.3.5}$$

For the four-point condition condition to be violated, the following pair of inequalities would need to hold:

$$d(x,u) + d(y,v) < d(x,y) + d(u,v) \tag{3.3.6}$$
$$d(x,v) + d(y,u) < d(x,y) + d(u,v) \tag{3.3.7}$$

From equations (3.3.2) – (3.3.5) we can derive

$$d(x,u) + d(v,y) = 2 \cdot d(r,s) + d(x,y) + d(u,v) \qquad \text{from (3.3.3), (3.3.5)} \tag{3.3.8}$$
$$d(x,v) + d(y,u) = 2 \cdot d(r,s) + d(x,y) + d(u,v) \qquad \text{from (3.3.2), (3.3.4)} \tag{3.3.9}$$

Even with $d(r,s) = 0$ we can see a contradiction between (3.3.8) and (3.3.6); also between (3.3.9) and (3.3.7).

For the 4-point condition, there are really two cases to consider. The first case is shown in figure 3.3. In case one, there is no overlap between the paths from x to y and u to v.

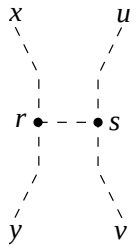


Figure 3.3: 4-point condition, case one

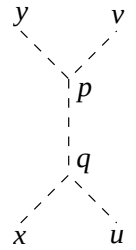


Figure 3.4: 4-point condition, case two

The second case is shown in figure 3.4. In case two there is overlap between the paths from x to y and u to v. In case two, we have:

$$\begin{aligned} d(x,y) &= d(x,q) + d(p,q) + d(p,y) \\ d(u,v) &= d(u,q) + d(p,q) + d(p,v) \end{aligned}$$

$$\begin{aligned} d(x,y) + d(u,v) &= d(x,q) + d(p,q) + d(p,y) + d(u,q) + d(p,q) + d(p,v) \\ &= 2 \cdot d(p,q) + d(x,q) + d(p,y) + d(u,q) + d(p,v) \end{aligned}$$

$$\begin{aligned} d(x,u) &= d(x,q) + d(q,u) \\ d(y,v) &= d(y,p) + d(p,v) \end{aligned}$$

3.3.3 Ultra-metrics

Suppose we have three numbers, a_1, a_2 and a_3 where the following inequality holds between any permutation of these numbers:

$$a_i \leq \max\{a_j, a_k\}$$

Two of $\{a_1, a_2, a_3\}$ must be equal and the third number must be no larger than the equal numbers.

Suppose a_j is the smallest.

$$\begin{aligned} a_i &\leq a_k \\ a_k &\leq \max\{a_i, a_j\} \end{aligned}$$

Because a_j is smallest,

$$\begin{aligned} a_k &\leq a_i \\ \therefore a_j &\leq a_i = a_k \end{aligned}$$

Let's apply this to $d(x,y) + d(u,v)$.

From the four-point condition, we can replace a_i, a_j and a_k with

$$d(x,y) + d(u,v)$$
$$d(x,u) + d(y,v)$$
$$d(x,v) + d(y,u)$$

and the inequalities will still hold.

Recall our definition of a metric. We have distance as $d\colon S \times S \to \mathbb{R} \geq 0$. A metric must meet three conditions:

$$d(x,y) = 0 \text{ if } x = y$$
$$d(x,y) = d(y,x)$$
$$d(x,y) \leq d(x,z) + d(z,y)$$

We can make a stronger assertion

$$d(x,y) \leq \max\{d(x,z), d(z,y)\} \tag{3.3.10}$$

Equation (3.3.10) is an ultra-metric.

Every triangle in ultra-metric space is an isosceles triangle.

Figure 3.5 shows the relationship between metrics, tree metrics and ultra-metrics. Every ultra-metric is

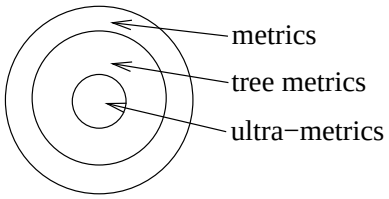


Figure 3.5: Relationship of metrics, tree metrics and ultra-metrics

a tree metric, and every tree metric is a metric.

Definition 3.3.3.1 (Equidistant Tree): A tree is equidistant if the distance from the root to every leaf is the same.

Ultra-metrics give us equidistant trees.

Theorem 3.3.3.2: Every ultra-metric is a tree metric.

To prove theorem 3.3.3.2 we must show that given

$$d(p,q) \leq \max\{d(p,r), d(r,q)\}$$

we have

$$d(x,y) + d(u,v) \leq \max \begin{cases} d(x,v) + d(y,u) \\ d(x,u) + d(y,v) \end{cases}$$

There are four cases to consider, as shown in table 3.1. Note that cases 3 and 4 are symmetric to cases 1 and 2. We only need to show cases 1 and 2 – the other two can be shown in the same way.

case	closest among u,v to x	closest among u,v to y
1	u	u
2	u	v
3	v	u
4	v	v

Table 3.1: 4 cases for proof of theorem 3.3.3.2

Case 1 We have $d(x,u) \leq d(x,v)$ and $d(y,u) \leq d(y,v)$. Therefore

$$d(x,u) \leq d(x,v) = d(u,v)$$
$$d(y,u) \leq d(y,v) = d(u,v)$$

Because $d(u,v) = d(y,v) = d(x,v)$ we can say

$$d(x,y) \leq \max\{d(y,u), d(x,u)\}$$

Case 2

$$d(x,y) \leq d(x,v) = d(u,v)$$
$$d(v,y) \leq d(x,u) = d(u,v)$$

$$d(x,y) + d(u,v) \leq \max\{d(u,v), d(x,u) + d(v,y)\}$$
$$d(x,y) \leq d(u,v)$$

$\therefore$ an ultra-metric is a tree metric.

Try picturing these cases with respect to Figure 3.6.

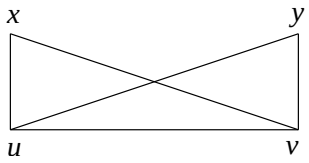


Figure 3.6: isosceles triangle diagram of tree

3.4 Lecture – 11/1/2006

3.4.1 Trees, Metrics, and Ultrametrics

Which kinds of trees produce ultra-metrics?

We have a weighting function defined on edges of trees:

$$w\colon E \to \mathbb{R} \geq 0$$

and the weighted distance

$$d_w(x, y) = \sum w(e)$$

such that e belongs to the simple path that joins x to y.

Suppose d were an ultra-metric (produced by a clustering algorithm). What kind of tree produces this kind of metric?

Definition 3.4.1.1 (Equidistant Tree): An equidistant tree is a tree where the distances between the root and any of the leaves are the same.

The property of equidistance is inherited from a tree to all of its sub-trees.

Given the tree shown in Figure 3.7, we can say that

$$d_w(u, l) = d(v_0, l) - d(v_0, u)$$
$$d_w(u, l_i) = \frac{d(l_1, l_2)}{2}$$

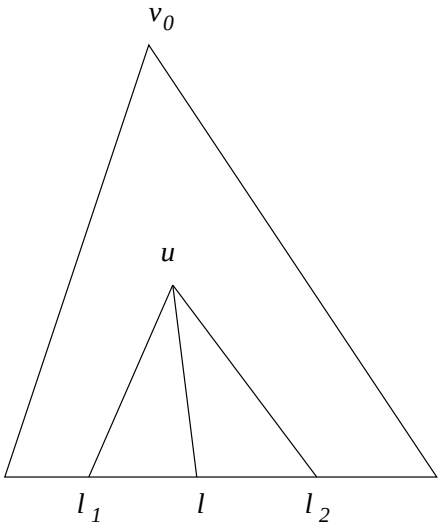


Figure 3.7: Equidistant Tree

Theorem 3.4.1.2: The metric generated by an equidistant tree is an ultrametric on the set of leaves of the tree.

Consider the tree shown in Figure 3.8. u is the closest ancestor of x and y (reminder – all leaves have a common ancestor, even if that ancestor is the root). If this is an equidistant tree, we can say

$$d_w(x, y) = 2 \cdot d_w(u, x) = 2 \cdot d_w(u, y)$$

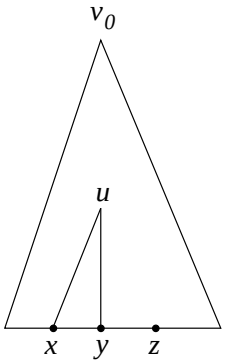


Figure 3.8:

Because all leaves have a common ancestor, y and z must also have a common ancestor. We propose that there are only two places that a common ancestor of y and z can exist: (1) on the path $u \cdots y$ or (2) on the path $v_0 \cdots u$.

Case 1 – On the path $u \cdots y$

For case 1 the tree is shown in figure 3.9, where v is the common ancestor. In Figure 3.9, we can say

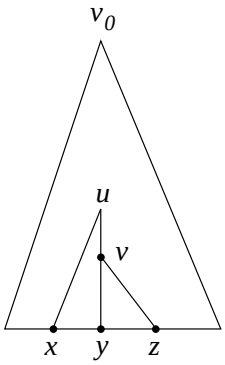


Figure 3.9: Common ancestor on the path $u \cdots y$

$$
\begin{aligned}
d_w(y,z) &= 2 \cdot d_w(v,y) = 2 \cdot d_w(v,z) \\
d_w(x,z) &= d_w(x,u) + d_w(u,v) + d(v,z) \\
d_w(x,u) &= d_w(u,v) + d_w(v,y) \\
&= d_w(u,v) + d_w(v,z) \\
d_w(x,z) &= 2 \cdot d_w(u,v) + d_w(v,y) + d_w(v,z)
\end{aligned}
$$

This tree is an ultra-metric.

$$d(y,z) \leq d(x,y) = d(x,z)$$

Case 2 – On the path $v_0 \cdots u$

For case 2 the tree is shown in figure 3.10. Again, v is the common ancestor of y and z. We have

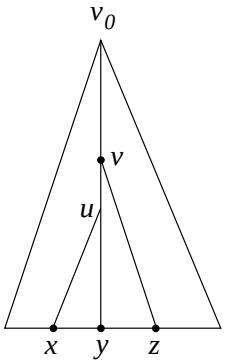


Figure 3.10: Common ancestor on the path $v_0 \cdots u$

$$d_w(x,y) = 2 \cdot d_w(u,x)$$
$$d_w(x,z) = 2 \cdot d_w(u,x) + 2 \cdot d_w(u,v)$$
$$d_w(y,z) = 2 \cdot d_w(u,v) + 2 \cdot d_w(y,y)$$

Again, this is an ultra-metric

$$d_w(x,y) \leq d_w(x,z) = d_w(y,z)$$

Theorem 3.4.1.3 (Semple & Steel): Given an ultra-metric d on the set S, there is an equidistant tree that has S as its leaves, and the distance induced by the tree on S equals d.

Proof. We can prove this claim by induction on the number of elements in the set S. Let $S = \{s_1 \ldots s_n\}$ for $n \geq 2$.

For $n = 2$, the tree appears as shown in figure 3.11. For the distances a and b, we have

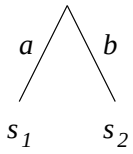


Figure 3.11: Case where $n = 2$

$$a = b = \frac{d_w(s_1, s_2)}{2}$$

Which trivially satisfies the ultra-metric property.

For $n = 3$, we have three points: s_1, s_2, s_3. We chose these points such that $d_w(s_1, s_2)$ is the minimal distance. If the distances $d_w(s_1, s_2)$, $d_w(s_1, s_3)$, $d_w(s_2, s_3)$ are all equal, we have an ultrametric because:

$$d_w(s_1, s_2) \leq d_w(s_1, s_3) = d_w(s_2, s_3)$$

If these distances are not all equal, the tree will look as shown in figure 3.12. In figure 3.12

$$c = \frac{d_w(s_1, s_2)}{2}$$
$$b = \frac{d_w(s_1, s_3)}{2}$$
$$a = \frac{d_w(s_1, s_3) - d_w(s_1, s_2)}{2}$$

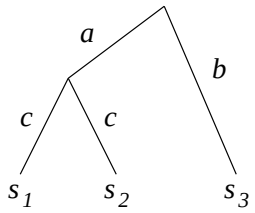


Figure 3.12: Case where $n = 3$

Because the tree is equidistant,

$$\begin{aligned} b - a &= \frac{d_w(s_1, s_2)}{2} \\ b + a + \frac{d_w(s_1, s_2)}{2} &= d_w(s_1, s_3) = d_w(s_2, s_3) \\ b + a &= \frac{-d_w(s_1, s_2)}{2} + d_w(s_1, s_3) \\ 2b &= d_w(s_1, s_3) \\ b &= \frac{d_w(s_1, s_3)}{2} \\ a &= \frac{d_w(s_1, s_3) - d_w(s_1, s_2)}{2} \end{aligned}$$

This is an ultra-metric.

$$d_w(s_1, s_2) \leq d_w(s_2, s_3) = d_w(s_1, s_3)$$

For $n \geq 4$, let us assume that the statement holds for sets with fewer than n elements.

We pick two elements of S; s_i and s_j, such that the distance $d_w(s_i, s_j)$ is the smallest distance between any pairs of nodes.

Let $S' = S - \{s_i, s_j\} \cup s$. Take s_i and s_j out and replace them with a new node s. We define a new weighting function for S',

$$d' \colon S' \times S' \to \mathbb{R} \geq 0$$

For two nodes s_k and s_l, we will have

$$d'(s_k, s_l) = d(s_k, s_l) \text{ if } s_k, s_l \in S' - \{s_i, s_j\}$$

In d',

$$d'(s_k, s) = d(s_k, s_i) = d(s_k, s_j)$$

Again, note how we have replaced s_i and s_j in S (and d) with $s \in S'$ (and d').

Also note that in the original set, we would have had

$$d(s_i, s_j) \leq d(s_k, s_i) = d(s_k, s_j)$$

For any leaf s_k, because $d(s_i, s_j)$ was the minimal distance in S.

To show that the distance d' is an ultra-metric on S', we need to look at $d'(s_k, s)$, $d'(s_l, s)$ and $d'(s_k, s_l)$. Note that

$$\begin{aligned} d'(s_k, s) &= d(s_k, s_i) \\ d'(s_l, s) &= d(s_l, s_i) \\ d'(s_k, s_l) &= d(s_k, s_l) \end{aligned}$$

This means that we can build an equidistant tree for the set S'.

$$d(s_i, s_j) \leq d'(s_k, s) = d(s_k, s_i)$$

We want to replace T' with T and add back s_i and s_j.

The general idea is shown in figure 3.13. However, Figure 3.13 is a little misleading – s_i and s_j would probably be the same distance from the root as s (I think ...). In Figure 3.13,

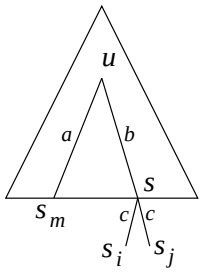


Figure 3.13: Putting s_i, s_j back to replace s

$$\begin{aligned} d(u, s_m) &= a \\ d(u, s_i) &= d(u, s_j) = b + c \\ d(s_i, s_j) &= 2 \cdot c \end{aligned}$$

$$\begin{aligned} c &= \frac{d(s_i, s_j)}{2} \\ b &= d(u, s_i) - c \\ &= d(u, s_i) - \frac{d(s_i, s_j)}{2} \\ a &= d(u, s_m) = \frac{d(s_m, s_i)}{2} \\ b &= \frac{d(s_m, s_i) - d(s_i, s_j)}{2} \leq 0 \end{aligned}$$

again, it's not drawn to scale

When going from T to T'

$$d'(v_0, u) = d(v_0, u) = d(v_0, s_m) - \frac{d(s_m, s_i)}{2}$$

s_i and s_j are the same distance from the root of T as s was from the root of T' (I think).

3.4.2 For Next Class

In the next class, we'll discuss algorithm's for building trees out of tree metrics. Read over the Saitou-Nei algorithm.

3.5 Dissimilarities and Metrics – 11/6/2006

Notes taken from Prof. Simovici's handout – Chapter 8: Dissimilarities, Metrics and Ultrametrics

3.5.1 Dissimilarities

Definition 3.5.1.1 (Dissimilarity): A *dissimilarity* on a set S is a function $d\colon S \times S \to \mathbb{R} \geq 0$ that satisifies two conditions:

1. $d(x, x) = 0$ for all $x \in S$
2. $d(x, y) = d(y, x)$ for all x, $y \in S$

Below are some properties that *may* be satisfied by dissimilarities:

Evenness $d(x, y) = 0$ implies that $d(x, z) = d(z, y)$ for all x, y, $z \in S$

Definiteness $d(x, y) = 0$ implies that $x = y$

Triangular Inequality $d(x, y) \leq d(x, z) + d(z, y)$

Ultrametric Inequality $d(x, y) \leq \max\{d(x, z), d(y, z)\}$

Bunneman's Inequality (also known as the 4-point condition)

$$d(x, y) + d(u, v) \leq \max \begin{cases} d(x, u) + d(y, v) \\ d(x, v) + d(y, u) \end{cases}$$

Together, the triangular inequality and definiteness imply evenness.

3.5.2 Types of Metrics

metric a metric is a dissimilarity that satisfies definiteness and the triangular inequality

Tree metric A tree metric is a dissimilarity that satisfies definiteness and the four-point condition.

Ultrametric An ultrametric is a dissimilarity that satisfies definiteness and the ultrametric inequality.

Three Inequalities Satisfied by Metrics

1. $d(x, y) = 0$ IFF $x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, y) \leq d(x, z) + d(z, y)$

If (1) is replaced by the weaker requirement that $d(x, x) = 0$, then we have *quasi-metric*. In a quasi-metric, having $d(x, y) = 0$ does not necessarily imply that $x = y$.

Relationships Between Metrics

Every tree metric is a metric.

Every ultrametric is a tree metric.

There is a link between ultrametrics and equivalence relations. Ultrametrics imply reflexivity, symmetry, and transitivity.

3.5.3 Graphs and Trees

Theorem 3.5.3.1: A graph $G = (V, E)$ is a tree IFF

1. G is connected
2. G contains no triangles
3. G's distance function satisifies Bunneman's inequality.

3.6 Lecture – 11/6/2006

3.6.1 Nei and Saitou's Algorithm

Nei and Saitou's algorithm is also known as the *NJ Algorithm.*

Input A tree metric

Output A tree

Procedure: Let n be the number of objects.

1. Compute
$$s_{ij} = (n-2) \cdot d_{ij} - r_i - r_j$$
where
$$r_i = \sum_{j=1}^{n} d_{ij}$$
2. Chose a pair of objects i, j such that s_{ij} is minimal.
3. Create a new node u (also denoted (ij) – we form u by merging i and j), and define
$$d_{iu} = \frac{d_{ij}}{2} + \frac{1}{2(n-2)}(r_i - r_j)$$
$$d_{ju} = \frac{d_{ij}}{2} + \frac{1}{2(n-2)}(r_j - r_i)$$
$$d_{ku} = \frac{d_{ik} + d_{jk} - d_{ij}}{2}$$
This step defines distances from u to all other objects.
4. Delete i, j, and replace them with u.
5. If more than two nodes remain, then go to step 1. Otherwise, stop.

One note about d_{iu} and d_{ju} – these tell us the distances from i to u and from j to u. However, we *don't* use them as part of the computation of d_{ku} (the distance from $u = (ij)$ to all other nodes k).

3.6.2 Discussion of the Algorithm

Because we choose i, j such that their distance is minimal, we have

$$s_{ik} - s_{ij} \geq 0 \tag{3.6.1}$$

for any other node k.

Let's take (3.6.1), expand and manipulate some of the terms.

$$s_{ik} - s_{ij} = ((n-2)d_{ik} - r_i - r_k) - ((n-2)d_{ij} - r_i - r_j) \quad \text{basic expansion} \quad (3.6.2)$$

$$= (n-2)(d_{ik} - d_{ij}) + r_j - r_k \quad \text{combine terms} \quad (3.6.3)$$

$$= (n-2)d_{ik} - (n-2)d_{ij} + r_j - r_k \quad (3.6.4)$$

$$= (n-2)d_{ik} - (n-2)d_{ij} + \sum_{l=1}^{n} d_{jl} - \sum_{l=1}^{n} d_{kl} \quad \text{definition of } r_i \quad (3.6.5)$$

$$= (n-3)d_{ik} - (n-3)d_{ij} + \sum_{l=1, l\neq i}^{n} d_{jl} - \sum_{l=1, l\neq i}^{n} d_{kl} \quad \text{see note 1} \quad (3.6.6)$$

$$= (n-3)d_{ik} - (n-3)d_{ij} + \sum_{l=1, l\neq i,j,k}^{n} (d_{jl} - d_{kl}) \quad \text{see note 2} \quad (3.6.7)$$

$$= \sum_{l\neq i,j,k} (d_{ik} - d_{ij} + d_{jl} - d_{kl}) \quad \text{see note 3} \quad (3.6.8)$$

1. Note 1: The restriction $l \neq i$ in the summations allow us to get rid of d_{ik} and d_{ij} terms – hence $n-3$
2. Note 2: The summation's restriction on l omits cases where d will be zero, or where the term will by canceled by another term in the equation.
3. Note 3: Notice that d_{ik} and d_{ij} do not depend on l.

If i, j are chosen such that s_{ij} is minimal, then i, j must be neighbors.

If i, j are neighbors, then s_{ij} must be smaller than any other value situated on s_{ij}'s row or column (think of a matrix of s_{ij} values). To reiterate:

$s_{ik} - s_{ij} \geq 0$ for any k

Consider the tree configuration in Figure 3.14.

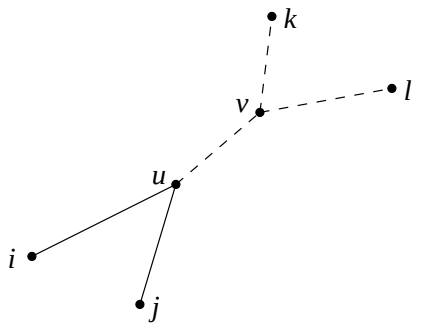


Figure 3.14: Tree with 5 nodes, i and j closest

Let's examine the terms

$d_{ik} - d_{ij} + d_{jl} - d_{kl}$

Expanding these four terms with respect to Figure 3.14, we have

$d_{ik} = d_{iu} + d_{uv} + d_{vk}$
$d_{ij} = d_{iu} + d_{uj}$
$d_{jl} = d_{ju} + d_{uv} + d_{vl}$
$d_{kl} = d_{kv} + d_{vl}$

If we substitute these four expansions into

$$d_{ik} - d_{ij} + d_{il} + d_{kl}$$

then we're left with

$$2 \cdot d_{uv} \geq 0$$

This tells us that the summation in (3.6.8) will be non-negative.

If s_{ij} is minimal, then i, j must be neighbors. This assertion is harder to prove.

A footnote to the discussion: we've been refering to tree nodes as "objects". In biological literature, these are commonly referred to as *OTU*s. OTU stands for *Operational Taxonomical Unit*.

3.6.3 Showing That i, j are neighbors

Let's do some more manipulation with $s_{kl} - s_{ij}$

$$\begin{aligned} s_{kl} - s_{ij} &= ((n-2)d_{kl} - r_k - r_l) - ((n-2)d_{ij} - r_i - r_j) \\ &= (n-2)d_{kl} - \sum d_{km} - \sum d_{lm} - (n-2)d_{ij} + \sum d_{im} + \sum d_{jm} \end{aligned}$$

In the next step, we restrict the summations by $m \neq i, j, k, l$. The distance terms excluded by the summation are put back separately. Note that equation (3.6.9) is a single multiline formula.

$$\begin{aligned} s_{kl} - s_{ij} =& (n-2)d_{kl} \\ & - \left(\sum_{m \neq i,j,k,l} d_{km} \right) - d_{ki} - d_{kj} - d_{kl} \\ & - \left(\sum_{m \neq i,j,k,l} d_{lm} \right) - d_{li} - d_{lj} - d_{lk} \\ & - (n-2)d_{ij} \\ & + \left(\sum_{m \neq i,j,k,l} d_{im} \right) + d_{ij} + d_{ik} + d_{il} \\ & + \left(\sum_{m \neq i,j,k,l} d_{jm} \right) + d_{ji} + d_{jk} + d_{jl} \end{aligned} \tag{3.6.9}$$

After simplification, this becomes:

$$\begin{aligned} s_{kl} - s_{ij} &= (n-4)d_{kl} - (n-4)d_{ij} + \sum_{m \neq i,j,k,l} (d_{im} + d_{jm} - d_{km} - d_{lm}) \\ &= \sum_{m \neq i,j,k,l} ((d_{im} + d_{jm} - d_{ij}) - (d_{km} + d_{lm} - d_{kl})) \end{aligned}$$

If s_{ij} is minimal, then i, j must be neighbors in this tree.

Assume this holds for $n \leq 4$. (Actually, this is being left as an excercise for the student. Try working out the different cases for $n \leq 4$).

Consider $n \geq 5$. For $n \geq 5$ we'll categorize points as category one, or category two, as shown by Figure 3.15.

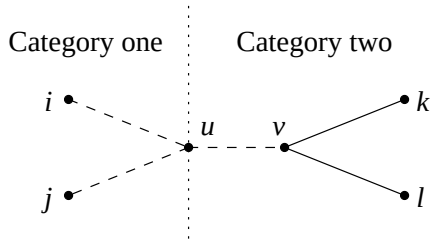


Figure 3.15: $n \geq 5$, showing Category one and two

In figure 3.15, we assume that i, j are nodes such that s_{ij} is minimal.

Because s_{ij} is minimal, we have $s_{kl} \geq s_{ij}$.

Suppose that m is a Category one point – we take m as being along the path $i \cdots u$ (the $j \cdots u$ case is similar).

$$\begin{aligned}&(d_{im} + d_{jm} - d_{ij}) - (d_{km} + d_{lm} - d_{kl}) \\ =& -2 \cdot d_{mu} - 2 \cdot d_{uv} < 0\end{aligned}$$

Now, suppose m is a Category Two point, along the path $u \cdots v$.

$$\begin{aligned}&(d_{im} + d_{jm} - d_{ij}) - (d_{km} + d_{lm} - d_{kl}) \\ =& 2 \cdot d_{um} - 2 \cdot d_{mv} \geq 0\end{aligned}$$

In order to get a positive contribution, we need more category two points than category one points.

(To be continued next lecture).

3.7 Lecture – 11/8/2006

Saitou-Nei Algorithm (cont'd)

Recall that we have started with

- A distance d_{ij}, which is a tree metric between i and j.
- A matrix s_{ij}, defined as

$$s_{ij} = (n-2)d_{ij} - r_i - r_j \tag{3.7.1}$$

$$r_i = \sum_{p=1}^{n} d_{ip} \tag{3.7.2}$$

If i and j are neighbors (the immediate descendants of a common ancestor), then s_{ij} is minimal. We proved this during our last lecture.

Today will we show the reverse: if s_{ij} is minimal, then i and j must be neighbors.

3.7.1 Showing s_{ij} minimal implies i, j neighbors

Suppose that s_{ij} were minimal but i and j were not neighbors. We can show that this leads to a contradiction.

If i and j are nodes of a tree, then we know that there is a unique path from i to j. If i and j are neighbors, then the path $i \cdots j$ has exactly one intermediate node. If i and j are not neighbors, then the path $i \cdots j$ must two or more intermediate nodes.

Let's consider a Category one node h where i and j are not neighbors.

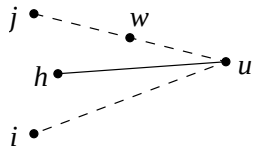


Figure 3.16: i, j not neighbors with category one node h

Recall any four objects must satisfy the following inequality:

$$s_{kl} - s_{ij} = \sum_{m \neq i,j,k,l} ((d_{im} + d_{jm} - d_{ij}) - (d_{km} + d_{lm} - d_{kl})) \tag{3.7.3}$$

Let's apply this inequality to Figure 3.16.

$$\begin{aligned}
&(d_{iw} + d_{jw} - d_{ij}) + (d_{iw} - d_{hw} - d_{ih}) \\
&= (d_{iu} + d_{uw} + d_{jw} - d_{ij}) - (d_{iu} + d_{uw} + d_{uh} + d_{uw} - d_{iu} - d_{uh}) \\
&= 0 - 2d_{uw} \\
&= -2d_{uw}
\end{aligned}$$

No matter where we choose w, the distance is negative.

3.7.2 Contributions of Category One and Category Two Points

Figure 3.17 shows a tree with a category one node m.

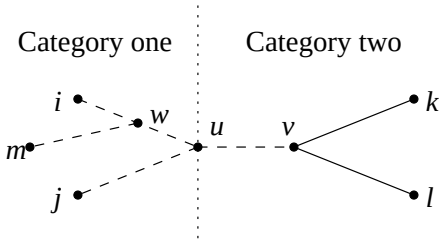


Figure 3.17: Category One node m

If m is a category one point, then $s_{kl} - s_{ij}$ breaks down to the summation of

$$\begin{aligned}
&(d_{im} + d_{jm} - d_{ij}) + (d_{km} + d_{lm} - d_{kl}) \\
=&[(d_{iu} + d_{uv} + d_{wm}) + (d_{jw} + d_{mw}) - d_{ij}] \\
&\quad - [(d_{mw} + d_{wu} + d_{uv} + d_{vh}) + (d_{mw} + d_{wu} + d_{uv} + d_{vl}) - d_{kl}] \\
=&2d_{wm} - (2d_{mw} + 2d_{uw} + 2d_{uv}) \\
=&-2d_{uw} - 2d_{uv}
\end{aligned}$$

$\therefore$ category one points give us a negative contribution.

Now consider Figure 3.18, where m is a category two point.

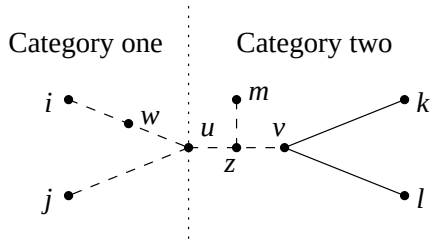


Figure 3.18: Category Two node m

We have

$$\begin{aligned}
&(d_{im} + d_{jm} - d_{ij}) - (d_{km} + d_{lm} + d_{kl}) \\
=&[(d_{iu} + d_{uz} + d_{zm}) + (d_{ju} + d_{uz} + d_{zm}) - d_{ij}] \\
&\quad - [(d_{kv} + d_{vz} + d_{zm}) + (d_{vl} + d_{vz} + d_{zm}) - dkl] \\
=&2d_{uz} - 2d_{vz}
\end{aligned}$$

Category two points give us a positive contribution, as long as $d_{uz} > d_{vz}$ (required to satisify $s_{kl} - s_{ij}$).

For category 2 points, we see that the positive contribution really depends on z.

In conclusion, having s_{ij} minimal means that the path $i \cdots j$ has at most one intermediate point, which means that i and j are neighbors.

The complexity of the Saitou-Nei algorithm is $\Theta(n^3)$.

3.8 The Parsimony Principle

Occam's Razor The most plausable explanation of a phenemenon is the simplest.

Parsimony Principle A technique for building phylogenetic trees that's based on the Occam's Razor approach.

Orthologous Genes These are families of genes that proceeded from a common ancestor, along the same timeline.

Walter Fitch The first person to describe parsimony and phylogenetic trees. To find his original paper, use *jstore* (one of the Healey Library online catalogs), and search by name.

Under the parsimony principle, we build trees using the following technique:

1. Start with a set of nucleotides (or amino acids) in corresponding positions.
2. Start with a proposed tree topology

The parsimony algorithm will tell us the plausibility of the topology.

Example: suppose we had four nucleotides, $ACAA$ in four corresponding positions. We might propose the tree structure shown in Figure 3.19.

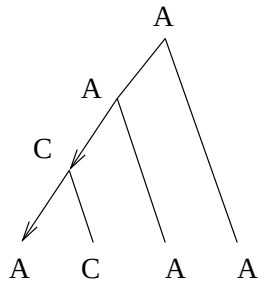


Figure 3.19: Proposed tree for $ACAA$ (two mutations)

Now consider the tree in Figure 3.20, which requires only one mutation. Figure 3.20 is more plausable than figure 3.19 because it requires only one mutation, not two.

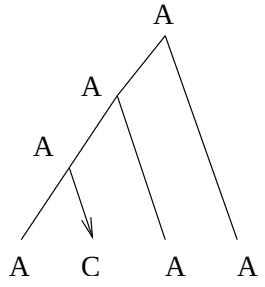


Figure 3.20: Proposed tree for $ACAA$ (one mutation)

Let us define the following operation on the sets u and v:

$$u * v = \begin{cases} u \cap v & \text{if } u \cap v \neq \emptyset \\ u \cup v & \text{if } u \cap v = \emptyset \end{cases} \tag{3.8.1}$$

The second case, $u \cup v$ applies when we have a mutation.

Given two nodes u, v, we will label their ancestor with $u * v$. Figure 3.21 shows an application of $u * v = u \cup v$ while Figure 3.22 shows $u * v = u \cap v$

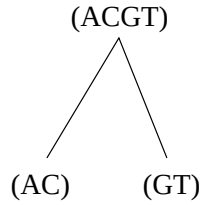


Figure 3.21: Example of $u * v = u \cup v$

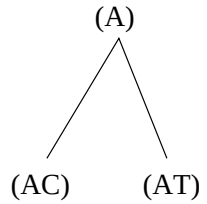


Figure 3.22: Example of $u * v = u \cap v$

Figure 3.23 shows an example with five mutations. In Figure 3.23, the parenthesized letters show where the mutations have occurred.

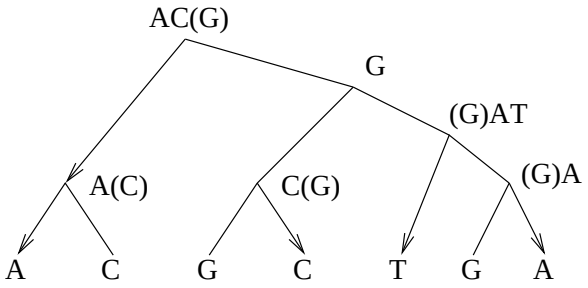


Figure 3.23: Example with 5 mutations

Reference Material

- There's a book on phylogeny by Felsenstien. It's not very mathematical, but it's a well-written and accessible work.
- *Phylogenetics* by Steel, published by Oxford Press is a more mathematical treatment of Phylogeny.

3.9 Lecture – 11/13/2006

Parsimony Algorithms

Last time, we looked at Fitch's parsimony algorithm. We defined a set operation

$$u * v = \begin{cases} u \cap v & \text{if } u \cap v \neq \emptyset \\ u \cup v & \text{if } u \cap v = \emptyset \end{cases}$$

The starting points for parsimony are (1) the topology of a tree and (2) a labeling of the leafs.

Trees found by the parsimony algorithm are not unique. Several different trees can have the same cost associated with them.

Phylogenetic trees are rooted binary trees. Given n leaf nodes, there are a total of $2n-1$ nodes in the tree. The algorithms described here visit trees in postorder.

3.9.1 Fitch's Parsimony Algorithm

In the algorithm below, c denotes the cost of a tree, and k denotes a node. The node $k = 2n-1$ denotes the root of the tree.

Initialization

Set $c = 0$ and set $k = 2n-1$ (the root of the tree)

Recursion

The recursive portion of the algorithm constructs sets of the form R_k.

If k is a leaf, then $R_k = \{x_u^k\}$. In the notation x_u^k, u is a sequence position, and k is a node.

If k is not a leaf, then k has two descendant nodes, i and j.

- Compute R_i, R_j for the descendants of k.
- Set $R_k = R_i * R_j$ ('*' is the set operation described above).
- If $R_i * R_j = R_i \cup R_j$, then increment c. ($R_i * R_j = R_i \cup R_j$ means that there must be a mutation).

Termination

The cost of the tree is retrieved from c.

In this algorithm, the cost of the tree is unrelated to the type of mutation that has to take place. From a biological standpoint, some mutations are less likely than others. For example, the purines are A and G; the pyrimidines are C and T. Mutations within the purines are more likely than mutations from purine to pyrimidine. Likewise, mutations within the pyrimidines are more likely than mutations from pyrimidines to purines.

Fitch's algorithm is not exhaustive – there are some tree structures that it will not find.

Fitch's Algorithm – Examples

Figure 3.24 shows Fitch's algorithm carried out on a tree whose leaves are $ABAB$. The cost of this tree is $c = 2$.

Figure 3.25 shows one way in which specific interior nodes can be chosen. Arrows denote mutations.

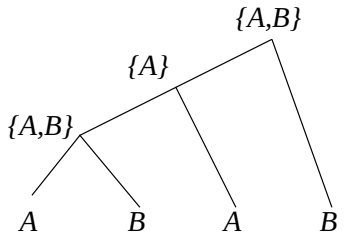


Figure 3.24: Fitch's Algorithm on tree with $ABAB$

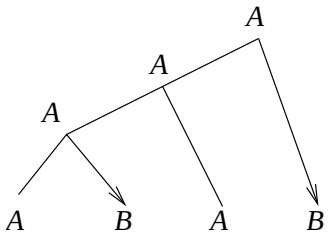


Figure 3.25: One Choice of Interior Nodes for $ABAB$

Figure 3.26 shows another way in which specific interior nodes can be chosen. Again, arrows denote mutations and the cost of the tree is $c = 2$ (two mutations).

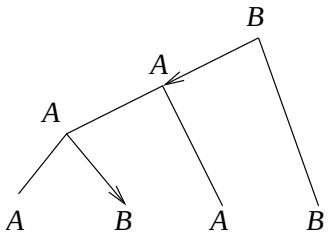


Figure 3.26: Second Choice of Interior Nodes for $ABAB$

Figure 3.27 shows another tree of cost two that is *not found* by the Fitch algorithm.

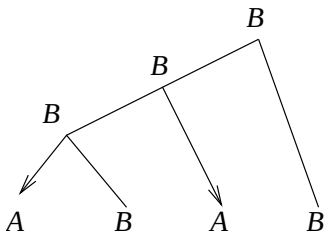


Figure 3.27: Tree that Fitch's algorithm doesn't find

3.9.2 Weighted Parsimony

Weighted parsimony helps account for the fact that certain mutations are more likely than others.

For every node k, and for every symbol $a_1 \dots a_k$, we will compute an array with components for each symbol $S_k(a_i)$.

$S_k(a_i)$ is a number in $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$

Let $s(a, b)$ denote the cost of mutating from a to b.

Weighted Parsimony Algorithm

Initialization

Set $k = 2n - 1$ (the root)

Recursion

If k is a leaf, then

$$S_k(a) = \begin{cases} 0 & \text{if } a = x_u^k \\ \infty & \text{otherwise} \end{cases}$$

If k is not a leaf, then compute $S_i(a)$, $S_j(b)$ for the descendants i, j of k. Then compute $S_k(a)$ as

$$S_k(a) = \min_b(S_i(b) + s(a, b)) + \min_b(S_j(b) + s(a, b))$$

Termination

The minimum cost of the tree is

$$\min_a S_{2n-1}(a)$$

Weighted Parsimony Example

The example below uses the alphabet $\{a, b\}$. For a cost function, we use the following

$$s(a, b) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In other words, $s(a, b) = 0$ if $a = b$; $s(a, b) = 1$ otherwise.

Figure 3.28 shows a tree with leaves $ABAB$. Nodes of the tree have been numbered in post-order.

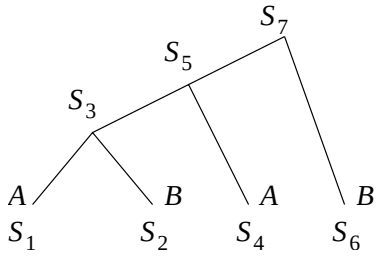


Figure 3.28: Tree for Weighted Parsimony Example

First, we compute the leaves. Note that each leaf is associated with an array of values. The first array element is for the symbol A and the second array element is for the symbol B.

$S_1 = (0, \infty)$	leaf is A
$S_2 = (\infty, 0)$	leaf is B
$S_4 = (0, \infty)$	leaf is A
$S_6 = (\infty, 0)$	leaf is B

Next, we turn to the interior nodes. Below, I'm going to use a slightly different notation that what was presented in class. I'm going to use k as the variable of minimization.

$$\begin{aligned}
S_3(A) &= \min_k(S_1(k) + s(A,k)) + \min_k(S_2(k) + s(A,k)) \\
&= \min(0,\infty) + \min(\infty,1) \\
&= 1 \\
S_3(B) &= \min_k(S_1(k) + s(B,k)) + \min_k(S_2(k) + s(B,k)) \\
&= \min(1,\infty) + \min(\infty,0) \\
&= 1 \\
\therefore S_3 &= (1,1)
\end{aligned}$$

$$\begin{aligned}
S_5(A) &= \min_k(S_3(k) + s(A,k)) + \min_k(S_4(k) + s(A,k)) \\
&= \min(1,2) + \min(0,\infty) \\
&= 1 \\
S_5(B) &= \min_k(S_3(k) + s(B,k)) + \min_k(S_4(k) + s(B,k)) \\
&= \min(2,1) + \min(1,\infty) \\
&= 2 \\
\therefore S_5 &= (1,2)
\end{aligned}$$

$$\begin{aligned}
S_7(A) &= \min_k(S_5(k) + s(A,k)) + \min_k(S_6(k) + s(A,k)) \\
&= \min(1,3) + \min(\infty,1) \\
&= 2 \\
S_7(B) &= \min_k(S_5(k) + s(B,k)) + \min_k(S_6(k) + s(B,k)) \\
&= \min(2,2) + \min(\infty,0) \\
&= 2 \\
\therefore S_7 &= (2,2)
\end{aligned}$$

This algorithm shows the minimal cost computation. To reconstruct the tree symbols, we'd need to keep pointers to the descendant symbols that produced the minimums.

3.10 Probabilistic Approaches to Phylogenetic Trees

Suppose we were dealing with the nucleotides $AGCT$. A probabilistic approach might model this as an HMM.

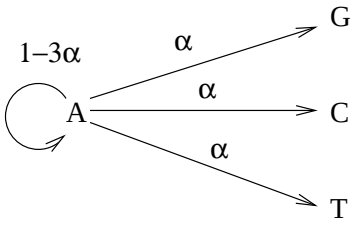


Figure 3.29: Partial Jukes-Cantor HMM for Phylogeny

Figure 3.29 is only partially drawn. Each $AGCT$ node would have four outbound edges, with the same set of probabilities that has been shown for A. In other words, we really have an HMM with four states. This type of model is called the Jukes-Cantor model. A mutation from $u \rightarrow v$ ($u \neq v$) occurs with probability α, regardless of the nucleotides that u, v represent.

The Kimura model is a refinement of the Jukes-Cantor model. Like the Jukes-Cantor model, the Kimura model is based on a 4-state HMM. However, here we make the distinction between purines and pyrimidines. Given a mutation $u \rightarrow v$, this model assigns a probability α if u and v are both purines (pyrimidines), and a probability β if the mutation crosses purine (pyrimidine) groups.

Figure 3.29 shows the Kimura model. This tends to produce better results than the Jukes-Cantor model.

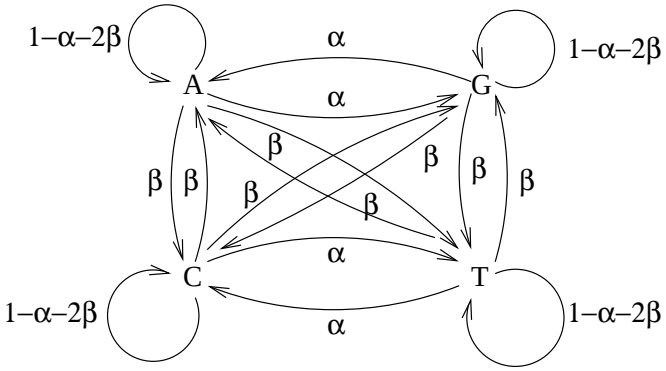


Figure 3.30: HMM for the Kimura Model

3.11 Lecture – 11/16/2006

Probabilistic Approaches to Phylogeny

There are two primary probabilistic approaches to phylogeny:

- Given a set of data, decide on the most plausable tree structure that is consistent with the data. This is the Baysean approach.
- Assume that we have a particular tree structure, and analyze the data with respect to that data structure. This is the approach that we will study.

We would like to look at probabilities of the form $P(\text{data}|\text{tree})$. In more concrete terms, let x and y be nodes of a tree, and let t be the edge that connects them:

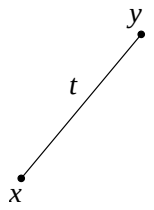


Figure 3.31: x, y and t

Our probabilities will look like $P(x|y,t)$: given ancestor y and time t, what is the probability of x.

Given a tree structure like the one in Figure 3.32

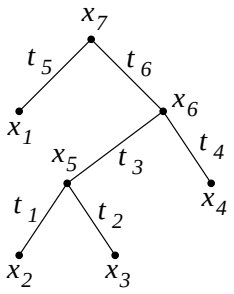


Figure 3.32: Example Tree

The probability of the tree would be calculated as

$$P(x_2|x_5,t_1)\cdot P(x_3|x_5,t_2)\cdot P(x_5|x_6,t_3)\cdot P(x_4|x_6,t_4)\cdot P(x_1|x_7,t_5)\cdot P(x_6|x_7,t_6)\cdot P(x_7)$$

3.11.1 Jukes-Cantor Model

The Jukes-Cantor model assumes that all mutations $u \rightarrow v$ for nucleotides u and v occur with equal probability (α). This is a big simplification.

We can do a discrete analysis of probabilities. Let $P_A(t)$ denote the probability of having nucleotide A at time t in a given position. What is $P_A(t+1)$?

If we denote $P(N_t = A)$ as the probability that the nucleotide at time t is A, we can say

$$\begin{aligned} P_A(t) &= P(N_t = A) \\ P(N_{t+1} = A) &= P(N_{t+1} = A, N_t = A) \\ &\quad + P(N_{t+1} = A, N_t = C) \\ &\quad + P(N_{t+1} = A, N_t = G) \\ &\quad + P(N_{t+1} = A, N_t = T) \end{aligned}$$

Put another way,

$$P(N_{t+1} = A) = P(N_{t+1} = A | N_t = T) \cdot P(N_t = T)$$

The $P(N_{t+1} = A | N_t = T)$ term represents α.

3.11.2 Treating t as a continuous parameter

Suppose we have the evolution

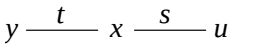


Figure 3.33: Evolution of y, x, u

Because these are probabilities, we have

$$P(u|y, t+s) = \left(\sum_x P(x|y,t) \right) \cdot P(u|x,s)$$

Given residues x and y, we can represent the substitution probabilities as a matrix $S(t)$. $S(t)$ will be an $n \times n$ matrix, where n is the number of residues. For nucleuotides, $S(t)$ will be a 4×4 matrix.

$$S(t) = \begin{pmatrix} P(A|A,t) & P(A|C,t) & P(A|G,t) & P(A|T,t) \\ P(C|A,t) & P(C|C,t) & P(C|G,t) & P(C|T,t) \\ P(G|A,t) & P(G|C,t) & P(G|G,t) & P(G|T,t) \\ P(T|A,t) & P(T|C,t) & P(T|G,t) & P(T|T,t) \end{pmatrix}$$

Note that the diagonal involve no substitution. Elements on the diagnonal will correspond to $1 - 3\alpha$ in the Jukes-Cantor model; elements not on the diagonal correspond to α. Given a unit of time t, we can express $S(t)$ as a matrix R, where R represents the rates of substitution:

$$R = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}$$

As matrices,

$$S(t+s) = S(t) \cdot S(s)$$

For a short unit of time, ϵ, this gives

$$S(t+\epsilon) = S(t) \cdot S(\epsilon)$$

Taking the first derivative of $S(t)$,

$$\begin{aligned} S'(t) &= \lim_{\epsilon \to 0} \frac{S(t+\epsilon) - S(t)}{\epsilon} \\ S(t+\epsilon) - S(t) &= S(t) \cdot S(\epsilon) - S(t) \\ \frac{S(t+\epsilon) - S(t)}{\epsilon} &= S(t) \cdot \frac{S(\epsilon) - I}{\epsilon} \\ S'(t) &= S(t) \cdot \lim_{\epsilon \to 0} \frac{S(\epsilon) - I}{\epsilon} \end{aligned}$$

In the last line, I is the identity matrix, $S(\epsilon)$ represents the probablity of change at time ϵ. The rate change matrix R will be a constant matrix.

$S(\epsilon) = I + R\epsilon$, so

$$I + R\epsilon = \begin{pmatrix} 1-3\alpha\epsilon & \alpha\epsilon & \alpha\epsilon & \alpha\epsilon \\ \alpha\epsilon & 1-3\alpha\epsilon & \alpha\epsilon & \alpha\epsilon \\ \alpha\epsilon & \alpha\epsilon & 1-3\alpha\epsilon & \alpha\epsilon \\ \alpha\epsilon & \alpha\epsilon & \alpha\epsilon & 1-3\alpha\epsilon \end{pmatrix}$$

In the limit of small ϵ, we have $S'(t) = S(t)R$, so we can write

$$S(t) = \begin{pmatrix} r_t & s_t & s_t & s_t \\ s_t & r_t & s_t & s_t \\ s_t & s_t & r_t & s_t \\ s_t & s_t & s_t & r_t \end{pmatrix}$$

Substituting this into $S'(t) = S(t)R$, we get the equations

$$\begin{aligned} r &= -3\alpha r + 3\alpha s \\ s &= \alpha s + \alpha r \end{aligned}$$

These are satisifed by

$$\begin{aligned} r_t &= \frac{1}{4} \cdot (1 + 3e^{-4\alpha t}) \\ s_t &= \frac{1}{4} \cdot (1 - e^{-4\alpha t}) \end{aligned}$$

We can also look at this in terms of eigenvalues and eigenvectors.

$$\begin{aligned} r' &= \begin{pmatrix} rsss \end{pmatrix} \cdot \begin{pmatrix} -3\alpha \\ \alpha \\ \alpha \\ \alpha \end{pmatrix} \\ r' &= -3\alpha r + 3\alpha s \\ s' &= \begin{pmatrix} srss \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ -3\alpha \\ \alpha \\ \alpha \end{pmatrix} \\ s' &= \alpha s + \alpha r \end{aligned}$$

This system can be simplified to

$$\begin{pmatrix} r' \\ s' \end{pmatrix} = \begin{pmatrix} -3\alpha & 3\alpha \\ \alpha & -\alpha \end{pmatrix} \cdot \begin{pmatrix} r \\ s \end{pmatrix}$$

To solve, we need to find determinants of the form

$$\begin{bmatrix} \lambda + 3\alpha & -3\alpha \\ -\alpha & \lambda + \alpha \end{bmatrix} = 0$$

Where λ is an eigenvalue *(I think ...)*.

$$\begin{aligned} (\lambda + 3\alpha)(\lambda + \alpha) - 4\alpha^2 &= 0 \\ \lambda^2 + 4\lambda\alpha &= 0 \\ \lambda &= (0, -4\alpha) \end{aligned}$$

$$\begin{pmatrix} r(t) \\ s(t) \end{pmatrix} = c \cdot v_0 + de^{-4\alpha t} \cdot v_1$$ *I think this is right ...*

$$(\lambda I - M)\mathbf{v} = 0$$

where $\mathbf{v}$ is an eigenvector.

$$\begin{bmatrix} \lambda + 3\alpha & -3\alpha \\ -\alpha & \lambda + \alpha \end{bmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} = 0$$

$$\begin{aligned} (\lambda + 3\alpha)u - 3\alpha w &= 0 \\ -\alpha u + (\lambda + \alpha)w &= 0 \end{aligned}$$

$u = w = 1$ is one solution.
$u + 3w = 0$ is another solution.

At the initial step, $r(0) = 1$ and $s(0) = 0$.

$$\begin{pmatrix} r(0) \\ s(0) \end{pmatrix} = c \begin{pmatrix} 1 \\ 1 \end{pmatrix} + d \begin{pmatrix} -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{aligned} c - 3d &= 1 \\ c + d &= 0 \\ d &= -c \\ 4c &= 1 \\ c &= 1/4 \\ d &= -1/4 \end{aligned}$$

Jukes-Cantor model Final Equations

$$
\begin{aligned}
r(t) &= v_0 c + v_1 d e^{-4\alpha t} \\
r(t) &= 1c + -3de^{-4\alpha t} \\
r(t) &= \frac{1}{4} - 3(-\frac{1}{4})e^{-4\alpha t} && \text{from } c = 1/4,\ d = -1/4 \\
r(t) &= \frac{1}{4}(1 + 3e^{-4\alpha t}) && \lozenge
\end{aligned} \tag{3.11.1}
$$

$$
\begin{aligned}
s(t) &= v_0 c + v_1 d e^{-4\alpha t} \\
s(t) &= \frac{1}{4} + (-\frac{1}{4})e^{-4\alpha t} && \text{from } c = 1/4,\ d = -1/4 \\
s(t) &= \frac{1}{4}(1 - e^{-4\alpha t}) && \lozenge
\end{aligned} \tag{3.11.2}
$$

To Do

- Review Eigenvalues and Eigenvectors

3.12 Review: Determinants, Eigenvalues, Eigenvectors

3.12.1 Determinants

For a 2×2 matrix

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

For a 3×3 matrix:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$$

- Each of the terms in a determinant is a product of a diagonal. For the 3×3 matrix, notice how the diagonal 'wraps' around to the other side.
- A matrix with a row of zeros has a determinant of zero.
- $\det(AB) = (\det(A)) \cdot (\det(B))$

3.12.2 Eigenvalues and Eigenvectors

The general form is

$$A\mathbf{x} = \lambda\mathbf{x}$$

where

- A is a square matrix
- λ is an *eigenvalue*
- $\mathbf{x}$ is an *eigenvector*

One of the uses of eigenvalues is their ability to simplify computations. It's easier to multiply a vector by a constant than to multiply it by a matrix.

Finding Eigenvalues

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ A\mathbf{x} - \lambda\mathbf{x} &= 0 \\ A\mathbf{x} - \lambda I\mathbf{x} &= 0 \\ (A - \lambda I)\mathbf{x} &= 0 \end{aligned}$$

In order for $\mathbf{x}$ to be an eigenvector, we must chose λ such that $\det(A - \lambda I) = 0$.

Example: Finding eigenvalues

Let

$$A = \begin{bmatrix} 1 & -2 \\ 2 & -4 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} 1 & -1 \\ 3 & -4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

$$= \begin{bmatrix} 1-\lambda & -2 \\ 3 & -4-\lambda \end{bmatrix}$$

Setting the determinant $\det(A - \lambda I) = 0$, we have

$$\begin{aligned} \det(A - \lambda I) &= \det \begin{bmatrix} 1-\lambda & -2 \\ 3 & -4-\lambda \end{bmatrix} \\ &= (1-\lambda)(-4-\lambda) - (-6) \\ &= \lambda^2 + 3\lambda - 4 + 6 \\ &= \lambda^2 + 3\lambda + 2 \end{aligned}$$

$$\lambda^2 + 3\lambda + 2 = 0$$

$$(\lambda + 2)(\lambda + 1) = 0$$

$$\lambda = \{-1, -2\}$$

$\therefore$ the eigenvalues are -1 and -2.

Finding Eigenvectors

- Start by finding the eigenvalues
- Substitute each eigenvalue λ into $A\mathbf{x} = \lambda\mathbf{x}$ (or equivalently, into $(A - \lambda I)\mathbf{x} = 0$.

Example: Finding Eigenvectors

Let

$$A = \begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix}$$

We have already found the eigenvalues for A: $\lambda = \{-1, -2\}$

Substituting $\lambda = -1$:

$$\begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x_1 - 2x_2 = -x_1$$

$$3x_1 - 4x_2 - x_2$$

$$2x_1 - 2x_2 = 0$$

$$3x_1 - 3x_2 = 0$$

This system is satisfied by any vector where $x_1 = x_2$:

$$c \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

For any c.

Substituting $\lambda = -2$:

$$\begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{aligned} x_1 - 2x_2 &= -2x_1 \\ 3x_1 - 4x_2 &= -2x_2 \end{aligned}$$

$$\begin{aligned} 3x_1 - 2_x 2 &= 0 \\ 3x_1 - 2_x 2 &= 0 \end{aligned}$$

This system is satisifed by

$$c \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

for any constant c.

3.13 Lecture – 11/20/2006

Logisitics

- We'll wrap up phylogeny this week
- We'll start formal language methods next week
- We'll have one more homework assignment on phylogeny
- Our last exam will be a take-home test, distibuted on the last day of class.
- For next lecture, look over Felsenstein's algorithm.

3.13.1 Jukes-Cantor Model

The Jukes-Cantor model uses functions $r(t)$, $s(t)$ and a matrix $S(t)$:

$$S(t) = \begin{pmatrix} r(t) & s(t) & s(t) & s(t) \\ s(t) & r(t) & s(t) & s(t) \\ s(t) & s(t) & r(t) & s(t) \\ s(t) & s(t) & s(t) & r(t) \end{pmatrix}$$

$S(t)$ has the property that $S(t) \cdot S(u) = S(t+u)$:

$$\begin{pmatrix} r(t) & s(t) & s(t) & s(t) \\ s(t) & r(t) & s(t) & s(t) \\ s(t) & s(t) & r(t) & s(t) \\ s(t) & s(t) & s(t) & r(t) \end{pmatrix} \cdot \begin{pmatrix} r(u) & s(u) & s(u) & s(u) \\ s(u) & r(u) & s(u) & s(u) \\ s(u) & s(u) & r(u) & s(u) \\ s(u) & s(u) & s(u) & r(u) \end{pmatrix} = \begin{pmatrix} r(t+u) & s(t+u) & s(t+u) & s(t+u) \\ s(t+u) & r(t+u) & s(t+u) & s(t+u) \\ s(t+u) & s(t+u) & r(t+u) & s(t+u) \\ s(t+u) & s(t+u) & s(t+u) & r(t+u) \end{pmatrix}$$

We provide direct definitions for $r(t+u)$, $s(t+u)$ (the symmetry of the matrix simplifies the job of doing this)

$$r(t+u) = r(t)r(u) + 3s(t)s(u) \tag{3.13.1}$$
$$s(t+u) = r(t)s(u) + s(t)r(u) + 2s(t)s(u) \tag{3.13.2}$$

(Note that the book uses r_t for $r(t)$ and s_t for $s(t)$).

The Jukes-Cantor model does not take into account the differences between mutations of different clases of nucleoties. (For example, mutations within the purines (AG) or within the pyrimidines (CT), are different than mutations that move from one of these groups to the other).

3.13.2 Kimura's Model of Phylogeny

The Kimura model assigns different probabilities, based on whether the mutation crosses nucleotide groups, as shown in Figure 3.34.

Where the Jukes-Cantor model uses two probabilities, the Kimura model will use three.

The $S(t)$ matrix for the Kimura model is

$$S(t) = \begin{array}{c} \begin{matrix} A & C & G & T \end{matrix} \\ \begin{pmatrix} r(t) & s(t) & u(t) & s(t) \\ s(t) & r(t) & s(t) & u(t) \\ u(t) & s(t) & r(t) & s(t) \\ s(t) & u(t) & s(t) & r(t) \end{pmatrix} \end{array} \begin{matrix} A \\ C \\ G \\ T \end{matrix}$$

Notes on $S(t)$

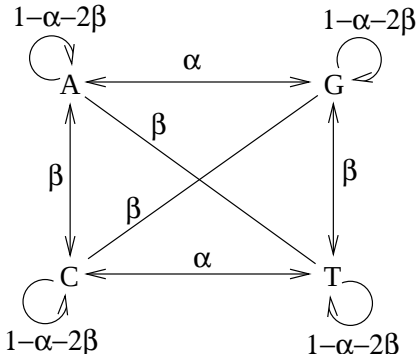


Figure 3.34: HMM For Kimura Model

- $r(t)$ – represents transitions to the same nucleotide $(1 - \alpha - 2\beta)$. There is one $r(t)$ per row
- $s(t)$ – represents transitions between the purines and pyrimidines (β). There are two of these per row.
- $u(t)$ – represents transactions to a different nucleotide within the same class (α). There is one $u(t)$ per row.

In the Kimura model, we'll still want $S(t + t') = S(t) \cdot S(t')$.

$$S(t + \epsilon) = S(t) \cdot S(\epsilon)$$

$$S(\epsilon) = \begin{array}{c} \begin{array}{cccc} \text{A} & \text{C} & \text{G} & \text{T} \end{array} \\ \begin{pmatrix} 1 - \alpha\epsilon - 2\beta\epsilon & \beta\epsilon & \alpha\epsilon & \beta\epsilon \\ \beta\epsilon & 1 - \alpha\epsilon - 2\beta\epsilon & \beta\epsilon & \alpha\epsilon \\ \alpha\epsilon & \beta\epsilon & 1 - \alpha\epsilon - 2\beta\epsilon & \beta\epsilon \\ \beta\epsilon & \alpha\epsilon & \beta\epsilon & 1 - \alpha\epsilon - 2\beta\epsilon \end{pmatrix} \end{array} \begin{array}{c} A \\ C \\ G \\ T \end{array}$$

$$\begin{aligned} S(t + \epsilon) - S(t) &= S(t) \cdot S(\epsilon) - S(t) \\ &= S(t)(S(\epsilon) - I) \end{aligned}$$

$$\begin{aligned} S'(t) &= \lim_{\epsilon \to 0} \frac{S(t + \epsilon) - S(t)}{\epsilon} \\ &= S(t) \cdot \lim_{\epsilon \to 0} \frac{S(\epsilon) - I}{\epsilon} \end{aligned}$$

Let us refer to the matrix R as

$$R = \lim_{\epsilon \to 0} \frac{S(\epsilon) - I}{\epsilon}$$

Kimura's R will be

$$R = \begin{array}{c} \begin{array}{cccc} \text{A} & \text{C} & \text{G} & \text{T} \end{array} \\ \begin{pmatrix} -\alpha - 2\beta & \beta & \alpha & \beta \\ \beta & -\alpha - 2\beta & \beta & \alpha \\ \alpha & \beta & -\alpha - 2\beta & \beta \\ \beta & \alpha & \beta & -\alpha - 2\beta \end{pmatrix} \end{array} \begin{array}{c} A \\ C \\ G \\ T \end{array}$$

$S'(t) = S(t) \cdot R$ is a system of linear differential equations. In what follows, we'll be able to reduce this to three equations and three unknowns.

Expressed as matrices, $S'(t) = S(t) \cdot R$ is

$$\begin{pmatrix} r' & s' & u' & s' \\ s' & r' & s' & u' \\ u' & s' & r' & s' \\ s' & u' & s' & r' \end{pmatrix} = \begin{pmatrix} r & s & u & s \\ s & r & s & u \\ u & s & r & s \\ s & u & s & r \end{pmatrix} \cdot \begin{pmatrix} -\alpha - 2\beta & \beta & \alpha & \beta \\ \beta & -\alpha - 2\beta & \beta & \alpha \\ \alpha & \beta & -\alpha - 2\beta & \beta \\ \beta & \alpha & \beta & -\alpha - 2\beta \end{pmatrix}$$

Because of the symmetry we can simpify this to three equations:

$$r' = (-\alpha - 2\beta)r + u\alpha + 2s\beta \tag{3.13.3}$$

$$\begin{aligned} s' &= r\beta + s(-\alpha - 2\beta) + u\beta + s\alpha \\ &= r\beta - 2s\beta + u\beta \end{aligned} \tag{3.13.4}$$

$$\begin{aligned} u' &= r\alpha + s\beta + u(-\alpha - 2\beta) + s\beta \\ &= r\alpha + 2s\beta + u(-\alpha - 2\beta) \end{aligned} \tag{3.13.5}$$

We can say

$$\begin{pmatrix} r' \\ s' \\ u' \end{pmatrix} = \begin{pmatrix} -\alpha - 2\beta & 2\beta & \alpha \\ \beta & -2\beta & \beta \\ \alpha & 2\beta & -\alpha - 2\beta \end{pmatrix} \cdot \begin{pmatrix} r \\ s \\ u \end{pmatrix} \tag{3.13.6}$$

In (3.13.6), the 3×3 matrix

$$M = \begin{pmatrix} -\alpha - 2\beta & 2\beta & \alpha \\ \beta & -2\beta & \beta \\ \alpha & 2\beta & -\alpha - 2\beta \end{pmatrix}$$

comes directly from equations (3.13.3), (3.13.4), and (3.13.5).

To solve this system, we start by computing eigenvalues:

$$\begin{aligned} \lambda &= \det(M - \lambda I) = 0 \\ M - \lambda I &= \begin{pmatrix} -\alpha - 2\beta - \lambda & 2\beta & \alpha \\ \beta & -2\beta - \lambda & \beta \\ \alpha & 2\beta & -\alpha - 2 - \lambda \end{pmatrix} \\ \det(M) &= \begin{pmatrix} -\alpha - 2\beta - \lambda & 2\beta & \alpha \\ \beta & -2\beta - \lambda & \beta \\ \alpha & 2\beta & -\alpha - 2 - \lambda \end{pmatrix} = 0 \\ \det(M) &= \begin{pmatrix} -\alpha - 2\beta - \lambda & 2\beta & -\lambda \\ \beta & -2\beta - \lambda & -\lambda \\ \alpha & 2\beta & -\lambda \end{pmatrix} \\ &= -\lambda \begin{pmatrix} -\alpha - 2\beta - \lambda & 2\beta & 1 \\ \beta & -2\beta - \lambda & 1 \\ \alpha & 2\beta & 1 \end{pmatrix} \\ &= -\lambda \begin{pmatrix} -\alpha - 2\beta - \lambda & 2\beta & 1 \\ \alpha + \beta + 2\beta + \lambda & -4\beta - \lambda & 0 \\ 2\alpha + 2\beta + \lambda & 0 & 0 \end{pmatrix} \end{aligned}$$

In the final line above, the only portion which contributes to the determinant is $a_{31}a_{22}a_{13}$, so we have

$$\det(M) = -\lambda(4\beta + \lambda)(2\alpha + 2\beta + \lambda) = 0$$

The solutions will take the form $e^{\lambda t}$ (a decay function), where $\lambda \leq 0$.

The eigenvalues are

$$\lambda_1 = 0$$
$$\lambda_2 = -4\beta$$
$$\lambda_3 = -2\alpha - 2\beta$$

Now that we have the eiqenvalues, we can find the eigenvectors, $\mathbf{v}$.

$$M\mathbf{v} = \lambda\mathbf{v}$$

For $\lambda_1 = 0$:

$$\begin{pmatrix} -\alpha - 2\beta & 2\beta & \alpha \\ \beta & -2\beta & \beta \\ \alpha & 2\beta & -\alpha - 2\beta \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix} = 0 \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix}$$

Therefore, for $\lambda_1 = 0$,

$$\mathbf{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

For $\lambda_2 = -4\beta$:

$$\begin{pmatrix} -\alpha - 2\beta & 2\beta & \alpha \\ \beta & -2\beta & \beta \\ \alpha & 2\beta & -\alpha - 2\beta \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix} = -4\beta \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix}$$

$$\begin{aligned} (-\alpha - 2\beta)v_1 + 2\beta v_2 + \alpha v_3 &= -4\beta v_1 \\ \beta v_1 - 2\beta v_2 + \beta v_3 &= -4\beta v_2 \\ \alpha v_1 + 2\beta v_2 + (-\alpha - 2\beta)v_3 &= -4\beta v_3 \end{aligned}$$

$$\begin{aligned} (-\alpha + 2\beta)v_1 + 2\beta v_2 + \alpha v_3 &= 0 \\ \beta v_1 + 2\beta v_2 + \beta v_3 &= 0 \\ \alpha v_1 + 2\beta v_2 + (-\alpha + 2\beta)v_3 &= 0 \end{aligned}$$

$$\begin{aligned} (-\alpha + 2\beta)v_1 + 2\beta v_2 + \alpha v_3 &= 0 \\ v_1 + 2v_2 + v_3 &= 0 \\ \alpha v_1 + 2\beta v_2 + (-\alpha + 2\beta)v_3 &= 0 \end{aligned}$$

For $\lambda_2 = -4\beta$, we have the eigenvector

$$\mathbf{v} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

For $\lambda_3 = -2\alpha - 2\beta$:

$$\begin{pmatrix} -\alpha - 2\beta & 2\beta & \alpha \\ \beta & -2\beta & \beta \\ \alpha & 2\beta & -\alpha - 2\beta \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix} = -2\alpha - 2\beta \cdot \begin{pmatrix} v_1 \\ v_2 \\ v3 \end{pmatrix}$$

For λ_3,

$$\mathbf{v} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

Now, we can put all of these pieces together.

$$\begin{pmatrix} r(t) \\ s(t) \\ u(t) \end{pmatrix} = c \cdot e^{0t} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + d \cdot e^{-4\beta t} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} + k \cdot e^{(-2\alpha - 2\beta)t} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

where c, d, and k are constants.

Let

$$\begin{aligned} r(0) &= 1 \qquad\qquad \text{at } t = 0, \text{ when we have no mutation} \\ s(0) &= 0 \\ u(0) &= 0 \end{aligned}$$

$$\begin{pmatrix} r(t) \\ s(t) \\ u(t) \end{pmatrix} = \begin{pmatrix} c \\ c \\ c \end{pmatrix} + \begin{pmatrix} d \\ -d \\ d \end{pmatrix} + \begin{pmatrix} k \\ 0 \\ -k \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Solving for c, d, k:

$$\begin{aligned} c + d + k &= 1 \\ c - d &= 0 \\ c &= d \\ c + d - k &= 0 \\ k &= 2c \\ c &= 1/4 \\ d &= 1/4 \\ k &= 1/2 \end{aligned}$$

Kimura model final equations

$$\begin{pmatrix} r(t) \\ s(t) \\ u(t) \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \frac{1}{4} e^{-4\beta t} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} + \frac{1}{2} e^{(-2\alpha - 2\beta)t} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} \tag{3.13.7}$$

$$r(t) = \frac{1}{4} + \frac{1}{4} e^{-4\beta t} + \frac{1}{2} e^{(-2\alpha - 2\beta)t} \tag{3.13.8}$$

$$s(t) = \frac{1}{4} - \frac{1}{4} e^{-4\beta t} \tag{3.13.9}$$

$$u(t) = \frac{1}{4} + \frac{1}{4} e^{-4\beta t} - \frac{1}{2} e^{(-2\alpha - 2\beta)t} \tag{3.13.10}$$

3.14 Lecture – 11/22/2006 (Part 1)

Logistics

- hw3 online, due 12/4/2006

3.14.1 Jukes-Cantor Example

Recall that the Jukes-Cantor model of phylogeny used two probabilities:

- $r(t)$ – denotes a transition of $a \to a$ for any nucleotide a.
- $s(t)$ – denotes a transition of $a \to b$ for any nucleotides a, b, $a \neq b$.

Over time $t + u$, r and s can be expressed as

$$r(t+u) = r(t)r(u) + 3s(t)s(u)$$
$$s(t+u) = r(t)s(u) + s(t)r(u) + 2s(t)s(u)$$

Let us use T to denote a tree, and t to denote an evolution time (assigned to edges of the tree). We are interested in the probability

$$P(x_0|T, t_0)$$

We know x_0 and assume T, t_0. Our goal is to maximize P over trees and time distributions.

Consider the tree in Figure 3.35, having three leaves.

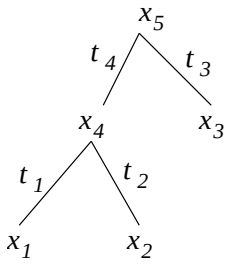


Figure 3.35: Tree T with three leaves

Our hypothesis – the tree has the shape shown in figure 3.35. We would like to evaluate the probability of our hypothesis. That is, we'd like to find the probability

$$P_T = P(x_1, x_2, x_4, x_4, x_5|T, t_1, t_2, t_3, t_4)$$

Applying conditional probabilities,

$$P_T = P(x_1|x_4, t_1) \cdot P(x_2|x_4, t_2) \cdot P(x_4|x_5, t_4) \cdot P(x_3|x_5, t_3) \cdot P(x_5)$$

This computation involves lots of very small numbers. In a computer program, we'd want to use logarithms.

Felsenstein's Algorithm computes the likelihood over a tree, by taking pairs of leaves at a time, and working bottom-up to the root of the tree.

See Durbin pg. 200–201 for Felsenstein's algorithm.

3.14.2 Calculating The likelihood for ungapped aligments

In this section, we'll see how to calculate the likelihood that a pair of sequences evolved from a common ancestor. As we go through this, it's helpful to have a concrete example to refer to. We'll use the example:

$x_1 =$ `CCGGCCGCGCG`
$x_2 =$ `CGGGCCGGCCG`

In this section, we deal with the Jukes-Cantor approach.

Given two sequences, x_1, x_2, there is only one tree. The tree has brances for each of x_1, x_2, and the root is their common ancestor. This is shown in figure 3.36. In figure 3.36 x_a means "ancestor".

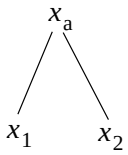


Figure 3.36: Tree for a pair of ungapped alignments

To compute the probability of the sequences, we'll have to start with the probabilites that *each pair* of nucleotides came from a common ancestor. For example:

- $x_{1_1} = C$, $x_{2_1} = C$ is a pair, so we compute that as a tree.
- $x_{1_2} = C$, $x_{2_2} = G$ is a pair, so we compute that as a tree.
- $x_{1_3} = G$, $x_{2_3} = G$ is a pair, so we compute that as a tree.

And so fourth.

In general

$$P(x_{1_i}, x_{2_i}, x_a | T, t_1, t_2) = P(x_a) \cdot P(x_{1_i} | x_a, t_1) \cdot P(x_{2_i} | x_a, t_2)$$

There is one more twist – while we know what x_{1_i} and x_{2_i} are, we don't know what their ancestor is. Thus, the probability that x_{1_i} and x_{2_i} came from a common ancestor is the probability that they both descended from an A, plus the probability that they both descended from a G, plus the probability that they both descended from a C, plus the probability that they both descended from a T.

In other words, we need to find P for every possible ancestor and sum them together:

$$P(x_{1_i}, x_{2_i} | T, t_1, t_2) = \sum_{a \in A,C,G,T} P(x_a) \cdot P(x_{1_i} | x_a, t_1) \cdot P(x_{2_i} | x_a, t_2) \tag{3.14.1}$$

The book uses q_A, q_C, q_G, q_T where I've used $P(x_a)$. I'm changing the notation slightly so that it makes more sense to me.

To find the probability of the two sequences:

- Find the probability that each pair of nucleotides came from a common ancestor. Again, we'll need to consider all 4 nucleotides as possible ancestors.
- To find the probability of the entire sequence, we take the product of the probabilities for the individual pairs.

The probability of the entire sequences (N pairs) is therefore:

$$P(x_1, x_2 | T, t_1, t_2) = \prod_{i=1}^{N} = P(x_{i_1}, x_{2_i} | T, t_1, t_2) \tag{3.14.2}$$

Note that we can save ourselves a little bit of work by looking at the different x_{1_i}, x_{2_i} combinations and the number of times they occur. For example, in x_1, x_2 the pair (C, C) occurs four times – we can compute the probability for (C, C) once and raise it to the fourth power.

Now, for the example. Recall

$$\begin{aligned} r(t) &= \tfrac{1}{4}(1 + 3e^{-4\alpha t}) && \text{see (3.11.1)} \\ s(t) &= \tfrac{1}{4}(1 - e^{-4\alpha t}) && \text{see (3.11.2)} \\ r(t+u) &= r(t)r(u) + 3s(t)s(u) && \text{see (3.13.1)} \\ s(t+u) &= r(t)s(u) + s(t)r(u) + 2s(t)s(u) && \text{see (3.13.2)} \end{aligned}$$

First, cases where $x_{1_i} = C$ and $x_{2_i} = G$.

$$\begin{aligned} P(C, G|T, t_1, t_2) = & \\ & P(x_A) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations: } A \to C,\ A \to G \\ & + P(x_C) \cdot r(t_1) \cdot s(t_2) && \text{1 mutation: } C \to G \\ & + P(x_G) \cdot s(t_1) \cdot r(t_2) && \text{1 mutation: } G \to C \\ & + P(x_T) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations: } T \to C,\ T \to G \end{aligned}$$

The Jukes-Cantor model assigns equal probabilities for each of the mutations, so

$$\begin{aligned} P(C, G|T, t_1, t_2) &= \tfrac{1}{4}(2 \cdot s(t_1)s(t_2) + r(t_1)s(t_2) + s(t_1)r(t_2)) \\ &= \tfrac{1}{4}s(t_1 + t_2) && \text{from (3.13.2)} \end{aligned}$$

Next, cases where $x_{1_i} = G$ and $x_{2_i} = G$.

$$\begin{aligned} P(G, G|T, t_1, t_2) = & \\ & P(x_A) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \\ & + P(x_C) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \\ & + P(x_G) \cdot r(t_1) \cdot r(t_2) && \text{0 mutations} \\ & + P(x_T) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \end{aligned}$$

$$\begin{aligned} P(G, G|T, t_1, t_2) &= \tfrac{1}{4}(3s(t_1)s(t_2) + r(t_1)r(t_2)) \\ &= \tfrac{1}{4}(s(t_1 + t_2)) && \text{from (3.13.2)} \end{aligned}$$

Next, cases where $x_{1_i} = C$ and $x_{2_i} = C$.

$$\begin{aligned} P(C, C|T, t_1, t_2) = & \\ & P(x_A) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \\ & + P(x_C) \cdot r(t_1) \cdot r(t_2) && \text{0 mutations} \\ & + P(x_G) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \\ & + P(x_T) \cdot s(t_1) \cdot s(t_2) && \text{2 mutations} \end{aligned}$$

$$\begin{aligned} P(C, C|T, t_1, t_2) &= \tfrac{1}{4}(3s(t_1)s(t_2) + r(t_1)r(t_2)) \\ &= \tfrac{1}{4}(r(t_1 + t_2)) && \text{from (3.13.1)} \end{aligned}$$

Finally, cases where $x_{1_i} = G$ and $x_{2_i} = C$.

$$
\begin{aligned}
P(G,C|T,t_1,t_2) = & \\
& P(x_A)\cdot s(t_1)\cdot s(t_2) && \text{2 mutations} \\
& + P(x_C)\cdot s(t_1)\cdot r(t_2) && \text{1 mutation} \\
& + P(x_G)\cdot r(t_1)\cdot s(t_2) && \text{1 mutation} \\
& + P(x_T)\cdot s(t_1)\cdot s(t_2) && \text{2 mutations}
\end{aligned}
$$

$$
\begin{aligned}
P(G,C|T,t_1,t_2) &= \tfrac{1}{4}(2\cdot s(t_1)s(t_2) + r(t_1)s(t_2) + s(t_1)r(t_2)) \\
&= \tfrac{1}{4}s(t_1+t_2) && \text{from (3.13.2)}
\end{aligned}
$$

Note that the probabilities for CC and GG are the same.
Likewise, the probabilities for CG, CG are the same.

x_1, x_2 have 8 of 11 pairs with CC or GG. These each have probability $\frac{1}{4}(r(t_1+t_2))$

x_1, x_2 have 3 of 11 pairs with GC or CG. These each have probability $\frac{1}{4}(s(t_1+t_2))$

The final probability is

$$P(x_1,x_2|T,t_1,t_2) = (\tfrac{1}{4})^{11}\cdot(s(t_1+t_2))^3\cdot(r(t_1+t_2))^8 \qquad \text{from (3.14.2)}$$

Part 4

Formal Language Methods

4.1 Lecture – 11/22/2006 (Part II)

Suppose we wanted to count then occurrences of a specific sequence of nucleotides. For example – how many times does CGG occur in a particular piece of DNA?

DFAs are good at *recognizing* sequences of symbols. However, the are not good at *counting* them.

One can create a finite state machine that works as a binary counter (if it counts to N, then it has $\log_2 N$ states). We might be able to count nucleotide sequences with by hooking two machines together: one that recognizes nucleotides, and one that counts them.

For next class: brush up on automata theory and grammars.

4.2 Logarithm Review

$$\ln(e^x) = x$$
$$e^{\ln x} = x$$
$$\lim_{x \to -\infty} e^x = 0$$
$$\lim_{x \to \infty} e^x = \infty$$
$$e^{x+y} = e^x e^y$$
$$e^{x-y} = \frac{e^x}{e^y}$$
$$(e^x)^y = e^{xy}$$
$$\frac{d}{dx} e^x = e^x$$
$$a^x = e^{x \ln a}$$
$$a^{x+y} = a^x a^y$$
$$\log_a x = \frac{\ln x}{\ln a}$$
$$\log_a x = y \Leftrightarrow a^y = x$$

4.3 Decay Functions

Let $y = f(t)$ denote the size of a population at time t. The rate of growth (or decay) will be proportional to the size of the population.

$$f'(t) = k \cdot f(t)$$

for some constant k.

If $y(t)$ is the value of quantity y at time t, then the rate of change in y is

$$\frac{dy}{dt} = ky$$

For $k > 0$, the is a growth function; for $k < 0$, this is a decay function.

Any formula that satisifies

$$\frac{dy}{dt} = ky$$

must have the form

$$y = ce^{kt}$$

In general,

$$y(t) = Ae^{kt}$$

where A represents the value at time zero: $y(0) = A$.

Therefore, we can say

$$y(t) = Ae^{kt} \tag{4.3.1}$$
$$y(t) = y(0)e^{kt} \tag{4.3.2}$$

4.4 Lecture – 11/27/2006

4.4.1 PROSITE

PROSITE is a swedish project that allows one to look up geneome research (you search for research papers by genome pattern). Some characteristics of PROSITE patterns:

- patterns are separated by dashes. For example

  `p0 - p1 - p2 - ... - pk`

  where p_i is a pattern (a sequence of protiens)
- `x(5)` – match any sequence of 5 characters
- `x(m,n)` – means match at least m occurrences, but no more than n.
- `[AHKPU]` – match any of the symbols A,H,K,P,U
- {`AHKPU`} – match any symbol that is not A,H,K,P,U (like `[^AHKPU]`)

An article on PROSITE: `http://bib.oxfordjournals.org/cgi/reprint/3/3/265`
Prosite syntax: `http://www.expasy.org/prosite/`

4.4.2 Regular Languages

The class of regular languages $\mathcal{R}$ is closed with respect to union, product and star. These are the regular operations.

Regular operations are monotonic. Negation is not a monotonic operation.

4.4.3 Context-Free Grammars for RNA

The spatial Structure of RNA

- RNA and DNA form coils. This entanglement is necessary for the protien to function properly.
- Given a succession of nucleotides, can we predict how they will coil? For DNA, there is affinity between the nucleotide pairs (AT, CG). For RNA, there is affinity between the pairs (AU, CG). These affinities affect the shape of the coil.

For example: given $CAGGAAACUG$, Figure 4.1 shows affinity between three nucleotide pairs. Figure 4.2 shows one possible shape for the coil.

Figure 4.1: Nucleotide affinity for $CAGGAAACUG$

In order to describe this kind of structure with a grammar, we need a context free grammar. For example

$S \to CSG \mid GSC \mid ASU \mid USA$
$S \to X$
$X \to CX \mid GX \mid AX \mid TX \mid \lambda$

The first production matches nucleotide pairs. The third production "fills in the middle".

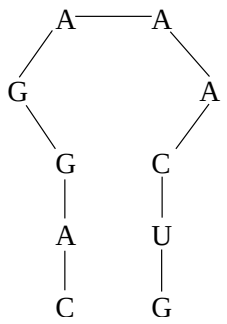


Figure 4.2: Possible Coil shape for $CAGGAAACUG$

Using the grammar above, a derivation for $CAGGAAACUG$ is

$$\begin{aligned} S &\Rightarrow CSG \\ &\Rightarrow CASUG \\ &\Rightarrow CAGSCUG \\ &\Rightarrow CAGXCUG \\ &\overset{4}{\Rightarrow} CAGGAAAXCUG \\ &\Rightarrow CAGGAAACUG \end{aligned}$$

We can use this idea to generate grammars for more complex structures, such as the one shown in Figure 4.3.

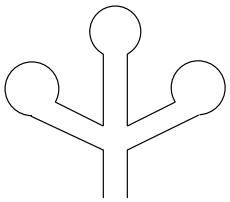


Figure 4.3: A more complicated coil structure

4.4.4 Review of CFGs

A CFG is a 4-tuple

$$G = (V_N, V_T, S, F)$$

where

- V_N is the set of non-terminals
- V_T is the set of terminals
- S is the start symbol
- F is the set of productions.

Productions typically have the form

$$X \to \alpha$$

An *erasure production* has the form

$$X \to \lambda$$

where λ is the null word. This production allows the non-terminal X to disappear.

If $\lambda \notin L(G)$, then there is a grammar G' such that G' has no erasure rules and $L(G) = L(G')$.

4.4.5 Stochastic Context Free Grammars

A traditional context-free grammar has productions of the form

$$\begin{aligned} &S \stackrel{*}{\Rightarrow} \ldots X \ldots \\ &X \to \alpha_1 \\ &X \to \alpha_2 \\ &\quad \vdots \\ &X \to \alpha_n \end{aligned}$$

One can apply any of the $X \to \alpha_i$ rules to the non-terminal X.

A stochastic context-free grammar (SCFG) assigns a *probability* to each production:

$$\begin{aligned} &X \to \alpha_1 \qquad P_1 \\ &X \to \alpha_2 \qquad P_2 \\ &\quad \vdots \\ &X \to \alpha_n \qquad P_n \end{aligned}$$

Because this is a probability distribution, $\sum_i P_i = 1$ where the LHS non-terminal is X.

With an SCFG, we can determine the probability of deriving a particular word, but only if the word has a finite number of derivations.

The question "is $x \in L(G)$" is decideable for context-free grammars, if all productions have the form $X \to YZ$ or $X \to a$.

If G is a CFG, $\lambda \notin L(G)$, then there is a grammar G' such that

1. $L(G') = L(G)$
2. Productions of G' have the form $X \to YZ$ or $X \to a$

Such a grammar G' is said to be in *Chomsky Normal Form*.

If a context-free grammar is in Chomsky Normal Form, then there is a bound on the number of derivations (the parse tree is a binary tree).

4.4.6 Coche-Younger-Kasami Algorithm

This algorithm is also known as the *CYK Algorithm*.

The CYK algorithm is a parsing algorithm for parsing context-free grammars. The algorithm can also be generalized to SCFGs. We'll start with the regular CFG algorithm first.

Let $x = x_1 \ldots x_n$, and let the set of non-terminals be $V_N = W_1 \ldots W_m$.

The algorithm is based on the following formula

$$\gamma(i, j, k) = \begin{cases} 1 & \text{if } W_k \stackrel{*}{\Rightarrow} x_i \ldots x_{i+j-1} \\ 0 & \text{otherwise} \end{cases} \tag{4.4.1}$$

We have $\gamma(1, L, 1) = 1$ where

$$W_1 \stackrel{*}{\Rightarrow} x_1 \ldots x_L = x$$

(L is the length of the word).

Given a grammar in Chomsky Normal form, we can subdivide the RHS as follows:

$$W_1 \stackrel{*}{\Rightarrow} W_{i_1} \dots W_{i_l} \Rightarrow x_{i_1} \dots x_{i_l}$$

Both the first and second parts of this derivation take place in $l-1$ steps, for a total derivation length of $2l-1$. There are a finite number of steps.

In CYK, γ has the following meaning

$$\gamma(i, l, W_j)$$

We generate a string of length l (middle term) starting at position i (first term), from the symbol W_j. (If we order the productions, then W_j could just be the index of a production).

If $l = 1$, the computation is easy – W_j either produces the non-terminal directly or it doesn't.

Algorithm 1 CYK Algorithm

```
1:  procedure CYK
2:      for i = 1 to L do                                   ▷ Handle l = 1 first
3:          for j = 1 to M do
4:              if  w_j → x_i ∈ F then
5:                  γ(i, 1, W_j) = 1
6:              else
7:                  γ(i, 1, W_j) = 0
8:              end if
9:          end for
10:     end for
11:     for i = 1 to L do
12:         for j = 1 to L − 1 do
13:             for k = 1 to M do                           ▷ for each production
14:                 γ(i, j, k) = max_{1≤p≤M, 1≤q≤M, 1≤l≤j−1} (γ(i, l, p) · γ(i + l, j − l, q) · t(k, p, q))   ▷ See below for t()
15:             end for
16:         end for
17:     end for
18:     return γ(1, L, 1)                                  ▷ starting from start symbol
19: end procedure
```

In algorithm 1, t is defined as follows

$$t(k, p, q) = \begin{cases} 1 & \text{if } W_k \to W_p W_q \in F \\ 0 & \text{otherwise} \end{cases}$$

$t(k, p, q)$ tells us whether there is a production that allows us to subdivide the derivation.

Notes on some Aspects of the Algorithm

When does $\gamma(i, j, k) = 1$? When

$$W_k \stackrel{*}{\Rightarrow} x_i \dots x_{i+j-1}$$

Given a production $W_k = W_p W_q$, we can derive

$$W_k \stackrel{1}{\Rightarrow} W_p W_q \stackrel{*}{\Rightarrow} x_i \dots x_{i+j-1}$$

such that

$W_p \overset{*}{\Rightarrow} x_i \dots x_{i+l-1}$ length l

$W_q \overset{*}{\Rightarrow} x_{i+l} \dots x_{i+j-1}$ length $j-l$

For W_p, $\gamma(i, l, p) = 1$
For W_q, $\gamma(i+l, j-l, q) = 1$

With this form of $W_k = W_p W_q$ subdivision, we'll have $\gamma(i, j, k) = 1$ if the following three conditions are met:

1. $\gamma(i, l, p) = 1$
2. $\gamma(i+l, j-l, q) = 1$
3. $W_k \to W_p W_q \in F$ (eg $t(k, p, q) = 1$)

In algorithm 1, this conditions are captured on line 14.

After calculating γ values, it's possible to reconstruct the parse tree with a traceback algorithm.

Complexity of CYK

- Space Requirements: $L^2 \cdot M$
- Time Requirements: $L^3 \cdot M^3$

(L is the length of the word, and M is the number of productions).

4.5 Lecture – 12/29/2006

4.5.1 Stochastic Context Free Grammars

Given a Stochastic CFG, there are two questions we might like to answer.

1. Given x, find $P(x|G)$ – what is the probability that the grammar G generated x?
2. For a nonterminal, U_v, find $P(U_v|x, G)$ – given x and G, what is the probability that the symbol U_v was used in the derivation of x?

Problem 1 can be solved using the *inside algorithm*. Problem 2 can be solved using the *outside algorithm*.

As noted last time, we assume that the grammar is in Chomsky Normal Form. A non-terminal X will appear as the LHS of two types of productions:

$$\begin{aligned} X &\to Y_1 Z_1 \\ X &\to Y_2 Z_2 \\ &\dots \\ X &\to Y_m Z_m \\ X &\to a \end{aligned}$$

To each production, there is an associated probability. We denote the probability of using $X \to Y_2 Z_2$ as

$$t_X(Y_2, Z_2)$$

In general, if we have productions whose LHS member is X

$$\begin{aligned} X &\to \alpha_1 \\ &\dots \\ X &\to \alpha_r \end{aligned}$$

Then we have probabilities $p_1 \dots p_r$ such that

$$\sum_{i=1}^{r} p_i = 1$$

It's a probability distribution.

Let x be a word such that $x = x_1 \dots x_L$. x is L symbols in length. Figure 4.4 shows part of a derivation tree that might produce x.

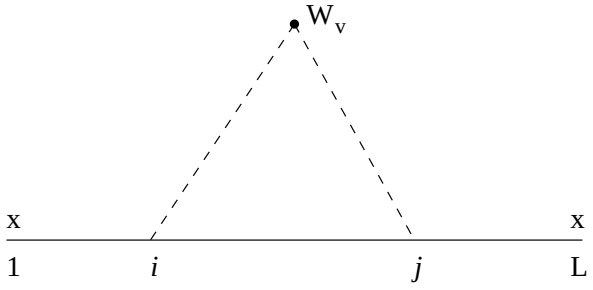


Figure 4.4: Partial parse tree showing derivation $W_v \stackrel{*}{\Rightarrow} x_i \dots x_j$

We will denote

$$P(W_v \stackrel{*}{\Rightarrow} x_i \dots x_j) = \alpha(i, j, v)$$

This represents the probability that production W_v is used to derive the substring $x_i \dots x_j$.

The overall length of x is L. The length of $x_i \dots x_j$ is $j - i + 1$ (i, j are inclusive indexes).

4.5.2 Inside Algorithm

The inside algorithm allows us to find the probability of generating a given string: $P(x|G)$. There are two cases to consider:

Case 1: We use the production $W_v \to a$, and immediately know that W_v generates a. This step will be captured by the initialization phase of the inside algorithm.

Case 2: We use $W_v \to W_y W_z$. Suppose $W_v \overset{*}{\Rightarrow} x_i \dots x_j$. A portion of $x_i \dots x_j$ will come from W_y and a portion will come from W_z. However, we don't know where the split occurs, so will we have to examine every possible break in order to find the one that is most probable.

Figure 4.5 shows the context in which Case 2 operates.

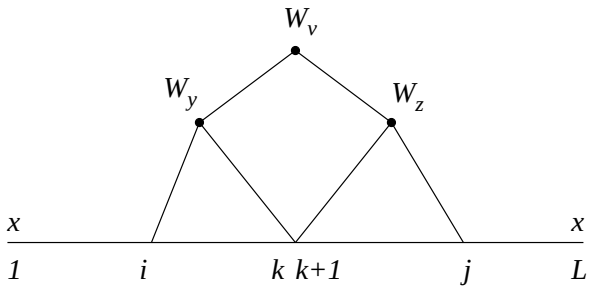


Figure 4.5: Inside algorithm, Case 2 (iteration phase)

Again, note that k can be anywhere between i and $j-1$, and we'll have to check each possible position.

In Figure 4.5, there are really three events taking place:

$W_v \to W_y W_z$
$W_y \overset{*}{\Rightarrow} x_i \dots x_k$
$W_z \overset{*}{\Rightarrow} x_{k+1} \dots x_j$

Because this is a context-free grammar, all three events are independent.

Algorithm 2 Inside Algorithm

```
1:  procedure INSIDE ALGORITHM(String x, Grammar G)
2:                                                    ▷ Initialization Phase
3:      for i = 1 to L do
4:          α(i, i, v) = P(W_v → x_i)
5:      end for
6:                                                    ▷ Iteration Phase
7:      for i = 1 to L do
8:          for j = i + 1 to L do
9:              for v = 1 to M do
10:                 α(i, j, v) = Σ_{y,z; k=i...j−1} α(i, k, y) · α(k + 1, j, z) · t_v(y, z)
11:             end for
12:         end for
13:     end for
14:     return P(W_1 ⇒* x_1 ... x_L) = P(x|G) = α(1, L, 1)
15: end procedure
```

Several points of algorithm 2 are worth discussing.

Look at the formula in line 10

$$\alpha(i,j,v) = \sum_{\substack{y,z \\ k=i\ldots j-1}} \alpha(i,k,y) \cdot \alpha(k+1,j,z) \cdot t_v(y,z) \tag{4.5.1}$$

- i and j are character indexes. v is a production index.
- y and z are RHS productions in $W_v \rightarrow W_y W_z$
- $\alpha(i,k,y)$ covers $x_i \ldots x_k$, produced by W_y (see figure 4.5 for reference).
- $\alpha(k+1,j,z)$ covers $x_{k+1} \ldots x_j$, produced by W_z
- $t_v(y,z)$ denotes the probability of using production $W_v \rightarrow W_y W_z$

In line 14

$$P(x|G) = \alpha(1,L,1) \tag{4.5.2}$$

$\alpha(1,L,1)$ is the probability of producing the word $x = x_1 \ldots x_L$, starting with production W_1 (the start symbol).

The inside algorithm works bottom up.

4.5.3 Outside Algorithm

Given a grammar G, suppose we were interested in

$$\beta(i,j,v) = P(x|W_v \overset{*}{\Rightarrow} x_i \ldots x_j)$$

This is the probability that x was generated by G, conditioned by W_v producing the subsequence $x_i \ldots x_j$

What would $\beta(1,L,1)$ mean? It means:

$$\beta(1,L,1) = P(x|S \overset{*}{\Rightarrow} x_1 \ldots x_L) = 1$$

If x was generated, then the start symbol must produce $x_1 \ldots x_L$.

Before getting into the details of the algorithm, let's look at some diagrams to provide context.

Figure 4.6 shows $W_v \overset{*}{\Rightarrow} x_i \ldots x_j$. W_v is not the start symbol of the grammar.

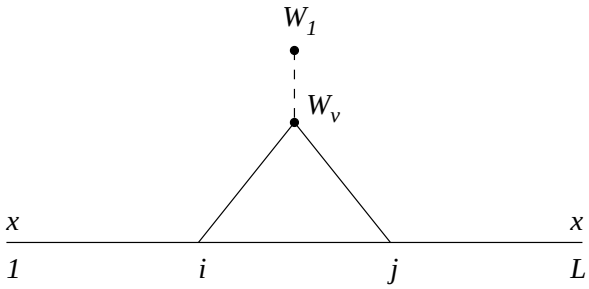


Figure 4.6: $W_v \overset{*}{\Rightarrow} x_i \ldots x_j$, $v \neq 1$

Suppose that W_v was produced by W_y. This can happen in two ways: $W_y \rightarrow W_v W_z$ or $W_y \rightarrow W_z W_v$. The former case is shown in figure 4.7, the latter case is shown in figure 4.8

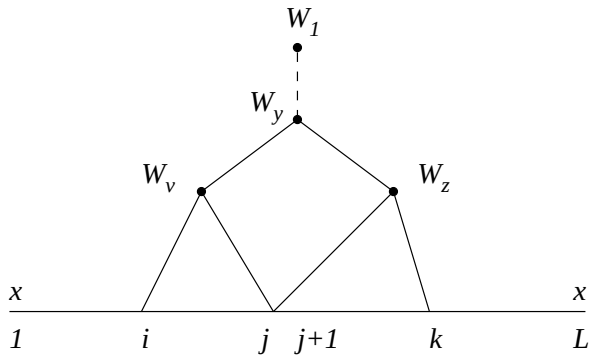


Figure 4.7: Case of $W_y \rightarrow W_v W_z$

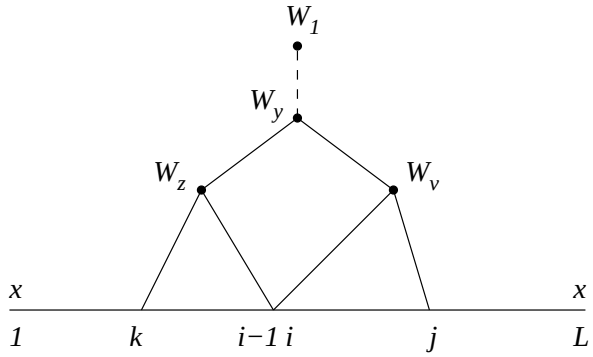


Figure 4.8: Case of $W_y \rightarrow W_z W_v$

Algorithm 3 Outside Algorithm

procedure OUTSIDE ALGORITHM

▷ Initialization Phase

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

$\beta(1, L, v) = 0$ if $v \neq 1$

▷ Iteration Phase

for $i = 1$ to L **do**

  for $j = i + 1$ to L **do**

    for $v = 1$ to M **do**

$$\beta(i,j,v) = \sum_{\substack{y,z \\ k=1\ldots i-1}} \alpha(k, i-1, z) \cdot \beta(k, j, y) \cdot t_y(z, v) + \sum_{\substack{y,z \\ k=1\ldots i-1}} \alpha(j+1, k, z) \cdot \beta(i, k, y) \cdot t_y(v, z)$$

    end for

  end for

end for

▷ termination phase

return $P(x|G) = \sum_{v=1}^{M} \beta(i, i, v) \cdot e_v(x_i)$

end procedure

Some discussion of algorithm 3.

- Lines 3 and 4 are effectively saying that "a derivation may only start with the start symbol".
- i, j, k are character indexes. v, y, z are production indexes.
- In

$$\beta(i,j,v) = \sum_{\substack{y,z \\ k=1\ldots i-1}} \alpha(k,i-1,z)\cdot\beta(k,j,y)\cdot t_y(z,v) + \sum_{\substack{y,z \\ k=1\ldots i-1}} \alpha(j+1,k,z)\cdot\beta(i,k,y)\cdot t_y(v,z) \tag{4.5.3}$$

  The first sum handles the case of $W_y \rightarrow W_z W_v$ (See figure 4.8). The second sum handles the case of $W_y \rightarrow W_v W_z$ (see figure 4.7).

- In the return statement,

$$e_v(x_i) = P(W_v \rightarrow x_i) \tag{4.5.4}$$

  $e_v(x_i)$ acts like an emission probability.

Terms like $\alpha(i,j,v)$ mean

$$P(W_v \overset{*}{\Rightarrow} x_i \cdots x_j)$$

Terms like $\beta(i,j,v)$ mean

$$P(x | W_v \overset{*}{\Rightarrow} x_i \cdots x_j)$$

Taken together

$$\alpha(i,j,v)\cdot\beta(i,j,v) = P(x \in L(G), W_v \overset{*}{\Rightarrow} x_i \cdots x_j)$$

The outside algorithm gives you the probability of a tree, assuming that certain portions have been generated.

4.5.4 Parameter estimation for SCFGs

Let

$$\phi_v = \sum_i \sum_j P(x \in L(G), W_v \overset{*}{\Rightarrow} x_i \cdots x_j)$$

What does ϕ_v mean?

If we divide

$$\frac{\phi_v}{P(x \in L(G))} = \sum_i \sum_j P(W_v \overset{*}{\Rightarrow} x_i \cdots x_j | x \in L(G)) \tag{4.5.5}$$

Equation (4.5.5) is the approximate frequency with which W_v appears in the generation of x. This formula can be used for parameter estimation (similar to the way we estimated parameters for HMMs).

Logistics

- Our next take-home exam will be distributed on 12/6.
- Our next take-home exam will be due on 12/13.

4.6 Lecture – 12/4/2006

4.6.1 Stochastic Context Free Grammars

When working with SCFGs, we assume that all grammars are in Chomsky Normal Form.

Last time, we looked at the probabilities:

$$\alpha(i,j,k) = P(W_v \overset{*}{\Rightarrow} x_i \dots x_j) \tag{4.6.1}$$

$$\beta(i,j,v) = P(S \overset{*}{\Rightarrow} x | W_v \overset{*}{\Rightarrow} x_i \dots x_j, G) \tag{4.6.2}$$

α (4.6.1) is the probability that W_v produced the substring $x_i \dots x_j$.

β (4.6.2) is the probability that x was produced from the start symbol S, under the condition that W_v produced the substring $x_i \dots x_j$.

How do we compute $P(S \overset{*}{\Rightarrow} x|G)$? Let's look at a diagram.

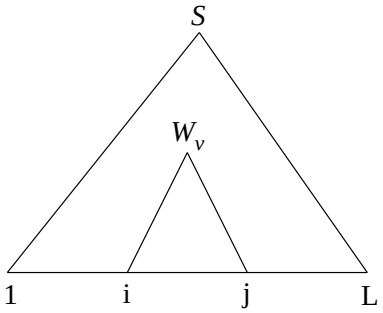


Figure 4.9: Partial derivation tree for $S \overset{*}{\Rightarrow} x$

The probability of $W_v \overset{*}{\Rightarrow} x_i \dots x_j$ is α.

The probability of $S \overset{*}{\Rightarrow} x$ is β (if $W_v \overset{*}{\Rightarrow} x_i \dots x_j$).

$$\begin{aligned} P(S \overset{*}{\Rightarrow} x|G) &= \sum_v \alpha(i,j,v) \cdot \beta(i,j,v) \\ &= \alpha(1,L,1) \qquad \text{for } \textbf{every } i \text{ and } j \end{aligned}$$

For the special case of $i = j$ (a single character), only the outside probabilities are required:

$$\begin{aligned} P(S \overset{*}{\Rightarrow} x|G) &= \sum_v \alpha(i,i,v) \cdot \beta(i,i,v) \\ &= \sum_v e_v(i) \cdot \beta(i,i,v) \end{aligned}$$

This holds for every i.

SCFG's have what is loosely equivalent to transition probabilities and emission probabilities in Hidden Markov Models.

$$\begin{aligned} t_v(y,z) &= \text{probability of } W_v \to W_y W_z && \text{like a transition probability} \\ e_v(i) &= \text{probability of } W_v \to i && \text{like an emission probability} \end{aligned}$$

Together, t_v and e_v form a probability distribution. For every W_v,

$$\sum_{yz} t_v(y,z) + \sum_i e_v(i) = 1$$

4.6.2 Parameter Estimation for SCFGs

As with HMMs, there is an iterative method that allows SCFGs to be "trained" against data. This method is similar to the Baum-Welsch process.

$$\alpha(i,j,v) \cdot \beta(i,j,v) = P(S \overset{*}{\Rightarrow} x, W_v \overset{*}{\Rightarrow} x_i \dots x_j | G) \tag{4.6.3}$$

For a fixed v.

We can use the definition of conditional probabily to rewrite this as

$$P(W_v \overset{*}{\Rightarrow} x_i \dots x_j | S \overset{*}{\Rightarrow} x, G) \cdot P(S \overset{*}{\Rightarrow} x | G) \tag{4.6.4}$$

Equation (4.6.4) is the probability that W_v is used in a deriviation – how do we estimate this?

Let P_v be the probability that W_v is used in a derivation. From (4.6.3) and (4.6.4) we can say

$$P_v = \frac{\sum_{i=1}^{L} \sum_{j=i}^{L-1} \alpha(i,j,v) \cdot \beta(i,j,v)}{P(S \overset{*}{\Rightarrow} x | G)} \tag{4.6.5}$$

Again, P_v is the probability that W_v is used in a derivation. This is something we can count by inspection. However, counting does *not* tell us which $W_v \to W_y W_z$ was used. So, we still need something else.

The probability that we're really interested in is

$$P(W_v \to W_y W_z \overset{*}{\Rightarrow} x_i \dots x_j | S \overset{*}{\Rightarrow} x, G) \tag{4.6.6}$$

Given the case where $W_v \overset{1}{\Rightarrow} W_y W_z \overset{*}{\Rightarrow} x_i \dots x_j$, there must be a number k such that

$$\begin{aligned} W_y &\Rightarrow x_i \dots x_k \\ W_z &\Rightarrow x_{k+1} \dots x_j \end{aligned}$$

In parameter estimation, we must consider every k from $i < k \leq j$.

The probabilities of these events are

$$P(W_y \overset{*}{\Rightarrow} x_i \dots x_k) = \alpha(i,k,y) \tag{4.6.7}$$

$$P(W_z \overset{*}{\Rightarrow} x_{k+1} \dots x_j) = \alpha(k+1,j,z) \tag{4.6.8}$$

Both (4.6.7) and (4.6.8) happen in the context of $W_v \to W_y W_z$.

Considering all possible k values, our probability computation looks like this:

$$\frac{\sum_{k=1}^{j-1} t_v(y,z) \cdot \alpha(i,k,y) \cdot \alpha(k+1,j,z) \cdot \beta(i,j,v)}{P(S \overset{*}{\Rightarrow} x | G)} \tag{4.6.9}$$

The one remaining question is how to express t_v.

$$\begin{aligned} t_v(y,z) &= P(W_v \to W_y W_z | W_v \text{ was used}) \\ &= \frac{P(W_v \to W_y W_z \text{ was used})}{P(W_v \text{ was used})} \end{aligned}$$

Our new estimate of t_v (we'll call it $\hat{t}_v$) is given is (4.6.10)

$$\hat{t}_v(y,z) = \frac{\sum_{i=1}^{L-1}\sum_{j=i}^{L} t_v(y,z)\cdot\alpha(i,k,y)\cdot\alpha(k+1,j,z)\cdot\beta(i,j,v)}{\sum_{i=1}^{L-1}\sum_{j=i}^{L}\alpha(i,j,v)\cdot\beta(i,j,v)} \tag{4.6.10}$$

A similar procedure exists for estimating $e_v(a)$. In this case, we're after

$$e_v(i) = \frac{P(W_v \rightarrow a \text{ was used})}{P(W_v \text{ was used})}$$

We estimate $\hat{e}_v(a)$ using (4.6.11) (from Durbin, pg 256)

$$\hat{e}_v(a) = \frac{\sum_{i|x_i=a}\beta(i,i,v)\cdot e_v(a)}{\sum_{i=1}^{L}\sum_{j=1}^{L}\alpha(i,j,v)\cdot\beta(i,j,v)} \tag{4.6.11}$$

4.6.3 HMM vs SCFG

- is it fair to compare HMMs and SCFGs?
- HMMs are simpler structures. They tend to have many fewer parameters.
- HMMs tend to be more popular, primarily because of their simplicity. SCFGs almost end up giving us "too much freedom".

4.6.4 CYK Algorithm For Stochastic Grammars

A few classes ago, we looked at the CYK algorithm for parsing context-free grammars. Here, we will look at a variation that handles stochastic context-free grammars. In spirit, this is similar to the Viterbi algorithm for Hidden Markov Models.

$\alpha(i,j,v)$ can be expressed as a recurrence:

$$\alpha(i,i,v) = e_v(i)$$
$$\alpha(i,j,v) = \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=i}^{j-1} t_v(z,y)\cdot\alpha(i,j,y)\cdot\alpha(k+1,j,z)$$

In the probabilistic CYK algorithm, our goal is to maximize probabilities.

Let's examine the quantity

$$t_v(y,z)\cdot\alpha(i,k,y)\cdot\alpha(k+i,j,z) \tag{4.6.12}$$

These terms represent the probability that

$$W_v \stackrel{1}{\Rightarrow} W_y W_z \stackrel{*}{\Rightarrow} x_i \ldots x_k x_{k+1} \ldots x_j$$

We are producing a string, but we are using a precisely-chosen derivation tree.

In the CYK algorithm for stochastic context-free grammars, we will seek to maximize (4.6.12).

Below, we use $\gamma(i,j,v)$ in the algorithm. This γ is a little different than the one used in the non-stochastic version of the algorigthm. Here, j is an index – in the earlier version j was a length.

$$\gamma(i,j,v) = \max_{y,z,k}\log\left[t_v(y,z)\cdot\alpha(i,k,y)\cdot\alpha(k+1,j,z)\right] \tag{4.6.13}$$

Because we deal with multiplication of small numbers, we'll want to use logarithms:

$$\gamma(i,j,v) = \max_{y,z,k} [\log t_v(y,z) + \log \alpha(i,k,y) + \log \alpha(k+1,j,z)] \tag{4.6.14}$$

We can replace $\alpha(i,k,y)$ with $\gamma(i,j,y)$, and we can replace $\alpha(k+1,j,z)$ with $\gamma(k+1,j,z)$. This substitution gives

$$\gamma(i,j,v) = \max_{y,z,k} [\log t_v(y,z) + \log \gamma(i,k,y) + \log \gamma(k+1,j,z)] \tag{4.6.15}$$

Of course, we'll need to remember $\text{argmax}_{y,z,k}$ in order to do backtracking (τ in the algorithm given below)

Algorithm 4 Stochastic CYK algorithm

```
1:  procedure STOCHASTIC CYK
2:                                                          ▷ Initialization Phase
3:      for  i = 1 to L do
4:          for  v = 1 to M do
5:              γ(i, i, v) = log e_v(i)
6:              τ(i, i, v) = (0, 0, 0)
7:          end for
8:      end for
9:                                                          ▷ Iteration Phase
10:     for  i = 1 to L − 1 do
11:         for j = i + 1 to L do
12:             for v = 1 to M do
13:                 γ(i, j, v) = max_{y,z} max_{1≤k≤j−1}  γ(i, k, y) + γ(k + 1, j, z) + log t_v(z, y)
14:                 τ(i, j, v) = argmax_{y,z,k}
15:             end for
16:         end for
17:     end for
18:                                                         ▷ termination phase
19:     return γ(1, L, 1)
20: end procedure
```

In line 13, note that log is only applied to the t_v term. (γ takes a log value in the initialization phase, so we just keep adding to it).

In line 14, τ is used to maintain traceback information. Page 257 of Durbin gives the traceback routine shown in algorithm 5

4.7 Secondary Structure of RNA

- RNA is a string of *ACGU* nucleotides.
- RNA is typically a single strand. It tends to coil itself into different shapes. As it turns out, these shapes have biological significance.
- Looking at a strand of RNA, can we predicat how the strand would coil? What secondary structure would it form?

For next class, read over the Nussinov algorithm.

Algorithm 5 Stochastic CYK traceback

procedure STOCHASTIC CYK TRACEBACK
    ▷ Initialization Phase
    Push $(1, L, 1)$ onto the stack.
    ▷ Iteration Phase
    Pop (i, j, v)
    $(y, z, k) = \tau(i, j, v)$
    if $\tau(i, j, v) = (0, 0, 0)$ **then** ▷ implies $i = j$
        attach x_i as the child of v.
    else
        Attach y,z to parse tree as children of v
        Push $(k + 1, j, z)$
        Push (i, k, y)
    end if
end procedure

4.8 Lecture – 12/6/2006

4.8.1 RNA Folding

RNA consists of a sequence of $AGCU$ nucleotides. Affinities between nucleotide pairs are

- $G - C$ (strongest)
- $A - U$ (less strong)
- $G - U$ (weakest)

Figure 4.10 shows some of the structural features of RNA.

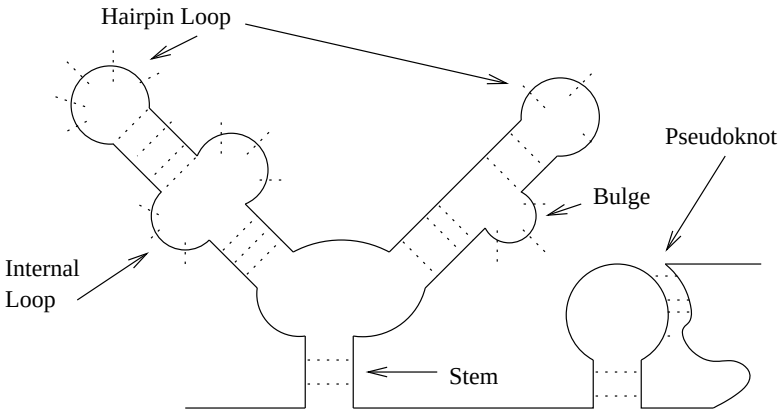


Figure 4.10: Structural Features of RNA

The challenge is to take a linear strand and predict the shape of the folding.

This folding, the *secondary structure* is caused by nucleotide affinity. In figure 4.11, affinity between G and C produces a stem.

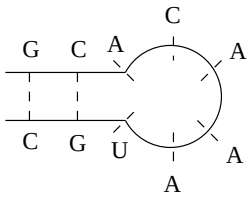


Figure 4.11: Secondary shape from $G - C$

The Nussinov algorithm is based on nucleotide affinity. It does not consider stability of the resulting structure – the algorithm can produce structures that would never occur in nature.

4.8.2 RNA and Parenthesis Matching

The table below uses balanced parenthesis to show different ways in which a single strand can fold. Dots indicate no folding.

Let's Illustrate a few of these examples.

Nussinov's algorithm will allow the structure of Figure 4.12. However, such a structure would never occur in nature – in nature, we never see hairpin loops whose length is less than three.

	A	*C*	*G*	*U*	*A*	*C*	*G*	*U*
Shape 1 (no folding)	·	·	·	·	·	·	·	·
Shape 2 (Figure 4.12)	·	·	·	·	(	·	·	)
Shape 3 (Figure 4.13)	·	·	(	·	·	·	·	)
Shape 4 (Figure 4.14)	·	·	(	(	·	·	)	)
Shape 5	(	·	·	·	·	·	·	)
Shape 6	(	·	·	(	·	·	)	)
Shape 7 (Figure 4.15)	(	·	·	)	(	·	·	)
Shape 8	(	(	·	·	·	·	)	)
Shape 9	(	(	(	·	·	)	)	)

Table 4.1: Examples of RNA Folding

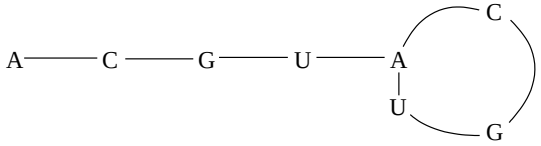


Figure 4.12: Shape 2 of table 4.1

Figure 4.13 is more plausable than Figure 4.12. Although the $G - U$ link is weaker, the loop length is something that could actually occur in nature.

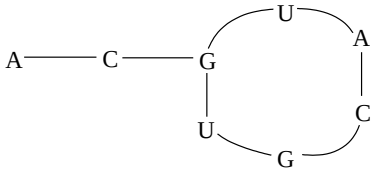


Figure 4.13: Shape 3 of table 4.1

Figure 4.14 shows another impossible hairpin.

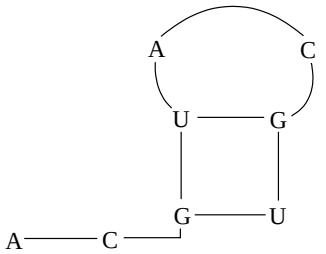


Figure 4.14: Shape 4 of table 4.1

Figure 4.15 has an interesting structure to it.

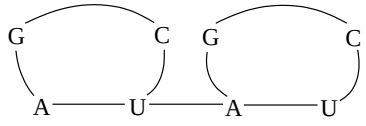


Figure 4.15: Shape 7 of table 4.1

The others are pretty easy to visualize.

4.8.3 Number of Coilings in an RNA String

Given n pairs of parenthesis, how many different ways can you arrange them so that the parens are balanced?

The answer is a number known as a *Catalan Number*

$$\frac{\binom{2n}{n}}{n+1} \sim \Theta(4^n)$$

Let $S(n)$ denote the number of possible coilings for an RNA string.

Base cases:

$$S(0) = 1$$
$$S(1) = 1$$
$$S(2) = 1$$

For the $(n+1)^{\text{st}}$ coiling, there are two ways it could happen: (1) the $(n+1)^{\text{st}}$ nucleotide is paired or (2) the $(n+1)^{\text{st}}$ nucleotide is not paired.

For case (2), there's no additional coiling. $S(n+1) = S(n)$.

For case (1), let k be a point somewhere along the RNA string. The $(n+1)^{\text{st}}$ nucleotide could coil back to any possible index for k. We need to count each of these points as a possible coiling for nucleotide $(n+1)$.

$$S(n+1) = S(n) + \left(\sum_{k=0}^{n-2} S(k) \cdot S(n-k-1) \right) \tag{4.8.1}$$

In (4.8.1), $S(n)$ represents the case of no coiling, and the summation represents the coiling with another nucleotide k.

Writing the $k = 0$ case explicitly,

$$S(n+1) = S(n) + S(n-1) + \left(\sum_{k=1}^{n-2} S(k) \cdot S(n-k-1) \right) \tag{4.8.2}$$

Equation (4.8.2) shows $S(n+1)$. For $S(n)$, this is

$$S(n) = S(n-1) + S(n-2) + \left(\sum_{k=1}^{n-2} S(k) \cdot S(n-k-2) \right) \tag{4.8.3}$$

We can plug (4.8.3) back into (4.8.2), giving

$$S(n+1) = S(n) + S(n-1) + S(n-2) + \left(\sum_{k=1}^{n-3} S(k) \cdot S(n-k-1) \right) \tag{4.8.4}$$

From (4.8.3) and (4.8.4) is pretty easy to see that if $p < q$, then $S(p) < S(q)$.

$\therefore S(n-k-2) < S(n-k-1)$

From (4.8.2) and (4.8.3) we can deduce

$$S(n+1) \geq 2 \cdot S(n)$$
$$S(n) \geq 2^{n-1}$$

Given a sequence of, say, 100 nucleotides this is an infeasable number of computations.

4.8.4 Nussinov Algorithm

Given a sequence $x = x_1 \dots x_l$, our goal is to find a folding that maximizes the number of pairs.

Figure 4.16 - 4.19 show four different ways in which folding can occur:

- Add unpaired position i onto best structure for subsequence $i+1$,j (Figure 4.16).
- Add unpaired position j on to the best structure for subsequence i, $j-1$ (Figure 4.17).
- Add i, j pair onto best structure found for subsequence $i+1$, $j-1$ (Figure 4.18).
- Combine two optimal substuctures i,k and $k+1$, j (Figure 4.19).

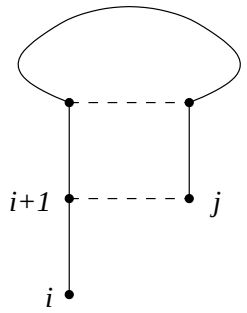


Figure 4.16: Sample Folding #1 – unpaired i

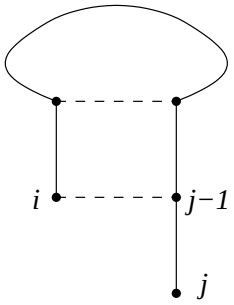


Figure 4.17: Sample Folding #2 – unpaired j

Let $\gamma(i,j)$ be the maximal number of joined nucleotides in a folding of $x_i \dots x_j$ (assuming $i \leq j$).

We define $\gamma(i, i-1) = 0$ (this is a technicality that simplifies the algorithm a little).

Finally, define $\gamma(i,i) = 0$.

This algorithm works by filling in an $(L \times L)$ matrix where L is the length of the RNA sequence.

- There will be zeros along the diagonal (from $\gamma(i,i) = 0$)
- There will be zeros immediately below the diagonal (from $\gamma(i, i-1) = 0$).
- Matrix cells below $(i, i-1)$ will be unused. Our computations take place in the upper right triangle of the matrix. Figure 4.20 shows this matrix. i indexes rows, j indexes columns and Cells with 'x' are unused.

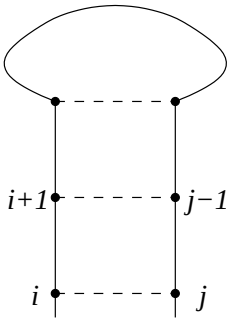


Figure 4.18: Sample Folding #3 – i, j pair

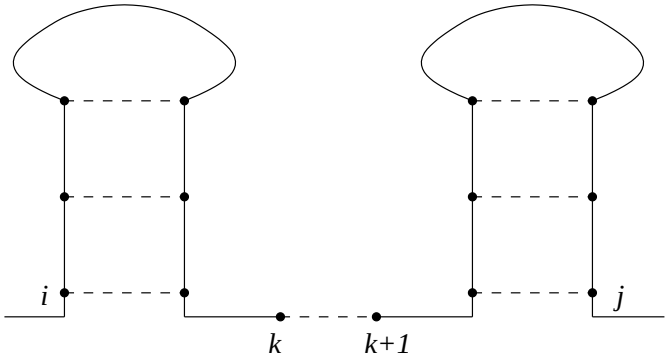


Figure 4.19: Sample Folding #4: bifurcation

$i \backslash j$	1	2	3	...	L
1	0				
2	0	0			
3	**x**	0	0		
⋮	**x**	**x**	0	0	
L	**x**	**x**	**x**	0	0

Figure 4.20: Matrix for Nussinov Algorithm

In line 14 of Algorithm 6, note the similarity between the max terms and figures 4.16–4.19.

Also, from line 14, we define $\delta(i, j)$ as

$$\delta(i,j) = \begin{cases} 1 & \text{if } i,\ j \text{ form a base pair: } GC,\ AU,\ GU \\ 0 & \text{otherwise} \end{cases} \tag{4.8.5}$$

Finally, examine the expression

$$\max_{i<k<j} \gamma(i,k) + \gamma(k+1,j)$$

This only comes into effect when $j - i > 1$ – it considers $j - i - 1$ different places where a bifurcated structure can occur. With each iteration of m the number of terms in the max expression increases by one.

Algorithm 6 Nussinov Algorithm

procedure NUSSINOV
▷ Initialization
  for $i = 1$ to L **do**
    $\gamma(i, i) = 0$
  end for
  for $i = 2$ to L **do**
    $\gamma(i, i-1) = 0$
  end for
▷ Recursion – fill by diagonals
  for $m = 1$ to $L - 1$ **do**
    for $n = 1$ to $L - m$ **do**
      $i = n$
      $j = n + m$

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1, j) \\ \gamma(i, j-1) \\ \gamma(i+1, j-1) + \delta(i,j) \\ \max_{i<k<j} \left[\gamma(i,k) + \gamma(k+1, j)\right] \end{cases}$$

    end for
  end for
  return $\gamma(1, L)$ ▷ Upper-right cell
end procedure

Traceback in the Nussinov Algorithm

$\gamma(1, L)$ tells us the maximum number of matchings. To recover the actual matches, we'd use a traceback algorithm: pointers and arrows, or a stack (as described in Durbin).

Algorithm 7 is Durbin's traceback algorithm.

4.8.5 To Do

- Look at Durbin's example, and try to find the mistake. (Possible exam question).
- Think about how to modify the algorithm, so that it will not consider hairpin loops of length ≤ 3.

Algorithm 7 Nussinov Traceback

```
1: procedure NUSSINOV TRACEBACK
2:                                                          ▷ Initialization
3:     Push (1, L) onto the stack
4:                                                          ▷ Recursion
5:     while Stack is not empty do
6:         pop (i, j)
7:         if i ≥ j then
8:             continue
9:         else if γ(i + 1, j) = γ(i, j) then
10:            push (i + 1, j)
11:        else if γ(i, j − 1) = γ(i, j) then
12:            push (i, j − 1)
13:        else if γ(i + 1, j − 1) + δ(i, j) = γ(i, j) then
14:            Record (i, j) base pair
15:            Push (i + 1, j − 1)
16:        else
17:            for k = i + 1 to j − 1 do
18:                if γ(i, k) + γ(k + 1, j) = γ(i, j) then
19:                    push (k + 1, j)
20:                    push (i, k)
21:                end if
22:            end for
23:        end if
24:    end while
25: end procedure
```

4.9 Lecture – 12/11/2006

4.9.1 hw3, problem 2

> Let d be an ultrametric on a set of objects subjected to the UPGMA algorithm. Prove that if clusters C_j and C_k are joined by the algorithm, then the distances between any member of C_j and any member of C_k are the same.

In order to prove this by induction, we must first state what we are doing induction on (a number, a structure, a step in an algorithm, etc). This holds true for *any* proof by induction.

In this section, we'll look at two different proofs by induction.

Induction of the Number of Algorithm Steps

Let k denote the number of steps in the clustering algorithm.

- At step 0 we have n clusters. (each is a single element)
- At step 1 we have $n-1$ clusters
- At step k we have $n-k$ clusters.

We can do induction on k.

At step k, we have clusters $C_1 \dots C_{n-k}$
At step $k+1$, we have clusters $C_1 \dots C_{n-k-1}$

If step $k+1$ fuses C_1 and C_2, then we must show that all points in C_1 are equidistant to all points in C_2.

Let d_k denote the set of distances in step k. d_{k+1} can be expressed as a function of d_k.

Claim: all distances $d_0 \dots d_k$ are ultrametric.

Inductive Hypothesis: if d_k is an ultrametric, the we can use this to prove that d_{k+1} is an ultrametric.

Let $C_{12} = C_1 \cup C_2$. At step $k+1$, what is $d_{k+1}(C_{12}, C_p)$ (for any other cluster C_p)? Applying the UPGMA's distance formula

$$d(C_{12}, C_p) = \frac{d(C_1, C_p) \cdot |C_1| + d(C_2, C_p) \cdot |C_2|}{|C_1| + |C_2|}$$

Because we start with an ultrametric, $d(C_1, C_p) = d(C_2, C_p)$, so we can factor:

$$\begin{aligned} d(C_{12}, C_p) &= \frac{d(C_1, C_p) \cdot (|C_1| + |C_2|)}{|C_1| + |C_2|} \\ d(C_{12}, C_p) &= d(C_1, C_p) \\ \therefore d(C_{12}, C_p) &= d(C_1, C_p) = d(C_2, C_p) \end{aligned}$$

In the UPGMA algorithm, C_1, C_2 would be joined if the distance between the two clusters was the smallest. Relative to any other cluster C_p, the distances form an isoceles triangle, as shown in figure 4.21.

By arguments above, if $d(C_1, C_p) = d(C_2, C_p)$ then

$$d_{k+1}(C_{12}, C_p) = d_k(C_1, C_p) = d_k(C_2, C_p)$$

In the UPGMA algorithm, if we start with an ultrametric distance measure, successive iterations will produce ultrametric distance measures.

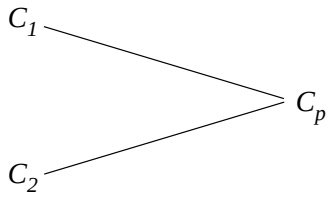


Figure 4.21: Isoceles triangle between C_1, C_2, C_p

Another Possible Proof

Claim: at any step k, if we fuse C_1 and C_2 then for every $x \in C_1$, and every $y \in C_2$, $d(x, y)$ is the same.

We are working with an arrangement like the one shown in figure 4.22.

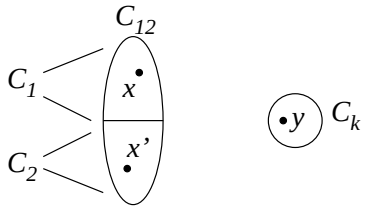


Figure 4.22: Points and clusters for second proof

In the first iteration, assume that $x \in C_p$ and $y \in C_q$.

$$d(C_p, C_q) = d(x, y)$$
$$\therefore d(x, y) = d(C_1, C_k)$$
$$\therefore d(x', y) = d(C_2, C_k)$$

At step $k + 1$ (no relationship to the subscript k in C_k),

$$\begin{aligned} d_{k+1}(C_{12}, C_k) &= \frac{d_k(C_1, C_k) \cdot |C_1| + d_k(C_2, C_k) \cdot |C_2|}{|C_1| + |C_2|} \\ &= d_k(C_1, C_k) \\ &= d_k(C_2, C_k) \end{aligned}$$

$$\begin{aligned} d(x, y) &= d(C_1, C_k) = d(C_{12}, C_k) \\ d(x', y) &= d(C_2, C_k) = d(C_{12}, C_k) \end{aligned}$$

4.9.2 Nussinov-Jacobson Algorithm

Nussinov's algorithm provides a good basis for understanding how to model RNA folding structures. However, the algorithm is unrealistically simple: it allows foldings that would never occur in nature, and it does not consider the molecular stability of the structure.

Several people have devised improved versions of Nussinov's algorithm. We'll examine one called the Nussinov-Jacobson Algorithm.

The Nussinov-Jacobson Algorithm seeks to produce coilings that store a minimal amount of energy. We'll assign energy weightings to the different base pairs:

- $C - G$ is the most stable. We assign an energy of -6.
- $A - U$ is less stable. We assign an energy of -5
- $G - U$ is least stable. We assign an energy of -1.
- For any other pairing, we assign an energy of 0.

Given a folding, we will algebraically sum the energies of the base pairs that are linked together. The most plausable coiling has the smallest energy.

As before, let $x = x_1 \dots x_L$. Let $E(i, j)$ denote the energy of $x_i \dots x_j$. There are four recurrences for the Nussinov-Jacobson algorithm.

Figure 4.23 shows an unpaired i. In this case, $E(i, j) = E(i + 1, j)$.

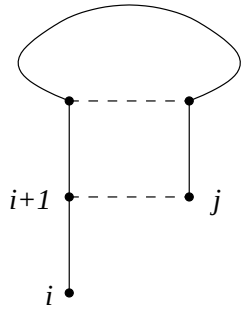


Figure 4.23: unpaired i

Figure 4.24 shows an unpaired j. In this case, $E(i, j) = E(i, j - 1)$.

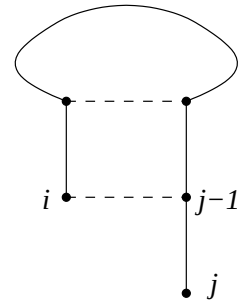


Figure 4.24: unpaired j

Figure 4.25 shows an i,j pair. Here, $E(i, j) = E(i + 1, j - 1) + a(i, j)$. The definition of $a(i, j)$ appears

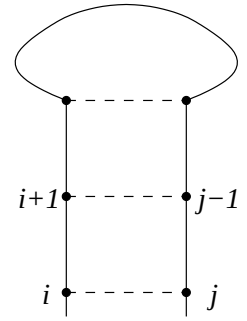


Figure 4.25: i, j pair

in equation (4.9.1).

$$a(i,j) = \begin{cases} -6 & \text{if } ij \text{ is } GC \text{ (or } CG) \\ -5 & \text{if } ij \text{ is } AU \text{ (or } UA) \\ -1 & \text{if } ij \text{ is } GU \text{ (or } UG) \\ 0 & \text{otherwise} \end{cases} \tag{4.9.1}$$

Finally, Figure 4.26 show a bifuration. Here, $E(i,j) = \min_{i<k<j}(E(i,k) + E(k+1,j))$

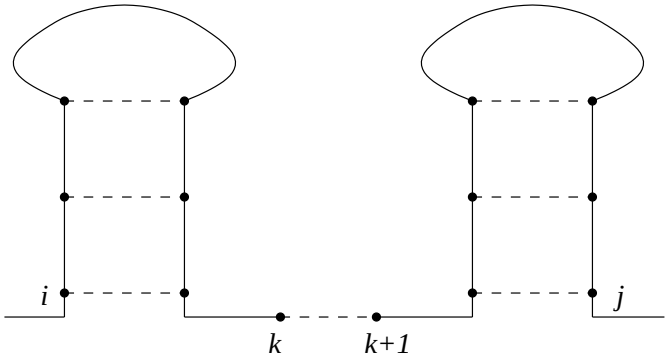


Figure 4.26: bifurcation

Put together, these give the recurrence in equation (4.9.2).

$$E(i,j) = \min \begin{cases} E(i+1,j) \\ E(i,j-1) \\ E(i+1,j-1) + a(i,j) \\ min_{i<k<j}(E(i,k) + E(k+1,j)) \end{cases} \tag{4.9.2}$$

The Nussinov-Jacobson algorithm works like the Nussinov algorithm, but with the following differences:

- Base pairs are weighted by potential energy, rather than the simple fact that the are a base pair.
- Because we seek to minimize energy, we use min of energy values rather than max of number of base pairs.

4.9.3 For Next Class

Read over the variant of the Nussinov algorithm that uses a Stochastic Context-Free Grammar (the parse tree will show the coiling structure).

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors,

and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.