Computational Molecular Biology

D. Durand

October 25, 2024

Contents

Chapter 1

Sequence Alignment

Pairwise sequence alignment seeks to establish a correspondence between the elements in a pair of sequences that share a common property, such as common ancestry or a common structural or functional role. In computational biology, the sequences under consideration are typically nucleic acid or amino acid polymers. We will consider three variants of the pairwise sequence alignment problem: global alignment, semi-global alignment, and local alignment.

Global alignment is used in cases where we have reason to believe that the sequences are related along their entire length. If, for example, sequences s_1 and s_2 are two independent sequencing runs of the same PCR product, then they should differ only at those positions where there are sequencing errors. In order to find those sequencing errors, we align all of sequence s_1 with all of sequence s_2. Other applications of global alignment include finding mutations in closely related gene or protein sequences and identification of single nucleotide polymorphisms (SNPs).

Semi-global alignment is a variant of global alignment that allows for gaps at the beginning and/or the end of one of the sequences. Semi-global alignment is used in situations where we believe that s_1 and s_2 are related along the entire length of the region where they overlap. For example, if s_1 is a segment of genomic DNA containing a prokaryotic gene and s_2 is the mRNA transcript produced when the gene in s_1 is expressed, every base in s_2 corresponds to a base in s_1, but not the reverse is not true. The bases immediately up- and downstream of the gene appear in the genomic DNA, but not in the transcript. Semi-global alignment "jumps" over those flanking regions in s_1 without exacting a penalty, but forces an alignment along the entire length of s_2.

In contrast, local alignment addresses cases where we only expect to find isolated regions of similarity. One example is alignment of genomic DNA upstream from two co-expressed genes to find conserved regions that may correspond to transcription factor binding sites. Another application is identification of conserved domains[1] in two amino acid sequences

Box 1: Notation for pairwise alignment

Alphabet:

An *alphabet*, denoted by Σ, is a finite, unordered set of symbols; e.g.,

DNA: $\Sigma_D = \{A, C, G, T\}$
RNA: $\Sigma_R = \{A, C, G, U\}$
Amino acids: $\Sigma_{AA} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$

Sequences or Strings:

A *sequence* or *string*, s, is a finite succession of the symbols in Σ.

Σ^* denotes the set of all sequences over alphabet Σ, including the empty sequence, $\varnothing$. For example, $\Sigma_R^* = \{\varnothing, A, C, G, U, AA, AC, AG, AU, CA, CC, CG, CU, \ldots\}$.

Given a sequence s of length n, we use $s[1]s[2]\cdots s[n]$ to denote the symbols in s.

Subsequences:

A *subsequence* of s is any sequence obtained by removing zero or more symbols from s. The sequences `CATA` and `CTG` are subsequences of `CATTAG`. `AATTCG` is not.

A *proper subsequence* is a subsequence obtained by removing one or more symbols from s.

Substrings:

A *substring* of s is a subsequence of s consisting of consecutive symbols in s. Given a sequence, s, of length n, the substring that begins with $s[i]$ and ends with $s[j]$ is denoted $s[i..j]$, $1 \leq i \leq j \leq n$. The sequence `CAT` is a substring of `CATTAG`. `CATA` is not.

A *prefix* of s is denoted $s[1..j]$, $j \leq n$.

A *suffix* of s is denoted $s[i..n]$, $1 \leq i$.

that encode proteins that share one or more domains, but are otherwise unrelated.

Prior to introducing algorithms for these pairwise alignment problems, we introduce some notation in Box 1. In Section 1.1, we introduce a formal definition of a global alignment. There are many ways to align a given pair of sequences. We consider scoring functions that assess the quality of an alignment as a basis for quantitative comparison of different alignments. Finally, we provide an efficient algorithm to find the alignment that is optimal with respect to the scoring function. Local and semi-global alignments are discussed in Sections 1.3 and 1.2.

1.1 Global pairwise alignment

Given a pair of sequences, s_1 and s_2, the goal of global sequence alignment is to insert gaps in s_1 and s_2 such that residues that share a property of interest are found in the same column. Suppose that $s_1 = \texttt{CATCAC}$ and $s_2 = \texttt{CTAGC}$ are sequences of lengths 6 and 5, respectively. By introducing a gap after the first symbol in s_2, we obtain an alignment, $\alpha(s_1, s_2)$, of length $l = 6$ with three matches, two mismatches, and a gap:

```
CATCAC
C-TAGC
```

This is just one of many ways to align these sequences. A different alignment can be obtained by introducing one gap in s_1 and two gaps in s_2:

```
CATCA-C
C-T-AGC
```

This alignment is longer ($l = 7$) and has four matches, three gaps and no mismatches. The second alignment implies that the `A`'s in s_1 and s_2 share a relationship; the first alignment does not.

Stacking the sequences vertically provides a visually intuitive representation that places related residues in the same column. The same information can be represented in a more compact form by simply indicating the positions of the gaps in the two sequences. For example, the first alignment can also be represented as $\alpha(s_1, s_2) = \{s'_1, s'_2\}$, where $s'_1 = \texttt{CATCAC}$ and $s'_2 = \texttt{C-TAGC}$. Note that although s_1 and s_2 have different lengths, the alignment $\alpha(s_1, s_2)$, s'_1, and s'_2 are all of the same length ($l = 6$).

Because there are many different alignments of the same pair of sequences, we use superscripts to distinguish between them when more than one alignment is under consideration. Using this notation, the two alignments we introduced above are $\alpha^1(s_1, s_2) = \{s^1_1, s^1_2\}$,

[1] A domain is a peptide sequence that encodes a protein module that will fold into its characteristic shape independent of the surrounding amino acid context and that is found in many different proteins.

where $s_1^1 =$ `CATCAC` and $s_2^1 =$ `C-TAGC` and $\alpha^2(s_1, s_2) = \{s_1^2, s_2^2\}$, where $s_1^2 =$ `CATCA-C` and $s_2^2 =$ `C-T-AGC`.

These ideas are the basis for the formal definition of a global sequence alignment. Given sequence $s_1 \in \Sigma^*$ of length n_1 and sequence $s_2 \in \Sigma^*$ of length n_2, $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$ is a global alignment of s_1 and s_2 if and only if

- $s_1^\kappa, s_2^\kappa \in (\Sigma')^*$, where $\Sigma' = \Sigma \cup \{\text{`-'}\}$ is the alphabet expanded to include the gap symbol;
- $|s_1^\kappa| = |s_2^\kappa| = l^\kappa$, where $\max(n_1, n_2) \leq l^\kappa \leq n_1 + n_2$;
- s_1 is the subsequence obtained by removing '-' from s_1^κ and s_2 is the subsequence obtained by removing '-' from s_2^κ;
- there is no value of i for which $s_1^\kappa[i] = s_2^\kappa[i] =$ '-'.

The length of the alignment is bounded below by $\max(n_1, n_2)$ because the alignment cannot be shorter than the longer of the two sequences. The longest possible alignment is one where every symbol in s_1 is aligned with a gap in s_2 and vice versa. In this case the length of the alignment is $l^\kappa = n_1 + n_2$. A longer alignment is not possible without introducing a site with gaps in both sequences which is a violation of the fourth criterion. The position in the alignment is indexed by columns; column i is often called *site* i. When considering site i, $s_1^\kappa[i]$ and $s_2^\kappa[i]$ are the symbols in s_1 and s_2 that are aligned. If $s_2^\kappa[i] =$ `-`, we say that there is a gap in s_2 at site i.

1.1.1 Global sequence alignment with similarity scoring

Among the many possible ways to align s_1 and s_2, our goal is to find the global alignment that best captures the relationship between them. This is fundamentally a biological question. From a practical perspective, we use a mathematical approach: We introduce an objective criterion that quantifies how well α^κ captures the relationship between s_1 and s_2. We then seek the alignment that optimizes that criterion.

Given sequences s_1 and s_2 and an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$, it is convenient to assign a score to alignment α^κ that quantifies how well α^κ captures the relationship between s_1 and s_2. One approach is to use a scoring function that reflects the similarity between the sequences. In this case, the alignment that maximizes the scoring function is considered to be the optimal alignment. Alternatively, we can introduce a function that quantifies the distance between the two sequences (for example, the number of changes needed to convert one sequence into the other) and seek the alignment that minimizes that distance.

In computational biology, similarity functions are more versatile and more widely used, so we will start with similarity scoring, where a higher score indicates a better alignment.

Given sequences s_1 and s_2 and an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$, the similarity of $\alpha^\kappa(s_1, s_2)$ is defined to be

$$\mathcal{S}(\alpha^\kappa(s_1, s_2)) = \sum_{i=1}^{l^\kappa} S[s_1^\kappa[i], s_2^\kappa[i]], \tag{1.1}$$

where $S[x, y]$ is a score that reflects the similarity of x and y and $S[x, \_]$ is the gap score. The optimal alignment is the alignment that maximizes the similarity between s_1 and s_2:

$$\alpha^*(s_1, s_2) = \underset{\kappa}{\operatorname{argmax}}\, \mathcal{S}\big(\alpha^\kappa(s_1, s_2)\big).$$

There may be more than one optimal alignment.

In general, amino acid alignments are scored with a substitution matrix, S, that assigns a similarity score to each pair of amino acid residues. Broadly speaking, a higher similarity score will be given to a site where the same amino acid is observed in both sequences ($S[x, x]$) than to a site where the amino acids in the two sequences are different ($S[x, y], x \neq y$). However, some amino acid matches will be assigned higher similarity scores than others. When different amino acids are aligned ($x \neq y$), $S[x, y]$ is typically higher when x and y have similar biochemical properties. Examples of amino acid substitution matrices used to score alignments include the PAM and BLOSUM matrices. Substitution matrices are used less often with nucleic acid sequences because the alphabet is smaller (four nucleotides versus 20 amino acids) and the biophysical properties of nucleotides are less varied. In Chapter 3, we will discuss how amino acid substitution matrices are derived.

In this chapter, we will ignore these biological nuances and consider a simple similarity scoring function that treats all symbols in Σ equally. In this simple system, the symbols at given site in an alignment may be the same (a match) or they may differ (a mismatch):

$$S[x, y] = \begin{cases} M & \text{if } x = y, \\ m & \text{if } x \neq y, \end{cases} \tag{1.2}$$

where M is a match score and m is the mismatch score. For this simple scoring function, we require that $M > m$, because matches are preferred over mismatches. The third case of interest arises when a residue is aligned with the gap character ($'\_'$). This case is scored with a gap penalty[2],

$$S[x, \_] = g$$

. The value of g is selected so that a substitution is preferred over two gaps (i.e., $m > 2g$). A scoring function with $m < 2g$ would exclude the possibility of an alignment with

[2]Strictly speaking, the gap score is not an integral part of the substitution matrix. We use this notation here for convenience, as in the sum in Equation 1.1.

substitutions, because the score of any substitution could be improved by replacing it with two gaps.

A final requirement is that the scoring function be symmetric; i.e., $S[x, y] = S[y, x]$ and $S[x, \_] = S[\_, x]$. Given a column with symbols x and y in a pairwise alignment, we have no way of knowing whether the ancestral symbol was an x that was later replaced by a y, or vice versa. (It is also possible that the ancestor was neither x nor y and some combination of substitutions gave rise to x in one sequence and to y in the other.) For this reason, the same score is assigned when x in s_1 is aligned with y in s_2 as when y in s_1 is aligned with x in s_2. Similarly, when a symbol, x, in sequence s_1 is aligned with a gap in sequence s_2 (or vice versa), there is no way to know whether x was inserted in s_1 or deleted from s_2. For this reason, sites with gaps are also called "indels" to indicate that either an insertion or a deletion may have occurred. Our very simple scoring function (Equation 1.2) is, by definition, symmetric because all mismatches have the same score. In Chapter 3, we will see that more complex similarity functions have this property.

Note that the score of an alignment is defined to be the sum of the scores for the individual positions in the alignment (Equations 1.1), which implies that each position in the alignment is independent of neighboring positions. This assumption is unrealistic: In real biomolecular sequences, there can be interactions between neighboring, or even distant, residues in the sequence. However, scoring functions that assume *positional independence* are widely used because they greatly simplify the calculation of alignment scores and other mathematical analyses.

1.1.2 A dynamic programming algorithm to align a pair of sequences

We now have a formal definition of an alignment and a way of assigning a numerical score to any given alignment. How do we find the alignment with the optimal score? We could generate all possible alignments, score each one, and choose the alignment with the best score. However, the computational cost would be prohibitive, since the size of the space of all possible alignments of s_1 and s_2 is $O(2^{n_1+n_2})$. (Convince yourself this is the case.)

Dynamic programming can be used to find the optimal alignment efficiently. This strategy takes advantage of the fact that every prefix of an optimal pairwise alignment is the optimal alignment of a prefix of s_1 and a prefix of s_2. This means that the optimal alignment of pairs of progressively longer prefixes of s_1 and s_2 can be obtained by extending the optimal alignment of shorter prefixes of s_1 and s_2. It is not necessary to examine a suboptimal alignment of prefixes in order to find the optimal alignment of the full length strings.

The dynamic programs for all three sequence alignment problems compute an n_1 x n_2 *alignment matrix* $\mathcal{A}$, where $\mathcal{A}[i, j]$ is the score of the optimal alignment of the prefixes $s_1[1..i]$ and $s_2[1..j]$, that is, the prefixes of s_1 and s_2 that end at positions i and j, respectively.

Dynamic programming algorithms for sequence alignment have four components:

- Initialization of the first row and column of $\mathcal{A}$.
- A recurrence relation that specifies how to calculate the value of $\mathcal{A}[i,j], i>0, j>0$, from the values of neighboring cells.
- Determination of the score of the optimal alignment from the entries in matrix $\mathcal{A}$.
- A procedure to trace back through the matrix to obtain the optimal alignment. This is achieved by storing the information required to construct the optimal alignment in an n_1 x n_2 *traceback matrix*, $\mathcal{T}$.

The calculation of the optimal global alignment of s_1 and s_2 requires two passes through the $\mathcal{A}$ and $\mathcal{T}$ matrices. In the first pass, the dynamic program proceeds from the upper left to the lower right corner of $\mathcal{A}$, populating the cells in $\mathcal{A}$. Whenever a value is assigned to a cell in $\mathcal{A}$, the associated pointer is entered into the corresponding cell in $\mathcal{T}$. In the second pass, the optimal alignment is constructed from the pointers in $\mathcal{T}$. The details of each of these steps are what differentiate global, semi-global, and local alignment.

A formal statement of the dynamic program for global sequence alignment is given below. In the first pass, the dynamic program progresses from $\mathcal{A}[0,0]$ to $\mathcal{A}[n_1,n_2]$, populating the cells in $\mathcal{A}$ with the scores of progressively longer optimal alignments of prefixes. At each iteration, the value of $\mathcal{A}[i,j]$ is calculated from the values of $\mathcal{A}[i-1,j]$, $\mathcal{A}[i-1,j-1]$, and $\mathcal{A}[i,j-1]$.

The initialization step calculates the values in the first row and column of $\mathcal{A}$. The score in $\mathcal{A}[0,j]$ reflects the alignment obtained by inserting gaps at the beginning of s_1 and aligning them to the first j symbols in s_2 . Similarly $\mathcal{A}[i,0]$ is the score of the alignment of i gaps inserted prior to the first symbol in s_2 with the first i symbols in s_1.

The internal entries in $\mathcal{A}$ are calculated by the recurrence relations. The value in $\mathcal{A}[i,j]$ is the score of the optimal alignment of the first i symbols in s_1 (i.e., $s_1[1..i]$) with the first j symbols in s_2 (i.e., $s_2[1..j]$). The optimal alignment of $s_1[1..i]$ and $s_2[1..j]$ is obtained by inserting one additional column at the end of a shorter optimal alignment. There are three optimal alignments of prefixes of s_1 and s_2 that, when extended in this way, will yield a candidate optimal alignment of $s_1[1..i]$ and $s_2[1..j]$.

1. The column $\begin{smallmatrix}-\\ s_2[j]\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i]$ and $s_2[1..j-1]$. The alignment score of this prefix is stored in $\mathcal{A}[i,j-1]$, the cell to the immediate left of $\mathcal{A}[i,j-1]$.
2. The column $\begin{smallmatrix}s_1[i]\\ s_2[j]\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i-1]$ and $s_2[1..j-1]$, which corresponds to $\mathcal{A}[i-1,j-1]$, the cell diagonally up and to the left of $\mathcal{A}[i,j]$.
3. The column $\begin{smallmatrix}s_1[i]\\ -\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i-1]$ and $s[1..j]$, which corresponds to the cell immediately above $\mathcal{A}[i,j]$.

Of these candidates, the alignment that optimizes the scoring function is the optimal alignment of $s_1[1..i]$ and $s_2[1..j]$. The score of this alignment is entered in $\mathcal{A}[i,j]$. The indices of the entry (or entries) in $\mathcal{A}$ that optimize the right hand side of the recurrence relation (Equation 1.3) are stored in the traceback matrix, $\mathcal{T}$. $\mathcal{T}[i,j]$ contains the index of an adjacent cell to the left $(i,j-1)$, upper left $(i-1,j-1)$, and/or above $(i-1,j)$ the current cell. In class, we use arrows ($\leftarrow$,$\nwarrow$, and $\uparrow$) to designate these indices. Note that more than one of the recurrence cases may optimize the value of $\mathcal{A}[i,j]$. In this case, more than one pointer will be added to $\mathcal{T}[i,j]$.

The first pass completes when all entries in the matrix $\mathcal{A}$ have been assigned values. At

DYNAMIC PROGRAM FOR GLOBAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0,j] &= \mathcal{A}[0,j-1]+g \qquad \text{(top row)} \\ \mathcal{A}[i,0] &= \mathcal{A}[i-1,0]+g \qquad \text{(left column)} \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i,j] = \max \begin{cases} \mathcal{A}[i,j-1]+g \\ \mathcal{A}[i-1,j-1]+S[s_1[i],\, s_2[j]] \\ \mathcal{A}[i-1,j]+g \end{cases} \tag{1.3}$$

$$\mathcal{T}[i,j] = \underset{i,j}{\text{argmax}} \begin{cases} \mathcal{A}[i,j-1]+g & \leftarrow \\ \mathcal{A}[i-1,j-1]+S[s_1[i],\, s_2[j]] & \nwarrow \\ \mathcal{A}[i-1,j]+g & \uparrow \end{cases} \tag{1.4}$$

Optimal alignment score:

The score of the optimal alignment is $\mathcal{A}[n_1,n_2]$.

Trace back:

Follow the pointers from $\mathcal{T}[n_1,n_2]$ to $\mathcal{T}[0,0]$ to obtain the optimal alignment.

this point, $\mathcal{A}[n_1, n_2]$ contains the score of the full length optimal alignment, but the actual alignment has not been explicitly determined. In the second pass, the alignment, α, is constructed in reverse order by following the pointers through $\mathcal{T}$ from the lower right corner to the upper left corner. This traceback procedure begins with an alignment consisting of only the last column of the alignment, which is determined from the indices in $\mathcal{T}[n_1, n_2]$, like this:

if $\mathcal{T}[n_1, n_2] = \leftarrow$
  $\alpha = \begin{smallmatrix} - \\ s_2[n_2] \end{smallmatrix}$
  $i = n_1,\ j = n_2 - 1$
if $\mathcal{T}[i, j] = \nwarrow$
  $\alpha = \begin{smallmatrix} s_1[n_1] \\ s_2[n_2] \end{smallmatrix}$
  $i = n_1 - 1,\ ,\ j = n_2 - 1$
if $\mathcal{T}[n_1, n_2] = \uparrow$
  $\alpha = \begin{smallmatrix} s_1[n_1] \\ - \end{smallmatrix}$
  $i = n_1 - 1,\ j = n_2$

The second pass reconstructs the optimal alignment(s) by tracing back though $\mathcal{T}$ from the lower right ($(i = n_1, j = n_2)$ corner to the upper left corner ($i = 0, j = 0$). At each iteration, α is extended by inserting an additional column at the beginning of α, according to the following rules:

if $\mathcal{T}[i, j] = \leftarrow$
  insert $\begin{smallmatrix} - \\ s_2[j] \end{smallmatrix}$ at the beginning of α
  $j = j - 1$
if $\mathcal{T}[i, j] = \nwarrow$
  insert $\begin{smallmatrix} s_1[i] \\ s_2[j] \end{smallmatrix}$ at the beginning of α
  $i = i - 1;\ j = j - 1$
if $\mathcal{T}[i, j] = \uparrow$
  insert $\begin{smallmatrix} s_1[i] \\ - \end{smallmatrix}$ at the beginning of α
  $i = i - 1$

If an entry in $\mathcal{T}$ with more than one pointer is encountered during the traceback, then there is more than one optimal alignment of s_1 and s_2. Multiple passes through $\mathcal{T}$ are required to

generate all optimal alignments, with additional bookkeeping to ensure that each optimal alignment is constructed once and only once.

This two pass alignment algorithm outputs an optimal alignment score, $\mathcal{A}[n_1, n_2]$, and one or more optimal global alignments. In the first pass, the dynamic program computes the scores of all pairs of prefixes in $O(n_1 \cdot n_2)$ time. The trace back through the alignment matrix to obtain the optimal alignment requires $O(n_1 + n_2)$ time for each optimal alignment. Note that with similarity scoring, the entries in $\mathcal{A}$ may be positive or negative; the optimal alignment score may also be positive or negative.

1.1.3 Global sequence alignment with distance scoring

With similarity scoring, we seek the alignment that maximizes the similarity between s_1 and s_2. With distance-based scoring, the optimal alignment is one that minimizes the distance between s_1 and s_2. One advantage of this approach is that distance functions are metrics, in the formal sense, and have nice mathematical properties. The alignment distance also has a concrete interpretation: the minimum alignment distance corresponds to the smallest number of operations required to transform one sequence into another. In computational biology, a major disadvantage of distance scoring is that it can be used for global and semi-global alignment, but not for local alignment. In addition, distance functions penalize mismatches and gaps, but do not reward matches.

We define the distance score of an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$ to be

$$\begin{aligned} D\big(\alpha^\kappa(s_1, s_2)\big) &= D(s_1^\kappa, s_2^\kappa) \\ &= \sum_{i=1}^{l^\kappa} d(s_1^\kappa[i], s_2^\kappa[i]), \end{aligned} \tag{1.5}$$

where $d(x, y)$ is the distance between a pair of symbols x and y in Σ' and l^κ is the length of the alignment. The optimal alignment, denoted α^*, is the alignment that minimizes the distance between s_1 and s_2:

$$\alpha^*(s_1, s_2) = \operatorname*{argmin}_\kappa D\big(\alpha^\kappa(s_1, s_2)\big).$$

The function specifying the distance between pairs of symbols must satisfy the following constraints, for all x, y, and z in Σ':

1. $d(x, x) = 0$
2. $d(x, y) > 0$
3. $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)
4. $d(x, y) = d(y, x)$

The first three constraints guarantee that $D(s_1^\kappa, s_2^\kappa)$ is a metric. In particular, D satisfies the triangle inequality. This means that the penalty for replacing x with y is never greater than the penalty for first replacing x with z and then replacing z with y. When $z =$ '_', this says that deleting x and then inserting y is never an improvement on a direct substitution of x with y. One consequence of the triangle inequality is that the cost of a substitution can never be greater than twice the cost of an indel. The symmetric property, $d(x, y) = d(y, x)$, implies that there is no directionality in the scoring system; we have no information about the order in which events occured. Like similarity scoring, distance scoring assumes positional independence; Equation 1.5 is the sum of the distance scores for the individual positions in the alignment. If $d(x, y) = 1$ and $d(x, \_) = 1$, $\forall x, y$, then

DYNAMIC PROGRAM FOR GLOBAL ALIGNMENT: DISTANCE SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0, j] &= \mathcal{A}[0, j-1] + d(\_, s_2[j]) \qquad \text{(top row)} \\ \mathcal{A}[i, 0] &= \mathcal{A}[i-1, 0] + d(s_1[i], \_) \qquad \text{(left column)} \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i, j] = \min \begin{cases} \mathcal{A}[i, j-1] + d(\_, s_2[j]) \\ \mathcal{A}[i-1, j-1] + d(s_1[i],\, [j]) \\ \mathcal{A}[i-1, j] + d(s_1[i], \_) \end{cases} \tag{1.6}$$

$$\mathcal{T}[i, j] = \underset{i,j}{\operatorname{argmin}} \begin{cases} \mathcal{A}[i, j-1] + d(\_, s_2[j]) & \leftarrow \\ \mathcal{A}[i-1, j-1] + d(s_1[i],\, [j]) & \nwarrow \\ \mathcal{A}[i-1, j] + d(s_1[i], \_) & \uparrow \end{cases} \tag{1.7}$$

Optimal alignment score:

The score of the optimal alignment is $\mathcal{A}[n_1, n_2]$.

Trace back:

Follow the pointers from $\mathcal{T}[n_1, n_2]$ to $\mathcal{T}[0, 0]$ to obtain the optimal alignment.

$D\big(\alpha^*(s_1, s_2)\big)$ corresponds to the minimum number of operations required to transform s_1 into s_2, where the operations are substitution, insertion, and deletion. This is called the *edit distance*. If $d(x, y) \neq 1$ or $d(x, \_) \neq 1$ or both, then $D(\alpha^*(s_1, s_2))$ is called the *weighted edit distance*.

Global alignment with a distance metric uses the same the two pass procedure described above for global alignment with similarity scoring. However, the details of the initialization and recurrence differ. And in contrast to similarity scoring, with distances, all entries in $\mathcal{A}$ are non-negative, since $d(x, y) \geq 0, \forall\, x, y$.

1.2 Local pairwise alignment

Global alignment is used in cases where we expect that s_1 and s_2 are related from end to end. Semi-global allows for some gaps at the beginning and/or end of one sequence, but the underlying assumption is the same: s_1 and s_2 share a relationship within the entire aligned region. In contrast, local alignment is used in cases where s_1 and s_2 share one or more local regions that are related, but are not related from end to end.

The alignment of any substring $s_1[h..i]$ of s_1 and any substring $s_2[j..k]$ of s_2 is a local alignment of s_1 and s_2. The optimal alignment of $s_1[h..i]$ and $s_2[j..k]$ is the highest scoring *global* alignment of those substrings, where $1 \leq h \leq i \leq n_1$, and $1 \leq j \leq k \leq n_2$ and S is the similarity scoring function defined in Equation 1.1. Note that there may be more than one. The optimal *local* alignment of s_1 and s_2 is the highest scoring optimal alignment of all possible substrings of s_1 and s_2, that is,

$$\alpha^*(s_1, s_2) = \underset{h,i,j,k}{\operatorname{argmax}}\, S(\alpha^*(s_1[h..i], s_2[j..k])).$$

Note that there may be more than one optimal local alignment. High scoring sub-optimal alignments may also be of interest.

For local alignment, the pairwise alignment dynamic programming algorithm (see p. 13) must be modified to allow the alignment to start and stop anywhere in s_1 and s_2. Unlike the dynamic program for global alignment, the local alignment recurrence (Equation 1.8) has a fourth term that sets the score $\mathcal{A}[i, j]$ to zero whenever adding a substitution or a gap to the alignment results in a negative score. This is what allows the local alignment algorithm to consider all possible starting positions in s_1 and in s_2.

The value in $\mathcal{A}[i, j]$ is the score of the optimal local alignment of a substring in $s_1[1..i]$ and a substring in $s_2[1..j]$. The score of the optimal local alignment over all possible substrings of s_1 and s_2 is the maximum value in $\mathcal{A}$, taken over all i and all j. The alignment itself is obtained by tracing back through $\mathcal{T}$, starting from the cell corresponding to the maximum score. The alignment is constructed in reverse order by following the pointers through $\mathcal{T}$ until end symbol "$\Diamond$" is encountered; this corresponds to reaching the first

DYNAMIC PROGRAM FOR LOCAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0,j] &= 0,\ \forall_j \\ \mathcal{A}[i,0] &= 0,\ \forall_i \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i,j] = max \begin{cases} \mathcal{A}[i,j-1] + g \\ \mathcal{A}[i-1,j-1] + S[s_1[i],\ s_2[j]] \\ \mathcal{A}[i-1,j] + g \\ 0 \end{cases} \tag{1.8}$$

$$\mathcal{T}[i,j] = \underset{i,j}{\text{argmax}} \begin{cases} \mathcal{A}[i,j-1] + g & \leftarrow \\ \mathcal{A}[i-1,j-1] + S[s_1[i],\ s_2[j]] & \nwarrow \\ \mathcal{A}[i-1,j] + g & \uparrow \\ 0 & \Diamond \end{cases} \tag{1.9}$$

Optimal alignment score:

The score of the optimal alignment is

$$\max_{i,j} \mathcal{A}[i,j].$$

Trace back:

Follow the pointers from $\mathcal{T}[i^*,j^*]$, where $(i^*,j^*) = \text{argmax}_{i,j}\, \mathcal{A}[i,j]$, and end the trace back at the first cell with value zero encountered.

zero-valued cell in $\mathcal{A}$ on the trace back path. This is in contrast to global alignment where the trace back starts in the lower right hand corner and proceeds to position $i = 1, j = 1$.

At each step,

If $\mathcal{T}[i,j] = \leftarrow$
  align $s_2[j]$ with a gap '_'
  $j = j-1$
If $\mathcal{T}[i,j] = \nwarrow$
  align $s_1[i]$ with $s_2[j]$
  $i = i-1$; $j = j-1$
If $\mathcal{T}[i,j] = \uparrow$
  align $s_1[i]$ with a gap '_'
  $i = i-1$
If $\mathcal{T}[i,j] = \Diamond$
  align $s_1[i]$ with $s_2[j]$ and end
  $i = 0$; $j = 0$

The point at which $\mathcal{A}[i,j]$ drops below zero and restarts depends on the scoring function and critically determines what the resulting alignment will look like. For this reason, scoring functions for local alignment are subject to more stringent constraints than scoring functions for global and semi-global alignment.

In order to find biologically meaningful conserved regions, a scoring function for local pairwise alignment must satisfy the following requirements:

- The scoring function must be a similarity function. Edit distance penalizes mismatches and gaps, but does not reward matches and is therefore not appropriate for finding local regions of similarity.
- There must be at least one pair of residues, x and y, for which the similarity score $S[x,y]$ is positive. Without this requirement, all entries in the alignment matrix, $\mathcal{A}$, would be set to zero.
- The expected alignment score of a pair of randomly generated sequences (i.e., sequences sampled from a background distribution) must be negative.
- The standard requirements for the similarity function must be upheld ($m > 2g$).

The rationale for a negative expected alignment score is as follows: The goal of local alignment is to find similar regions in a pair of sequences, $s_1, s_2 \in \Sigma^*$. The optimal local alignment obtained from the dynamic programming algorithm will depend on the function used to score matches and mismatches. We seek a scoring function that will yield local alignments that correspond to biologically meaningful features. However, even

unrelated sequences, when aligned, will match at a few positions. In order to have a strong presumption that a high-scoring local alignment is biologically meaningful, the match and mismatch scores should be chosen in such a way that chance matches contribute little to the alignment score.

If a pair of symbols, sampled at random, were appended to the end of an existing alignment, how much would the alignment score increase? The increase, on average, is given by the expected alignment score per position. Let x be a symbol in Σ and let p_x be the background frequency of x in the genomes from which sequences s_1 and s_2 were sampled. Then, using our simple scoring function, the expected alignment score per position is

$$\begin{aligned}\overline{\mathcal{S}} =& \sum_{x\in\Sigma}\sum_{y\in\Sigma} p_x p_y \; S[x,y] \\ =& \sum_{x\in\Sigma} p_x^2 \; M + \sum_{x\in\Sigma}\sum_{\substack{y\in\Sigma,\\ y\neq x}} p_x p_y \; m,\end{aligned}$$

where M and m are the match and mismatch scores, respectively.

For example, suppose that s_1 and s_2 are DNA sequences with uniform nucleotide frequencies; i.e., $p_A = p_G = p_C = p_T = 0.25$. Since the nucleotide frequencies are uniform, the nucleotide pairs also have uniform frequencies: $p_x p_y = (0.25)^2, \forall x, y$. There are 16 possible pairs of nucleotides; four pairs consist of the same nucleotide (AA, GG, CC, and TT) and 12 pairs are made up of different nucleotides. In this case, the expected alignment score per position is

$$\begin{aligned}\overline{\mathcal{S}} =& 4\;(0.25)^2\; M + 12\;(0.25)^2\; m \\ =& 0.25\; M + 0.75\; m.\end{aligned}$$

Why is it important that the expected alignment score be negative? If the expected score were positive, then extending a local alignment with unrelated pairs of symbols would increase its score. Such an alignment would have a higher score because it is longer, not because it contains stronger evidence of a shared biological relationship. In the nucleotide example given above, M and m should be selected so that

$$0.25\; M + 0.75\; m < 0.$$

The rules above apply to general scoring functions, where the score for each pair of residues can have a different numerical value. For our simple similarity function, we achieve these goals by requiring a positive score for matches ($M > 0$) and a negative score for mismatches ($m < 0$) and gaps ($g < 0$).

1.3 Semi-global alignment

Global alignment seeks the best, full length alignment of a pair of sequences; that is, the best way to match up two sequences along their entire length. For some applications, it is desirable to relax this requirement and not penalize gaps at the beginning and/or end of an alignment. For example, for sequence assembly, we seek sequence fragments that overlap; that is, we expect to be able to align the end of one fragment with the beginning of another. Very occasionally, we may find sequence fragments that start and end at the same position, but, in general, we expect some gaps at the beginning and at the end of the alignment. Another example is aligning cDNAs with genomic DNA to identify gene structure. Because the cDNA corresponds to a small region in the genome, the cDNA fragment will be flanked by gaps at both ends when aligned with the genomic DNA.

Semi-global alignment is a modification of global alignment that allows the user to specify that gaps will be penalty-free at the beginning of one of the sequences and/or at the end of one of the sequences. The optimal semi-global alignment can be found using dynamic programming, with modifications to adapt the dynamic program for global pairwise alignment to the semi-global alignment problem. These modifications are described in terms of alignment with similarity scoring. Similar modifications can be made to obtain a semi-global alignment algorithm that uses distances.

For penalty-free prefixes, the gap penalties in the first column or row are replaced with zeros. To achieve penalty-free gaps at the beginning of s_1, the entries in the first row are set to zero. Allowing penalty-free gaps at the beginning of s_2 requires zeros in the first column. Penalty-free suffixes are achieved by re-defining the optimal alignment score to be the maximum in the last row or column, rather than the value in the cell at the lower right hand corner of the alignment matrix. Considering all cells in the last row allows for penalty-free gaps at the end of s_1 For penalty-free gaps at the end s_2, the algorithm may consider all cells in the last column. As with global alignment, the traceback begins from the cell associated with the optimal score. This means that, unlike global alignment, the traceback can start from a cell in the last row or column, rather than being required to start from the lower right corner.

There may be more than one optimal semi-global alignment. First, two or more cells in the last row (respectively, column) may contain the maximum value. In addition, as with global alignment, for each maximum-valued cell in the last row (respectively, column) there may be more than one path that traces back through in the alignment matrix.

DYNAMIC PROGRAM FOR SEMI-GLOBAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

To allow penalty-free gaps at the beginning of s_1 (as in Case 1), set the first row to zero and initialize the first column as in global alignment.

$$\begin{aligned} \mathcal{A}[0,j] &= 0,\ \forall_j && \text{(top row)} \\ \mathcal{A}[i,0] &= \mathcal{A}[i-1,0]+g && \text{(left column)} \end{aligned}$$

To allow penalty-free gaps at the beginning of s_2 (as in Case 2), set the first column to zero and initialize the first row as in global alignment.

$$\begin{aligned} \mathcal{A}[0,j] &= \mathcal{A}[0,j-1]+g && \text{(top row)} \\ \mathcal{A}[i,0] &= 0,\ \forall_i && \text{(left column)} \end{aligned}$$

Recurrence relation:

Same as global.

Optimal alignment score and trace back:

To avoid trailing gap penalties at the end of s_1 (as in Case 3), we define the optimal score to be the optimal score in the bottom row

$$\max_j \mathcal{A}[n_1,j].$$

Trace back from $\mathcal{T}[n_1,j^*]$, where $j^* = \text{argmax}_j\, \mathcal{A}[n_1,j]$. In other words, trace back from the cell(s) in the last row with optimal score.

To avoid trailing gap penalties at the end of s_2 (as in Case 4), we define the optimal score to be the optimal score in the last column

$$\max_i \mathcal{A}[i,n_2].$$

Trace back from $\mathcal{T}[i^*,n_2]$, where $i^* = \text{argmax}_i\, \mathcal{A}[i,n_2]$. In other words, trace back from the cell(s) in the last column with optimal score.

	0		S		P		E		L		L		B		I		N		D		I		N		G
0	0		0		0		0		0		0		0		0		0		0		0		0		0
		\	|	\	|	\	|	\	|	\	|	\						\	|	\	|	\	|	\	|
B	-1		-1		-1		-1		-1		-1		2	--	1	--	0	--	-1		-1		-1		-1
		\	|	\	|	\	|	\	|	\	|		|	\		\		\		\	|	\	|	\	|
O	-2		-2		-2		-2		-2		-2		1		1	--	0	--	-1	--	-2		-2		-2
		\	|	\	|	\	|	\	|	\	|		|	\	|	\		\		\		\	|	\	|
U	-3		-3		-3		-3		-3		-3		0		0		0	--	-1	--	-2	--	-3		-3
		\	|	\	|	\	|	\	|	\	|		|	\	|	\					\			
N	-4		-4		-4		-4		-4		-4		-1		-1		2	--	1	--	0		0	--	-1
		\	|	\	|	\	|	\	|	\	|		|	\	|		|	\							
D	-5		-5		-5		-5		-5		-5		-2		-2		1		4	--	3	--	2	--	1

Figure 1.1: Semi-global alignment matrix with similarity scoring ($M = 2, m = -1, g = -1$) and penalty-free gaps at the beginning and end of s_1. With this configuration, there are two optimal semi-global alignments. The traceback starts at the cell corresponding to the maximum value in the last row, highlighted in pink. Arrows along the traceback are indicated in red. Zeros in the first row allow the traceback to terminate any position in the top row.

As an example, Figure 1.1 shows the alignment matrix for the semi-global alignment of $s_1 =$ `BOUND` and $s_2 =$ `SPELLBINDING`, with penalty-free gaps at the beginning and end of s_1. Zeros in the top row allow the algorithm to start late in `SPELLBINDING`. This results in five gaps that are not penalized before the first character in s_1. The alignment terminates at the highest-scoring cell in the last row, resulting in three penalty-free gaps after the last character in s_1. In this example, there is a unique maximum in the bottom row.

With this scoring function, there are two optimal semi-global alignments because there are two paths from the maximum value in the last row to the a zero-valued cell in the first row.

```
s_1:             B O U N D
s_2:   S P E L L B I _ N D I N G
```

and

```
s_1:             B O U N D
s_2:   S P E L L B _ I N D I N G
```

Multiple optimal semi-global alignments could also arise if there are two or more maximum-valued cells in the last row, but that is not the case here. To emphasize that indels at the beginning and end of s_1 are penalty-free, we do not show the gap symbols before and after

BOUND. This distinguishes leading and lagging indels from the indel in SPELLBINDING that *is* penalized, because it is in the middle of the alignment.

In the example in Fig. 1.1, the semi-global alignment algorithm was configured to allow gaps to be penalty-free at the beginning and end of s_1. In general, semi-global alignment can be set up to allow penalty-free gaps at the beginning of either sequence. There are eight possible cases to consider:

1. Gaps are penalty-free at the beginning of s_1 ($\mathcal{A}[0, j] = 0, \forall_j$); *e.g.*,

```
s_1:       D O
s_2:   R E D O
```

2. Gaps are penalty-free at the beginning of s_2 ($\mathcal{A}[i, 0] = 0, \forall_i$); *e.g.*,

```
s_1:   R E D O
s_2:       D O
```

3. Gaps are penalty-free at the end of s_1 (opt $= \max_j \mathcal{A}[n_1, j]$); *e.g.*,

```
s_1:   D O
s_2:   D O N E
```

4. Gaps are penalty-free at the end of s_2 (opt $= \max_i \mathcal{A}[i, n_2]$); *e.g.*,; *e.g.*,

```
s_1:   D O N E
s_2:   D O
```

5. Gaps are penalty-free at the beginning and end of s_1; ($\mathcal{A}[0, j] = 0, \forall_j$ and opt $= \max_j \mathcal{A}[n_1, j]$.); *e.g.*,

```
s_1:       D O
s_2:   R E D O N E
```

6. Gaps are penalty-free at the beginning and end of s_2; ($\mathcal{A}[i, 0] = 0, \forall_i$ and opt $= \max_i \mathcal{A}[i, n_2]$.); *e.g.*,

```
s_1:   R E D O N E
s_2:       D O
```

7. Gaps are penalty-free at the beginning of s_1 and at the end of s_2; ($\mathcal{A}[0, j] = 0, \forall_j$ and opt $= \max_i \mathcal{A}[i, n_2]$.); *e.g.*,

```
s_1:       D O N E
s_2:   R E D O
```


8. Gaps are penalty-free at the beginning of s_2 and at the end of s_1; ($\mathcal{A}[i,0] = 0, \forall_i$ and opt $= \max_j \mathcal{A}[n_1, j]$.); *e.g.*,

```
s_1:   R E D O
s_2:       D O N E
```

In semi-global alignment, we do not allow penalty-free gaps at the beginning of s_1 *and* the beginning of s_2 in the same alignment. Nor do we allow penalty-free gaps at the end of s_1 and the end of s_2. Why not?

Chapter 2

Sequence Evolution Models

In the previous lectures, we introduced two simple scoring functions for pairwise alignments:

- a similarity function, that assigns a score of M to matches ($M > 0$), m to mismatches ($m < 0$), and g to indels ($g < 0$) and
- an edit distance, which does not reward matches ($M = 0$) and assigns a unit cost to mismatches and gaps ($m, g > 0$).

These scoring functions allow us to compare two alignments by comparing their scores, but are less useful for assessing a pairwise alignment in an absolute sense. Given a pair of aligned sequences with a particular collection of matches, mismatches, and indels, does the alignment reflect enough similarity to suggest that it is of biological interest? One way of assessing an alignment in an absolute sense is to determine whether it reflects more similarity than we would expect by chance. In developing this approach, we must take into account the divergence of related sequences due to mutation. With that in mind, we will explore models of sequence evolution and then discuss how they are used to assess alignments. Sequence evolution models are typically based on Markov chains, so we will begin with a general introduction to Markov models.

Box 2 includes a summary of some notation that will be used throughout this chapter to describe Markov chains

2.1 Finite discrete Markov chains

In various computational biology applications, it is useful to track the stochastic variation of a random variable. Here are some examples:

1. **Time-dependent system:** For models of sequence evolving by substitution, the random variable of interest is the nucleotide (or amino acid) observed at a fixed

position, or *site*, in the sequence at time t. The goal is to characterize how this random variable changes over time.

2. **Space-dependent system:** It is also useful to consider how the residues in a sequence change as one moves along the sequence from one site to the next. In this case, the random variable is the amino acid (or nucleotide) at site i. We are interested in how the probability of observing a given amino acid at site i depends on the amino acid observed at site $i-1$.

For each of these examples, we can model how the value of the random variable (the nucleotide or amino acid) changes with respect to an independent variable (time or position), using a *Markov* chain with a finite number of *states*, $E_1, E_2, \ldots E_s$. Each state corresponds to one of the possible values of the random variable:

- In a nucleic acid sequence, there are four states, each corresponding to the event of observing one of the four nucleotides at the site of interest, e.g., $E_1 = A$, $E_2 = G$, $E_3 = C$, $E_4 = T$.
- In a protein sequence, there are 20 states, each of which corresponds to the event of observing a given amino acid; for example $E_1 = \text{Ala}$, $E_2 = \text{Cys}$, $\ldots$ $E_{20} = \text{Tyr}$.

In our examples above, the states are defined as follows:

1. In a time-dependent system, we say the system is in state E_x at time t.
2. In a spatially varying system, we say the system is in state E_i at site i without concerning ourselves with time. This is in contrast to the previous example, where time varies and the position, i, is held fixed.

The probability that a Markov chain is in state E_x at time t is designated[1] $\varphi_x(t)$. Consider the example of modelling the evolution of a given nucleotide site over time. In this example, $\varphi_1(t)$ is the probability of observing an A at site i at time t. The vector $\varphi(t) = (\varphi_1(t), \varphi_2(t), \ldots \varphi_s(t))$ describes the *state probability distribution* over all states at time t. The *initial* state probability distribution is given by $\varphi(0)$. Note that Ewens and Grant[2] use π to denote the initial state distribution: $\pi = (\varphi_1(0), \varphi_2(0), \ldots \varphi_s(0))$.

In order to capture the stochastic variation of the system, we must also define the probability of making a transition from one state to another. The *transition probability*, P_{xy}, is $\Pr(E_y \text{ at } t+1 \mid E_x \text{ at } t)$, the probability that the chain will be in state E_y at time $t+1$, given that it was in state E_x at the previous time step, t. In the time-dependent, nucleotide sequence example, P_{12} is the probability of an A-to-G substitution at site i.

[1] To simplify the exposition, we will focus on models where time is the independent variable. However, the framework is more general, and can be used to model variation with respect to other independent variables, such as the position in a sequence.

[2] Statistical Methods in Bioinformatics: An Introduction. W. Ewens, G. Grant. Springer 2001.

P is an $s \times s$ matrix specifying the probability of making a transition from any state to any other state. The rows of this matrix sum to one ($\sum_y P_{xy} = 1$) since the chain must be in some state at every time step. The columns do not have to add up to one, since there is no guarantee that the system will end up in a particular state, y.

The *Markov property* states that Markov chains are memoryless. In other words, the probability that the chain is in state E_x at time $t+1$, depends only on the state at time t and not on the past history of the states visited at times $t-1$, $t-2$...

In this course, we will focus on discrete, finite, time-homogeneous Markov chains. These are models with a *finite* number of states, in which time (or space) is split into *discrete* steps. The assumption of discrete steps is quite natural for a spatially varying system, because sequences of symbols are inherently discrete, but somewhat artificial for the sequence evolution over time model, since time is continuous. In Section 2.4, we will derive two continuous DNA Markov models from their discrete counterparts discussed in Section 2.3. Our models are *time-homogeneous*, because the transition matrix does not change over time.

Box 2: Summary of Markov chain notation

A Markov chain has *states* $E_1, \ldots, E_s$ corresponding to the range of the associated random variable.

$\varphi_x(t)$ is the probability that the chain is in state E_x at time t. The vector $\varphi(t) = (\varphi_1(t), \ldots \varphi_s(t))$ is the *state probability distribution* at time t.

$\pi = \varphi(0)$ is the *initial state probability distribution.*

P is the *transition probability matrix.* P_{xy} gives the probability of making a transition to state E_y at time $t+1$, given that the chain was in state E_x at time t. The rows of this matrix sum to one: $\sum_y P_{xy} = 1$.

The state probability distribution at time $t+1$ is given by $\varphi(t+1) = \varphi(t) \cdot P$. The probability of being in state E_y at $t+1$ is

$$\varphi_y(t+1) = \sum_x \varphi_x(t) P_{xy}$$

The *Markov property* states that Markov chains are memoryless. The probability that the chain is in state E_y at time $t+1$, depends only on $\varphi(t)$ and is independent of $\varphi(t-1), \varphi(t-2), \varphi(t-3) \ldots$

2.1.1 Higher Order Markov Chains

The memoryless requirement that the probability of occupying state E_x at time $t+1$ depends only on $\varphi(t)$ can sometimes be relaxed to allow for a more general Markov process. Markov chains that uphold the memoryless property are often called *first order* Markov chains. Higher-order dependencies can also be modelled. For example, consider the case where the nucleotide at time t depends not only on the nucleotide observed at time $t-1$ but also the nucleotide observed at time $t-2$. This case can be modeled with a *second order* Markov chain because the nucleotide at time t depends on the previous two time points. The transition matrix P would be of size 16×4; columns would represent the nucleotide at time t and rows would represent all combinations of the two preceeding nucleotides. More generally, an *n-th order* Markov chain is a system where the probability of a given state at time t depends on the previous n sites. The transition matrix P would be of size $s^n \times s$.

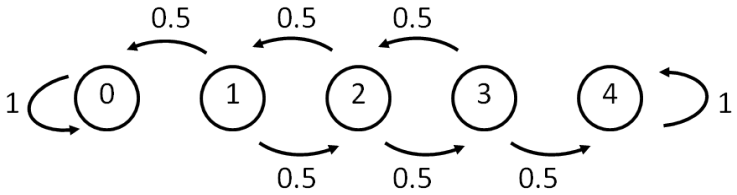
2.2 Random walks

To illustrate the concepts of Markov chains, let us consider a simple example: A drunk is staggering about on a very short railway track with five ties on top of a mesa (a high hill with a flat top and steep sides.) Here, state E_x corresponds to the event that the drunk is standing on the x^{th} tie, where $0 \leq x \leq 4$. At each time step, the drunk staggers either to the left or to the right with equal probability. If the drunk reaches either end of the track (either the 0^{th} or the 4^{th} tie), he falls off the mesa. This model is called a *random walk with absorbing boundaries*, because once the drunk falls off the mesa, he can never get back on the railroad track. States E_0 and E_4 are *absorbing* states. Once the system enters one of these states, it remains in that state forever, since $P_{00} = P_{44} = 1$. This results in the following transition probability matrix:

$$P = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 1 & 0 & 0 & 0 & 0 \\ E_1 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ E_4 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Note that each row sums to one, consistent with the definition of a Markov chain.

The transition matrix of a Markov chain can be represented as a graph, where the nodes represent states and the edges represent transitions with non-zero probability. For example, the random walk with absorbing boundaries can be modeled like this:



Note that the sum of the weights on all outgoing edges from any given state sum to 1, just as row sums are equal to 1 in the transition matrix.

How does the state probability distribution change over time? If we know the state probability distribution at time t, the distribution at the next time step is given by:

$$\varphi_y(t+1) = \sum_x \varphi_x(t) P_{xy} \tag{2.2}$$

or

$$\varphi(t+1) = \varphi(t)\ P \tag{2.3}$$

in matrix notation.

For example, suppose that at time $t = 0$, the drunk is standing on the middle tie (state E_2); that is, $\pi = \varphi(0) = (0, 0, 1, 0, 0)$. To obtain the state probability distribution after one time step, we apply Equation 2.2:

$$\varphi_y(1) = \sum_{x=0}^{4} \varphi_x(0) P_{xy}.$$

Thus, the probability of being in state E_1 when $t = 1$ is given by

$$\begin{aligned}
\varphi_1(1) &= \sum_{x=0}^{4} \varphi_x(0) P_{x1}. \\
&= 0 \cdot 0 + 0 \cdot 0 + 1 \cdot \frac{1}{2} + 0 \cdot 0 + \cdot 0 \cdot 0 \\
&= \frac{1}{2}.
\end{aligned}$$

Note this is equivalent to multiplying the vector $(0, 0, 1, 0, 0)$ by the second column of the transition matrix in Equation 2.1.

Since this Markov chain is symmetrical, it is easy to show that $\varphi_3(1)$ is also equal to $1/2$. (Try it.) It is not possible to reach state E_0 or state E_4 in a single step from state E_2, so $\varphi_0(1) = \varphi_4(1) = 0$. Nor is it possible to remain in state E_2 for two consecutive time steps since $P_{22} = 0$, so $\varphi_2(1) = 0$. Since state E_2 is the only state with non-zero probability at time $t = 0$, we obtain,

$$\varphi(1) = (0, \frac{1}{2}, 0, \frac{1}{2}, 0).$$

Now that we have the probability distribution at time $t = 1$, we can calculate the probability distribution at time $t = 2$ using the same procedure

$$\varphi_y(2) = \sum_{x=0}^{4} \varphi_x(1) P_{xy}.$$

The probability of being in state E_0 at $t = 2$ is given by

$$\begin{aligned}\varphi_0(2) &= \sum_{x=0}^{4} \varphi_x(1) P_{x0} \\ &= 0 \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} + 0 \cdot 0 + \frac{1}{2} \cdot 0 + 0 \cdot 0 \\ &= \frac{1}{4}.\end{aligned}$$

Again, because the matrix is symmetrical, $\varphi_4(2) = \varphi_0(2)$. The probability of being in state E_2 is

$$\begin{aligned}\varphi_2(2) &= \sum_{x=0}^{4} \varphi_x(1) P_{x2} \\ &= 0 \cdot 0 + \frac{1}{2} \cdot \frac{1}{2} + 0 \cdot 0 + \frac{1}{2} \cdot \frac{1}{2} + 0 \cdot 0 \\ &= \frac{1}{2}.\end{aligned}$$

It is not possible to reach state E_1 in a single step from state E_3, or vice versa; nor is it possible to remain in state E_1 or E_3 for two consecutive time steps since $P_{11} = 0$ and $P_{33} = 0$. Thus the probabilities of being in state E_1 or E_3 at time $t = 2$ are zero.

The probability distribution vector at time $t = 2$ is, therefore,

$$\varphi(2) = (\frac{1}{4}, 0, \frac{1}{2}, 0, \frac{1}{4}). \tag{2.4}$$

2.2.1 Calculating "n-step" Transition Probabilities

Suppose that we wish to know the state of the system after two time steps. In the previous section, we used Equation 2.2 to calculate $\varphi(1)$, given $\pi = \varphi(0)$, and then we applied

Equation 2.2 again to calculate $\varphi(2)$, from $\varphi(1)$. We can approach this from another linear algebra perspective by constructing a *two-step* transition probability matrix in which each time step corresponds to two time steps in the original Markov chain.

Here, we derive a general expression for $\varphi(t+2)$ in terms of $\varphi(t)$ and P^2. From Equation 2.2, we obtain

$$\varphi_l(t+1) = \sum_{x=0}^{s} \varphi_x(t) P_{xl} \tag{2.5}$$

and

$$\varphi_y(t+2) = \sum_{l=0}^{s} \varphi_l(t+1) P_{ly}. \tag{2.6}$$

Substituting the right hand side of Equation 2.5 for $\varphi_l(t+1)$ in Equation 2.6 yields

$$\varphi_y(t+2) = \sum_{l=0}^{s} \left(\sum_{j=0}^{s} \varphi_x(t) P_{xl} \right) P_{ly}.$$

We can reverse the order of the summations since the terms may be added in any order:

$$\varphi_y(t+2) = \sum_{x=0}^{s} \left(\sum_{l=0}^{s} \varphi_x(t) P_{xl} \right) P_{ly}.$$

Since $\varphi_x(t)$ does not depend on l, it can be moved out of the summation over l, yielding:

$$\varphi_y(t+2) = \sum_{x=0}^{s} \varphi_x(t) \left(\sum_{l=0}^{s} P_{xl} P_{ly} \right). \tag{2.7}$$

The term in the inner summation is simply the element in row x and column y of the matrix obtained by multiplying matrix P by itself. In other words,

$$\sum_{l=0}^{s} P_{xl} P_{ly} = P^2_{xy},$$

where $P^2 = PP$, so that Equation 2.7 may be rewritten as

$$\varphi_y(t+2) = \sum_{x=0}^{s} \varphi_x(t) P^2_{xy}.$$

Matrix P^2 is the transition matrix for moving two time steps over the Markov chain described by P. In other words, a single time step in P^2 is equivalent to two time steps

in P. Similarly, the *n-step* transition probability matrix, P^n, models change after n time steps such that:

$$P^n = \underbrace{PP \cdots P}_{n \text{ times}}.$$

The n-step equivalent of Equation 2.3 is

$$\varphi(t+n) = \varphi(t) \cdot P^n.$$

As an example, let's apply this approach to our 5-state random walk with absorbing boundaries. Recall that the transition matrix for the random walk, given in Equation 2.1, is

$$P = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 1 & 0 & 0 & 0 & 0 \\ E_1 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ E_4 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying P times itself yields the two-step transition matrix, P^2:

$$P^2 = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 1 & 0 & 0 & 0 & 0 \\ E_1 & \frac{1}{2} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ E_2 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_3 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} \\ E_4 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

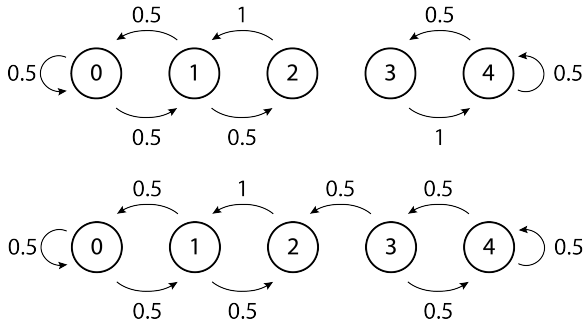
Try this matrix multiplication to convince yourself that this is correct. The state probability distribution at $t = 2$ can be calculated by applying P^2 to π:

$$\begin{aligned} \varphi(2) &= \pi P^2 \\ &= (0, 0, 1, 0, 0)P^2 \\ &= (\frac{1}{4}, 0, \frac{1}{2}, 0, \frac{1}{4}). \end{aligned}$$

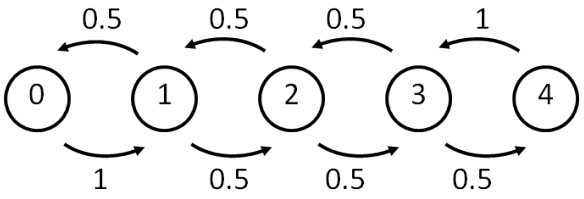
Note that this gives the same result as Equation 2.4, which we got by applying the original Markov chain twice.

2.2.2 Irreducible Markov chains

In our random walk with absorbing states, we will not eventually reach every state from every other state — we cannot travel from state E_0 to any other state, no matter how much time has passed. Markov chains where every state is connected to every other state via a series of zero or more states are called *irreducible*. Markov chains with absorbing states are never irreducible. Two other examples of random walk Markov chains that are not irreducible are:



Let us consider a different example, that is irreducible. In order to save the drunk from an early death, we introduce a random walk with *reflecting* boundaries. At each step, the drunk moves to the left or to the right with equal probability. When the drunk reaches one of the boundary states (E_0 or E_4), he returns to the adjacent state (E_1 or E_3) at the next step, with probability one — hence, E_0 and E_4 are reflecting boundaries.



Notice that we can now get to every state, including E_0, from E_0 in one, two, three, or four steps. Convince yourself that every state can reach every other state in a finite number of steps; thus, this Markov chain is irreducible.

We can see this in the n-step transition probability matrices. This Markov chain yields

the following matrix:

$$P = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 0 & 1 & 0 & 0 & 0 \\ E_1 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ E_4 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.9}$$

Notice that E_0 can reach E_1, but no other state. Calculating the two-step transition matrix, P^2 yields:

$$P^2 = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ E_1 & 0 & \frac{3}{4} & 0 & \frac{1}{4} & 0 \\ E_2 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_3 & 0 & \frac{1}{4} & 0 & \frac{3}{4} & 0 \\ E_4 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \tag{2.10}$$

Notice that this matrix, while similar to the two-step matrix in 2.8, is not the same. What entries are different and why does this make sense given the changes in the Markov chain?

This shows that state E_0 can reach itself and state E_2, in two steps. What about states E_3 and E_4? The three- and four-step transition matrices, P^3 and P^4 are:

$$P^3 = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 0 & \frac{3}{4} & 0 & \frac{1}{4} & 0 \\ E_1 & \frac{3}{8} & 0 & \frac{1}{2} & 0 & \frac{1}{8} \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & \frac{1}{8} & 0 & \frac{1}{2} & 0 & \frac{3}{8} \\ E_4 & 0 & \frac{1}{4} & 0 & \frac{3}{4} & 0 \end{bmatrix} P^4 = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & \frac{3}{8} & 0 & \frac{1}{2} & 0 & \frac{1}{8} \\ E_1 & 0 & \frac{5}{8} & 0 & \frac{3}{8} & 0 \\ E_2 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_3 & 0 & \frac{3}{8} & 0 & \frac{5}{8} & 0 \\ E_4 & \frac{1}{8} & 0 & \frac{1}{2} & 0 & \frac{3}{8} \end{bmatrix} \tag{2.11}$$

We can see that state E_0 can reach E_3 in three steps and E_4 and four steps. If we look over all four matrices, we see that every entry has a non-zero entry in at least one of these matrices, proving the Markov chain is irreducible.

2.2.3 Periodic Markov chains

Comparing the original transition matrix with the two-, three-, and four-step transition matrices, we see a pattern begin to emerge. In fact, once n reaches a high enough value,

this pattern becomes clearer. When n is even, the n-step transition matrix will be:

$$\begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_1 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_2 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_3 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_4 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} \tag{2.12}$$

And, when n is odd, the n-step transition matrix will be:

$$\begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_1 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & \frac{1}{4} & 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ E_4 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{bmatrix} \tag{2.13}$$

Thus, this random walk with reflecting boundaries is a periodic Markov chain.

A Markov chain is *periodic* if there is some state that can only be visited, with any probability greater than 0, in multiples of m time steps, where $m > 1$. In other words, state E_x has period m if it can only be visited every m time steps. Formally, state E_x has period

$$m = \gcd\{n > 0 : P_{xx}^n > 0\},$$

where "gcd" is the greatest common divisor. In our example, $P_{xx}^n > 0$ for $n = 2, 4, 6, 8, ...$ for all x; therefore, each state has a period of 2, which is the gcd of $\{2, 4, 6, 8, ...\}$. (Note, the values of P_{xx}^n do not have to be the same; they simply have to be non-zero.) If $m = 1$, the state is aperiodic. Note that the definition of periodicity does not depend on the initial state distribution π. If any state in the Markov chain is aperiodic, then the Markov chain is aperiodic.

To show that an irreducible Markov chain is aperiodic, it is sufficient to show one of the following:

1. Any state has a self-loop in P, the original (1-step) transition matrix; i.e., $P_{xx} > 0$ for some state x.
2. All elements of the n-step transition matrix P^n are greater than 0 for some positive integer n.
3. If $P_{xx}^k > 0$ and $P_{xx}^l > 0$, and the $\gcd(k, l) = 1$.
4. Any state in the Markov chain is aperiodic.

We do not require periodic Markov chains for modeling sequence evolution and will only consider aperiodic Markov chains going forward.

2.2.4 Stationary distributions

A state probability distribution, φ^*, that satisfies the equation

$$\varphi^* = \varphi^* P \tag{2.14}$$

is called a *stationary* distribution. This is equivalent to satisfying the expression

$$\sum_x \varphi^*_x P_{xy} = \varphi^*_y$$

for all E_y.

A key question for a given Markov chain is whether such a stationary distribution exists. Intuitively, we know that the stationary state can be found by examining the n-step transition matrix as $n \to \infty$

$$Q = \lim_{n\to\infty} P^n$$

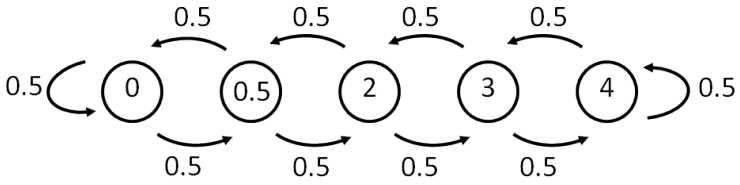
For some Markov chains, Q may not exist, even though a stationary distribution does. The random walk with reflecting boundaries does not have a solution for $\lim_{n\to\infty} P^n$, but does has a stationary distribution of $(\frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8})$. Prove this to yourself by multiplying the stationary distribution and the transition matrix.

Another approach is to recognize that Equation 2.14 is equivalent to a system of s equations with s unknowns. One way to determine the stationary distribution is to solve that system of equations. The stationary distribution can also be obtained using linear algebra, but that approach is beyond the scope of this course.

The random walk with absorbing boundaries does not have a unique stationary distribution; both $(1, 0, 0, 0, 0)$ and $(0, 0, 0, 0, 1)$ are stationary distributions of the random walk with absorbing boundaries.

For the rest of this course, we will concern ourselves only with irreducible, aperiodic Markov chains that do not have absorbing states. If a finite Markov chain is aperiodic and irreducible, it has a *unique* stationary distribution. We will not attempt to prove this or even to state the theorem in a rigorous way. For those who are interested, a very nice treatment can be found in Chapter 15 of *Probability Theory and its Applications (Volume I)* by William Feller (John Wiley & Sons).

As an example of an irreducible, aperiodic Markov chain, we introduce a third random walk model that has neither absorbing, nor reflecting boundaries. In this model, if the drunk is in one of the boundary states (E_0 or E_4) at time t, then at time $t + 1$ he remains in the boundary state with a probability of 0.5 or returns to the adjacent state (E_1 or E_3) with a probability of 0.5.



This results in the following state transition matrix:

$$P = \begin{bmatrix} & E_0 & E_1 & E_2 & E_3 & E_4 \\ E_0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ E_1 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ E_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ E_3 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ E_4 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{2.15}$$

Yet again, the weights on outgoing edges sum to 1 for every state.

We can determine the stationary state distribution for this random walk model by substituting this transition matrix into Equation 2.14. The probability of being in state E_0 is

$$\begin{aligned} \varphi_0^* &= \sum_{x=0}^{4} \varphi_x^* P_{x0} \\ &= \varphi_0^* P_{00} + \varphi_1^* P_{10} + \varphi_2^* P_{20} + \varphi_3^* P_{30} + \varphi_4^* P_{40}. \end{aligned}$$

This reduces to

$$\varphi_0^* = \frac{1}{2}\varphi_0^* + \frac{1}{2}\varphi_1^*, \tag{2.16}$$

since P_{20}, P_{30} and P_{40} are all equal to zero. The other stationary distribution state probabilities are derived similarly, yielding

$$\varphi_1^* = \frac{1}{2}\varphi_0^* + \frac{1}{2}\varphi_2^* \tag{2.17}$$

$$\varphi_2^* = \frac{1}{2}\varphi_1^* + \frac{1}{2}\varphi_3^* \tag{2.18}$$

$$\varphi_3^* = \frac{1}{2}\varphi_2^* + \frac{1}{2}\varphi_4^* \tag{2.19}$$

$$\varphi_4^* = \frac{1}{2}\varphi_3^* + \frac{1}{2}\varphi_4^*. \tag{2.20}$$

In addition, the stationary distribution probabilities must sum to 1, since at any point in time, the drunk must be *somewhere*. This imposes an additional constraint:

$$\varphi_0^* + \varphi_1^* + \varphi_2^* + \varphi_3^* + \varphi_4^* = 1. \tag{2.21}$$

The Markov model specified by Equation 2.15 has a stationary distribution if the above equations have a solution. By repeated substitution, it is possible to show that Equations 2.16 - 2.20 reduce to $\varphi_0^* = \varphi_1^* = \varphi_2^* = \varphi_3^* = \varphi_4^*$. (Do the algebra to convince yourself that this is true.) Applying the constraint in Equation 2.21, we see that $\varphi^* = (0.2, 0.2, 0.2, 0.2, 0.2)$ is a unique solution to the above equations.

In this example, we found a unique solution to Equation 2.21, demonstrating that our third random walk has a unique stationary state. Solving Equation 2.14 is a general approach to finding the stationary distribution. Alternatively, if we know the stationary state distribution, or have an educated guess, it is sufficient to verify that it indeed satisfies Equation 2.14. For example, it is easy to verify that $(0.2, 0.2, 0.2, 0.2, 0.2) \cdot P = (0.2, 0.2, 0.2, 0.2, 0.2)$.

A stationary distribution of $\varphi^* = (0.2, 0.2, 0.2, 0.2, 0.2)$ does not mean that we expect to find 20% of a drunk standing on each railroad tie. Imagine instead that there are an infinite number of co-existing universes and that in each universe, we have a mesa with a railroad track with five ties and a drunk. These drunks are lurching back and forth according to the same Markov model, but they are not synchronized; at any given time point, some of the drunks will be on the 2nd tie, other drunks will be on the 4th tie, and so on. At steady state, for every x, $0 \leq x \leq 4$, 20% of the parallel universes will have a drunk on the x^{th} railroad tie.

Limiting distributions and stationary distributions

If a Markov chain is finite, irreducible, and aperiodic, then it has a *limiting distribution* and the chain will converge to the stationary distribution φ^*, independent of the starting distribution π. Formally

$$\lim_{n \to \infty} P_{xy}^n = \varphi_y^*.$$

In other words, as $n \to \infty$ the n-step transition matrix will be

$$P^n = \begin{bmatrix} & E_1 & \cdots & E_y & \cdots & E_s \\ E_1 & \varphi_1^* & \cdots & \varphi_y^* & \cdots & \varphi_s^* \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ E_x & \varphi_1^* & \cdots & \varphi_y^* & \cdots & \varphi_s^* \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ E_s & \varphi_1^* & \cdots & \varphi_y^* & \cdots & \varphi_s^* \end{bmatrix}. \tag{2.22}$$

2.2.5 Time reversibility

Most sequence substitution models are *time reversible*. A time reversible model exhibits the same steady state behavior if we run it backward, instead of forward. This is a convenient property that makes many calculations simpler. Most, if not all, of the Markov models we encounter in this course are time reversible. However, when analyzing a data set involving genomes with very different G+C content, a time reversible model of sequence evolution may not provide accurate results. It is therefore helpful to understand the concept of time reversibility and be aware of whether or not the models you are using have this property.

Formally, a Markov chain is *time reversible* if

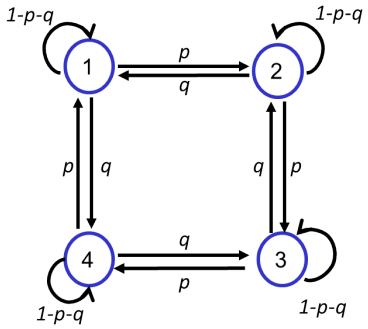
$$\varphi_x^* P_{xy} = \varphi_y^* P_{yx}$$

for all states x and y. This criterion is called the detailed balance equation. Earlier, we introduced parallel universes as a metaphor for the stationary state probability distribution, φ^*. This metaphor is also helpful in understanding time reversibility: If a Markov chain satisfies the detailed balance equation, then the number of universes that are moving from E_x to E_y is equivalent to the number of universes that are moving from E_y to E_x.

Kolmogorov proposed an alternate criterion for time reversibility that depends only on the transition probability matrix. Let $\mathcal{M}$ be a Markov chain with a unique stationary distribution and let $x_1, \ldots, x_n$ be a sequence of states (a path of length n) through $\mathcal{M}$. Then, $\mathcal{M}$ is time reversible if and only if

$$P_{x_1,x_2}\, P_{x_2,x_3}\, \ldots P_{x_{(n-1)},x_n}\, P_{x_n,x_1} = P_{x_1,x_n}\, P_{x_n,x_{(n-1)}}\, \ldots P_{x_3,x_2}\, P_{x_2,x_1}. \tag{2.23}$$

Time reversibility and the use of Kolmogorov's criteria are illustrated by the Markov model in the figure below. The four transitions associated with a clockwise circuit have



probability p^4, while the probability of a counterclockwise circuit is q^4. When $p = q$, the transition probabilities satisfy Kolmogorov's criterion; the model is time reversible. When $p \neq q$, the probabilities of the clockwise and counterclockwise circuits are not the

same ($p^4 \neq q^4$). Kolmogorov's criterion is violated, indicating that the model is not time reversible.

What is the best way to test time reversibility in practise? Kolmogorov's criterion is useful if you have a sequence of states that violates Equation 2.23, providing a direct demonstration that the model at hand is not time reversible. However, it is less useful as a general, systematic test of time reversibility. Given a Markov model with an $s \times s$ transition probability matrix, P, you could use the detailed balance equations to test for time reversibility by checking that $\varphi^*_x P_{xy} = \varphi^*_y P_{yx}$ for all combinations of x and y. However, it is more efficient to do all of these tests using a single matrix product. First, determine the stationary distribution φ^* by solving the system of equations in Equation 2.14. Then, construct an $s \times s$ diagonal matrix, D, where the entries on the main diagonal are $\varphi^*_1, \varphi^*_2, \ldots, \varphi^*_s$ and the off-diagonal elements are zero. The detailed balance equations hold if and only if $D \times P$ is a symmetric matrix. Convince yourself that this is the case.

2.3 Markov models of sequence evolution

Now that we have the Markov chain machinery under our belts, let's return to the question of modeling sequence evolution. The process of substitution at a single site in a nucleotide sequence can be modeled as a Markov chain, where each state represents a single nucleotide. The transition probability, P_{xy}, is the probability that nucleotide x will be replaced by nucleotide y in one time step. Similarly, Markov chains can be constructed to model the evolution of amino acid sequences. Although in principle Markov models of sequence evolution are general and can be applied to nucleotide sequences and to amino acid sequences in exactly the same way, in practice working with a twenty-letter alphabet poses challenges that do not arise with a four-letter alphabet. In addition, the biophysical properties of the amino acids are more varied than those of the nucleotides. For these reasons, the Markov chain framework is applied somewhat differently in amino acid sequence models. For the moment, we will focus on nucleotide models and postpone amino acid models until later in the course.

Markov models of sequence substitution are used to answer a wide range of questions that arise in molecular evolution, including correcting for multiple substitutions at the same site, simulating sequence evolution, estimating rates of evolution, deriving substitution scoring matrices, estimating the likelihood of observing a pair of aligned nucleotides, and maximum likelihood estimation methods for reconstructing evolutionary trees.

2.3.1 The Jukes-Cantor model

The simplest Markov model of sequence evolution for DNA is the *Jukes-Cantor model*[3], which assumes that all substitutions ($A \to C$, $A \to G$, $A \to T$, $C \to A$...) are equally probable and occur at a rate, α. Since DNA sequences are made up of four nucleotides, there are three possible substitutions for any given base. Thus, the overall rate of substitution is $\lambda = 3\alpha$. That is, λ is the probability that a given nucleotide will be replaced by *some* other nucleotide in one time step. The probability that the nucleotide remains unchanged is $1 - 3\alpha$. A graphical representation of this model is shown in Fig. 2.1. The transition probability matrix for this Markov model is:

$$\begin{bmatrix} & A & G & C & T \\ A & 1-3\alpha & \alpha & \alpha & \alpha \\ G & \alpha & 1-3\alpha & \alpha & \alpha \\ C & \alpha & \alpha & 1-3\alpha & \alpha \\ T & \alpha & \alpha & \alpha & 1-3\alpha \end{bmatrix}$$

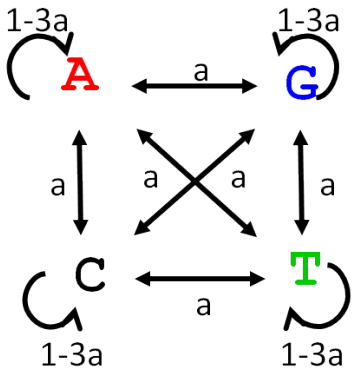


Figure 2.1: Jukes Cantor substitution model

The rate, α, of each possible substitution is an explicit parameter of the Jukes-Cantor model. In addition, the frequencies of A's, G's, C's and T's are implicitly specified by the model, since this is determined by the stationary distribution. The stationary distribution of this Markov chain is $\varphi^* = (0.25, 0.25, 0.25, 0.25)$. (Verify that this is so by checking that $\varphi^* = \varphi^* P$).

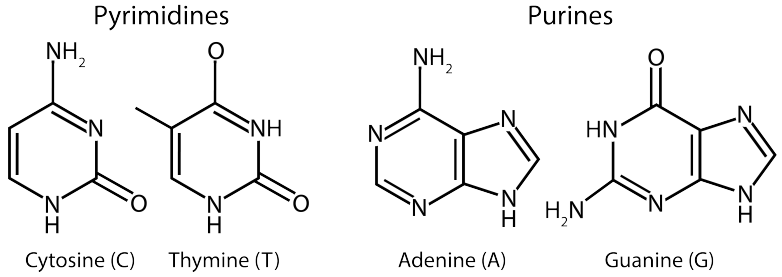
Nucleotide substitution models can be made more realistic in two directions. First, the assumption that all substitutions occur at the same rate can be relaxed. Second, the

[3]Jukes and Cantor, Evolution of protein molecules. In H. N. Munro, (ed.) *Mammalian Protein Metabolism*, 21-123, Academic Press, NY, 1969.

specification of the rates can be adjusted to yield a non-uniform stationary distribution, since the assumption that all four bases have the same frequency ($\varphi_A = \varphi_C = \varphi_G = \varphi_T$) is unlikely to hold in most data sets.

2.3.2 Non-uniform transition probabilities

The Kimura 2 Parameter (K2P) model assumes that transitions and transversions occur at different rates. A *transition* is the substitution of a purine for another purine or a pyrimidine for another pyrimidine. A *transversion* is the substitution of a purine for a pyrimidine or a pyrimidine for a purine. Recall that the pyrimidines, including cytosine and thymine, are nucleotides with a ring with six elements. The purines, including adenine and guanine, have a pyrimidine ring fused to a five-sided imidazole ring.



It make sense that transversions would proceed at a different rate than transitions, since substituting a purine with a pyrimidine, or vice versa, involves a greater change in size and shape than a substitution of two nucleotides from the same class.

The transition matrix for the K2P model is

$$\begin{bmatrix} & A & G & C & T \\ A & 1-\alpha-2\beta & \alpha & \beta & \beta \\ G & \alpha & 1-\alpha-2\beta & \beta & \beta \\ C & \beta & \beta & 1-\alpha-2\beta & \alpha \\ T & \beta & \beta & \alpha & 1-\alpha-2\beta \end{bmatrix},$$

where α is the rate of transitions and β is the rate of transversions. In this model, the overall substitution rate is $\lambda = \alpha + 2\beta$, since of the three possible substitutions for any given base, one is a transition and two are transversions. Like the Juke Cantor model, the K2P model has a uniform stationary distribution, $\varphi^* = (0.25, 0.25, 0.25, 0.25)$. (Work through the algebra to convince yourself that this is true.)

2.3.3 Non-uniform stationary distributions

A stationary distribution with uniform base frequencies is not a realistic model for the many genomes in which the G+C content deviates from 50%. The 1981 Felsenstein (F81)

model allows for an arbitrary stationary distribution, $\varphi^* = (\varphi^*_A, \varphi^*_C, \varphi^*_G, \varphi^*_T)$, where the frequencies of the four bases are allowed to deviate from the uniform distribution. Like the Jukes-Cantor model, the F81 model has a single scaling parameter, α, and does not make a distinction between transitions and transversions. The *F81* transition matrix is

$$\begin{bmatrix} & A & G & C & T \\ A & 1-\alpha\,(\varphi^*_C+\varphi^*_G+\varphi^*_T) & \alpha\,\varphi^*_G & \alpha\,\varphi^*_C & \alpha\,\varphi^*_T \\ G & \alpha\,\varphi^*_A & 1-\alpha\,(\varphi^*_A+\varphi^*_C+\varphi^*_T) & \alpha\,\varphi^*_C & \alpha\,\varphi^*_T \\ C & \alpha\,\varphi^*_A & \alpha\,\varphi^*_G & 1-\alpha\,(\varphi^*_A+\varphi^*_G+\varphi^*_T) & \alpha\,\varphi^*_T \\ T & \alpha\,\varphi^*_A & \alpha\,\varphi^*_G & \alpha\,\varphi^*_C & 1-\alpha\,(\varphi^*_A+\varphi^*_C+\varphi^*_G) \end{bmatrix}$$

2.3.4 More general models

The *Hasegawa, Kishino, Yano (HKY) model* combines both innovations. It allows for different rates for transitions and transversions and an arbitrary stationary distribution, $\varphi^* = (\varphi^*_A, \varphi^*_C, \varphi^*_G, \varphi^*_T)$, where the individual base frequencies may deviate from a uniform distribution. The HKY transition matrix is:

$$\begin{bmatrix} & A & G & C & T \\ A & 1-\alpha\,\varphi^*_G-\beta\,(\varphi^*_C+\varphi^*_T) & \alpha\,\varphi^*_G & \beta\,\varphi^*_C & \beta\,\varphi^*_T \\ G & \alpha\,\varphi^*_A & 1-\alpha\,\varphi^*_A-\beta\,(\varphi^*_C+\varphi^*_T) & \beta\,\varphi^*_C & \beta\,\varphi^*_T \\ C & \beta\,\varphi^*_A & \beta\,\varphi^*_G & 1-\alpha\,\varphi^*_T-\beta\,(\varphi^*_A+\varphi^*_G) & \alpha\,\varphi^*_T \\ T & \beta\,\varphi^*_A & \beta\,\varphi^*_G & \alpha\,\varphi^*_C & 1-\alpha\,\varphi^*_C-\beta\,(\varphi^*_A+\varphi^*_G) \end{bmatrix}$$

The *General Time Reversible (GTR) model* is an even more general model that allows a different rate for each of the six possible substitutions and an arbitrary stationary distribution, $\varphi^* = (\varphi^*_A, \varphi^*_C, \varphi^*_G, \varphi^*_T)$ The GTR transition matrix is:

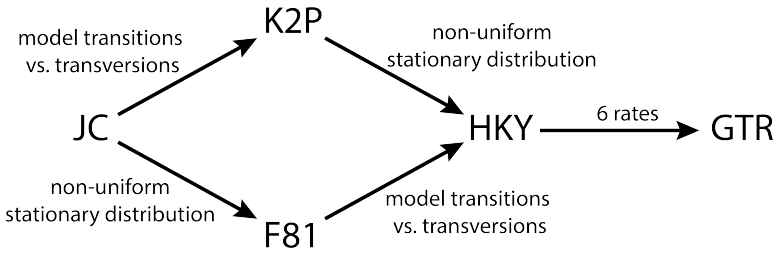
$$\begin{bmatrix} & A & G & C & T \\ A & 1-\alpha\,\varphi^*_G-\beta\,\varphi^*_C-\gamma\,\varphi^*_T & \alpha\,\varphi^*_G & \beta\,\varphi^*_C & \gamma\,\varphi^*_T \\ G & \alpha\,\varphi^*_A & 1-\alpha\,\varphi^*_A-\delta\,\varphi^*_C-\epsilon\,\varphi^*_T & \delta\,\varphi^*_C & \epsilon\,\varphi^*_T \\ C & \beta\,\varphi^*_A & \delta\,\varphi^*_G & 1-\beta\,\varphi^*_A-\delta\,\varphi^*_G-\eta\,\varphi^*_T & \eta\,\varphi^*_T \\ T & \gamma\,\varphi^*_A & \epsilon\,\varphi^*_G & \eta\,\varphi^*_C & 1-\gamma\,\varphi^*_A-\epsilon\,\varphi^*_G-\eta\,\varphi^*_C \end{bmatrix}$$

All of these models are discussed in greater detail in various molecular evolution textbooks; see, for example, Li's *Molecular Evolution*, (Sinauer Associates, 1997).

2.3.5 Model Selection

In deciding which model to use for a particular data set, we face a trade-off that arises with many statistical models. More general models with more parameters provide a more accurate representation of the underlying evolutionary process. However, with more complex

models, more data is required to estimate the parameter values and the danger of overfitting the parameters is greater.



Analyses of alignments of present-day sequences suggest that, in many sequence families, the rate of change varies from site to site. This is typically addressed by assuming that sequence substitution in a given family can be captured by a single model with a small number of rate categories. For example, one might model substitution in a given family using the Jukes Cantor model with four rates, $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$. For each site, i, maximum likelihood estimation is used to estimate probabilities $(p_1(i), p_2(i), p_3(i), p_4(i))$, where $p_r(i)$ is the probability that site i is evolving at rate α_r. Ziheng Yang discusses this approach in his textbook *Computational Molecular Evolution* (Oxford University Press, 2006).

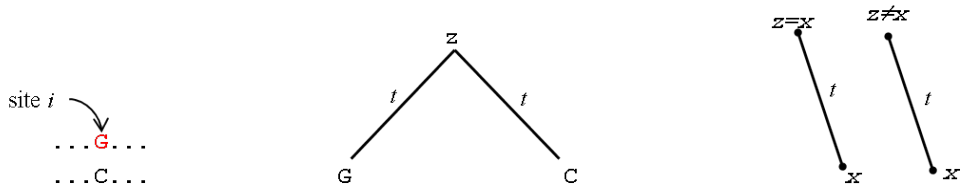
These models do not allow for changes in rate or in GC-content over time. Developing models to account for temporal changes in rate or nucleotide composition is currently an active area of research.

2.4 Applications of DNA substitution models

There are many applications of Markov models of sequence substitution. Here, we demonstrate how DNA substitution models can be used to estimate the likelihood of observing a pair of aligned nucleotides, given a phylogenetic model, and to correct for multiple substitutions. In the next chapter, we will use an amino acid substitution model to derive a scoring matrix.

2.4.1 The likelihood of a pair of aligned nucleotides

First, let's consider the problem of estimating the likelihood of a pair of aligned sequences. This problem arises in maximum likelihood approaches to estimating a phylogenetic tree. Maximum likelihood estimation (MLE) is a general method for estimating parameters of a model. It is based on the assumption that the observed data is best explained by the model that maximizes its likelihood; that is, the model for which the probability of the data is greatest. Given a parameterized model, the parameter values are estimated by determining the values that maximize the probability of the data.



(a) (b) (c)

Figure 2.2: A hypothetical evolutionary scenario: (a) A pairwise alignment of two sequences with residues $x = G$ and $y = C$ at site i. (b) These residues in present-day nucleotide sequences have been diverging from a common, unknown ancestral nucleotide, z, for a period of time, t. (c) In the Jukes Cantor model, the calculation of the probability of seeing a given nucleotide at site i after elapsed time, t, can be reduced to two cases: either the nucleotides in the ancestral and present-day sequences are the same (left, Equation 2.27) or they are not (right, Equation 2.28).

In the context of phylogeny estimation, the observed data is a set of k aligned sequences. The model has two components: a Markov model of sequence substitution and a rooted, binary tree with k leaves. The likelihood is the probability of observing the multiple alignment, under the assumption that the sequences evolved along the branches of the tree, sustaining mutations according to the rates specified by the substitution model. For a fixed tree topology, the branch lengths and the substitution rates are estimated by maximizing the probability of the multiple sequence alignment.

We demonstrate this calculation for the case where $k = 2$. Suppose we have two residues, x and y, that are the descendants of an ancestral nucleotide, z, and that time t has elapsed since their divergence. Fig. 2.2 illustrates this situation for the case where x is a guanine and y is a cytosine. The probability of observing x aligned with y is

$$\Pr\begin{pmatrix} x \\ y \end{pmatrix} \mid t = \sum_{z \in \{A,C,G,T\}} p_z \; p_{zx}(t) \; p_{zy}(t), \tag{2.24}$$

where p_z is an estimation of the frequency of z in the ancestral sequence and $p_{zx}(t)$ and $p_{zy}(t)$ are the respective probabilities of observing an x after time t, and a y at time t, given that the ancestral residue was z. Since the base in the ancestral sequence is unknown, we estimate the probability by summing over all possible values of z.

To evaluate the right hand side of Equation 2.24, requres expressions for $p_{zx}(t)$ and $p_{zy}(t)$. Here, we derive expressions for the probability $p_{zx}(t)$ under the assumption that the sequence is evolving according to the Jukes Cantor model. Analogous expressions are derived from the Kimura 2 parameter model in Section2.4.3.

The Jukes Cantor transition probability matrix,

$$\begin{bmatrix} 1-3\,\alpha & \alpha & \alpha & \alpha \\ \alpha & 1-3\,\alpha & \alpha & \alpha \\ \alpha & \alpha & 1-3\,\alpha & \alpha \\ \alpha & \alpha & \alpha & 1-3\,\alpha \end{bmatrix}$$

is defined in terms of an instantaneous substitution rate, α. If the duration of a single time step in this Markov chain is δt, $\delta t \ll \frac{1}{\alpha}$, then the probability of a substitution between a given pair of nucleic acids in a single time step is $\alpha\,\delta t$. We can define a transition probability matrix for the Jukes Cantor Markov model with this time interval as follows:

$$\begin{bmatrix} 1-3\,\alpha\,\delta t & \alpha\,\delta t & \alpha\,\delta t & \alpha\,\delta t \\ \alpha\,\delta t & 1-3\,\alpha\,\delta t & \alpha\,\delta t & \alpha\,\delta t \\ \alpha\,\delta t & \alpha\,\delta t & 1-3\,\alpha\,\delta t & \alpha\,\delta t \\ \alpha\,\delta t & \alpha\,\delta t & \alpha\,\delta t & 1-3\,\alpha\,\delta t \end{bmatrix}.$$

We use this transition matrix to derive an expression describing how changes accumulate at site i over a period of time t. First, we consider the event of observing, for example, an A at site i after a single time step; i.e., at time $t+\,\delta t$. This event can occur in two ways: either site i contained an A at time t and no substitution occurred during the most recent time step or site i contained some other nucleotide at time t and a substitution resulted in an A one time step later. Accounting for both of these scenarios, the probability of observing A at time $t+\,\delta t$ is

$$\varphi_A(t+\,\delta t\,) = (1-3\,\alpha\,\delta t\,)\varphi_A(t) + \alpha\,\delta t\,\varphi_C(t) + \alpha\,\delta t\,\varphi_G(t) + \alpha\,\delta t\,\varphi_T(t)$$

where $\varphi_x(t)$ is the probability of observing nucleotide x (i.e. of being in state E_x) at time t. Since in the stationary distribution of the Jukes Cantor model, all nucleotides have the same frequency, we can combine the 2nd, 3rd and 4th terms, yielding

$$\varphi_A(t+\,\delta t\,) = (1-3\,\alpha\,\delta t\,)\,\varphi_A(t) + \alpha\,\delta t\,\left[1-\varphi_A(t)\right].$$

Here, the first term gives the probability that the residue at site i at time t was an A and no substitution occurred. The second term is the probability that the residue at time t was not an A and a substitution did occur, replacing that residue with A. Since the model is symmetric, this equation applies equally well to C, G or T. We can therefore rewrite the equation using the parameter x, where $x \in \{A, C, G, T\}$, and combine terms to obtain

$$\varphi_x(t+\,\delta t\,) = \varphi_x(t) + (1-4\,\varphi_x(t))\,\alpha\,\delta t\,. \tag{2.25}$$

Having obtained an expression for the probability of observing a given nucleotide (x) after one time step, we next need to derive an expression for the probability of observing x

after a longer time interval. Subtracting $\varphi_x(t)$ from both sides of Equation 2.25 and some algebraic manipulation yields

$$\frac{\varphi_x(t+\ \delta t\ ) - \varphi_x(t)}{\delta t} = (1 - 4\ \varphi_x(t))\ \alpha.$$

Taking the limit as $\delta t\ \rightarrow 0$, we obtain a differential equation

$$\frac{d\varphi_x(t)}{dt} = (1 - 4\ \varphi_x(t))\ \alpha \tag{2.26}$$

that we can use to obtain an expression for the probability of observing nucleotide x at site i after an arbitrary time interval t. This differential equation has a standard form ($f'(t) = a - bf(t)$) with a known solution, from which we obtain

$$\varphi_x(t) = \frac{1}{4} + \left(\varphi_x(0) - \frac{1}{4}\right) e^{-4\alpha t}.$$

See Box 3 for a more detailed explanation of solving the standard form differential equation.

We now have an expression that gives the probability of observing nucleotide x in terms of initial state probability, $\varphi_x(0)$. So how to solve the probability of observing nucleotide x, *given* the ancestral nucleotide z? We have two cases. Either the nucleotide at this site in the ancestral sequence was also x (i.e., $\varphi_x(0) = 1$) or the nucleotide at this site in the ancestral sequence was some residue other than x (i.e., $\varphi_x(0) = 0$). Substituting 1 for $\varphi_x(0)$ in the expression for $\varphi_x(t)$, we can derive an expression for the probability that the present-day residue is the same as the ancestral nucleotide after time t:

$$p_{xx}(\alpha, t) = \frac{1}{4}\ +\ \frac{3}{4} e^{\ -4\alpha t}. \tag{2.27}$$

Similarly, setting $\varphi_x(0) = 0$ in the the expression for $\varphi_x(t)$ gives the probability that the present-day nucleotide differs from the ancestral residue after time t:

$$p_{zx}(\alpha, t) = \frac{1}{4}\ -\ \frac{1}{4} e^{\ -4\alpha t}. \tag{2.28}$$

In general, the event that z was present in the ancestral sequence t million years ago and x are is observed at site i today may have a different probability for each of the 16 possible pairs $z, x \in \Sigma \times \Sigma$, where $\Sigma = \{A, G, C, T\}$. For the Jukes Cantor model, with a single parameter for all possible nucleotide substitutions, there are only two possibilities: the value of p_{zx} is the same for all 12 pairs in $\Sigma \times \Sigma$ such that $x \neq z$. Similarly, for the case where $x = z$, the probability, denoted p_{xx}, is the same for the four cases where the ancestor and present-day residues are the same.

Box 3: Solving the standard form differential equation

The standard form linear differential equation

$$\frac{df(x)}{dx} = q(x) - p(x)f(x)$$

can be solved with the formula

$$f(x) = g(x)^{-1} \int (g(x)q(x)dx) + C \cdot g(x)^{-1},$$

where $g(x) = e^{\int (p(x)dx)}$.
In our case, the differential equation is

$$\frac{d\varphi_x(t)}{\delta t} = \alpha - 4\alpha\, \varphi_x(t).$$

So, when substituting in the standard form, $x = t$, $f(x) = \varphi_x(t)$, $q(t) = \alpha$, and $p(t) = 4\alpha$. Solving $g(t) = e^{\int p(t)dt}$ yields $e^{\int 4\alpha dt} = e^{4\alpha t}$. From the standard form solution, we get

$$\begin{aligned}
\varphi_x(t) &= g(t)^{-1} \int (g(t)q(t)\, \delta t\,) + C \cdot g(t)^{-1} \\
&= e^{-4\alpha t} \int (e^{4\alpha t}\alpha\, \delta t\,) + C \cdot e^{-4\alpha t} \\
&= e^{-4\alpha t} \frac{1}{4\alpha} e^{4\alpha t} \cdot \alpha + C \cdot e^{-4\alpha t} \\
&= \frac{1}{4} + C \cdot e^{-4\alpha t}
\end{aligned}$$

Solving for constant C at $t = 0$ gives

$$\begin{aligned}
\varphi_x(0) &= \frac{1}{4} + C \cdot e^{-4\alpha 0} \\
&= \frac{1}{4} + C \\
C &= \varphi_x(0) - \frac{1}{4}.
\end{aligned}$$

Combining these equations, we get

$$\varphi_x(t) = \frac{1}{4} + \left(\varphi_x(0) - \frac{1}{4}\right) e^{-4\alpha t}.$$

As an example, we use Equations 2.27 and 2.28 to obtain an expression, in terms of α and t, for the likelihood of observing G aligned with C. The likelihood of observing a guanine in one sequence and a cytosine in the other is

$$\Pr\left(\begin{matrix}G\\C\end{matrix} \mid \alpha, t\right) = \sum_{z\in\{A,C,G,T\}} p_z \; p_{zG}(\alpha,t) \; p_{zC}(\alpha,t), \tag{2.29}$$

where p_z is an estimate of the frequency of z in the ancestral sequence. Expanding the right hand side of Equation 2.29, we obtain

$$p_A \; p_{AG}(\alpha,t) \; p_{AC}(\alpha,t) + p_C \; p_{CG}(\alpha,t) \; p_{CC}(\alpha,t) + p_G \; p_{GG}(\alpha,t) \; p_{GC}(\alpha,t) + p_T \; p_{TG}(\alpha,t) \; p_{TC}(\alpha,t). \tag{2.30}$$

Substituting the right hand sides of Equations 2.27 and 2.28 into Equation 2.30, we obtain the likelihood for observing G aligned with C

$$\Pr\left(\begin{matrix}G\\C\end{matrix} \mid \alpha, t\right) = \frac{1}{2}\left(\frac{1}{4} \; - \; \frac{1}{4}e^{\,-4\alpha t}\right)^2 \; + \; \frac{1}{2}\left(\frac{1}{4} \; - \; \frac{1}{4}e^{\,-4\alpha t}\right) \cdot \left(\frac{1}{4} \; + \; \frac{3}{4}e^{\,-4\alpha t}\right),$$

assuming that $p_z = \frac{1}{4}$ for all z.

Equations 2.27 and 2.28 can be used to derive the probability of observing x aligned with y in a pair of sequences that have been diverging from a common ancestor for t million years. There are two cases: the residues at site i in s_1 and s_2 may differ or the the same residue may appear at site i in both sequences. We consider each case in turn.

Thus, the probability of aligning two different nucleotides is:

$$\begin{aligned}
\Pr\left(\begin{matrix}x\\y\end{matrix} \mid \alpha, t\right) &= \frac{1}{4}\left(2p_{xx}(t)p_{zx}(t) \; + \; 2p_{zx}(t)^2\right) \\
&= \frac{1}{2}p_{zx}(t) \; \left(p_{xx}(t) \; + \; p_{zx}(t)\right) \\
&= \frac{1}{8}\left(1 \; - \; e^{\,-4\alpha t}\right) \; \left(\frac{1}{4}(1 \; + \; 3e^{\,-4\alpha t}) \; + \; \frac{1}{4}(1 \; - \; e^{\,-4\alpha t})\right) \\
&= \frac{1}{32}\left(1 \; - \; e^{\,-4\alpha t}\right) \; \left(2 \; + \; 2e^{\,-4\alpha t}\right) \\
&= \frac{1}{16}\left(1 \; - \; e^{\,-4\alpha t}\right) \; \left(1 \; + \; e^{\,-4\alpha t}\right) \\
&= \frac{1}{16}\left(1 \; - \; e^{\,-8\alpha t}\right).
\end{aligned} \tag{2.31}$$

The probability that both sequences will have an x at site i is

$$\begin{aligned}
\Pr\left(\begin{matrix}x\\x\end{matrix} \mid \alpha, t\right) &= \frac{1}{4}\left(p_{xx}(t)^2 + 3p_{zx}(t)^2\right) \\
&= \frac{1}{4}\left(\frac{1}{16}(1 + 3e^{-4\alpha t})^2 + \frac{3}{16}(1 - e^{-4\alpha t})^2\right) \\
&= \frac{1}{64}\left(1 + 6e^{-4\alpha t} + 9e^{-8\alpha t} + 3 - 6e^{-4\alpha t} + 3e^{-8\alpha t}\right) \\
&= \frac{1}{64}\left(4 + 12e^{-8\alpha t}\right) \\
&= \frac{1}{16}\left(1 + 3e^{-8\alpha t}\right). \qquad (2.32)
\end{aligned}$$

We now have expressions for the probability of observing nucleotide x aligned with nucleotide y that depends on two parameters: the branch length, t, and the substitution rate, α. These parameter values are typically estimated by finding the values of t and α that maximize $\Pr\left(\begin{matrix}x\\y\end{matrix} \mid \alpha, t\right)$.

This approach can be expanded to alignments of more than two sequences by nesting multiple expressions with the same form as the right hand side of Equation 2.24. Under the assumption of positional independence, the likelihood for multiple sites is simply the product of the likelihoods for each site, individually.

2.4.2 Correcting for multiple substitutions.

Another problem that arises in molecular evolution is estimating the amount of sequence divergence between a pair of sequences, s_1 and s_2. A simple approach to estimating sequence divergence would be to count the number of positions that are not identical in the pairwise alignment of s_1 and s_2. If only a few changes have occurred, then the observed number of mismatches may, in fact, be the actual number of substitutions. However, as the divergence increases, so does the probability of two or more substitutions at the same site. In this case, the number of observed changes will underestimate the actual divergence, as shown in Fig. 2.3.

Suppose that we have an ungapped[4] pairwise alignment of length n of two nucleotide sequences, s_1 and s_2, that disagree at m positions. We wish to estimate the number of substitutions that actually occurred over t, the time interval that elapsed since they diverged from a common ancestor.

Here, we use the Jukes-Cantor model to derive a more accurate estimate of the number of substitutions. Recall that the Jukes-Cantor model assumes that all substitutions ($A \rightarrow C$, $A \rightarrow G$, $A \rightarrow T$, $C \rightarrow A$...) are equally likely and occur at a rate α, resulting in an

[4]None of the substitution models we have discussed account for insertions and deletions.

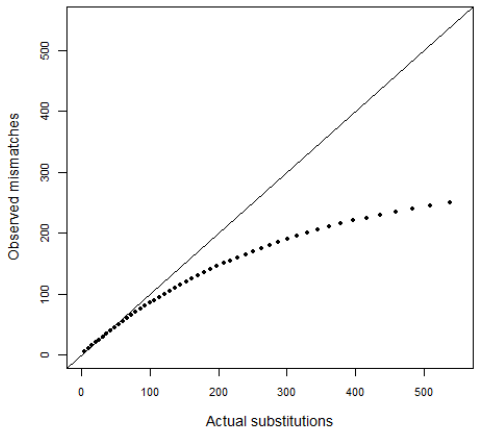


Figure 2.3: The observed number of mismatches versus the actual number of substitutions in a sequence of 400 nucleotides.

overall rate of substitution $\lambda = 3\alpha$. If we assume a constant rate of substitution, λ, in both lineages then the expected number of substitutions per site is

$$\begin{aligned} P_s &= 2\lambda t \\ &= 6\alpha t. \end{aligned}$$

However, both α and t are unknown. Given a Markov model of sequence substitution, we use the observed frequency of mismatches to estimate $2\lambda t$ using the following strategy:

First, using the expressions for $p_{xx}(t)$ and $p_{zx}(t)$ that we derived in the previous section (Eqns. 2.27 and 2.28), we estimate the frequency of mismatches as a function of αt,

$$P_m = f(\alpha t).$$

We do this by estimating P_M, the frequency of matches, and subtracting to obtain $P_m = 1 - P_M$. Next, we invert this function to obtain an expression for the expected number of substitutions per site in terms of the number of mismatches per site:

$$\alpha t = f^{-1}\left(P_m\right).$$

The actual 'frequency of mismatches can be approximated by the observed frequency of mismatches, $\frac{m}{n}$, yielding an equation of the form

$$\alpha t \approx f^{-1}\left(\tfrac{m}{n}\right).$$

From this, we obtain an approximation for the frequency of substitutions:

$$\begin{aligned} P_s &= 6\alpha t \\ &\approx 6f^{-1}\left(\tfrac{m}{n}\right). \end{aligned}$$

Now we apply this strategy to obtain an estimate of the number of substitutions that occurred assuming that sequences are evolving according to the Jukes Cantor model. First, we derive an expression for the probability of observing a match; for example, for observing two adenines aligned at site i. Given two sequences evolving independently from a common ancestral sequence, the probability of a match at site i is

$$P_M = \sum_{x \in \{A,G,C,T\}} \Pr\left(\begin{matrix} x \\ x \end{matrix} \mid \alpha, t\right)$$

The term inside the summation is given by Eqn 2.32. Under the Jukes-Cantor model, this probability is the same for all residues, and results in a match probability

$$P_M = \frac{1}{4}\left(1 + 3e^{-8\alpha t}\right). \tag{2.33}$$

The probability of observing a mismatch at site i is simply $1 - P_M$ or

$$P_m = \frac{3}{4}\left(1 - e^{-8\alpha t}\right). \tag{2.34}$$

We solve the above equation to obtain an expression for αt in terms of P_m:

$$\alpha t = -\frac{1}{8}\ln\left(1 - \frac{4}{3}P_m\right).$$

Multiplying both sides of the equation by 6 yields $6\alpha t$, which is the expected frequency of substitutions per site in terms of the probability of observing a mismatch:

$$P_s = -\frac{3}{4}\ln\left(1 - \frac{4}{3}P_m\right).$$

P_m can be estimated by the observed frequency of mismatches, allowing us to obtain an estimate of the probability of substitution in terms of the fraction of sites with an observable difference:

$$P_s \approx -\frac{3}{4}\ln\left(1 - \frac{4}{3}\frac{m}{n}\right).$$

Multiplying by n yields an estimate of the expected number of substitutions that actually occurred in the entire alignment of length n:

$$-\frac{3}{4}\ln\left(1 - \frac{4}{3}\frac{m}{n}\right) \cdot n. \tag{2.35}$$

For example, if we observe mismatches at 100 sites in a pairwise nucleotide sequence alignment of length 1,000, then the Jukes-Cantor model predicts that the actual number of substitutions per site is 0.107 or 107 substitutions.

2.4.3 Applications with the K2P model

In previous sections, we used the Jukes Cantor model to derive expressions for the probability of observing a given nucleotide at site i in a present day sequence (Equations 2.27 and 2.28). We then used these equations to obtain an estimate for the number of substitutions that occurred, given the number of mismatches observed in an ungapped alignment (Equation 2.35). Analogous equations can be derived for the K2P model, which assumes that transitions and transversions occur at different rates.

In this case, we will use the observation for continuous Markov chains that the continuous rate probabilities can be computed from the derivative transition matrix $P'(t)$, which can be calculated from the product of the transition matrix $P(t)$ and the instantaneous rates at $t = 0$. Specifically, $P'(t) = P(t)P'(0)$

For the K2P model, the transition matrix $P(t)$ is:

$$\begin{bmatrix} & A & G & C & T \\ A & p_{AA}(t) & p_{AG}(t) & p_{AC}(t) & p_{AT}(t) \\ G & p_{GA}(t) & p_{GG}(t) & p_{GC}(t) & p_{GT}(t) \\ C & p_{CA}(t) & p_{CG}(t) & p_{CC}(t) & p_{CT}(t) \\ T & p_{TA}(t) & p_{TG}(t) & p_{TC}(t) & p_{TT}(t). \end{bmatrix}$$

The probability of observing different nucleotides at site i in the ancestral and present-day sequences depends on whether the nucleotides belong to the same class or to different classes. If the ancestral nucleotide z belongs to the same class as the present-day nucleotide (i.e., there was a transition), I denote the present-day nucleotide as x and the probability of $z \rightarrow x$ over time t as $p^s_{zx}(t)$. If the ancestral nucleotide z belongs to a different class as the present-day nucleotide (i.e., there was a transversion), I denote the present-day nucleotide as y and the probability of $z \rightarrow y$ over time t as $p^v_{zy}(t)$. Under the K2P model, the probability that the ancestral nucleotide z at site i is the same as the present-day nucleotide after time t is $p_{zz}(t) = 1 - p^s_{zx}(t) - 2p^v_{zy}(t)$. The transition matrix $P(t)$ can then be written as:

$$\begin{bmatrix} & A & G & C & T \\ A & 1 - p^s_{zx}(t) - 2p^v_{zy}(t) & p^s_{zx}(t) & p^v_{zy}(t) & p^v_{zy}(t) \\ G & p^s_{zx}(t) & 1 - p^s_{zx}(t) - 2p^v_{zy}(t) & p^v_{zy}(t) & p^v_{zy}(t) \\ C & p^v_{zy}(t) & p^v_{zy}(t) & 1 - p^s_{zx}(t) - 2p^v_{zy}(t) & p^s_{zx}(t) \\ T & p^v_{zy}(t) & p^v_{zy}(t) & p^s_{zx}(t) & 1 - p^s_{zx}(t) - 2p^v_{zy}(t). \end{bmatrix} \tag{2.36}$$

The derivative matrix $P'(t)$ is found by taking the derivative of each entry with respect to t, yielding:

$$\begin{bmatrix} & A & G & C & T \\ A & -p^s_{zx}{}'(t) - 2p^v_{zy}{}'(t) & p^s_{zx}{}'(t) & p^v_{zy}{}'(t) & p^v_{zy}{}'(t) \\ G & p^s_{zx}{}'(t) & -p^s_{zx}{}'(t) - 2p^v_{zy}{}'(t) & p^v_{zy}{}'(t) & p^v_{zy}{}'(t) \\ C & p^v_{zy}{}'(t) & p^v_{zy}{}'(t) & -p^s_{zx}{}'(t) - 2p^v_{zy}{}'(t) & p^s_{zx}{}'(t) \\ T & p^v_{zy}{}'(t) & p^v_{zy}{}'(t) & p^s_{zx}{}'(t) & -p^s_{zx}{}'(t) - 2p^v_{zy}{}'(t). \end{bmatrix}$$

Setting $p^s_{zx}{}'(0) = \alpha$ and $p^v_{zy}{}'(0) = \beta$, then the derivative transition matrix at $t = 0$, $P'(0)$ is

$$\begin{bmatrix} & A & G & C & T \\ A & -\alpha - 2\beta & \alpha & \beta & \beta \\ G & \alpha & -\alpha - 2\beta & \beta & \beta \\ C & \beta & \beta & -\alpha - 2\beta & \alpha \\ T & \beta & \beta & \alpha & -\alpha - 2\beta. \end{bmatrix} \tag{2.37}$$

Our goal is to solve the derivatives $\frac{dp^s_{zx}(t)}{dt}$ and $\frac{dp^v_{zy}(t)}{dt}$. We do this by taking any transition and transversion entry, respectively, in $P'(t) = P(t) \times P'(0)$ (i.e., $p'_{AG}(t)$ and $p'_{AC}(t)$) and solving using the method for standard form linear differential equations, as we did in the example with Jukes-Cantor. Looking at matrices 2.36 and 2.37, we see that these derivatives are

$$\begin{aligned} \frac{dp^s_{zx}(t)}{dt} &= \alpha\left(1 - p^s_{zx}(t) - 2p^v_{zy}(t)\right) - (\alpha + 2\beta)\, p^s_{zx}(t) + \beta p^v_{zy}(t) + \beta p^v_{zy}(t) \\ &= \alpha - \alpha p^s_{zx}(t) - 2\alpha p^v_{zy}(t) - \alpha p^s_{zx}(t) - 2\beta p^s_{zx}(t) + 2\beta p^v_{zy}(t) + \beta p^v_{zy}(t) \\ &= \alpha - 2\,(\alpha + \beta)\, p^s_{zx}(t) - 2\,(\alpha - \beta)\, p^v_{zy}(t) \end{aligned}$$

$$\begin{aligned} \frac{dp^v_{zy}(t)}{dt} &= \beta\left(1 - p^s_{zx}(t) - 2p^v_{zy}(t)\right) + \beta p^s_{zx}(t) - (\alpha + 2\beta)\, p^v_{zy}(t) + \alpha p^v_{zy}(t) \\ &= \beta - \beta p^s_{zx}(t) - 2\beta p^v_{zy}(t) + \beta p^s_{zx}(t) - \alpha p^v_{zy}(t) - 2\beta p^v_{zy}(t) + \alpha p^v_{zy}(t) \\ &= \beta - 4\beta p^v_{zy}(t). \end{aligned}$$

Since $\frac{dp^v_{zy}(t)}{dt}$ only depends on $p^v_{zy}(t)$, we will solve the transversion case first. In fact, this is exactly the same form as differential equation 2.26, which we solved for the JC model. Thus, $p^v_{zy}(t) = \frac{1}{4} + C \cdot e^{-4\beta t}$ and $C = -\frac{1}{4}$ since $p^v_{zy}(0) = 0 = \frac{1}{4} + C$. Therefore, the probability of observing a *transversion* substitution at site i in a descendant sequence after elapsed time t is

$$p^v_{zy}(t) = \frac{1}{4} \; - \; \frac{1}{4}e^{-4\beta t}. \tag{2.38}$$

We can then substitute this value in $\frac{dp^v_{zx}(t)}{dt}$ to get

$$\begin{aligned} \frac{dp^s_{zx}(t)}{dt} &= \alpha - 2\,(\alpha + \beta)\, p^s_{zx}(t) - \tfrac{1}{2}\,(\alpha - \beta)\left(1 - e^{-4\beta t}\right) \\ &= \alpha - 2\,(\alpha + \beta)\, p^s_{zx}(t) - \tfrac{1}{2}\alpha + \tfrac{1}{2}\beta + \tfrac{1}{2}\,(\alpha - \beta)\, e^{-4\beta t} \\ &= \tfrac{1}{2}\,(\alpha + \beta) + \tfrac{1}{2}\,(\alpha - \beta)\, e^{-4\beta t} - 2\,(\alpha + \beta)\, p^s_{zx}(t) \end{aligned}$$

Solving the standard form differential equation (see Box 4) yields:

$$p^s_{zx}(t) = \frac{1}{4} \; + \; \frac{1}{4}e^{-4\beta t} \; - \; \frac{1}{2}e^{-2(\alpha+\beta)t}, \tag{2.39}$$

the probability of observing a *transition* substitution at site i in a descendant sequence after elapsed time t.

Equations 2.38 and 2.39 can be combined to solve $p_{zz}(t) = 1 - p^s_{zx}(t) - 2p^v_{zy}(t)$. The probability of observing the same nucleotide at site i in an ancestral sequence and in a descendant sequence after elapsed time t is

$$p_{zz}(t) = \frac{1}{4} + \frac{1}{4}e^{-4\beta t} + \frac{1}{2}e^{-2(\alpha+\beta)t}. \tag{2.40}$$

This equation is analogous to Equation 2.27.

The likelihood of a pair of aligned nucleotides

When inferring the likelihood of observing a pair of nucleotides aligned at site i, we are actually considering a tree with two leaves and inferring the probability that some ancestral nucleotide z at site i evolved independently over time t into the nucleotides observed in the present-day sequences at site i. Let x and y be the observed, present-day nucleotides.

The probability of observing x in the one present-day sequence is $p_{zx}(t)$ and the probability of observing y in the other present-day species is $p_{zy}(t)$. The probability of observing x aligned with y is a product of the probabilities $p_{zx}(t)$ and $p_{zy}(t)$. Since the ancestral nucleotide z is unknown, we consider all four possibilities, weighted by the probability of observing z:

$$\Pr\binom{x}{y} = \sum_{z\in\{A,C,G,T\}} p_z \; p_{zx}(t) \; p_{zy}(t). \tag{2.41}$$

Under the K2P model, transitions and transversions are weighted differently. So, we'll consider three cases for the aligned nucleotides x and y: (1) they are the same, (2) they are from the same class (i.e., A aligned with G or C aligned with T), or (3) they are from different classes (i.e., G aligned with C).

The aligned nucleotides are the same.
In this case, we can rewrite Eqn. 2.41 as

$$\Pr\binom{x}{x} = \frac{1}{4} \sum_{z\in\{A,C,G,T\}} p_{zx}(t)^2.$$

When considering possible values of z, one will be the same as x, one will be in the same class as x, and two will be in different classes. Therefore, the probability of aligning x and x is

$$\tfrac{1}{4}\left(p_{zz}(t)^2 + p^s_{zx}(t)^2 + 2p^v_{zy}(t)^2\right)$$

Box 4: Solving the standard form equation for K2P

Recall that $\frac{df(t)}{dt} = q(t) - h(t)f(t)$ is the standard form linear differential equation, which can be solved using the formula

$$f(x) = g(x)^{-1} \int (g(x)q(x)dx) + C \cdot g(x)^{-1},$$

where $g(x) = e^{\int (p(x)dx)}$.
In our case, $f(t) = p^s_{zx}(t)$, $q(t) = \frac{1}{2}(\alpha + \beta) + \frac{1}{2}(\alpha - \beta)e^{-4\beta t}$, and $h(t) = 2(\alpha + \beta)$. Solving $g(t) = e^{\int h(t)dt}$ yields

$$g(t) = e^{\int 2(\alpha+\beta)dt} = e^{2(\alpha+\beta)t}.$$

We then need to solve $\int g(t)q(t)dt$:

$$\begin{aligned}
\int g(t)q(t)dt &= \int e^{2(\alpha+\beta)t} \left(\tfrac{1}{2}(\alpha + \beta) + \tfrac{1}{2}(\alpha - \beta)e^{-4\beta t}\right) dt \\
&= \int \tfrac{1}{2}(\alpha + \beta)e^{2(\alpha+\beta)t}dt + \int \tfrac{1}{2}(\alpha - \beta)e^{-4\beta t}e^{2(\alpha+\beta)t}dt \\
&= \frac{\alpha + \beta}{2} \int e^{2(\alpha+\beta)t}dt + \frac{\alpha - \beta}{2} \int e^{2(\alpha-\beta)t}dt \\
&= \frac{\alpha + \beta}{4(\alpha + \beta)}e^{2(\alpha+\beta)t} + \frac{\alpha - \beta}{4(\alpha - \beta)}e^{2(\alpha-\beta)t} \\
&= \tfrac{1}{4}\left(e^{2(\alpha+\beta)t} + e^{2(\alpha-\beta)t}\right).
\end{aligned}$$

Plugging these values into the standard form solution yields

$$\begin{aligned}
f(t) &= g(t)^{-1} \int g(t)q(t)dt + C \cdot g(t)^{-1} \\
p^s_{zx}(t) &= e^{-2(\alpha+\beta)t}\, \tfrac{1}{4}\left(e^{2(\alpha+\beta)t} + e^{2(\alpha-\beta)t}\right) + C \cdot e^{-2(\alpha+\beta)t} \\
&= \tfrac{1}{4}\left(e^{-2(\alpha+\beta)t}e^{2(\alpha+\beta)t} + e^{-2(\alpha+\beta)t}e^{2(\alpha-\beta)t}\right) + C \cdot e^{-2(\alpha+\beta)t} \\
&= \tfrac{1}{4}\left(1 + e^{-4(\alpha+\beta)t}\right) + C \cdot e^{-2(\alpha+\beta)t}.
\end{aligned}$$

Solving for constant C at $t = 0$, $p^s_{zx}(0) = 0 = \frac{2}{4} + C$ and $C = -\frac{1}{2}$. Therefore, $p^s_{zx}(t)$ is

$$\frac{1}{4} + \frac{1}{4}e^{-4\beta t} - \frac{1}{2}e^{-2(\alpha+\beta)t}.$$

Plugging in Eqn.s 2.38, 2.39 and 2.40, yields

$$\begin{aligned}\Pr\binom{x}{x} &= \frac{1}{4}\left(\tfrac{1}{16}(1+e^{-4\beta t}+2e^{-2(\alpha+\beta)t})^2+\tfrac{1}{16}(1+e^{-4\beta t}-2e^{-2(\alpha+\beta)t})^2+\tfrac{2}{16}(1-e^{-4\beta t})^2\right)\\ &= \frac{1}{4}\frac{1}{16}\left((1+e^{-4\beta t}+2e^{-2(\alpha+\beta)t})^2+(1+e^{-4\beta t}-2e^{-2(\alpha+\beta)t})^2+2(1-e^{-4\beta t})^2\right)\\ &= \frac{1}{16}\left(1+e^{-8\beta t}+2e^{-4(\alpha+\beta)t}\right) \qquad (2.42)\end{aligned}$$

The aligned nucleotides are in the same class.
Let's consider aligning A and G, which give us

$$\Pr\binom{A}{G} = \tfrac{1}{4}\left(p_{AA}(t)p_{AG}(t)+p_{GA}(t)p_{GG}(t)+p_{CA}(t)p_{CG}(t)+p_{TA}(t)p_{TG}(t)\right)$$

Of the four products, two of them are of the form $p_{zz}(t)p^s_{zx}(t)$ and two are of the form $p^v_{zy}(t)^2$. This is because in two cases, the ancestral nucleotide will be the same as one of the nucleotides in the alignment; since x and y are in the same class, the other present-day nucleotide will be a transition. In the other two cases, the ancestral nucleotide will be in a different class from x and y, and thus represent a transversion. Therefore, the probability of aligning x and y when x and y are in the same class is

$$\Pr\binom{x}{y} = \tfrac{1}{4}\left(2p_{zz}(t)p^s_{zx}(t)+2p^v_{zy}(t)^2\right)$$

Plugging in Eqn.s 2.38, 2.39 and 2.40, yields

$$\begin{aligned}\Pr\binom{x}{y} &= \tfrac{1}{2}\left(p_{zz}(t)p^s_{zx}(t)+p^v_{zy}(t)^2\right)\\ &= \tfrac{1}{32}\left((1+e^{-4\beta t}+2e^{-2(\alpha+\beta)t})(1+e^{-4\beta t}-2e^{-2(\alpha+\beta)t})+(1-e^{-4\beta t})^2\right)\\ &= \tfrac{1}{16}\left(1+e^{-8\beta t}-2e^{-4(\alpha+\beta)t}\right) \qquad (2.43)\end{aligned}$$

The aligned nucleotides are in different classes.
Let's consider aligning A and C, which give us

$$\Pr\binom{A}{C} = \tfrac{1}{4}\ \left(p_{AA}(t)p_{AC}(t)+p_{GA}(t)p_{GC}(t)+p_{CA}(t)p_{CC}(t)+p_{TA}(t)p_{TC}(t)\right)$$

Of the four products, two of them are of the form $p_{zz}(t)p^v_{zy}(t)$ and two are of the form $p^s_{zx}(t)p^v_{zy}(t)$. This is because in two cases, the ancestral nucleotide will be the same as one of the nucleotides in the alignment; since x and y are in different classes, the other present-day

nucleotide will be a transversion. In the other two cases, the ancestral nucleotide will be in the same class as one present-day nucleotide (i.e., a transition) but in a different class from the other present-day nucleotide (i.e., a transversion). Therefore, the probability of aligning x and y when x and y are in different classes is

$$\Pr\begin{pmatrix} x \\ y \end{pmatrix} = \tfrac{1}{4}\left(2p_{zz}(t)p^v_{zy}(t) + 2p^s_{zx}(t)p^v_{zy}(t)\right)$$

Plugging in Eqn.s 2.38, 2.39 and 2.40, yields

$$\begin{aligned}
\Pr\begin{pmatrix} x \\ y \end{pmatrix} &= \tfrac{1}{2}p^v_{zy}(t)\left(p_{zz}(t) + p^s_{zx}(t)\right) \\
&= \tfrac{1}{32}(1-e^{-4\beta t})\left(1 + e^{-4\beta t} + 2e^{-2(\alpha+\beta)t} + 1 + e^{-4\beta t} - 2e^{-2(\alpha+\beta)t}\right) \\
&= \tfrac{1}{16}(1-e^{-4\beta t})\left(1 + e^{-4\beta t}\right) \\
&= \tfrac{1}{16}\left(1 - e^{-8\beta t}\right) \qquad (2.44)
\end{aligned}$$

Correcting for multiple substitutions.

With the JC model, we calculated the expected number of substitutions per site based on the probability of a mismatch. In the K2P model, we also have two mismatch probabilities, one for transitions and one for transversions. Recall that Eqn.s 2.42, 2.43, and 2.44 give the probabilities of observing a specific alignment of nucleotides, based on whether the nucleotides aligned at site i are the the same, in the same class, or in different classes, respectively. We can use these equations to calculate the probabilities of a match and the two types of mismatches in general. In particular, there are four ways to have a match, four ways to have a mismatch within the same class, and 8 ways to have a mismatch across different classes.

Given an alignment of two sequences that have been diverging for time t from a common ancestor and have since been evolving according to the K2P model, the probability of observing a match (i.e., `AA`, `GG`, `CC`, `TT`) at a given site i is

$$P_M = \frac{1}{4}\left(1 \; + \; 2e^{-4(\alpha+\beta)t} \; + \; e^{-8\beta t}\right). \qquad (2.45)$$

The probability of observing a transition (i.e., `AG`, `GA`, `CT`, `TC`) at a given site i is

$$P^s_m = \frac{1}{4}\left(1 \; - \; 2e^{-4(\alpha+\beta)t} \; + \; e^{-8\beta t}\right). \qquad (2.46)$$

The probability of observing a transversion at site i is given by

$$P^v_m = \frac{1}{2}\left(1 \; - \; e^{-8\beta t}\right). \qquad (2.47)$$

The expected number of substitutions is $E = 2\lambda t = 2(\alpha + 2\beta)t$, with the expected number of transitions $E^s = 2\alpha t$ and the expected number of transversions $E^v = 4\beta t$. We can solve for βt and αt from Eqn.s 2.46 and 2.47.

$$\begin{aligned} 2P_m^v &= 1 - e^{-8\beta t} \\ 1 - 2P_m^v &= e^{-8\beta t} \\ -8\beta t &= \ln[1 - 2P_m^v] \\ \beta t &= -\frac{1}{8}\ln[1 - 2P_m^v] \end{aligned}$$

$$\begin{aligned} 4P_m^s &= 1 - 2e^{-4(\alpha+\beta)t} + 1 - 2P_m^v \\ 4P_m^s + 2P_m^v - 2 &= -2e^{-4(\alpha+\beta)t} \\ 1 - 2P_m^s - P_m^v &= e^{-4(\alpha+\beta)t} \\ -4(\alpha+\beta)t &= \ln[1 - 2P_m^s - P_m^v] \\ \alpha t + \beta t &= -\frac{1}{4}\ln[1 - 2P_m^s - P_m^v] \end{aligned}$$

$$\begin{aligned} \alpha t &= \tfrac{1}{8}\ln[1 - 2P_m^v] - \tfrac{1}{4}\ln[1 - 2P_m^s - P_m^v] \\ \alpha t &= \frac{1}{8}\left(\ln[1 - 2P_m^v] - 2\ln[1 - 2P_m^s - P_m^v]\right). \end{aligned}$$

Given an alignment of length n, with m_s transitions and m_v transversions, the expected numbers of transitions and transversions that actually occurred is approximately

$$\frac{1}{4}\left[\ln\left(1 - \frac{2m_v}{n}\right) - 2\ln\left(1 - \frac{2m_s}{n} - \frac{m_v}{n}\right)\right] \cdot n \tag{2.48}$$

and

$$-\frac{1}{2}\ln\left(1 - \frac{2m_v}{n}\right) \cdot n, \tag{2.49}$$

respectively. Summing these two quantities, we obtain the expected number of substitutions of all types:

$$-\frac{1}{4}\left[\ln\left(1 - \frac{2m_v}{n}\right) + 2\ln\left(1 - \frac{2m_s}{n} - \frac{m_v}{n}\right)\right] \cdot n. \tag{2.50}$$

Chapter 3

Amino Acid Substitution Matrices

In prior lectures, we introduced Markov models of nucleotide substitution. We derived expressions for the probability that nucleotide x will change to nucleotide y after elapsed time t. Further, we used the model to account for multiple substitutions, by estimating the number of actual substitutions that occurred, given the number of observed mismatches.

Here, we focus on Markov models of amino acid replacement and their use in deriving amino acid substitution matrices. An amino acid substitution matrix assigns a score to a pair of aligned amino acids, x and y. A good substitution matrix should have the following properties:

- *Evolutionary divergence:* The substitution matrix should be appropriate for the degree of evolutionary divergence of the proteins under consideration. The observation of identical or functionally similar amino acids at the same site is more surprising in highly diverged protein families than in families characterized by little sequence divergence. The best results are obtained using a substitution matrix based on amino acid replacement frequencies that are typical of the protein family. Therefore, a set of matrices that is parameterized by sequence divergence is desired.

- *Multiple substitutions:* The score associated with an amino acid pair, x and y, should reflect the probability of observing x aligned with y, taking into account the possibility of multiple replacements at the same site.

- *Biophysical properties of residues:* Amino acids differ in size and charge. Some are acidic, some are basic, some have aromatic side chains. Generally, replacement of an amino acid with another amino acid with similar properties is less likely to cause dramatic changes in function than replacement with an amino acid with different properties. A substitution matrix should reflect this.

There are several families of amino acid substitution matrices that have these properties. Two that are widely used are the PAM matrices (Dayhoff *et al.*, 1978) and the BLOSUM

matrices (Henikoff and Henikoff, 1992.) Both of these families of substitution matrices are parameterized by sequence divergence. The PAM matrices account for evolutionary divergence using a formal Markov model of sequence evolution. The BLOSUM matrices use an *ad hoc* approach. Although the details differ, both matrix families were derived according to the following general strategy:

1. Use a "trusted" set of ungapped, multiple sequence alignments to infer model parameters.
2. Count observed amino acid pairs in the trusted alignments, correcting for sample bias.
3. Estimate substitution frequencies from amino acid pair counts.
4. Construct a log likelihood scoring matrix from substitution frequencies.

3.1 A log likelihood ratio framework for scoring alignments

Before introducing the PAM and BLOSUM matrices, we briefly introduce the log likelihood framework in which these matrices were developed. Suppose $\alpha(s_1, s_2)$ is an ungapped alignment of sequences s_1 and s_2 of length n[1]. Under the assumption of positional independence, we can assign a similarity score to $\alpha(s_1, s_2)$ by adding the similarities of the symbols in each position in the alignment,

$$\mathcal{S} = \sum_{i=1}^{n} p(s_1[i], s_2[i]), \tag{3.1}$$

where $p(x, y)$ is a quantitative measure of the similarity of x and y. Recall that earlier in the semester, we used a simple scoring scheme with a single match score, $p(x, x) = M$, $\forall x \in \Sigma$, and a single mismatch score, $p(x, y) = m$, $\forall x, y \in \Sigma$, where $x \neq y$. Since all matches (respectively, mismatches) have the same score, with this scoring scheme

$$\mathcal{S} = n_m \cdot m + (n - n_m) \cdot M,$$

where n_m is the number of mismatches in α.

This simple scoring scheme has limitations, especially for amino acids. First, since all mismatches are assigned the same score, it cannot reflect differences in the biochemical similarity of various amino acid pairs. Second, if M and m are chosen arbitrarily, then alignment scores have no intuitive meaning in an absolute sense. For example, if I tell you that a given alignment has a score of 14, you know that it is better than some other alignment that has a score of 12, but you have no way of assessing whether the alignment is inherently good or bad.

[1]We only consider ungapped alignments in this chapter, because substitution matrices do not model insertions and deletions.

Third, this scoring scheme does not take the evolutionary divergence of s_1 and s_2 into account. If we are testing the hypothesis that s_1 and s_2 are related and have changed very little since they diverged from their common ancestor, then we might interpret any mismatch as evidence that s_1 and s_2 are unrelated, even if the mismatch is a conservative replacement (i.e., involves amino acids with similar biochemical properties). In contrast, if we are testing the hypothesis that s_1 and s_2 are related and have changed a great deal since their divergence, then we might interpret mismatches that represent conservative replacements as evidence s_1 and s_2 are indeed related. In order to capture these nuances, we require a scoring method that is parameterized by evolutionary divergence.

One way of assessing whether an alignment is good in an absolute sense is to ask whether $\alpha(s_1, s_2)$ reflects more similarity than we expect to see by chance. Let H_0 be the null hypothesis that s_1 and s_2 are unrelated sequences. The alternate hypothesis, H_A, is that s_1 and s_2 are related sequences with a given amount of evolutionary divergence. We can assess whether $\alpha(s_1, s_2)$ reflects more than chance similarity by calculating the ratio of the probabilities of the alignment under H_A and H_0:

$$\mathcal{LR}(\alpha) = \frac{p(\alpha|H_A)}{p(\alpha|H_0)}. \tag{3.2}$$

This *likelihood ratio* will be less than one, if the alignment of s_1 and s_2 represents less similarity than expected by chance, and greater than 1, if the alignment represents is more similarity than expected by chance. If the ratio is much greater than 1, then we have strong evidence that the sequences share common ancestry.

Under the assumption of positional independence, the probability of the alignment is equivalent to the product of probabilities of the individual positions in the alignment

$$\mathcal{LR}(\alpha) = \frac{\prod_{i=1}^{n} p(\alpha[i]|H_A)}{\prod_{i=1}^{n} p(\alpha[i]|H_0)}, \tag{3.3}$$

where $\alpha[i]$ is the alignment of $s_1[i]$ and $s_2[i]$. This formulation provides a way to assess alignments based on the probabilities of individual amino acid pairs in the alignment. (Recall that α is an ungapped alignment.) However, it requires calculating the product of a sequence of numbers between 0 and 1, with the concomitant challenge of working with smaller and smaller numbers as the length of the alignment increases.

This problem can be addressed by calculating the *log* of the likelihood ratio, instead of the likelihood ratio, itself. Note that since $\log(x)$ increases monotonically with x, the alignment that maximizes $\mathcal{LR}(\alpha)$, also maximizes $\log \mathcal{LR}(\alpha)$. Thus, $\log \mathcal{LR}(\alpha)$ can also be used to assess the extent to which $\alpha(s_1, s_2)$ represents more than chance similarity. Taking

the log of both sides of Equation 3.3 yields

$$\log \mathcal{LR}(\alpha) = \log \frac{\prod_{i=1}^{n} p(\alpha[i]|H_A)}{\prod_{i=1}^{n} p(\alpha[i]|H_0)}, \tag{3.4}$$

$$= \log \prod_{i=1}^{n} \frac{p(\alpha[i]|H_A)}{p(\alpha[i]|H_0)} \tag{3.5}$$

$$= \sum_{i=1}^{N} \log \frac{p(\alpha[i]|H_A)}{p(\alpha[i]|H_0)}. \tag{3.6}$$

Since $\mathcal{LR}(\alpha)$ is non-negative, $\log \mathcal{LR}(\alpha)$ ranges from $-\infty$ to ∞. If $\log \mathcal{LR}(\alpha) > 0$, then $\alpha(s_1, s_2)$ reflects more similarity than expected by chance; if $\log \mathcal{LR}(\alpha) < 0$, then $\alpha(s_1, s_2)$ reflects less similarity than expected.

The right hand side of this equation looks very similar to the right hand side of Equation 3.1: in both cases, we have a sum of values, one for each position in the alignment. By defining the similarity score of x aligned with y to be

$$p(x, y) = \log \frac{p(\binom{x}{y}|H_A)}{p(\binom{x}{y}|H_0)},$$

we obtain an alignment score that is equivalent to the log of the ratio of the probabilities of that alignment under the alternate and null hypotheses:

$$\mathcal{S} = \log \mathcal{LR}(\alpha).$$

This yields a scoring scheme that has a natural, biological interpretation, that can be adjusted to account for evolutionary divergence, and that can be interpreted in an absolute, as well as a relative, context.

To define similarity scores in this way, requires estimates of $p(\binom{x}{y}|H_A)$ and $p(\binom{x}{y}|H_0)$, $x, y \in \Sigma,$, for a range of evolutionary distances. For amino acid substitution matrices, these quantities are estimated from trusted amino acid alignments. In the following sections, we discuss how amino acid pair probabilities are estimated in derivation of the PAM matrices and the BLOSUM matrices.

3.2 PAM matrices

In 1978, Margaret Dayhoff and her colleagues developed a family of substitution matrices that are parameterized by PAM distance, a unit of evolutionary divergence. The term "PAM" is an abbreviation of "percent accepted mutation." The divergence between two sequences is N PAMs, if, on average, N amino acid replacements (possibly at the same site)

per 100 residues occurred since their separation. Note that this is distinct from percent identity, which reflects the number of matches per 100 residues.

The derivation of these matrices requires estimating amino acid pair frequencies in sequences that are diverged by N PAMs, for a range of values of N. Given alignments of sequences that are separated by N PAMs, amino acid pair frequencies can be estimated simply by tabulating the number of instances of each amino acid pair in those alignments. However, it is not clear how to obtain such alignments, because determining the PAM distance associated with a given alignment is not straightforward. The number of *mismatches* can easily be determined by inspection, but inferring the number of *replacements* that occurred requires a method for estimating multiple replacements at the same site. To address this problem, Dayhoff first constructed a model of amino acid replacement using alignments with high levels of sequence similarity, in which multiple substitutions at the same site are unlikely. She then used higher-order Markov models to obtain models of amino acid replacements in more diverged sequences.

Dayhoff developed this model using the four step approach described above. Specifically:

Step 1: Training data. As training data, Dayhoff *et al* used a set of ungapped, global multiple sequence alignments of 71 groups of closely related sequences. Within each group, the sequence identity was 85% or greater. The rationale is that sequences with at least 85% identity will contain no site that has sustained more than one mutation.

Step 2: Count amino acid pairs. Observed amino acid pair frequencies were tabulated from the 71 multiple alignments. Sample bias was corrected by counting the minimum number of changes required to fit the data to a tree. This requires inferring the unrooted tree that describes the evolutionary relationships between the sequences in each aligned family and then estimating the number of amino acid replacements that occurred on each branch of that tree.

We will demonstrate how this works in practice using the following alignment of four amino acid sequences of length four:

```
1:  AEIR
2:  DEIR
3:  QKLH
4:  AHLH
```

For an alignment with four sequences, there are three unrooted trees with four leaves, shown in Fig. 3.1. Tree I corresponds to the hypothesis that Sequence (1) is more closely related to Sequence (2) than to either Sequence (3) or Sequence (4). According to Tree II, Sequence (1) and Sequence (3) are most closely related, while Tree III says that Sequence (1) and Sequence (4) are closest. For each tree, the leaves are annotated with the corresponding

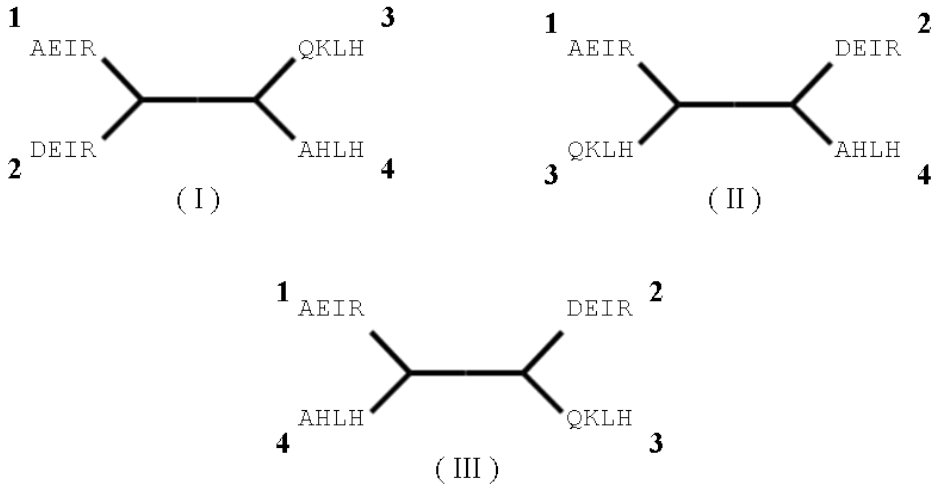


Figure 3.1: Three unrooted trees representing the three possible hypotheses for evolution of four sequences. The leaves of each tree are labeled with the corresponding present-day sequences. The internal nodes are not labeled; the sequences associated with internal nodes correspond to ancestral sequences and are unknown.

present-day sequences. The sequences on internal nodes are unknown, since they correspond to ancestral sequences.

First, we will illustrate how to estimate the number of substitutions for a given evolutionary tree. Then, we will return to the question of how to infer the tree that best explains a given alignment. Dayhoff inferred the sequences on the internal nodes according to the *parsimony criterion*, which states that the best hypothesis is the one that requires the fewest amino acid replacements to explain the data. Consistent with this criterion, sequences were assigned to the internal nodes of each tree in such a way that the total number of changes along branches of the tree is minimized.

For example, suppose that we have determined that Tree I is the best hypothesis for the evolutionary history of the four sequences in the alignment. Ancestral sequences that satisfy the parsimony criterion for Tree I are shown in Fig. 3.2 (and Fig. 3.4). With these ancestral sequences, six substitutions (shown on their respective branches) are required to explain the evolution of the four present day sequences. Convince yourself that there is no assignment of labels to the internal nodes that allows for fewer than six substitutions.

Once ancestral sequences have been inferred, the counts for each amino acid pair are tabulated. A_{xy}, the number of x,y pairs observed, is determined by counting the number of edges connecting x and y, for $x \neq y$. Note that $A_{xy} = A_{yx}$, since every edge connecting x

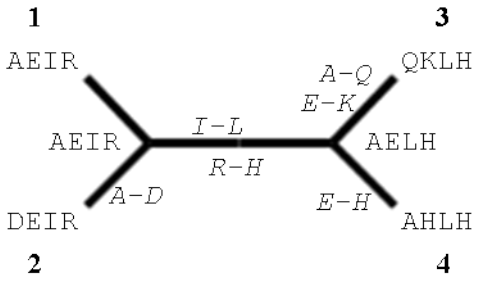


Figure 3.2: Tree I from Fig. 3.1 with ancestral sequences inferred according to the parsimony criterion. Six amino acid replacements, shown on the branches of the tree, are required to explain the present day sequences. This set of most parsimonious ancestral sequences is not unique. There are two other most parsimonious hypotheses for the ancestral sequences, shown in Fig. 3.4.

with y also connects y with x. A_{xx} is defined to be twice the number of edges connecting x and x. This is because the edges connecting two dissimilar residues are also counted twice, once in the xy direction and once in the yx direction. For example, there are 6 `EE` pairs in Fig. 3.2: Two counts are contributed by each of the three edges connecting `AEIR` and `AEIR`, `AEIR` and `DEIR`, and `AEIR` and `AELH`. The tabulated counts for all amino acid pairs are given in the table in Fig. 3.3.

In general, there can be more than one way to assign sequences to internal nodes such that the total change is minimized. Each most parsimonious set of internal node labels will result in different amino acid pair counts. In our example, there are two additional

	A	D	E	H	I	K	L	Q	R
A	6	1						1	
D	1								
E			6	1		1			
H			1	4					1
I					4		1		
K			1						
L					1		4		
Q	1								
R				1					4

Figure 3.3: Amino acid pair counts derived according to Dayhoff's counting scheme from the tree in Fig. 3.2. Only amino acids that are present in at least one sequence are shown in the table.

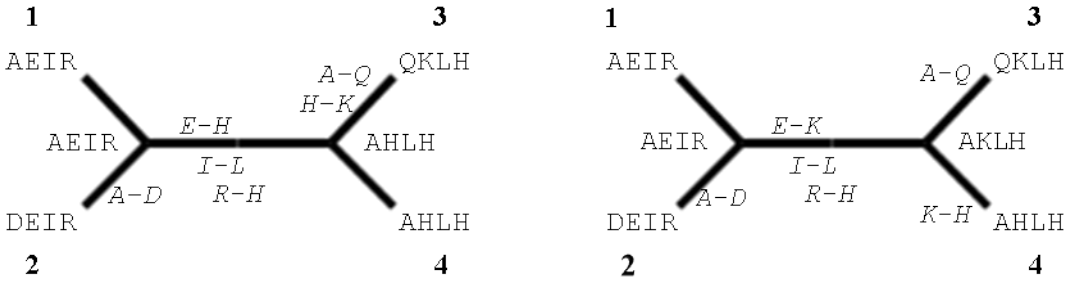


Figure 3.4: Two other sets of most parsimonious ancestral sequences for Tree I from Fig. 3.1. The associated amino acid replacements are shown on the branches of the tree.

assignments of ancestral sequences for which six substitutions are sufficient to explain the present-day sequences, shown in Fig. 3.4. The pair counts resulting from these two alternate sets of labels are given in the tables in Fig. 3.5. Since there is no way of knowing which set of inferred ancestral sequences is the best estimate, all possibilities must be considered. Dayhoff does this by averaging the counts over all most parsimonious labelings. For our example, the matrix in Fig. 3.6 shows the average of the pair counts in Figs. 3.3 and 3.5.

Comparison of the amino acid pairs in each column in the multiple alignment on page 61 with the pair counts derived from the trees in Figs. 3.2 and 3.4 demonstrates how this approach compensates for sample bias and leads to different amino acid pair statistics. If we derived amino acid pairs directly from the alignment, each sequence would be compared to three other sequences, effectively counting the replacement of the same amino acid more than once. In contrast, when counting amino acid pairs on a tree, each sequence is compared to one other sequence, i.e., the inferred ancestral sequence. For example, since `D` and `Q`

	A	D	E	H	I	K	L	Q	R
A	6	1						1	
D	1								
E			4	1					
H			1	6		1			1
I					4		1		
K				1					
L					1		4		
Q	1								
R				1					4

	A	D	E	H	I	K	L	Q	R
A	6	1						1	
D	1								
E			4			1			
H				4		1			1
I					4		1		
K			1	1		2			
L					1		4		
Q	1								
R				1					4

Figure 3.5: Amino acid pair counts derived according to Dayhoff's counting scheme from the trees in Fig. 3.4.

	A	D	E	H	I	K	L	Q	R
A	6	1						1	
D	1								
E			4.7	1		0.7			
H			1	4.7		0.7			1
I					4		1		
K			0.7	0.7		0.7			
L					1		4		
Q	1								
R				1					4

Figure 3.6: Average amino acid pair counts over all most parsimonious labelings of Tree I. Each entry represents the mean of the corresponding entries in the tables in Figs. 3.3 and 3.5.

both appear in the first column of the alignment, obtaining amino acid pair counts directly from the alignment would result in a non-zero value of A_{DQ}. However, no `D-Q` replacement appears on the branches of the labeled trees in Figs. 3.2 and 3.4 and $A_{DQ} = 0$ in the table in Fig. 3.6.

Having demonstrated how to infer ancestral sequences for a given evolutionary tree, we return to the question of how to infer the tree that is the best hypothesis for the aligned sequences. For a given tree, the minimum number of changes required to explain the present day sequences, over all possible internal labelings, is called the *parsimony score* of that tree. In our example, Tree I has a parsimony score of six. This score provides a basis for recovering the tree that explains the data according to the parsimony criterion. Given an alignment of a family of k sequences, all unrooted trees with k leaves are considered. The parsimony score is estimated for each tree and the tree(s) with the lowest parsimony score are reported.

In general, there can be more than one most parsimonious tree for a given set of present-day sequences, although for this example there is only one. (Convince yourself that there is no assignment of sequences to the internal nodes of Tree II that requires fewer than seven replacements. Check that the same is true of Tree III.) Having found the set of most parsimonious trees, Dayhoff estimated amino acid pair frequencies by averaging the counts over all most parsimonious labelings of all most parsimonious trees, yielding

$$A_{xy} = \frac{1}{n_T} \sum_T A_{xy}^T,$$

where n_T is the number of labeled trees with an optimal parsimony score and T is an indicator variable that enumerates such trees.

Step 3: Estimate substitution frequencies. To estimate substitution frequencies from amino acid pair counts, Dayhoff constructed a family of Markov models representing evolution at a single site, i, in an amino acid sequence. (Note that this model assumes site independence.) All models in the family have twenty states, one for each amino acid. If the model visits state x at time t, we say that the amino acid at site i was an x at time t. The models differ in their transmission probability matrices, which reflect the propensity for amino acid replacement at various evolutionary divergences.

Dayhoff derived $P_{xy}^{(1)}$, the transition matrix for the 1 PAM model, from closely related alignments that may be assumed to contain no multiple substitutions. $P_{xy}^{(1)}$ is the probability that amino acid x will be replaced by amino acid y in sequences separated by 1 PAM of evolutionary distance. Next, Dayhoff derived the PAM-N transition matrix, $P_{xy}^{(N)}$, by extrapolating from the PAM-1 transition probability, as described in detail below.

The transition matrix $P_{xy}^{(1)}$ is derived from the counts, A_{xy}, obtained in step 2, as follows:

$$P_{xy}^{(1)} = m_x \frac{A_{xy}}{\sum_{h \neq x} A_{xh}}, \quad x \neq y \tag{3.7}$$

$$P_{xx}^{(1)} = 1 - m_x \tag{3.8}$$

Here, m_x is the "mutability" of amino acid x and is defined to be

$$m_x = \frac{1}{L p_x z} \sum_{l \neq x} A_{xl}, \tag{3.9}$$

where p_x is the background frequency of x, L is the length of the alignment, and z is a scaling that guarantees that the transition matrix will correspond to exactly 1 PAM. We select the scaling factor, z, so that

$$\sum_{x=1}^{20} (p_x m_x) = \frac{1}{100}. \tag{3.10}$$

This scaling factor is required because although the training alignments are sufficiently conserved to contain no multiple substitutions, the frequency of replacements in each alignment may not be exactly one in a hundred.

We obtain an expression for the scaling factor, z, by substituting the right hand side of Equation 3.9 for m_x in Equation 3.10 and solving for z. This yields

$$z = \frac{100}{L} \sum_{x=1}^{20} \sum_{l \neq x} A_{xl}. \tag{3.11}$$

We now replace the z in Equation 3.9 with the right hand side of Equation 3.11 to obtain the mutability of x,

$$m_x = \frac{0.01}{p_x} \frac{\sum_{l \neq x} A_{xl}}{\sum_h \sum_{l \neq h} A_{hl}}.$$

Substituting the expression for m_x into the right hand side of Equation 3.7, we obtain the PAM1 transition probability

$$P_{xy}^{(1)} = \frac{0.01}{p_x} \frac{A_{xy}}{\sum_h \sum_{l \neq h} A_{hl}}.$$

Note that $P_{xy}^{(1)}$ in Equation 3.7 is consistent with the definition of a Markov chain: the rows of the transition matrix sum to 1 and it is history independent. This Markov chain is finite, aperiodic and irreducible ("connected"). Therefore, it has a stationary distribution.

We now derive the PAM-2 transition matrix. Note that the residue at site i can change from x to y in two time steps via several state paths: $x \to x \to y$, $x \to y \to y$, or $x \to l \to y$, where l is a third amino acid, not equal to x or y. Recall that the probability of changing from x to y in two time steps is

$$P_{xy}^{(2)} = \sum_l P_{xl}^{(1)} P_{ly}^{(1)}.$$

$P^{(2)}$ can be derived by squaring the matrix $P^{(1)}$ by matrix multiplication. This is the 2-step transition probability that models amino acid replacements that occur in two time steps. Similarly, we can use matrix multiplication to derive the PAM-N, N-step transition matrix for any $N \geq 2$ as follows:

$$P^{(N)} = \left(P^{(1)}\right)^N.$$

Step 4: Construct a log likelihood scoring matrix. We obtain a log likelihood scoring matrix from the transition probability matrix as follows. Let $q_{xy}^{(N)} = p_x P_{xy}^{(N)}$ be the probability that we see amino acid x aligned with amino acid y at a given position in an alignment of sequences with N PAMs of divergence; i.e., that amino acid x has been replaced by amino acid y after N PAMs of mutational change. Then, we define the PAM-N scoring matrix to be

$$\begin{aligned} S^N[x,y] &= \lambda \log \frac{q_{xy}^{(N)}}{p_x p_y} \\ &= \lambda \log \frac{P_{xy}^{(N)}}{p_y}, \end{aligned} \tag{3.12}$$

where λ is a constant chosen to scale the matrix to a convenient range. Typically $\lambda = 10$ and the entries of S^N are rounded to the nearest integer. Note that Equation 3.12 is a log likelihood ratio, where $q_{xy}^{(N)}$ is the probability of seeing x and y aligned under the alternate hypothesis that x and y share common ancestry with divergence N and $p_x p_y$ is the probability that x and y are aligned by chance.

It is easy to verify that the PAM-N transition matrix is not symmetric; that is, $P_{xy}^{(N)} \neq P_{yx}^{(N)}$. This makes sense since replacing amino acid x with amino acid y may have different consequences than replacing y with x. In contrast, the substitution matrix *is* symmetric; that is, $S^N[x, y] = S^N[y, x]$. This makes sense because in an alignment, we cannot determine direction of evolution, so we assign the same score when x is aligned with y, and when y is aligned with x.

3.3 BLOSUM Matrices

The BLOSUM (BLOck SUbstitution Matrices) matrices were derived by Steven and Jorja Henikoff in 1992[2]. BLOSUM matrix construction follows the general framework used for constructing the PAM matrices, but differs in the details of the individual steps.

Step 1: Training data. The BLOSUM matrices are based on a much larger data set than the PAM matrices. They were constructed using conserved local alignments or "blocks," rather than global alignments of very closely related sequences. The *"trusted"* alignments used to construct the BLOSUM matrices consisted of roughly 2000 blocks of conserved regions representing 500+ groups of proteins.

The full BLOSUM procedure combines information from many aligned blocks. Below, we first present the procedure for constructing a substitution matrix in the BLOSUM framework from a single aligned block. We then discuss how to extend this to multiple blocks.

Step 2: Count amino acid pairs. In the BLOSUM framework, the sequences in each block are grouped into clusters, such that sequences that belong to different clusters are always less than N% identical. Clustering with different values of N, ranging from 45% to 90%, produces a parameterized set of matrices representing different degrees of sequence divergence. Next, for every column in every pair of clusters, amino acid pairs consisting of one amino acid from each cluster are tabulated. Pairs of amino acids within the same cluster are ignored. The resulting amino acid pair counts are normalized by cluster size so that all clusters contribute equally to the pair statistics.

The clustering step in BLOSUM matrix construction has two purposes: parameterizing evolutionary divergence and accounting for sample bias. First, restricting the tabulation

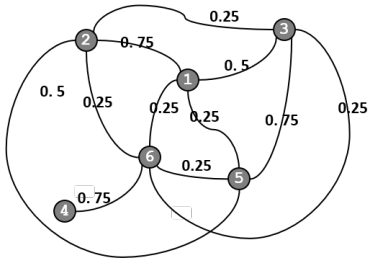
[2] *Amino acid substitution matrices from protein blocks*, PNAS, 1992 Nov 15;89(22):10915-9

1: KKRK
2: KKKK
3: KNRN
4: NRNR
5: KNKN
6: KRNR

(a)

	2	3	4	5	6
1	75	50	0	25	25
2		25	0	50	25
3			0	75	25
4				0	75
5					25

(b)



(c)

Figure 3.7: A BLOSUM example. **(a)** A block in a hypothetical multiple sequence alignment. **(b)** Matrix showing the percent identity of all pairs of sequences in (a). **(c)** Graph representation of the percent identity matrix. Nodes correspond to sequences. Each edge is labeled with the percent identity of the sequences corresponding to the nodes connected by the edge. Edges with weight zero are not shown.

to sequences from different clusters ensures that the matrix is constructed from amino acid pairs observed in alignments of sequences with a particular divergence (i.e., sequences that are less than $N\%$ identical). Second, the contribution of each residue in a cluster is normalized by the number of sequences in that cluster. This controls for sample bias, because with normalization each cluster contributes the same amount of information to the estimation of amino acid pair frequencies, even though clusters may contain different numbers of sequences. The specific procedure for obtaining amino acid pair counts is as follows:

Partitioning sequences into clusters with $N\%$ identity: The clustering step takes as input a block of k sequences of length L (no gaps) and generates n_C non-overlapping clusters. The ith cluster, C_i, has k_i sequences of length L, such that $k = \sum k_i$. The sequences in the block are partitioned in such a way that every sequence in a cluster is at least $N\%$ identical to at least one other sequence in the cluster. The sequences in the block are partitioned in such a way that the percentage of sites with identical residues in an alignment of sequences from different clusters will always be less than $N\%$.

One way to obtain such a clustering is to represent the block as a weighted graph, where the nodes correspond to sequences. Each pair of nodes is connected by an edge that is weighted by the percent identity of the associated sequences. To obtain clusters with an $N\%$ identity threshold, all edges with weights less than $N\%$ are removed. Each connected component in the resulting graph corresponds to a cluster. Note that this approach yields clusters in which every sequence in a cluster is at least $N\%$ identical to at least one other sequence in the cluster. It does not guarantee that every pair of sequences in the same cluster will be at least $N\%$ identical. The construction of the percent identity graph for a

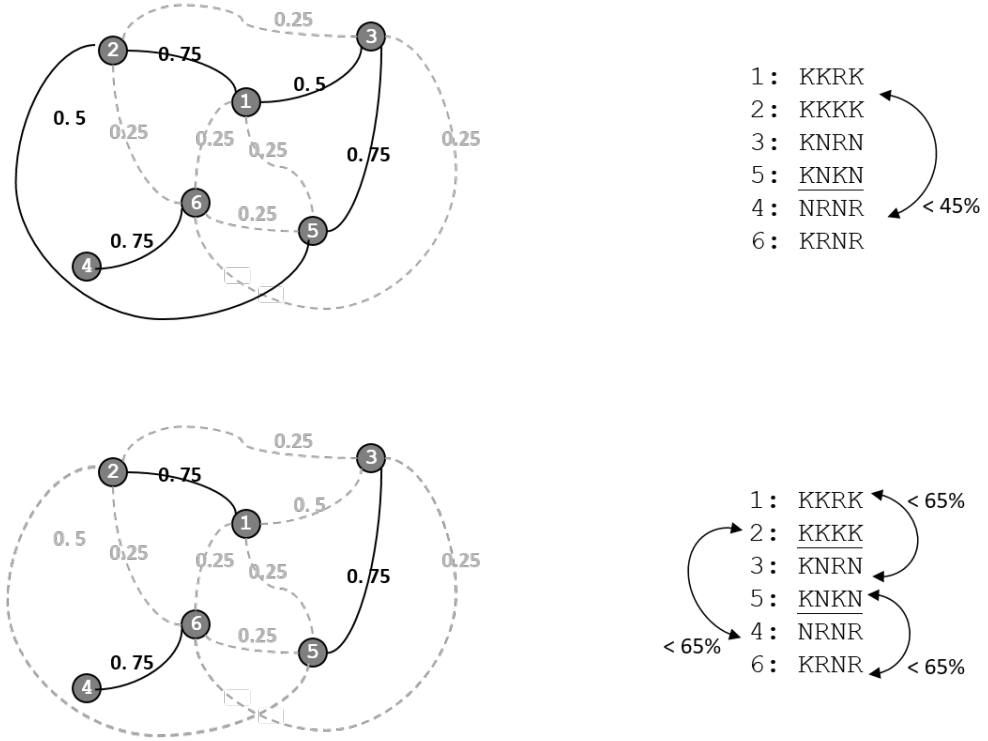


Figure 3.8: Clustering sequences with an $N\%$ identity threshold. (Top: $N = 45\%$) All edges with weight less than 45% are removed from the graph, shown at left, resulting in two clusters of size $k_1 = 4$ and $k_2 = 2$ respectively. These are depicted as sub-alignments in the sequence block at right. (Bottom: $N = 65\%$) Removing all edges with weight less than 65% results in three connected components. The corresponding sub-alignments are shown in the block at right.

block of aligned sequences is shown in in Fig. 3.7. Fig. 3.8 illustrates the use of the resulting percent identity graph for clustering sequences.

If the weight of every edge is less than N, then all edges are removed from the graph and each sequence constitutes a separate cluster. If the weight of every edge is greater than N, then no edges are removed, resulting in a single cluster containing all sequences in the block. Since at least two clusters are needed to obtain amino acid pair counts, it is not possible to construct a BLOSUM matrix when N is lower than the minimum edge weight.

Amino acid pair counts: Following the clustering step, the number of instances of amino acid x aligned with amino acid y is calculated as follows. For each pair of clusters, C_i and C_j, we determine the number of x, y and y, x pairs, where x and y are in the same column, but in different clusters. Let $\mathcal{N}_l(C_i, x)$ be the number of times that residue x appears in column l of cluster C_i. Then, the total number of pairs in column l involving an x in one cluster and a y in the other cluster is

$$\mathcal{N}_l(C_i, x) \cdot \mathcal{N}_l(C_j, y) + \mathcal{N}_l(C_i, y) \cdot \mathcal{N}_l(C_j, x).$$

The above expression gives a raw pair count, which is then adjusted to control for differences in cluster sizes. Suppose clusters C_i and C_j contain k_i and k_j sequences, respectively. Then, the adjusted contribution of column l in clusters C_i and C_j to the pair count for x and y is

$$\frac{\mathcal{N}_l(C_i, x) \cdot \mathcal{N}_l(C_j, y) + \mathcal{N}_l(C_i, y) \cdot \mathcal{N}_l(C_j, x)}{k_i \cdot k_j}.$$

To obtain the total x, y pair count from this block, we sum over all pairs of clusters and over all columns, yielding

$$A^N_{xy} = \sum_{i=1}^{n_C - 1} \sum_{j=i+1}^{n_C} \sum_{l=1}^{L} \frac{\mathcal{N}_l(C_i, x) \cdot \mathcal{N}_l(C_j, y) + \mathcal{N}_l(C_i, y) \cdot \mathcal{N}_l(C_j, x)}{k_i \cdot k_j}, \tag{3.13}$$

where $x \neq y$. The superscript N indicates that these are pair counts for a BLOSUM-N matrix, where N is the threshold used in the clustering. When $x = y$, the pairs are only counted in one direction:

$$A^N_{xx} = \sum_{i=1}^{n_C - 1} \sum_{j=i+1}^{n_C} \sum_{l=1}^{L} \frac{\mathcal{N}_l(C_i, x) \cdot \mathcal{N}_l(C_j, x)}{k_i \cdot k_j} \tag{3.14}$$

As an illustration of this calculation, consider the block in Fig. 3.7. Clustering with a 45% threshold results in $n_C = 2$ clusters of size $k_1 = 4$ and $k_2 = 2$, respectively (Fig. 3.8, top). In the first column of the clustered alignment, there are four `K`'s in cluster C_1 and one `K` in cluster C_2: $\mathcal{N}_1(C_1, \texttt{K}) = 4$ and $\mathcal{N}_1(C_2, \texttt{K}) = 1$. Since there are only two clusters when $N = 45$, the two outer sums in Equation 3.14 reduce to a single case ($i = 1, j = 2$). Further, column 1 is the only column in which `K` appears in both clusters. Since $\mathcal{N}_l(C_2, \texttt{K}) = 0, l \in \{2, 3, 4\}$, there is only one non-zero term in the inner sum. This yields

$$\begin{aligned} A^{45}_{\texttt{KK}} &= \frac{\mathcal{N}_1(C_1, \texttt{K}) \cdot \mathcal{N}_1(C_2, \texttt{K})}{k_1 \cdot k_2} \\ &= \frac{4 \cdot 1}{4 \cdot 2} \\ &= 0.5. \end{aligned}$$

The calculation for different amino acids is somewhat more complex. As an example, we calculate $A^{45}_{\mathtt{KN}}$. In the first column of the clustered alignment in Fig. 3.8 (top),

$$\mathcal{N}_1(C_1,\mathtt{K}) = 4, \qquad \mathcal{N}_1(C_2,\mathtt{N}) = 1 \qquad \mathcal{N}_1(C_1,\mathtt{N}) = 0 \qquad \mathcal{N}_1(C_2,\mathtt{K}) = 1.$$

In the third column,

$$\mathcal{N}_3(C_1,\mathtt{K}) = 2, \qquad \mathcal{N}_3(C_2,\mathtt{N}) = 2, \qquad \mathcal{N}_3(C_1,\mathtt{N}) = 0, \qquad \mathcal{N}_3(C_2,\mathtt{K}) = 0.$$

Since neither K nor N appear in Cluster 2 in the 2nd or 4th column (i.e., $\mathcal{N}_2(C_2,\mathtt{K}) = \mathcal{N}_2(C_2,\mathtt{N}) = \mathcal{N}_4(C_2,\mathtt{N}) = \mathcal{N}_4(C_2,\mathtt{K}) = 0$), these columns do not contribute counts to $A^{45}_{\mathtt{KN}}$. The inner sum in Equation 3.14 only contains terms for Column 1 and Column 3:

$$\begin{aligned} A^{45}_{\mathtt{KN}} &= \frac{(\mathcal{N}_1(C_1,\mathtt{K})\cdot\mathcal{N}_1(C_2,\mathtt{N}) + \mathcal{N}_1(C_1,\mathtt{N})\cdot\mathcal{N}_1(C_2,\mathtt{K}))}{k_1\cdot k_2} + \\ &\quad \frac{(\mathcal{N}_3(C_1,\mathtt{K})\cdot\mathcal{N}_3(C_2,\mathtt{N}) + \mathcal{N}_3(C_1,\mathtt{N})\cdot\mathcal{N}_3(C_2,\mathtt{K}))}{k_1\cdot k_2} \\ &= \frac{((4\cdot 1)+(0\cdot 1))}{4\cdot 2} + \frac{((2\cdot 2)+(0\cdot 0))}{4\cdot 2} \\ &= 1.0. \end{aligned}$$

Step 3: Estimate substitution frequencies. The frequencies of amino acid pairs are derived from the pair counts by normalizing by the total number of possible pairs. For a single block, we normalize by the product of the number of sites in the block and the number of pairs of clusters:

$$q^N_{xy} = \frac{A^N_{xy}}{L\cdot\binom{n_{\mathrm{C}}}{2}}. \tag{3.15}$$

In general, a BLOSUM matrix is constructed from many aligned blocks. To determine amino acid pair frequencies with many blocks, the counts for each block are summed to obtain the total number of counts in the numerator. The combined counts are normalized by the total number of possible pairs, combined over all blocks, yielding

$$q^N_{xy} = \frac{\sum_{b=1}^{n_{\mathrm{B}}} A^N_{xy}(b)}{\sum_{b=1}^{n_{\mathrm{B}}} L(b)\binom{n_{\mathrm{C}}(b)}{2}}, \tag{3.16}$$

where n_{B} is the total number of blocks and $A^N_{xy}(b)$, $L(b)$, and $n_{\mathrm{C}}(b)$ are the number of observed amino acid pairs, columns, and clusters in block b, respectively.

Step 4: Construct a log likelihood scoring matrix. The expected frequency of x aligned with y in unrelated sequences is the product of the *background* probabilities of observing x and y independently. In PAM matrix construction, the background frequency of an amino acid is assumed to be the frequency of that amino acid in typical proteins, for example, as tabulated by Robinson and Robinson[3]. In contrast, in BLOSUM matrix construction, the expected frequencies are estimated from the block data and adjusted for the current divergence, N.

In order to get the *expected* frequency of x aligned with y, we first estimate the frequencies of the individual residues in the current block, again using the clusters to correct for sample bias. As above, the counts from each cluster are "discounted" by a factor of $1/k_i$, and then normalized by the total number of elements, $L \cdot n_{\mathrm{C}}$, to obtain the amino acid background frequency. For a single block, the frequency of amino acid, x, is

$$p_x^N = \frac{1}{L\, n_{\mathrm{C}}} \sum_{i=1}^{n_{\mathrm{C}}} \sum_{l=1}^{L} \frac{\mathcal{N}_l(C_i, x)}{k_i}. \tag{3.17}$$

Given a data set with multiple blocks,

$$p_x^N = \frac{\sum_{b=1}^{n_{\mathrm{B}}} A_x^N(b)}{\sum_{b=1}^{n_{\mathrm{B}}} L(b)\, n_{\mathrm{C}}(b)}. \tag{3.18}$$

where

$$A_x^N(b) = \sum_{i=1}^{n_{\mathrm{C}}(b)} \sum_{l=1}^{L(b)} \frac{\mathcal{N}_l(C_i(b), x)}{k_i(b)}.$$

is the number of copies of x contributed by block b. The expected pair frequencies are then obtained from the products of the background frequencies:

$$E_{xy}^N = p_x^N p_y^N + p_y^N p_x^N \tag{3.19}$$

$$E_{xx}^N = (p_x^N)^2. \tag{3.20}$$

Finally, the BLOSUM-N log likelihood scoring matrix is calculated from the ratios of the observed and expected frequencies:

$$S^N[x, y] = 2 \log_2 \frac{q_{xy}^N}{E_{xy}}. \tag{3.21}$$

[3] *Distribution of glutamine and asparagine residues and their near neighbors in peptides and proteins*, PNAS, 1991 Oct;88:8880-4

3.4 Comparing PAM and BLOSUM Matrices

We began this endeavor with the goal of deriving substitution matrices that are parameterized by evolutionary divergence. In other words, a given alignment should be scored with a matrix with scores that are appropriate for the evolutionary divergence of the sequences being compared. In addition, these scores should implicitly account for multiple substitutions per site, consistent with the typical evolutionary divergence associated with each matrix in the family. A further goal is that the matrices should reflect the biophysical properties of amino acids. The scores for amino acid pairs with similar biophysical properties (i.e., conservative replacements) should be greater than scores for amino acid pairs with divergent biophysical properties (i.e., non-conservative or radical replacements).

The PAM and BLOSUM matrices were both constructed in an explicit log-likelihood framework, with entries of the form

$$S^N[x,y] = c\ log_2 \frac{q^N_{xy}}{p_x p_y},$$

where the numerator, q^N_{xy}, is the frequency of the amino acid pair (x, y) in alignments of related sequences with divergence N and the denominator, $p_x p_y$, is the frequency with which the pair (x, y) will occur if amino acids are sampled according to their background frequencies. The constant c is a scaling factor chosen for convenience. Multiplying every entry in the matrix by a constant changes the value of the entries in an absolute sense, but does not change the ratio between any two entries of the matrix. As a result, the constant does not change the extent to which one amino acid pair is preferred over another. Scaling a matrix with a constant, c, can be used to obtain scores in a convenient range, *e.g.* between 1 and 20.

Although the PAM and BLOSUM matrices have the general log-likelihood framework in common, they differ in many aspects of their construction, as summarized in Table 3.1. In both cases, the frequencies of amino acid pairs, q^N_{xy}, were estimated from amino acid pair counts in *"trusted"* alignments, but these trusted alignments are different in nature. In contrast to the PAM alignments, the BLOSUM matrices are based on locally conserved regions (ungapped blocks) in multiple alignments of sequences that were not highly conserved along their entire length. The PAM matrices were constructed from full length alignments of closely related sequences with at least 85% identity. These sequences are assumed to contain no site at which more than one substitution has occurred. The trusted alignments used to construct the BLOSUM matrices consisted of roughly 2000 blocks of conserved regions representing 500+ groups of proteins. In other words, some protein families contribute more than one block.

Both matrix families are parameterized by sequence divergence, but this is achieved using very different methods. The PAM matrices are based on a Markov chain that

	PAM	BLOSUM
Evolutionary model	Explicit evolutionary model	None
Data	Full length MSAs	Conserved blocks
Bias correction	Trees	Clustering
Multiple substitutions	Markov model: $P^N = (P^1)^N$	Implicitly represented in data
Evolutionary distance	Markov model: $P^N = (P^1)^N$	Clustering
Matrices	Transition & log likelihood scoring matrices	Log likelihood scoring matrix only
Parameter N	Distance increases with N	Distance decreases with N
Biophysical properties	Derived indirectly from data	Derived indirectly from data

Table 3.1: Properties of the PAM and BLOSUM matrices.

models amino acid replacement explicitly. The use of a Markov model allowed Dayhoff and her colleagues to address several challenges in matrix construction. A PAM-1 transition matrix is constructed from amino acid pair counts obtained from the trusted alignments. The effect of sample bias on these pair frequencies was mitigated by counting changes on the branches of maximum parsimony trees. Dayhoff accounted for both evolutionary divergence and multiple substitutions by deriving higher N-step transition matrices from the PAM-1 transition matrix. With PAM matrices, the divergence parameter increases with evolutionary divergence. A rough equivalence between PAMs and percent identity can be determined through simulations, as shown in Table 3.2.

The BLOSUM matrices have no underlying mathematical model. In BLOSUM matrix construction, clustering is used to address sample bias and to obtain different degrees of divergence. Sequences with at least $N\%$ identity are placed in the same cluster. Amino acid pairs are only counted across clusters, not within clusters. In contrast to the PAM matrices, the BLOSUM divergence parameter *decreases* as evolutionary divergence increases. BLOSUM matrices can also be roughly calibrated by percent identity using empirical methods, providing an approximate mapping between the PAM divergence scale and the BLOSUM divergence scale (Table 3.2).

Neither matrix family explicitly considers biophysical properties. The PAM and BLOSUM matrices are constructed from aligned sequences that are conserved because the amino acids in each column are under selective constraints. Nevertheless, the matrices favor amino acid pairs that share biochemical properties. Inspection of the BLOSUM62 matrix, for example, shows that alignments of residues in the same biochemical group tend to have positive log likelihood scores. These residues are more likely to be observed together in alignments of related sequences than by chance. Residues from different biochemical groups tend to have negative scores. These residues are less likely to be observed together in related

Sequence identity	PAM	BLOSUM
83%	20	-
-	30	-
63%	60	-
-	70	-
43%	100	90
38%	120	80
30%	160	60
25%	200	50
20%	250	45

Table 3.2: Correspondence between percent identity and the divergence of PAM and BLOSUM matrices.

sequences than in chance alignments. A score of zero means that this pair of residues is equally likely in related and chance alignments.

Chapter 4

Modeling motifs: Position Specific Scoring Matrices

Local multiple sequence alignment involves the discovery, modeling, and recognition of conserved patterns or motifs in multiple (and potentially very many) DNA or protein sequences.

- In *discovery*, we are given *unlabeled* sequences. The task is to identify one or more shared, conserved motifs in these sequences. In machine learning terms, this is equivalent to *labeling* the sequences. For example, each symbol in a sequence might be labeled "1" if it is in the conserved pattern and "0" if it is not. More complex labeling schemes representing more than one motif or different substructures within a motif are also possible.

- In *modeling*, we are given a local multiple alignment as input. The task is to construct a probabilistic model that represents the properties of each column in the alignment (i.e., the symbols we are likely to observe at that position) in an efficient manner and that can be used for searching for new instances of the motif.

- In *recognition*, we are given a new unlabeled sequence containing zero, one, or more than one instance of the motif of interest. A probabilistic model of the motif is used to search the unlabeled sequence for instances of the motif. The location and extent of each motif identified are reported.

We will first discuss *Position Specific Scoring Matrices* (PSSMs), a formalism for modeling local multiple alignments, and the Gibbs Sampler, a discovery method that uses the PSSM formalism. PSSMs and the Gibbs Sampler are suitable for ungapped motifs only. The *Hidden Markov Model (HMM)* is a formalism that can be used for both modeling and discovery of patterns that contain gaps. We will discuss HMMs immediately following the Gibbs Sampler.

In these notes, PSSMs and Gibbs Sampler are presented in terms of amino acid motifs. Both formalisms can be equally well applied to patterns in nucleic acid sequences. In fact, discovering and modeling transcription factor binding sites in DNA sequences is a common application of the Gibbs Sampler in bioinformatics.

4.1 Position Specific Scoring Matrices

PSSMs are a formalism for *modeling* ungapped local alignments. Like scoring matrices used for pairwise alignments, PSSMs are based on a log-odds formalism. Recall that both the PAM and BLOSUM matrices are defined in terms of an alternate hypothesis, H_a, that a pair of sequences are related at a given evolutionary divergence and a null hypothesis, H_0, that the sequences are unrelated and any observed similarity is due to chance. In such matrices, $\mathbb{S}[x, y]$ is the logarithm of the likelihood ratio:

$$\mathbb{S}[x, y] = \log_2 \frac{\mathbb{P}[x \text{ aligned with } y|H_a]}{\mathbb{P}[x \text{ aligned with } y|H_0]}. \tag{4.1}$$

A PSSM is similarly defined in a log-likelihood framework. In this case, the scoring matrix is used to score a candidate instance of a motif in a single sequence. The focus is on the probability of observing of a particular amino acid at a particular position in the motif. A PSSM is constructed from training data representing examples of the motif. Given an ungapped local alignment representing k instances of a motif of width w, we derive a *propensity* matrix, $\mathbb{P}$, representing the likelihood ratio

$$\mathbb{P}[x, i] = \frac{q[x, i]}{p_x}, \tag{4.2}$$

where $q[x, i]$ is the probability of observing amino acid x at position i under the alternate hypothesis that this is an instance of the motif. The probability of the same event under the null hypothesis is p_x, the background distribution of amino acid x. The log odds position specific scoring matrix is

$$\mathbb{S}[x, i] = \log_2 \mathbb{P}[x, i]. \tag{4.3}$$

Note that both $\mathbb{P}$ and $\mathbb{S}$ have $|\Sigma|$ rows and w columns, where Σ is the alphabet (in this case the 20 amino acids).

To complete the definition of $\mathbb{P}[x, i]$, we need an expression for the numerator, $q[x, i]$. The frequency of amino acid $x \in \Sigma$ at position i in the alignment is defined to be

$$q[x, i] = \frac{c[x, i]+b}{k+b \cdot |\Sigma|}, \tag{4.4}$$

where $c[x, i]$ is the number of x's at position i and b is a *pseudocount*.

Pseudocounts are introduced to account for examples of the motif that are not represented in the training data. It is possible that a particular amino acid, x, does sometimes occur at position i in this motif, but that this case does not arise in any of the sequences in the input alignment. If $\mathbb{P}[x,i] = 0$, then the resulting PSSM will assign a score of zero to any sequence with an x at position i, preventing the future discovery of this variant of the motif. To account for this, pseudocounts are used to give every amino acid a small, but non-zero probability at every position in the motif. The normalization in the denominator is adjusted accordingly by the term $b|\Sigma|$. In this class, we typically use $b = 1$. There are more complex approaches to selecting a pseudocount. For those interested in exploring this further, a more general treatment of pseudocounts is given in Section 5.6 of Durbin's book. This is not required for this course.

Given a new, unlabeled sequence, t, of length n, we can search for an instance of the motif in t by scoring the w residues at each possible startion position in the sequence as follows:

$$\mathcal{S}(t,o) = \sum_{i=1}^{w} \mathbb{S}[t[o+i],i]. \tag{4.5}$$

Here the *offset*, o, ranges from 0 to n-w and refers to the position *before* the first symbol in the motif. For a given value of o, the motif is the subsequence from position $o+1$ to $o+w$. The offset with the highest score is most likely to be an instance of the motif. To be convincing, the score must also be high in an absolute sense, not just higher that the scores associated with other offsets. In cases where there may be more than one instance of the motif in t, offsets with near optimal scores should also be considered.

Note that the score of a window of length w following offset o in t is a log likelihood ratio

$$\mathcal{S}(t,o) = \log_2 \frac{\Pr(t[(o+1)\ldots(o+w)]|H_a)}{\Pr(t[(o+1)\ldots(o+w)]|H_0)}, \tag{4.6}$$

where H_a is the alternate hypothesis that t contains the motif at position $o+1$ and H_0 is the null hypothesis that there is no instance of the pattern at this location and the residues in the window occur with typical background frequencies. To see this, consider that

$$\begin{aligned}
\mathcal{S}(t,o) &= \sum_{i=1}^{w} \mathbb{S}[t[o+i],i] \\
&= \sum_{i=1}^{w} \log_2 \frac{q[t[o+i],i]}{p_{t[o+i]}} \\
&= \log_2 \prod_{i=1}^{w} \frac{q[t[o+i],i]}{p_{t[o+i]}} \\
&= \log_2 \frac{\prod_{i=1}^{w} q[t[o+i],i]}{\prod_{i=1}^{w} p_{t[o+i]}}.
\end{aligned}$$

The numerator is the probability that the w residues starting at position $o+1$ in t represent an instance of the motif. The denominator is the probability of observing those residues under the null hypothesis.

4.2 Gibbs Sampler for motif discovery

A PSSM provides a compact, probabilistic representation of an ungapped motif, based on a training data set consisting of k representative sequences of length w. But how do we discover a motif in unlabeled sequences, when the motif is not known in advance?

The *Gibbs Sampler* is an algorithm for *discovery* of ungapped local alignments that uses the PSSM formalism as its basic data structure. The Gibbs Sampler takes as input k sequences, $t_1 \dots t_k$, of lengths $n_1 \dots n_k$, that share an ungapped motif of length w. The underlying assumption of the Gibbs Sampler is that each sequence contains exactly one instance of the motif. The length of the motif, w, must be supplied by the user. The output is a set of k subsequences of length w, one in each input sequence, that are "most similar" to each other. Here, our measure of "most similar" is a likelihood function derived from the propensity matrix, $\mathbb{P}$, defined in Equation 4.2.

Sequences $t_1 \dots t_k$ contain $O(n^k)$ candidate motifs, corresponding to all possible sets of offsets $\{o_1 \dots o_k\}$, where $0 \leq o_z \leq n_z - w$ is the offset in sequence t_z. Since the motif is ungapped, for a fixed width, w, each candidate is completely defined by a set of offsets $\{o_1 \dots o_k\}$ that specify starting points of the subsequences in the k sequences. Each set of offsets defines a local alignment consisting of the k subsequences of length w,

$$A = \begin{cases} t_1[(o_1+1)\dots(o_1+w)] \\ t_2[(o_2+1)\cdots(o_2+w)] \\ \cdots \\ t_k[(o_k+1)\cdots(o_k+w)], \end{cases}$$

where the notation $t[u \cdots v]$ denotes the substring of t, starting at position u and ending at and including position v.

A brute force approach to identifying the true motif is exhaustive enumeration of all candidates. For each candidate ungapped alignment, a score can be assigned that captures the extent to which columns in the alignment reflect similar sequence features. A PSSM, $\mathbb{S}[\cdot,\cdot]$, is constructed as specified in Equations 4.2 - 4.4, and used to score each sequence in the alignment:

$$\mathcal{S}(t_z, o_z) = \sum_{i=1}^{w} \mathbb{S}[t_z[o_z+i], i]. \tag{4.7}$$

The sum of these scores, $\sum_{z=1}^{k} \mathcal{S}(t_z, o_z)$, gives the overall score of the candidate motif.

In most cases, the PSSM will be meaningless in the sense that the individual subsequences will be unrelated to each other and the residues in each column will not reflect a significant degree of conservation. In this case, the individual subsequences will not receive scores that are better than chance. Only a candidate alignment that does, in fact, correspond to a conserved motif will produce a PSSM that yields good $\mathcal{S}(t_z, o_z)$ scores.

The computational cost of this brute force approach is prohibitive for all, but the smallest problem instances. The Gibbs Sampler is a more efficient approach to searching the space of candidate motifs that does not require explicit consideration of all possible alignments. The Gibbs Sampler uses an iterative approach in which a new estimate of the motif is generated from the current estimate of the motif at each iteration. The Gibbs Sampler has a robust theoretical basis and can be proven to converge to the best estimate. This convergence guarantee results from the specific features of the procedure for generating a new estimate from the current estimate. First, the new estimate must be different from the current estimate, but not too different. Second, the new estimate must be chosen in such a way that the algorithm can find the global optimum (i.e., the best estimate) and not just a local optimum. (A local optimum is an estimate that is better than all estimates that could be obtained in one step of the algorithm; that is, an estimate that has a higher score than all estimates resulting from a single modification of the current estimate.)

We introduce these ideas in two steps. We first consider a Hill Climbing algorithm (Algorithm 1) for motif discovery that has the same iterative structure as the Gibbs Sampler, but is not guaranteed to converge to the global optimum. The term "hill-climbing" reflects the fact that this algorithm finds an estimate with a higher score at each step and stops when it reaches the top of the hill (i.e., when it has found a local optimum). This Hill Climbing algorithm has the same data structures as the Gibbs Sampler and exemplifies the procedure for generating new candidate estimates. We then extend this algorithm to obtain a full Gibbs Sampler (Algorithm 2) that has a more sophisticated procedure for generating a new estimate, which allows it to converge to a global optimum.

Both algorithms generate a new estimate from the current estimate by modifying the contribution of one of the k sequences, called t^*, to A, while holding the subsequences from the remaining $k-1$ sequences fixed. The current estimate is represented by $(k-1) \times w$ matrix, A' which holds the subsequences of length w that represent the current best estimate in the $k-1$ sequences that will not change in this iteration. A PSSM is then constructed from A' and used to score all candidate offsets in t^*. Based on these scores, a new offset o^* is selected for sequence t^*. A new estimate of the motif is then constructed by removing one row in A' and replacing it with the new subsequence of length w with offset o^* in t^*. In order to simplify the bookkeeping associated with selecting a sequence at random from the $k-1$ sequences represented in the current iteration of A', we introduce an array called `index` that contains the indices of the sequences currently in A'. The row to be removed is selected by generating a random number, r, between 1 and $k-1$; the index of the sequence to be removed is stored in `index[r]`.


```
Algorithm:  Hill-Climbing
Input:
  Sequences t_1,...,t_k of lengths n_1,...,n_k.

Initialization:
  z = 1                                   # index of special sequence.
  t* = t_z, n* = n_z                      # t1 is the special sequence, initially.
  for (j = 2 to k) {
     index[j-1] = j                       # index of non-special sequences
     o_j = rand(1, n_j - w)               # Guess starting offsets
     A'[j-1, 1...w] = t_j[(o_j+1)...(o_j+w)]
  }
  Calculate P[x,i], the propensity matrix of A' with pseudocounts

Search for motif:
  Repeat
  {
     o* = argmax_o{S(t*, o)}              # Select starting offset in t*
     r = rand(1, k-1)                     # Select new special sequence
     A'[r, 1...w] = t*[(o*+1)...(o*+w)]   # Replace new special with t* in A'
     y = index[r]; index[r] = z; z = y    # store ptr to t* in index
     t* = t_z; n* = n_z                   # initialize new t*
     Calculate P[x,i], the propensity matrix of A' with pseudocounts
     S[x,i] = log_2 P[x,i]
  } until(P[·,·] stops changing)
  Obtain A by adding t*[(o*+1)...(o*+w)] to A'
  Compute the log odds scoring matrix, S, from A.

Output:
  Local multiple sequence alignment A with scoring matrix S.
```

Algorithm 1: **Hill Climbing** The matrices $\mathbb{P}$ and $\mathbb{S}$ are the propensity and log odds matrices defined in Equations 4.2 and 4.3. Note that A' and $\mathbb{P}$ are $(k-1) \times w$ matrices, whereas the output matrices A and $\mathbb{S}$ are $k \times w$ matrices. The use of pseudocounts when calculating $\mathbb{P}$ and $\mathbb{S}$ is recommended to ensure all symbols in the alphabet are represented.

The selection of a new offset, o^*, in t^* is a crucial aspect of the convergence of this algorithm. In the Hill-Climbing algorithm (Algorithm 1), the subsequence with the highest score is selected. Initially, selecting the subsequence with the highest score might seem an attractive strategy, but this could trap the algorithm in a local optimum. In fact, this is why the Hill Climbing algorithm is not guaranteed to converge to a global optimum.

The Gibbs Sampler (Algorithm 2), instead, selects a window in t^* at random from all windows starting at offsets ranging from 0 to n^*-w. The probability of selecting a particular offset, o, is biased by the probability of the subsequence at that offset so that higher scoring windows have a greater chance of being selected. The probability of the subsequence starting at offset $o+1$, defined in terms of its propensity with respect to the current estimate, is

$$pmf(o) = \frac{\prod_{j=1}^{w} \mathbb{P}[t^*[o+j], j]}{\sum_{i=0}^{n^*-w} \prod_{j=1}^{w} \mathbb{P}[t^*[i+j], j]}. \tag{4.8}$$

The numerator is the propensity of the subsequence starting at $o+1$. The denominator is the sum of propensities for all possible offsets from $o = 0$ to $o = n^*-w$. This is a normalization factor that ensures that $\sum_{o=0}^{n^*-w} pmf(o) = 1$. Note that here we score offsets using the propensity matrix, $\mathbb{P}$, and not the log odds scoring matrix, $\mathbb{S}$. Since $pmf(o)$ is a probability it must always have a non-negative value; $\mathbb{S}$ cannot be used because $\mathbb{S}$ can be negative.

Selecting a value of o with probability $pmf(o)$ requires a method for obtaining a random number conditioned on an arbitrary probability distribution. This random number can be obtained by calculating the cumulative distribution function

$$cmf(o) = \sum_{l=0}^{o} pmf(l),$$

a monotonically increasing function with domain $[0, n^*-w]$ and range $[0, 1]$. Its inverse, $cmf^{-1}(r)$, is a function defined on the domain $[0, 1]$ with range $[0, n^*-w]$. The offset of the new subsequence is defined to be $o^* = cmf^{-1}(r)$, where r is a uniformly distributed random number in the interval $[0, 1]$. The index o^* generated by this procedure is a random number with distribution $pmf(o)$.

Potential pitfalls: There are various potential pitfalls associated with the Gibbs Sampler, as with any algorithmic attempt to discover biological truth. For one thing, you could find a statistically significant or biologically meaningful motif that is not the motif you are looking for. In addition, problems can arise if one or more sequences have no copy of the motif or have more than one copy.

Using this algorithm to obtain meaningful solutions requires a number of decisions that are not programmatically determined and require *ad hoc* solutions, possibly guided by the user's biological intuition:


```
Algorithm:  Gibbs Sampler
Input:
  Sequences t_1,...,t_k of lengths n_1,...,n_k.

Initialization:
  z = 1                                        # index of special sequence.
  t* = t_z, n* = n_z                           # t1 is the special sequence, initially.
  for (j = 2 to k) {
     index[j-1] = j                            # index of non-special sequences
     o_j = rand(1, n_j - w)                    # Guess starting offsets
     A'[j-1, 1...w] = t_j[(o_j+1)...(o_j+w)]
  }
  Calculate P[x,i], the propensity matrix of A' with pseudocounts

Search for motif:
  Repeat
  {
     for (o = 0 to (n* - w))
     {
        pmf(o) = ∏_{j=1}^{w} P[t*[o+j], j] / Σ_{i=0}^{n*-w} ∏_{j=1}^{w} P[t*[i+j], j]
     }
     With probability pmf[o], o* = o           # Select starting offset in t*
     r = rand(1, k-1)                          # Select new special sequence
     A'[r, 1...w] = t*[(o*+1)...(o*+w)]        # Replace new special with t* in A'
     y = index[r]; index[r] = z; z = y         # store ptr to t* in index
     t* = t_z; n* = n_z                        # initialize new t*
     Calculate P[x,i], the propensity matrix of A' with pseudocounts
  } until(P[·,·] stops changing)
  Obtain A by adding t*[(o*+1)...(o*+w)] to A'
  Compute the log odds scoring matrix, S, from A.

Output:
  Local multiple sequence alignment A with scoring matrix S.
```

Algorithm 2: Gibbs Sampler

- Selecting the window size, w
- Selecting the starting configuration
- Selecting values for pseudocounts
- Termination condition: how should the algorithm decide when to stop?

These issues are discussed in greater detail in Lawrence et al. (1993), which is available via the "optional readings" column of the syllabus.

Convergence: The Gibbs Sampler models the search for an optimal local alignment as a Markov Chain, in which each state is a set of k subsequences of length w. It can be shown that this Markov Chain has a stationary distribution and that the state corresponding to the most likely motif has high probability in that distribution. In theory, this process is guaranteed to converge to the optimal solution, given "enough time." In practice, the Sampler can get bogged down in local optima for long periods of time. An approach to avoiding this problem is to run the procedure several times with different starting configurations. This is discussed in greater detail in the materials listed under "optional reading."

Background: The Gibbs Sampler is a general method for estimating a joint probability distribution by repeated calculations of a conditional distribution using a Markov Chain Monte Carlo (MCMC) approach. The application of the Gibbs Sampler for motif finding in biomolecular sequences was proposed first by Chip Lawrence[1] and his colleagues in 1993. For those interested in more theoretical aspects, Ewens and Grant discuss the Gibbs Sampler for biomolecular motif discovery in the MCMC framework (Section 10.5 in the first edition). A general introduction to the Gibbs Sampler[2] in a statistical context can be found under "optional readings" on the syllabus page. These readings are not required for the course.

Another probabilistic search procedure called *Expectation Maximization (EM)* can also be used to identify conserved, ungapped motifs. We will discuss EM briefly in the context of HMM's later in the course. EM is discussed in detail in 03-712.

[1] *Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment.* Lawrence et al., Science. 1993 262(5131):208-14.

[2] *Explaining the Gibbs Sampler*, G. Casella & E. I. George, The American Statistician, 46:167-174, 1992

Chapter 5

Hidden Markov Models

5.1 Introduction

We have been discussing conserved motifs or patterns found in multiple sequences, with a focus on three major tasks:

Discovery: Given a set of unlabeled sequences, identify an unknown pattern that is common to the input sequences.

Modeling: Given a set of sequences containing instances of a pattern, construct a compact, probabilistic representation of the pattern.

Recognition: Given a pattern model and a new unlabeled sequence, determine whether the pattern is present in the sequence and, if so, label the sites that correspond to the pattern.

In previous lectures, we considered Position Specific Scoring Matrices (PSSMs) for modeling and recognition of ungapped patterns or motifs and the Gibbs Sampler for discovering such motifs in unlabeled data. PSSMs work well for fixed length patterns in which the sites are more or less independent and the length of the motif can be predicted in advance. For example, the Gibbs sampler is an effective approach to finding transcription factor binding sites, which conform fairly well to these criteria. However, PSSMs's have limitations:

1. *PSSMs cannot model positional dependencies:* Suppose, for example, that a motif has a positional dependency like this one,


```
WEIRD
WEIRD
WEIQH
WEIRD
WEIQH
WEIQH
```

in which we see either `RD` or `QH` in the last two positions, but never `QD` or `RH`. A PSSM for this motif would give a high score to `WEIRD`, which conforms to the positional dependency, but also to `WEIRH`, which does not.

2. *PSSMs cannot recognize motif instances containing insertions or deletions:* A PSSM for the `WEIRD` motif would not assign high scores to pattern instances with insertion and deletions, such as `WECIRD` and `WERD`.

In the following lectures, we discuss *Hidden Markov models (HMMs)*, a more general and powerful formalism that can model a wider variety of conserved patterns. HMMs can capture interactions between sites in conserved motifs and recognize new motif instances with indels. In addition to conserved motifs, HMMs can model more diffuse, variable length patterns. These patterns are characterized by an underlying sequence composition that is representative of their biological role. Membrane-bound regions in a transmembrane protein (Fig. 5.1) are one example of a variable length pattern of this type. Sequence segments that are localized to the membrane are enriched for hydrophobic residues, relative to segments in the cytosol or the extracellular matrix, providing a signal that can be exploited for detecting transmembrane regions. A PSSM is not suitable for recognizing this type of feature. Similar recognition problems include identifying sequences that encode specific secondary structures (e.g., α helices, β sheets), distinguishing between coding and non-coding sequences, and delimiting genomic regions that exhibit compositional bias. For example, many bacterial

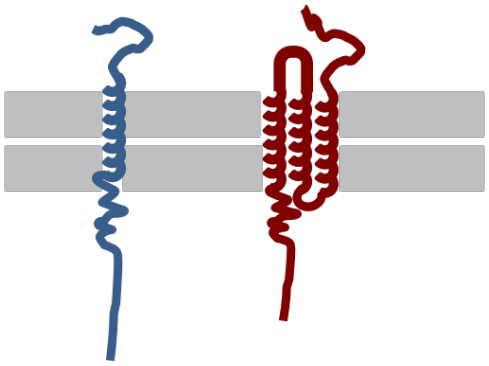


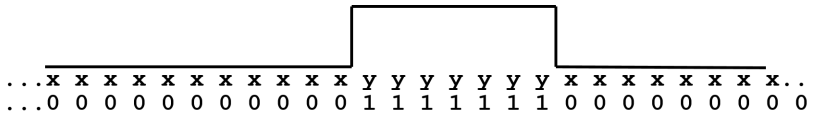
Figure 5.1: Transmembrane proteins

genomes have regions of GC skew, where one strand is enriched for guanine relative to the other strand.

Identification of patterns like these can be cast as a classification problem. We consider two problems:

1. We are given a sequence and we wish to assign it to a class. For example, is the given sequence a coding sequence?
2. We are given a sequence in which different segments belong to different classes. We wish to label each symbol in the sequence according to its class. For example, given a nucleic acid sequence, label each base as a coding nucleotide, if it is in a coding region, or a non-coding nucleotide, if it is in an intron or intergenic region.

The first problem can be addressed with a Markov chain. The second problem requires identification of sharp boundaries between classes. Attempting to classify segments with a Markov chain would require scoring a sliding window, which is not an effective approach for boundary detection (Fig. 5.2). However, as we shall see, HMM's are able to predict boundaries between classes precisely.



(a) Precise boundaries

(b) Fuzzy boundaries

Figure 5.2: Boundary detection. (a) Perfect boundary detection identifies the exact position of a pattern (e.g., a membrane bound segment) in a sequence. (b) Scoring a sliding window identifies the general neighborhood of the pattern, but cannot identify its precise position.

The theoretical basis of Hidden Markov models dates to the 1960's. In the 1970's, HMMs were broadly applied to problems in speech recognition. This laid the foundation for the use of HMMs for statistical inference, generally. Much of the HMM terminology is informed by early work on the speech recognition problem. This work is described in Rabiner's 1989 tutorial article, which is posted in the HMM module on Canvas for those who would like to probe further or prefer a more statistical treatment of the material. This article is *not* required reading.

In the speech recognition context, HMMs are used to infer the meaning of or "decode" a series of observations, $O = O_1, O_2, \ldots, O_T$. An observation is a sequence of sounds (i.e.,

speech). The goal of decoding is to determine the words that were uttered. An HMM is a generative model, i.e., a model that emits a sequence of symbols. In this framework, the decoding problem is akin to asking, what is the probability that a given model emitted the observed sequence?

Your textbooks adopt this framework and notation and we will do the same. In biological sequence analysis, an observation, $O = O_1, O_2, \ldots, O_T$, is a sequence of nucleotides or amino acids. The goal is to determine the probability that O was emitted by a model that corresponds to the biological question at hand (e.g., a transmembrane model, a coding sequence model, etc.)

5.2 Modeling variable length patterns with Markov chains

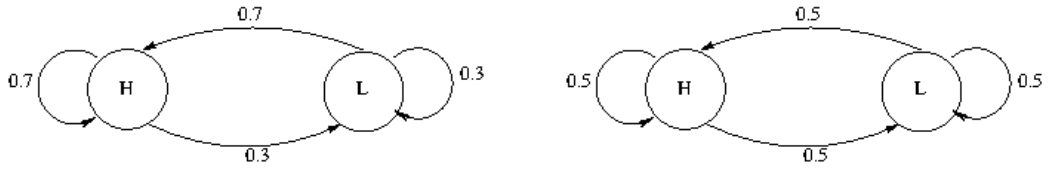
Hidden Markov models are a type of Markov model. Since we are already familiar with Markov chains, we will start by demonstrating how a Markov chain can be used to model a variable length pattern. However, as we will see, while Markov chains can be used to solve the first problem (classifying an entire sequence), they are not well suited to the second problem (assigning a class to individual residues). We next introduce Hidden Markov models (HMMs) and show how the the additional features of HMMs allow for the classification of individual residues and the inference of precise boundaries. In subsequent lectures, we will explore how HMMs can be used to model indels and positional dependence in motifs.

Recall that a finite, discrete Markov chain is defined by

- a finite set of *states*, $E_0, E_1, \ldots, E_N$;
- a *transition probability,* P_{xy}, which is the probability that the chain will be in state E_y at site $t+1$ given that it was in state E_x at site t; and
- an *initial state probability distribution*, $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$, where π_x gives the probability of being in state E_x at site $t = 1$.

In prior chapters, we used Markov chains to model the progression of substitutions at a single site over time. Here, we model the progression along the sequence. The transition probability gives the probability of observing a particular residue at the current site, as a function of the residue observed immediately before it. The independent variable t designates the position in the sequence, rather than time. We continue to use t to denote this independent variable to be consistent with the notation in Rabiner.

Let us first consider the problem of classifying an entire sequence, using transmembrane proteins as an example. To simplify the exposition, we will assume that protein sequences have been recoded in a two letter alphabet, $\Sigma_{H,L} = \{H, L\}$, in which each amino acid is replaced with an H if the residue is hydrophobic and an L if it is hydrophilic. We are given a sequence segment that is either a subsequence of a transmembrane (TM) region or a



(a) Transmembrane model ($\pi_H = 0.7, \pi_L = 0.3$) (b) ECM/Cytosol model ($\pi_H = \pi_L = 0.5$)

Figure 5.3: Markov models of sequence fragments localized to (a) the membrane or (b) the extracellular matrix (ECM) or cytosol.

subsequence of an extracellular or cytosolic (E/C) region. We wish to determine whether or not this segment is localized to the membrane.

For this sequence classification problem, we construct a transmembrane Markov model in which hydrophobic residues are preferred, consistent with frequencies observed in experimentally determined membrane-bound sequences (Fig. 5.3a). Our model has two states, H and L, corresponding to the symbols in $\Sigma_{H,L}$. Thus, a sequence of symbols corresponds to a sequence of states in the Markov chain. The probability that a given sequence $O \in (\Sigma_{H,L})^*$ is localized to the membrane can be estimated by the probability of visiting the states corresponding to symbols in O:

$$\Pr(O|\text{TM}) = \pi_{O_1} \prod_{t=2}^{T} P^{\text{TM}}_{O_{t-1}O_t},$$

where P^{TM} is the transition matrix for the transmembrane Markov model in Fig. 5.3a. For example, the probability of the sequence `HHLHH` is

$$\begin{aligned} \Pr(\texttt{HHLHH}|TM) &= \pi_H \times P_{HH} \times P_{HL} \times P_{LH} \times P_{HH} \\ &= 0.7 \times 0.7 \times 0.3 \times 0.7 \times 0.7 \\ &\approx 0.072. \end{aligned}$$

This calculation gives us the probability that `HHLHH` is a membrane-bound sequence according to the TM model, but how do we interpret that probability? We need a basis for selecting the minimum likelihood required to assert with confidence that a given sequence is membrane-bound.

A second difficulty is that the likelihood of a sequence is sensitive to its length; the longer the sequence, the lower its probability. Intuitively, this makes sense since the number of sequences of length T grows exponentially with T. Consider, for example, a sequence of ten consecutive hydrophobic residues. The probability of this sequence according to the transmembrane model is

$$\begin{aligned} \Pr(\texttt{HHHHHHHHHH}|\text{TM}) &= \pi_H \times (P_{HH})^9 \\ &\approx 0.03. \end{aligned}$$

Thus, the likelihood of HHHHHHHHHH according to the TM model is *lower* than the likelihood of HHLHH, even though HHHHHHHHHH looks more like a transmembrane segment than HHLHH.

Both of these issues can be addressed be adressed by comparison with the likelihood of O under a null model to obtain a likelihood ratio. For this purpose, we introduce a model of extracellular and cytosolic sequences (Fig. 5.3b) in which hydrophobic and hydrophilic residues occur with equal frequencies. Using these models, we can classify an observed sequence, $O = O_1, O_2, \ldots, O_T$, by its likelihood ratio

$$\frac{\Pr(O|\text{TM})}{\Pr(O|\text{E/C})}. \tag{5.1}$$

For example, $\Pr(\texttt{HHLHH}|\text{E/C}) = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \approx 0.031$ under this null model, yielding a likelihood ratio of

$$\begin{aligned} \frac{\Pr(\texttt{HHLHH}|\text{TM})}{\Pr(\texttt{HHLHH}|\text{E/C})} &= \frac{0.072}{0.031} \\ &\approx 2.3. \end{aligned}$$

Thus, it is a little more than twice as likely that HHLHH is a transmembrane sequence than an E/C sequence. In contrast, it can be shown that the likelihood ratio for HHHHHHHHHH is roughly 29, which is much greater than 2.3. This is consistent with our intuition that HHHHHHHHHH is a more convincing membrane-bound sequence than HHLHH.

5.3 Hidden Markov Models

Now we turn to the second problem: classifying individual residues in a sequence. We have demonstrated that the Markov chain models in Fig. 5.3 can be used to classify a sequence segment where all residues are of the same class (i.e., all TM or all E/C). However, we cannot apply these models to residue classification. If we knew the boundaries of transmembrane regions in O, we could switch back and forth between the E/C model and the TM model, but we do not have a procedure for identifying those boundaries. We could use a sliding window approach and calculate a likelihood ratio to score successive overlapping windows along the sequence, but this would result in a fuzzy boundary like that shown in Fig. 5.2.

In order to determine the location of the transmembrane regions in an unlabeled sequence, we need a model that explicitly labels each residue with its class (TM or E/C). For this purpose, we construct a new model that combines the two Markov chains by adding low probability transitions between E/C and TM states, as shown in Fig. 5.4.

This four-state model differs from the Markov chains in Fig. 5.3 in some fundamental ways. In the Markov chain models, each state associates a probability with a single piece of information: the amino acid at the current site. This means that there is a 1-1

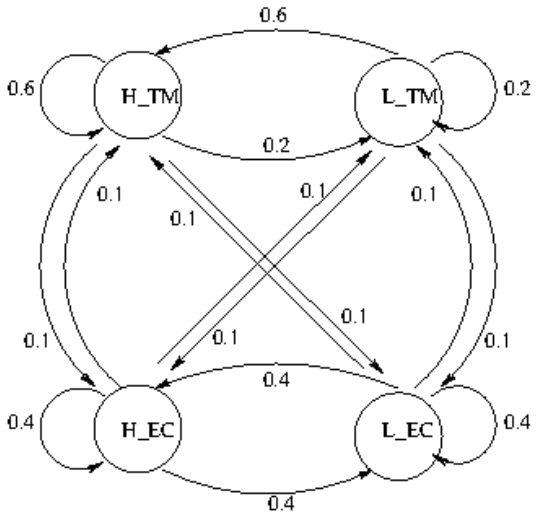


Figure 5.4: A four-state transmembrane HMM

correspondence between symbols and states. Every sequence of symbols, O, corresponds to a unique sequence of states, or *state path*, through the model. The probability of the sequence, $P(O|\text{model})$, is specified by product of the transition probabilities along this state path. Thus, we can use these models to solve Problem 1 by calculating the ratio of the likelihood under the assumption that the sequence is a transmembrane sequence and the likelihood that it is localized to the cytosol or the extracellular matrix.

In contrast, in the four-state model in Fig. 5.4, every state associates a probability with *two* pieces of information: the residue at the current site and the cellular location of that residue. This model does not have a one-to-one correspondence between symbols and states. Rather, there are two states associated with hydrophobic residues and two states associated with hydrophilic residues. As a result, for any given sequence of `H`'s and `L`'s, there are multiple paths through the states of the model. If we are given a particular state path (i.e., a sequence of states), then we can use the model to calculate the probability of observing this sequence of amino acids in the cellular locations specified by the state path. However, in general, the cellular localitions of the amino acids, and hence the true state path, are not known. In order to classify residues, we need a procedure for finding a state path that best estimates the cellular localization of the residues in O (the "true" state path). This estimate gives a precise boundary detection method. There are several algorithmic approaches to estimating the true path. We will discuss these in the next section.

The four-state model extends the Markov chain models in Fig. 5.3 by allowing a many-to-one correspondence between states and symbols. As a result, this model is no longer

a simple Markov chain. It is a Hidden Markov model. The word "hidden" refers to the fact that the true sequence of states for a given sequence of symbols is unknown; i.e., it is hidden from us. We can generalize further by also allowing one state to emit more than one symbol, resulting in a many-to-many relationship between symbols and states. This gives us the full definition of a Hidden Markov model, formally defined as follows:

a finite set of N *states* E_1, E_2, ..., E_N.

an alphabet, $\Sigma = \{\sigma_1, \sigma_2 \ldots \sigma_M\}$, of symbols that can be emitted from states.

a *transition probability matrix,* a, where a_{ij} is the probability of making a transition from state E_i to E_j; i.e., $a_{ij} = \Pr(E_j \text{ at } t+1 | E_i \text{ at } t)$. The rows of this matrix sum to one: $\sum_j a_{ij} = 1$.

the *emission probabilities,* $e_i(\sigma_k)$, for all states E_i and all symbols $\sigma_k \in \Sigma$ such that $e_i(\sigma_k)$ is the probability that state E_i emits σ_k. The emission probabilities for a single state sum to one: $\sum_k e_i(\sigma_k) = 1$.

the *initial state probability distribution*, $\pi = (\pi_1 \cdots \pi_N)$ for all states. The initial probabilities sum to one: $\sum_i^N \pi_i = 1$.

We refer to the transition probabilities, the emission probabilities, and the initial distribution, collectively, as *the parameters of the model*, designated $\lambda = (a_{ij}, e_i, \pi_i)$. Further, we use $Q = q_1, q_2, q_3, \ldots, q_T$ to denote a sequence of states visited when emitting a given sequence, $O = O_1, O_2, O_3, \ldots, O_T$. When considering more than one state path, we will use superscripts to distinguish between them: $Q^b = q_1^b, q_2^b, q_3^b, \ldots$. Similarly, we use superscripts to distinguish between multiple sequences of symbols: $O^d = O_1^d, O_2^d, O_3^d, \ldots$

An HMM is a *generative* model. A state in an HMM emits symbols from a fixed alphabet according to the specified emission probabilities. In the four-state model in Fig. 5.4, we used two states to encode residues in transmembrane sequences, one for hydrophobic residues and one for hydrophilic residues. Although not stated explicitly, the emission probabilities in the transmembrane states are $e_{H\_TM}(H) = 1, e_{H\_TM}(L) = 0, e_{L\_TM}(H) = 0$ and $e_{L\_TM}(H) = 1$. The emission probabilities in the E/C states are defined similarly. This does not take full advantage of the descriptive power of the HMM formalism. We can refine this model by collapsing states E_{H_TM} and E_{L_TM} into a single transmembrane state and using the emission probabilities to distinguish between hydrophobic and hydrophilic residues. This approach is illustrated in the two-state HMM shown in Fig. 5.5.

We can further refine the model by introducing separate E and C states as shown in Fig. 5.6. Using separate states to represent the cytosol and the extracellular matrix makes it possible to model differences in amino acid frequencies inside and outside the cell. In addition, biological intuition can be encoded in the topology. The model does not have a transition between the C and E states, representing the fact that proteins cannot tunnel from

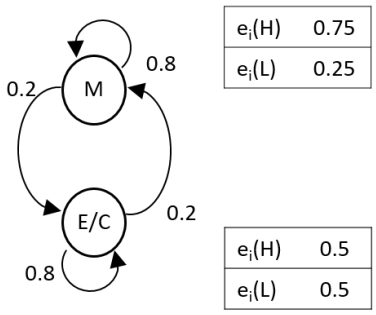


Figure 5.5: A two state Hidden Markov model of a transmembrane sequence.

the cytosol to the ECM without passing through the membrane. Typically, transmembrane proteins start in the cytosol and end in the the ECM, or vice versa; transmembrane proteins do not originate in the membrane. This is reflected in the zero-valued initial probability for the M state ($\pi_M = 0$). In this example, we assume that transmembrane sequences are equally likely to start in the cytosol or in the extracellular matrix (ECM); i.e., $\pi_C = 0.5$ and $\pi_E = 0.5$.

In this HMM model, the subcellular location of each residue is represented by the state that emitted the symbol associated with that residue. There are many state paths that can generate a given sequence of amino acids. If we are given both the observed sequence and the state path, then calculating the probability is straight-forward. Given a sequence $O = O_1, O_2, \ldots, O_T$ and a state path $Q = q_1, q_2, \ldots, q_T$, the probability of visiting the

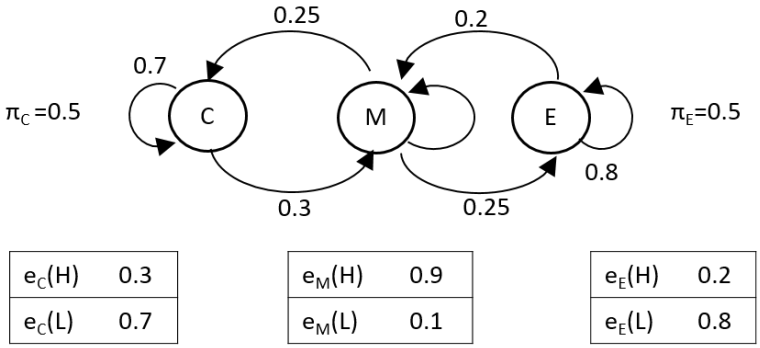


Figure 5.6: A three state Hidden Markov model of a transmembrane sequence.

states in Q and emitting O is

$$\begin{aligned}\Pr(O,Q|\lambda) &= \pi_{q_1}\cdot e_{q_1}(O_1)\cdot a_{q_1q_2}\cdot e_{q_2}(O_2)\cdot a_{q_1q_2}\cdot e_{q_3}(O_3)\ldots a_{q_{T-1}\,q_T}\cdot e_{q_T}(O_T)\\ &= \pi_{q_1}\cdot e_{q_1}(O_1)\prod_{t=2}^{T} a_{q_{t-1}q_t}\cdot e_{q_t}(O_t).\end{aligned}$$

For example, suppose $O =$`LHHHL` and $Q = CMMME$, then $\Pr(\texttt{LHHHL}, CMMME)|\lambda) =$

$$\pi_C\cdot e_C(\texttt{L})\cdot a_{CM}\cdot e_M(\texttt{H})\cdot a_{MM}\cdot e_M(\texttt{H})\cdot a_{MM}\cdot e_M(\texttt{H})\cdot a_{ME}\cdot e_E(\texttt{L}).$$

An HMM defines a probability distribution over all possible combinations of sequences and state paths for a hypothetical HMM, illustrated as a cartoon in Fig. 5.7a. Every point on the horizontal plane corresponds to a particular sequence, O^d, and a particular state path, Q^b. The value on the vertical axis is the joint probability, $\Pr(O^d, Q^b|\lambda)$, that the HMM will visit the states on path Q^b and emit sequence O^d. In the three-state TM model example (Fig. 5.6), the set of all possible sequences, $O^1, O^2, O^3, \ldots$ corresponds to `H, L, HH, HL, LH, LL, HHH, ...` and the set of all possible state paths, $Q^1, Q^2, Q^3, \ldots$ corresponds to *C, M, E, CC, CM, CE, MC, ...* Note that $\Pr(O^d, Q^b) = 0$ for many (O^d, Q^b) pairs. For example, in this model, $\Pr(O^d, Q^b) = 0$ for any state path that contains C immediately followed by E, because $a_{CE} = 0$. In any model, $\Pr(O^d, Q^b) = 0$ when O^d and Q^b have different lengths.

An HMM emits each sequence $O^d \in \Sigma^*$ with probability $\Pr(O^d) \geq 0$. Since a sequence can, potentially, be emitted from more than one state path, in order to obtain the total probability of a sequence, O, we must sum over the all possible paths:

$$\Pr(O) = \sum_b \Pr(O|Q^b,\lambda)\cdot\Pr(Q^b) = \sum_b \Pr(O,Q^b|\lambda).$$

Fig. 5.7b shows a cartoon representation of $\Pr(O, Q)$ for a single sequence, say O^5, for the set of all possible state paths, Q. The area under the curve is equal to $\Pr(O)$, the total probability of sequence O.

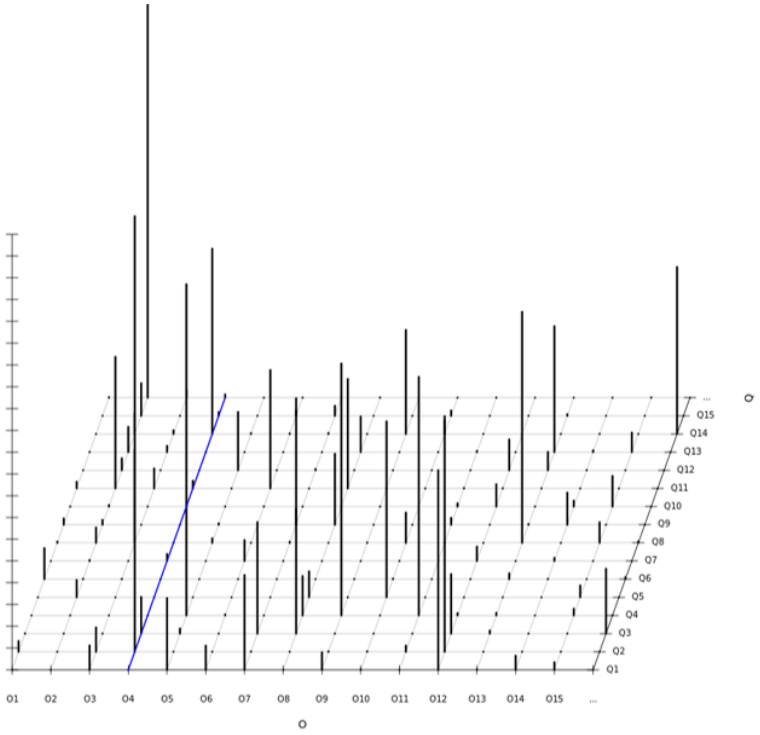
When all possible sequences and all possible paths are considered, the probability distribution shown in Fig. 5.7a sums to one:

$$\sum_d\sum_b \Pr(O^d, Q^b|\lambda) = 1.$$

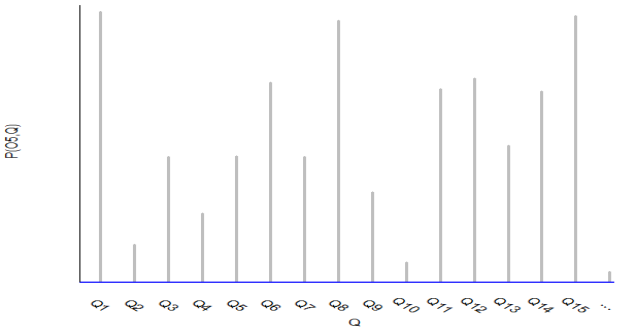
5.4 Using HMMs for recognition

In this section, we focus on motif recognition using HMMs. We will discuss parameter estimation, motif discovery, and modeling using HMMs in future sections. Here, we assume that we are given an HMM with known parameter values.

Our goal is to use the HMM to answer the various recognition questions, including:



(a) The joint probability, $\Pr(O^d, Q^b)$.



(b) $\Pr(O^5, Q^b)$

Figure 5.7: **(a)** The joint probability $\Pr(O^d, Q^b)$ for every sequence O^d and state path Q^b. The volume under this curve is one. **(b)**The probability of sequence $O = O^5$ for every state path $Q^1, Q^2, Q^3, \ldots$. This curve corresponds to the intersection of the probability distribution in Fig. 5.7a and the vertical plane at $O = O^5$ (shown as a blue line in Fig. 5.7a). The area under this curve is $\Pr(O^5|\lambda)$, the probability of O^5. The maximum point on the curve is the most probable path, $Q^* = \text{argmax}_Q \Pr(Q|O, \lambda)$.

1. What is the probability that a given sequence, O, was generated by the HMM?
   Example: Is sequence O a transmembrane protein?
2. Given a sequence, O, what is the true path? Otherwise stated, we wish to assign labels to an unlabeled sequence.
   Example: Identify the cytosolic, transmembrane, and extracellular regions in sequence O. In this case, we wish to assign the labels E, M, or C to each amino acid residue in the sequence.
3. What is the probability of being in state E_i when symbol O_t is emitted?
   Example: Is a given residue localized to the membrane?

Although this is not a focus of this class, we should point out that, since HMMs are generative models, an HMM can also be used for simulation; for example, to generate sequences with properties similar to real transmembrane sequences.

5.4.1 Calculating the total probability of sequence O

In order to answer the first question,

1. What is the probability that a given sequence, O, was generated by the HMM?

we must calculate $\Pr(O|\lambda)$, the total probability of the sequence given the model. The total probability is the probability of visiting states in path Q and emitting sequence O, summed over all possible state paths:

$$\Pr(O|\lambda) = \sum_b \Pr(O|Q^b, \lambda) \cdot \Pr(Q^b|\lambda) = \sum_b \Pr(O, Q^b|\lambda).$$

We could calculate this quantity by enumerating all paths, Q^b, and calculating $\Pr(O, Q^b|\lambda)$ for each one, but this brute force approach becomes intractable as the number of states gets large, since the number of state paths grows as $O(N^T)$. Instead, we use a dynamic programming algorithm called the *Forward* algorithm, which recursively calculates the probability of emitting prefixes of O. At each step, the Forward algorithm calculates the probability of emitting the first t symbols, $O_1, O_2, \ldots, O_t$, summing over all possible paths that end in state E_i. We designate this quantity

$$\alpha(t, i) = \Pr(O_1, O_2, O_3, \ldots, O_t, q_t = E_i|\lambda).$$

The variable α is an $T \times N$ matrix, where the rows correspond to prefixes of O and the columns correspond to states. At the t^{th} iteration, the algorithm calculates the entries in row t of the matrix, based on the entries in row $t-1$ and the parameters of the model. The entries in the final row contain the probability of emitting the entire sequence and ending in state E_i, for $i = 1$ to N. The probability of emitting the entire sequence, independent of

the final state, is obtained by the summing the entries in the last row. The algorithm to calculate $\alpha(t, i)$ for all $t \in (1, T)$ proceeds as follows:

Algorithm: Forward

Initialization:

$$\alpha(1, i) = \pi_i e_i(O_1)$$

Recursion:

$$\alpha(t, i) = \sum_{j=1}^{N} \alpha(t-1, j) \cdot a_{ji} \cdot e_i(O_t)$$

Final:

$$\Pr(O) = \sum_{i=1}^{N} \alpha(T, i)$$

The computational complexity of the Forward algorithm is $O(TN^2)$: There are $T \times N$ cells in the α matrix and the computational cost of computing each cell is $O(N)$.

In class, we worked an example based on the three-state transmembrane model shown in Fig. 5.6. A worksheet for this exercise is linked to the class syllabus page. The solution is also available. I recommend that you try to work through the Forward algorithm before looking at the solution.

5.4.2 Viterbi decoding

Next, we tackle the second recognition question:

2. Given a sequence O, what is the true path?

Given an unlabeled sequence, our goal is to classify or *label* each symbol in the sequence by inferring the state path. This process is called "decoding" because we decode the sequence of symbols to determine their meaning. HMMs were developed in the field of speech recognition, where recorded speech is "decoded" into words or phonemes to determine the meaning of the utterance. In our application, we decode an amino acid sequence to infer the functional role of each residue. There are two common approaches to decoding: Viterbi decoding and posterior decoding. We discuss Viterbi decoding here and postpone posterior decoding until after discussing Question 3.

Viterbi decoding is based on the assumption that the *most probable path*,

$$Q^* = \underset{b}{\operatorname{argmax}} \Pr(Q^b | O, \lambda),$$

is a good estimation of the sequence of states that generated the observed sequence O.[1] In practice, we maximize the joint probability $\Pr(Q, O|\lambda)$, rather than the conditional $\Pr(Q|O, \lambda)$, but this will still give us the most probable path because the path that maximizes $\Pr(Q, O|\lambda)$ also maximizes $\Pr(Q|O, \lambda)$. To see this, note that

$$\Pr(Q|O, \lambda) = \frac{\Pr(Q, O|\lambda)}{\Pr(O|\lambda)}.$$

Since $\Pr(O|\lambda)$ is independent of Q,

$$\operatorname*{argmax}_b \Pr(Q^b|O, \lambda) = \operatorname*{argmax}_b \Pr(Q^b, O|\lambda).$$

As in the case of the Forward algorithm, the brute approach of enumerating all paths and calculating $\Pr(Q|O, \lambda)$ for each one is intractable, because the number of state paths grows as $O(N^T)$. Instead, we calculate $\operatorname{argmax}_b \Pr(Q^b, O|\lambda)$ using a dynamic programming algorithm called the *Viterbi* algorithm. Let $\delta(t, i)$ be the probability of emitting the first t residues via the most probable path that ends in E_i. We calculate $\delta(t, i)$ as follows:

Algorithm: Viterbi

Initialization:

$$\delta(1, i) = \pi_i \cdot e_i(O_1)$$

Recursion:

$$\delta(t, i) = \max_{1 \le j \le N} \delta(t-1, j) \cdot a_{ji} \cdot e_i(O_t)$$

$$j^*(t, i) = \operatorname*{argmax}_{1 \le j \le N} \delta(t-1, j) \cdot a_{ji} \cdot e_i(O_t)$$

Final:

$$\Pr(Q^*, O|\lambda) = \max_{1 \le j \le N} \delta(T, j)$$

$$q_T^* = \operatorname*{argmax}_{1 \le j \le N} \delta(T, j).$$

At each step in the recursion, we save $j^*(t, i)$, the value of j that maximizes $\delta(t-1) \cdot a_{ji} \cdot e_i(O_t)$. These values are used to reconstruct the most probable path. The final state on the most probable path, q_T^*, is the state that maximizes $\delta(T, j)$. The rest of the state path is reconstructed by tracing back through the dynamic programming matrix, a procedure similar to the traceback in pairwise sequence alignment:

$$q_{t-1}^* = j^*(t, q_t^*), \qquad t = T \ldots 2.$$

[1] Note that the most probable path is not the same as the path that maximizes the likelihood of O!

The running time of the Viterbi algorithm is $O(TN^2)$. There are TN entries in the dynamic programming matrix. Each entry requires calculating N terms.

In class, we used the three-state HMM shown in Fig. 5.6 as an example. As an exercise, try applying the Viterbi algorithm to determine the most probable path through this model for the sequence `HHH`. A worksheet for this exercise is linked to the class syllabus page. The solution is also available. I recommend that you try to work through the Viterbi algorithm before looking at the solution.

5.4.3 The probability that state E_i emitted O.

The third question

3. What is the probability of being in state E_i when O_t is emitted?

is a special case of the decoding problem, where the focus is on classifying one specific residue. The probability of being in state E_i when O_t is emitted is the product of two probabilities: (1) the total probability of emitting $O_1 \ldots O_t$ over all paths that end in E_i and (2) the total probability emitting $O_{t+1} \ldots O_T$ over all paths, given that the model was in state E_i at time t:

$$\Pr(q_t = E_i, O) = \Pr(O_1, O_2, O_3, \ldots, O_t, q_t = E_i) \cdot \Pr(O_{t+1}, O_{t+2}, \ldots, O_T | q_t = E_i). \quad (5.2)$$

Note that the first term is just $\alpha(t, i)$, as defined in the section on the Forward algorithm. To calculate the second term, we introduce $\beta(t+1, i)$, the probability of emitting $O_{t+1}, O_{t+2}, \ldots, O_T$ given that O_t was emitted from state E_i:

$$\beta(t+1, i) = \Pr(O_{t+1}, O_{t+2}, \ldots, O_T | q_t = E_i, \lambda).$$

Substituting α and β for the first and second terms in Equation 5.2, we obtain the following expression for the probability of emitting O_t from state E_i

$$\Pr(q_t = E_i, O_t | \lambda) = \alpha(t, i) \cdot \beta(t+1, i). \quad (5.3)$$

The first term, $\alpha(t, i)$, is calculated using the Forward algorithm. The second term, $\beta(t+1, i)$, is calculated using an algorithm called the Backward algorithm. Like the Forward and Viterbi algorithms, the Backward algorithm is a dynamic programming algorithm. However, the Backward algorithm is different in that we start by calculating the probability of emitting the last symbol, O_T, and then work backwards from O_T to O_{t+1}.

Algorithm: Backward

Initialization:

$$\beta(T,i) = \sum_{j=1}^{N} a_{ij} \cdot e_j(O_T)$$

Recursion:

$$\beta(t,i) = \sum_{j=1}^{N} a_{ij} \cdot e_j(O_t) \cdot \beta(t+1,j)$$

In addition to determining the probability that O_t was emitted from a given state, the Backward algorithm has several other applications. Although the Forward algorithm is usually used to calculate the probability of a sequence, O, the Backward algorithm can also be used for this purpose. To calculate the probability of the entire sequence, we use the Backward algorithm to calculate $\beta(t,i)$ recursively, starting with $\beta(T,i)$. The total probability of O is given by:

$$\Pr(O) = \sum_{j=1}^{N} \pi_j e_j(O_1)\beta(2,j).$$

In motif discovery, both the Forward and the Backward algorithm are needed in order to learn parameters from unlabeled data using the Baum Welch procedure, which is a form of Expectation Maximization. The Backward algorithm is also used in another approach to inferring the true state path, called "Posterior decoding".

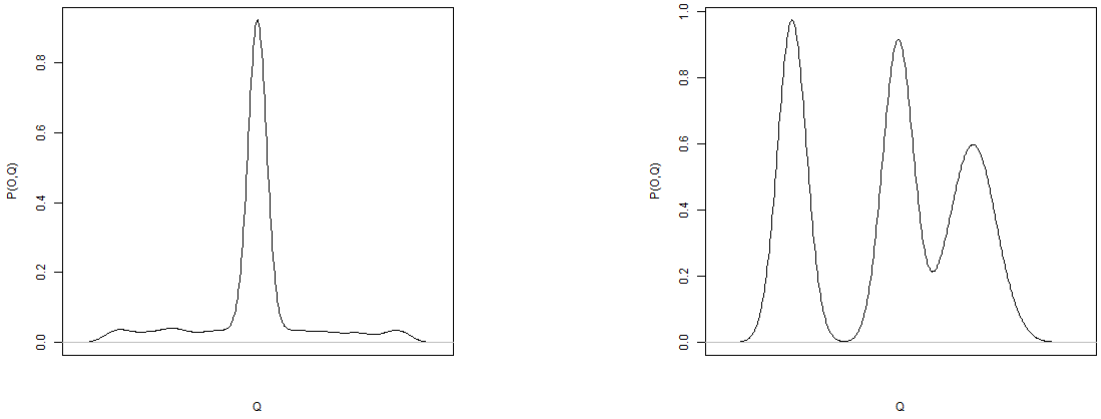
5.4.4 Posterior decoding

Let us revisit the question of estimating the path through an HMM that corresponds to the true labeling of the data. In Viterbi decoding, the *most probable path* is considered the best estimate of the true path. An alternative is to use $\hat{Q}$, the sequence of *most probable states*, as an estimate of the true path. This approach is referred to as *posterior decoding* because it is based on the posterior probability of emitting O_t from state i, when the emitted sequence is known. The most probable state at time t is the state that has the highest probability of emitting O_t when all possible state paths are considered:

$$\begin{aligned}
\hat{q}_t &= \operatorname*{argmax}_i \Pr(q_t = E_i, O_t) \\
&= \operatorname*{argmax}_i \Pr(O_1, O_2, O_3, \ldots, O_t, q_t = E_i) \cdot \Pr(O_{t+1}, O_{t+2}, \ldots, O_T | q_t = E_i) \\
&= \operatorname*{argmax}_i \alpha(t,i) \cdot \beta(t+1,i).
\end{aligned}$$

Note that the most probable state for emitting O_t is independent of the most probable state for any other symbol in O. In fact, the sequence of most probable states, $\hat{Q} = \hat{q}_1, \hat{q}_2, \ldots \hat{q}_T$ may not correspond to any legitimate path through the model.

Posterior decoding may give better results than Viterbi decoding in some cases, such as when suboptimal paths are almost as probable as the most probable path. If there is only one state path with high probability (e.g., Fig. 5.8a), then it is likely that Q^* and $\hat{Q}$ will represent the same sequence of states. However, when there are two or more state paths with high probability (e.g., Fig. 5.8b), each of those paths contributes some information about the classification of each symbol, O_t. Posterior decoding takes advantage of the information encoded in all state paths, while Viterbi decoding does not.



(a) (b)

Figure 5.8: (a) The probability distribution of paths for a given sequence of symbols, O^1, for a hypothetical hidden Markov model. In this hypothetical case, the probability of the most probable path is much greater than the probability of all other paths. (b) The probability distribution of paths for a given sequence of symbols, O^2, for a hypothetical hidden Markov model. In this hypothetical case, there are several paths with relatively high probability. One of these is almost as probable as the most probable path

5.5 Summary

We started by introducing three recognition questions:

1. What is the probability that a given sequence, O, was generated by the HMM?

Example: Is sequence O a transmembrane protein?

2. Given a sequence O, what is the true path? Otherwise stated, we wish to assign labels to an unlabeled sequence.
   Example: Identify the cytosolic, transmembrane, and extracellular regions in O. In this case, we wish to assign the labels E, M, or C to each amino acid residue in the sequence.
3. What is the probability of being in state E_i when O_t is emitted?
   Example: Is a given residue localized to the membrane?

We then discussed several approaches to answering these questions:

- Calculating $\Pr(O|\lambda)$ using the Forward or Backward algorithms
- Inferring the state path that emitted O using Viterbi or Posterior decoding
- Inferring the state that emitted O_t using the Forward and Backward algorithms

These tools can be used to answer biological questions in a variety of ways. For example, one approach to predicting whether O is a transmembrane protein is to calculate $\Pr(O|\lambda_{TM})$, the probability that O was emitted by the transmembrane model. However, the resulting probability can be difficult to interpret. How big must the probability be to convince us that O is in fact a transmembrane sequence? To answer the question, it is useful to construct a model representing a null hypothesis and to calculate $\Pr(O|\lambda_0)$, the probability that O was emitted by this null model. If the resulting likelihood ratio

$$\frac{\Pr(O|\lambda_{TM})}{\Pr(O|\lambda_0)}$$

is much greater than one, then we can infer that O is a transmembrane sequence.

An alternate approach would be to infer the state path that emitted O using the Viterbi or posterior decoding. If the resulting path includes membrane states, then we can conclude that O is a transmembrane sequence. If the entire sequence is labeled with C states or with E states, then we conclude that it is not.

Box 5: Summary of Hidden Markov model notation

Hidden Markov models, or HMMs, are defined formally by:

a finite set of N *states* E_1, E_2, ..., E_N.

an alphabet, $\Sigma = \{\sigma_1, \sigma_2 \ldots \sigma_M\}$, that can be emitted from states.

a *transition probability matrix,* a, where a_{ij} is the probability of making a transitioning from state E_i to E_j ($\Pr(E_j \text{ at } t+1 | E_i \text{ at } t)$). The rows of this matrix sum to one: $\sum_j a_{ij} = 1$.

the *emission probabilities,* $e_i(\sigma_k)$, for all states E_i and all symbols $\sigma_k \in \Sigma$ such that $e_i(\sigma_k)$ is the probability that state E_i emits σ_k. The emission probabilities for a single state sum to one: $\sum_k e_i(\sigma_k) = 1$.

the *initial state probability distribution,* $\pi = (\pi_i \cdots \pi_N)$ for all states. The initial probabilities sum to one: $\sum_i \pi_i = 1$.

Additional terminology:

the *parameters of the model*: $\lambda = (a_{ij}, e_i(a), \pi_i)$.

the *sequence of observed symbols*: $O = O_1, O_2, O_3, \ldots, O_T$. When considering more than one sequence, superscripts are used to distinguish them: $O^d = O_1^d, O_2^d, \ldots, O_T^d$

the *state path* or sequence of states visited: $Q = q_1, q_2, q_3, \ldots, q_T$. When considering more than one state path, superscripts are used to distinguish them: $Q^b = q_1^b, q_2^b, \ldots, q_T^b$

5.5.1 Summary of recognition algorithms

Algorithm: Forward

Initialization:

$$\alpha(1,i) = \pi_i e_i(O_1)$$

Recursion:

$$\alpha(t,i) = \sum_{j=1}^{N} \alpha(t-1,j) \cdot a_{ji} \cdot e_i(O_t)$$

Final:

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

Algorithm: Viterbi

Initialization:

$$\delta(1,i) = \pi_i \cdot e_i(O_1)$$

Recursion:

$$\delta(t,i) = \max_{1 \le j \le N} \delta(t-1,j) \cdot a_{ji} \cdot e_i(O_t)$$

$$j^*(t,i) = \operatorname*{argmax}_{1 \le j \le N} \delta(t-1,j) \cdot a_{ji} \cdot e_i(O_t)$$

Final:

$$P(Q^*, O|\lambda) = \max_{1 \le j \le N} \delta(T,j)$$

$$j^*(T) = \operatorname*{argmax}_{1 \le j \le N} \delta(T,j).$$

Algorithm: Backward

Initialization:

$$\beta(T,i) = \sum_{j=1}^{N} a_{ij} \cdot e_j(O_T)$$

Recursion:

$$\beta(t,i) = \sum_{j=1}^{N} a_{ij} \cdot e_j(O_t) \cdot \beta(t+1,j)$$

Final:

$$P(O) = \sum_{i=1}^{N} \pi_i \cdot e_j(O_1) \cdot \beta(2,i)$$

Algorithm for calculating $P(q_t = E_i, O_t)$

Initialization: Calculate matrices α and β.

Recursion:

$$P(q_t = E_i, O_t) = \alpha(t,i) \cdot \beta(t+1,i)$$

Algorithm: Posterior Decoding

Initialization: Calculate matrices α and β.

Recursion:

$$\hat{q}_t = \operatorname*{argmax}_{1 \le i \le N} \alpha(t,i) \cdot \beta(t+1,i)$$

Final:

$$\hat{Q} = \hat{q}_1, \hat{q}_2, \ldots \ \hat{q}_T$$

5.6 Designing HMMs: Motif discovery and modeling

There are three major computational tasks associated with conserved motifs found in multiple sequences: Discovery, Modeling, and Recognition. In previous sections, we discussed the recognition problem: Given an HMM, how do we use it to ask questions about patterns in a new, unlabeled sequence? Here, we consider modeling and discovery. For HMMs, modeling and discovery are closely coupled. There are two major issues to consider: designing the HMM topology and estimating the parameters of the model. A fundamental tradeoff drives HMM design: On the one hand, more complex models, with more parameters, can yield more accurate and biologically realistic models. On the other hand, as the number of parameters increases, so does the amount of data needed to estimate parameters without overfitting.

5.6.1 HMM topology

The topology of an HMM is determined by the set of states, $E_1, \ldots, E_N$, and how they are connected; in other words, we must specify which pairs of states will be connected by edges with non-zero transition probabilities. We could just choose a fully connected graph, but typically this has too many parameters to estimate. Instead, we can exploit biological knowledge. The goal is to choose a topology that limits the number of states and edges required, while still being expressive enough to represent the structure of the biological pattern of interest.

The choice of model topology can have a strong impact on the properties of the patterns we will discover. HMMs map symbols to states. Since changes in state define motif boundaries, how states are defined will influence the results of boundary detection. Inter-site dependencies and flexibility in pattern length can also be encoded in the topology of an HMM.

The transmembrane models we have discussed illustrate some of the issues to consider. For example, the three state model in Fig. 5.6 is not sufficiently restrictive to emit only transmembrane protein sequences. It can emit sequences that are entirely cytosolic or sequences that pass from the cytosol into the membrane and back to the cytosol, without ever passing through the extracellular region.

We can impose additional order dependencies on sequences generated by the model by modifying the topology. Suppose the goal is to generate sequences that always start and end in the cytosol, with one or more passes through the cell membrane into the extracellular matrix, and back through membrane to the cytosol. By adding additional states, we can obtain a model that only emits sequences that satisfy these conditions (Fig. 5.9). Note that this model has silent *Start* and *End* states, which we have not encountered before. These states do not emit symbols. They serve to ensure that the entry and exit from the model occur in specific states.

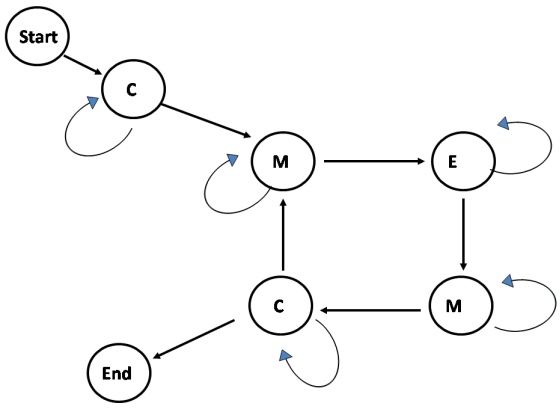


Figure 5.9: An HMM that emits transmembrane sequences that start and end in the cytosol and make at least one pass through the membrane to the extracellular matrix and back through the membrane to the cytosol. Note the use of silent Start and End states ensures that sequences start and end in the cytosol.

The topology of the model also influences the distribution of the lengths of sequences that the model can emit. For example, a simple self loop with probability p (Fig. 5.10) results in sequences with lengths that follow an exponentially decaying (geometric) distribution. The probability that this model will emit a sequence of length l is $(1-p)p^{l-1}$. This does not correspond to the length distribution of real amino acid sequences. More realistic length distributions can be obtained with more complex topologies. Some of these are described in Durbin, section 3.4.

In addition to specifying the model topology, we must choose the alphabet and decide which states will emit which symbols. The larger the alphabet, the greater the number of emission probabilities that must be estimated from training data. The transmembrane models we discussed in class used a two letter alphabet of hydrophobic (`H`) and hydrophilic

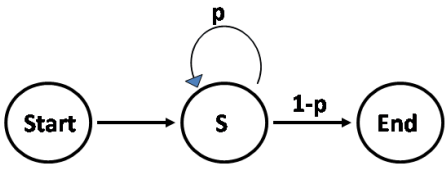


Figure 5.10: An HMM with one non-silent state that emits a single symbol, σ, with unit emission probability ($e_s(\sigma) = 1$). The probability that a sequence is emitted by this HMM decreases exponentially with the length of the sequence.

(L) residues to represent sequences, instead of the full 20 letter alphabet for amino acids. This not only gives a simpler representation, it also requires fewer training sequences to learn the parameters since all hydrophobic (resp., hydrophilic) residues contribute to the estimation of a single parameter.

5.6.2 Parameter estimation

Once the states and connectivity have been chosen, the parameters of an HMM are estimated from training data. We are given observed sequences, $O^1, O^2, \ldots, O^k$, and wish to construct an HMM with parameters, λ, to model these sequences. If the sequences are labeled, the transition and emission probabilities can be estimated easily from the observed transition and emission frequencies. If the sequences are unlabeled, we must first discover the conserved pattern using a machine learning algorithm.

Labeled sequences:

When we are given labeled sequences in which every symbol O_t^d is associated with a state, $q_t^d = E_i$, the parameters can be estimated by tabulating the emission and transition frequencies in the data. Note that inferring parameters from counts in labeled data is a form of maximum likelihood estimation; we are assuming that the emission and transition probabilities that best model the motif of interest are those that maximize the probability of the observed symbols and states in the training data.

The transition probabilities are calculated by tabulating the number of observed state changes in the data:

$$a_{ij} = \frac{\sum_{d=1}^{k} A_{ij}^d}{\sum_{d=1}^{k} \sum_{j\prime} A_{ij\prime}^d},$$

where A_{ij}^d is the number of pairs of adjacent symbols, $O_t^d\ O_{t+1}^d$, that are labeled $E_i E_j$. The emission probabilities are given by

$$e_i(\sigma) = \frac{\sum_{d=1}^{k} \mathcal{E}_i^d(\sigma)}{\sum_{d=1}^{k} \sum_{\alpha \in \Sigma} \mathcal{E}_i^d(\alpha)}$$

where $\mathcal{E}_i^d(\sigma)$ is the number of instances in O^d where the symbol σ is labeled with state E_i. Finally, the initial probability π_i is given by

$$\pi_i = \frac{1}{k} \sum_{d=1}^{k} I^d(i), \tag{5.4}$$

where

$$I^d(i) = \begin{cases} 1, & \text{if } q_1^d = E_i \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

is an indicator variable that is equal to one when the first symbol in O^d is labeled with state E_i and zero otherwise.

We may wish to include pseudocounts to account for cases not observed in the training data. Pseudocounts are incorporated into the emission probabilities in the same way that we used pseudocounts in the definition of the frequency matrix for PSSMs. The probability of emitting σ from state E_i is

$$e_i(\sigma) = \frac{\sum_{d=1}^{k} \mathcal{E}_i^d(\sigma) + b}{\sum_{\alpha \in \Sigma} \left(\sum_{d=1}^{k} e_i^d(\alpha) + b \right)}$$

$$= \frac{\sum_{d=1}^{k} \mathcal{E}_i^d(\sigma) + b}{\left(\sum_{\alpha \in \Sigma} \sum_{d=1}^{k} e_i^d(\alpha) \right) + |\sum| b} \tag{5.6}$$

where b is a pseudocount. In class, we have used $b = 1$ as a pseudocount. There are more sophisticated approaches to selecting a pseudocount. We will not cover them in this course; for those interested in learning more about this on their own, Durbin's discussion of Dirichlet mixtures in Section 11.5 of his book provides a starting point.

We can also use pseudocounts to account for state transitions that are allowed, but not observed in the training data. Let $\mathcal{N}(i)$, the neighborhood of state E_i, be the set of states that can be reached from E_i in a single transition. In other words, $\mathcal{N}(i)$ is the set of states, E_j such that a_{ij} has not been explicitly defined to be zero in the design of the topology. Then,

$$a_{ij} = \frac{\left(\sum_{d=1}^{k} A_{ij}^d \right) + b}{\sum_{j' \in \mathcal{N}(i)} \left((\sum_{d=1}^{k} A_{ij'}^d) + b \right)}$$

$$= \frac{\left(\sum_{d=1}^{k} A_{ij}^d \right) + b}{\left(\sum_{j' \in \mathcal{N}(i)} \sum_{d=1}^{k} A_{ij'}^d \right) + |\mathcal{N}(i)| b} \tag{5.7}$$

As an example, consider the three-state transmembrane model in Fig. 5.6. For this model, seven transition probabilities must be inferred: a_{CC}, a_{CM}, a_{MC}, a_{MM}, a_{ME}, a_{EM}, and a_{EE}. To estimate A_{CC}, for example, given the following labeled sequence

H	H	H	L	L	H	L	H	L	L	H	H	H	H	H
C	C	C	C	C	C	C	C	C	C	M	M	M	M	M

we count the number of CC pairs and normalize by the number of pairs of the form $C*$ where $*$ can be any state. Since there are nine pairs of adjacent symbols labeled CC and one pair labeled CM, for this sequence $a_{CC} = 0.9$ without pseudocounts. With pseudocounts,

$$a_{CC} = \frac{A_{CC} + b}{\sum_{j' \in \mathcal{N}(C)} [A_{Cj'} + b]} ,$$

where $\mathcal{N}(C) = \{C, M\}$. With a pseudocount of $b = 1$, we obtain

$$\begin{aligned} a_{CC} &= \frac{9+1}{(9+1)+(1+1)}\ , \\ &= \frac{10}{12}. \end{aligned}$$

To obtain the emission probabilities from state C, note that C is associated with five hydrophobic and five hydrophilic residues. Thus,

$$\begin{aligned} e_C(\texttt{H}) &= \frac{\mathcal{E}_C(\texttt{H}) + b}{\sum_{\alpha \in \{\texttt{H},\ \texttt{L}\}} \mathcal{E}_C(\alpha) + 2b}\ , \\ &= \frac{5+1}{10+2} \end{aligned}$$

or $e_C(\texttt{H}) = 0.5$, again assuming a pseudocount of $b = 1$.

The other transition and emission probabilities are estimated similarly. We estimate the initial probability π_C by counting the number of sequences that begin in the cytosol, and normalizing by the total number of sequences.

Unlabeled sequences:

If the sequences are *unlabeled*, then it is necessary to both discover the motif and learn the model parameters. The motif is discovered automatically, but implicitly, through the process of parameter inference. Once the parameters have inferred, the parameters are used to obtain an explicit model of the motif via Viterbi or posterior decoding.

Parameters are inferred using maximum likelihood estimation. Given sequences $O^1, O^2, \ldots, O^k$, we seek $\lambda_l = \{a_{ij}, e_i(\cdot), \pi_i\}$ such that the probability of observing the input sequences is maximized. Stated formally,

$$\begin{aligned} \lambda &= \operatorname*{argmax}_{\lambda_l} \mathcal{L}(\lambda_l) \\ &= \operatorname*{argmax}_{\lambda_l} \sum_d \Pr(O^d|\lambda_l) \\ &= \operatorname*{argmax}_{\lambda_l} \sum_d \sum_Q \Pr(O^d|\lambda_l, Q)\ . \end{aligned}$$

Except for very small problem instances, finding a global maximum is intractable. We would have to calculate $\mathcal{L}(\lambda_l)$ for all possible combinations of parameters, λ_l, to find the parameters that maximize $\Pr(O^1 \ldots O^k|\lambda_l)$. Instead, heuristics are used. These are typically guaranteed to find at least a local maximum. Since these are heuristics, evaluation is usually done empirically by withholding some of the training data for testing, but we will not discuss this further.

The *Baum-Welch* algorithm (Algorithm 3) is used to estimate the parameters of a Hidden Markov model from unlabeled training data. Baum-Welch belongs to a family of algorithms, called Expectation Maximization (EM) algorithms, that work by alternating between estimating the likelihood of the data, given the current estimate of the parameters and re-estimating the parameters from the current likelihoods. Baum-Welch is based on algorithms that we have already encountered: Given labeled data, we can estimate the model parameters using Equations 5.4-5.7, as described in the previous section. Given a model with parameters, we can label unlabeled sequences using Viterbi or posterior decoding.

Informally, Baum-Welch is an iterative algorithm that alternately applies these two procedures. First, an initial estimate of the parameter values is required, for example, based on prior knowledge of the biology underlying the model or on a uniform prior. With this initial estimate, the model is used to label the training data, typically using posterior decoding with the Forward and Backward algorithm. Once the sequences have been labeled, the parameters are re-estimated from the labeled data. The training sequences are then re-labeled using this new estimate of the parameters. The algorithm iterates, alternately labeling the data with the current estimate of the parameter values and then re-estimating the parameters from the labeled data. At each iteration, the likelihood is guaranteed to remain unchanged or increase. This iterative process terminates when the likelihood ceases to improve.

It is instructive to note the similarities and differences between the Baum-Welch algorithm and the Gibbs sampler. Like Baum-Welch, the Gibbs sampler alternates between re-estimating parameters (i.e., a PSSM) from the current estimate of the motif and inferring a new instance of the motif from the updated parameters. However, unlike Baum-Welch, where every training sequence is relabeled at each iteration, in the Gibbs sampler, only one sequence is relabeled at each iteration. A second major difference between the two methods is that the Gibbs sampler is guaranteed to converge to a global optimum given enough time. In contrast, the Baum-Welch algorithm is only guaranteed to find a local optimum.

Given the observed, unlabeled sequences, the parameters are re-estimated in the inner loop of the algorithm. A_{ij} is the expected number of transitions from E_i to E_j. For a given sequence, O^d, probability of transiting from state E_i to E_j at time t is $\Pr(q_t^d = i, q_{t+1}^d = j|O^d, \lambda)$. The number of total transitions from E_i to E_j can be obtained by summing over all time steps, $t = 1$ to T and all input sequences:

$$A_{ij} = \sum_d \sum_t \Pr(q_t^d = i, q_{t+1}^d = j|O^d, \lambda) \tag{5.8}$$

To facilitate the calculation, we again use the trick of converting the conditional probability

Algorithm 1: Baum-Welch

Input:

A set of observed sequences, $O^1, O^2, \ldots, O^k$

Initialization:

Select arbitrary model parameters, $\lambda = (a_{ij}, e_i(\cdot), \pi_i)$.

Iteration:

Repeat

{

For each sequence, O^d,

{

Calculate $\alpha(t,i)$, $\beta(t,i)$ and $\Pr(O_d)$ using Forward and Backward algorithms.

$A^d_{ij} = \frac{1}{\Pr(O^d)} \cdot \sum_t \alpha(t,i) a_{ij} e_j(O^d_{t+1}) \beta(t+2, j)$

$\mathcal{E}^d_i(\sigma) = \frac{1}{\Pr(O^d)} \sum_{\{t | O^d_t = \sigma\}} \alpha(t,i)\, \beta(t+1, i).$

}

$a_{ij} = \frac{\sum_d A^d_{ij}}{\sum_d \sum_l A^d_{il}}$

$e_i(\sigma) = \frac{\sum_d \mathcal{E}^d_i(\sigma)}{\sum_d \sum_\alpha \mathcal{E}^d_i(\alpha)}$

$\pi_i = \sum_{d=1}^k \frac{I^d(i)}{\Pr(O^d)}$

$\lambda = (a_{ij}, e_i(\cdot), \pi_i)$.

$\mathcal{L}(\lambda) = \prod_d \Pr(O^d | \lambda)$

}

Until ($\mathcal{L}(\lambda)$ stops changing.)

Algorithm 3: **Baum Welch**

into a joint probability:

$$\begin{aligned}\Pr(q_t^d = i, q_{t+1}^d = j|O^d, \lambda) &= \frac{\Pr(q_t^d = i, q_{t+1}^d = j, O^d)}{\Pr(O^d)} \\ &= \frac{\alpha(t,i)a_{ij}e_j(O_{t+1}^d)\beta(t+2,j)}{\Pr(O^d)} .\end{aligned}$$

The term $\alpha(t,i)$ is the probability that the model has emitted symbols $O_1^d \ldots O_t^d$ and is in state E_i at time t. This probability can be obtained using the Forward algorithm. The term in the denominator, $\Pr(O^d)$, is also calculated with the Forward Algorithm. The terms a_{ij} and $e_j(O_{t+1}^d)$ give the probability of making the transition from E_i to E_j and emitting O_{t+1}. The Backward algorithm yields $\beta_{t+2}(j)$, the probability of emitting the rest of the sequence if the model was in state E_j at time $t+1$. From this we can estimate

$$A_{ij} = \sum_d \frac{\sum_t \alpha(t,i)\, a_{ij}\, e_j(O_{t+1}^d)\, \beta(t+2,j)}{\Pr(O^d)} \tag{5.9}$$

Note that instead of explicitly labeling the data and then counting state transitions as we do with labeled data, the association of symbols and states is implicit in the re-estimation process in the inner loop of the algorithm.

$\mathcal{E}_i(\sigma)$ is the expected number of times that σ is emitted from state E_i:

$$\mathcal{E}_i(\sigma) = \sum_d \Pr(q_t^d = E_i, O_t^d = \sigma|O^d, \lambda) \tag{5.10}$$

$$= \sum_d \frac{\sum_{\{t|O_t^d=\sigma\}} \alpha(t,i)\, \beta(t+1,i)}{\Pr(O^d)} . \tag{5.11}$$

Again, the quantities on the right hand side can be calculated using the Forward and Backward algorithms. Finally, the initial probability π_i is given by

$$\pi_i = \sum_{d=1}^{k} p(q_1^d = S_i|O_d, \lambda) \tag{5.12}$$

$$= \sum_{d=1}^{k} \frac{I^d(i)}{\Pr(O^d)} \tag{5.13}$$

It can be proven that if current estimate is replaced by these new estimates then the likelihood of the data will not decrease (i.e. will increase unless already at a local maximum/critical point). See Durbin, Section 11.6 for discussion of avoiding local maxima and other typical pitfalls with this algorithm.

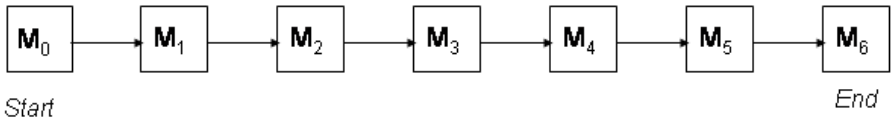


Figure 5.11: A HMM for modeling fixed length motifs. Note that this HMM is equivalent to a Position Specific Scoring Matrix.

The *Baum-Welch* algorithm estimates the values of the parameters from training data and, thus, implicitly discovers the motif. *Baum-Welch* does output an explicit description of the motif. To determine the motif explicitly, the Viterbi or posterior decoding are used to label each of the input sequences.

5.7 Profile HMMs

In 1994, Krogh, Haussler[2] and colleagues introduced a generic HMM topology specifically designed to model conserved sequence motifs. It captures the propensity to observe specific amino acids or nucleotides at each position in a pattern and allows for insertions and deletions. This topology, called a *Profile HMM*, can be customized for a broad range of conserved motifs by selecting the appropriate length for a given motif and initializing the parameters to capture the specific properties of the motif.

Here, we introduce the features of the Profile HMM model by showing how it could be used to model the `WEIRD` motif based on the following alignment, which has no gaps and no positional dependencies:

```
WEIRD
WEIRD
WEIRE
WEIQH
```

We can recognize the `WEIRD` motif using an HMM with the simple topology shown in Fig. 5.11, where the transitions probabilities are

$$a_{M_i,M_j} \begin{cases} 1, & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases}.$$

[2]Krogh, A., Brown, M., Mian, I. S., Sjolander, K., and Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. J. Mol. Biol., 235:1501-1531.

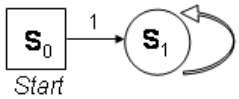


Figure 5.12: An HMM representing the null model. Each symbol, σ, is emitted with probability $p(\sigma)$, the background frequency of σ.

The emission probabilities can be estimated from labeled training sequences. Given an ungapped multiple alignment of k sequences, the emission probabilities are

$$e_{M_i}(\sigma) = \frac{c[\sigma, i]+b}{k+b|\Sigma|}, \tag{5.14}$$

where $c[\sigma, i]$ is the number of σ's at position i in the training motif and b is a pseudocount. The Start and End states (M_0 and M_6 in Fig. 5.11) are silent. Note that $e_{M_i}(\sigma)$ is equivalent to $q[\sigma, i]$, the frequency matrix that we derived for the PSSM example using pseudocounts. Moreover, when $\mathcal{E}_{M_i}(\sigma) = c[\sigma, i]$, Equation 5.14 is equivalent to the general definition of emission probabilities for an HMM given in Equation 5.6.

In order to assess whether a new sequence, O, contains an instance of the `WEIRD` motif, we calculate a likelihood ratio:

$$\frac{\Pr(O|H_A)}{\Pr(O|H_O)}.$$

Our alternate hypothesis, H_A, is that O is an instance of the motif represented by the HMM in Fig. 5.11. In order to obtain a likelihood ratio, we also need a model of the null hypothesis, H_0, that the amino acids in O were sampled from the background distribution. Fig. 5.12 shows a very simple null model. In this model, all transition probabilities are equal to one. The emission probability $e(\sigma)$ is defined to be $p(\sigma)$, where $p(\sigma)$ is the background frequency of residue σ.

Given these two models, we can score O by calculating the probability that O was emitted by the Profile HMM in Fig. 5.11 and comparing it with the probability that O was emitted by the background model in Fig. 5.12. For example, if $O = O_1O_2O_3O_4O_5$, then $\Pr(O|H_A)$ is equal to

$$\pi_{M_O} \cdot e_{M_1}(O_1) \cdot a_{M_O M_1} \cdot e_{M_2}(O_2) \cdot a_{M_2 M_3} \cdot e_{M_3}(O_3) \cdot a_{M_3 M_4} \cdot e_{M_4}(O_4) \cdot a_{M_4 M_5} \cdot e_{M_5}(O_5) \cdot a_{M_5 M_6}.$$

Since the initial and transition probabilities are equal to one ($\pi_{M_O} = 1; a_{M_i M_{i+1}} = 1, 0 \leq i \leq 5$), this reduces to

$$\Pr(O|H_A) = e_{M_1}(O_1) \cdot e_{M_2}(O_2) \cdot e_{M_3}(O_3) \cdot e_{M_4}(O_4) \cdot e_{M_5}(O_5)$$

or

$$\Pr(O|H_A) = \prod_{t=1}^{5} e_{M_t}(O_t) \ .$$

The probability that O was emitted by the background model is $\prod_{t=1}^{5} p(O_t)$. The score of sequence O is the log likelihood ratio

$$\sum_{t=1}^{5} \log \frac{e_{M_t}(O_t)}{p(O_t)} \ ,$$

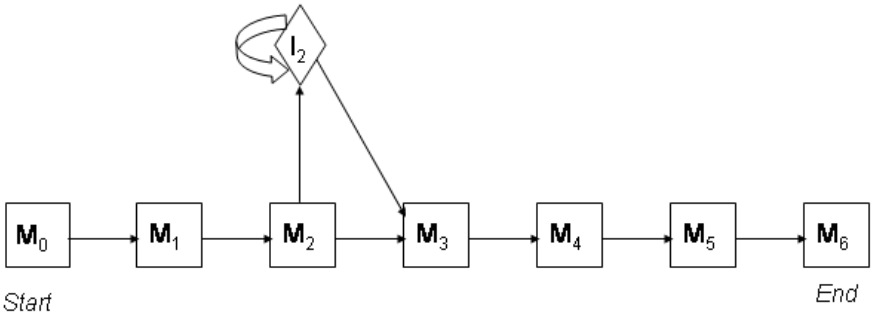
which is equivalent to $\sum_{t=1}^{5} S[O_t, i]$, the score we would have obtained with the PSSM for the `WEIRD` motif.

We now have an HMM that is equivalent to a PSSM for a conserved motif. It can be used to identify motifs of a fixed size, but not cannot handle variations in length. We next extend this model to accommodate insertions and deletions. We can modify the basic HMM to recognize motif instances with insertions, such as $O =$ `WECIRD`, by adding an insertion state between any two match states, M_i and M_{i+1}, as shown in Fig. 5.13a. The emission probabilities for the insertion state are the background frequencies.

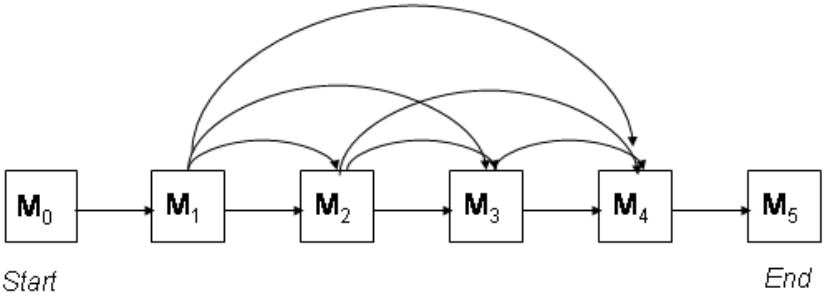
We also wish to recognize motif instances with deletions, e.g., $O =$ `WERD`. One approach to capturing such deletions would be to add edges allowing us to jump over any set of match states. An example of this is shown in Fig. 5.13b. The disadvantage to this approach is that the number of transitions grows rapidly as the number of match states increases. To infer the transition probabilities, we would need a very large set of training data, in which all deletions of all possible sizes were represented. An alternative approach that requires fewer parameters is to model long deletions as sequences of short ones, as seen in the HMM in Fig. 5.13c.

The canonical Profile HMM, shown in Fig. 5.14, combines these features. A Profile HMM has a column containing a Match, an Insertion, and a Deletion state for each position in the conserved pattern. States M_i, I_i, and D_i correspond to the ith position in the pattern. We refer to the number of Match states, not including the silent Start and End states, as the *length* of a Profile HMM. A leading insertion state, I_0, allows for patterns that occur in the middle of a longer sequence. If the pattern ends before the end of the sequence, the remaining sequence is emitted by the insertion state I_n, where n is the last position in the pattern.

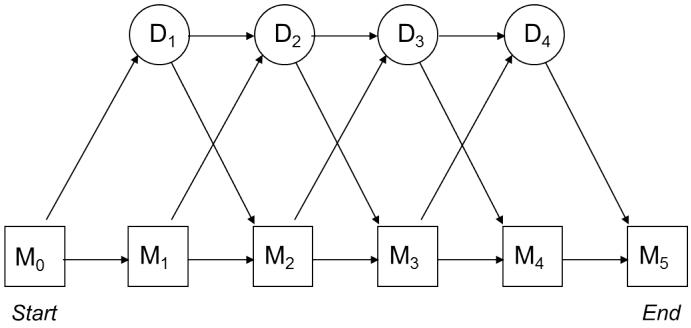
Note that in a Profile HMM there is a path from the Start state, M_0, to the End state, M_{n+1}, that passes only through Insertion and Deletion states. Thus, a Profile HMM can emit a sequence that does not contain an instance of the pattern. Such a sequence would have a low probability, compared with a sequence generated by the Match states.



(a) Insertion model



(b) Deletion model with many transitions



(c) Deletion model with fewer transitions

Figure 5.13: (a) Additional insertion states enable recognition of pattern instances with insertions. This example allows for the insertion of one or more symbols between positions 2 and 3 in the pattern. (b) Adding an arc between every pair of sequences allows for deletions, but the number of transitions grows rapidly with the number of Match states. (c) In this topology, the number of transitions grows linearly with the number of Match states.

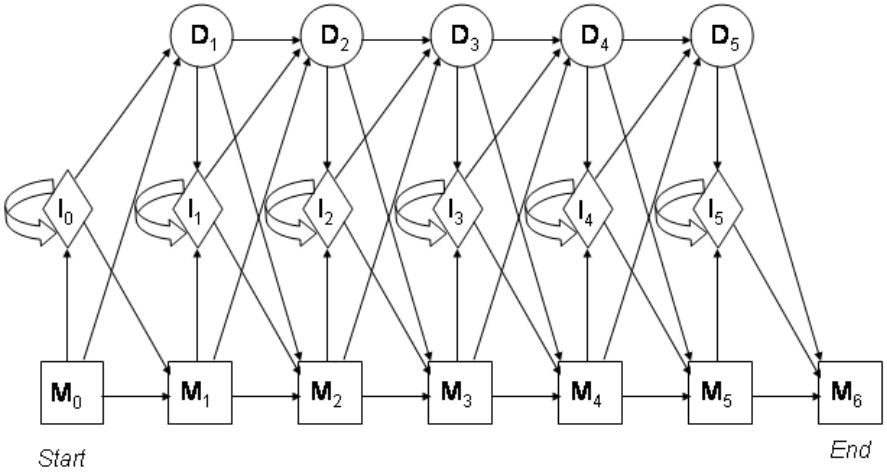


Figure 5.14: A profile HMM of length 5

Parameter estimation

The emission and transition probabilities of a Profile HMM must be estimated from data. If the sequences are aligned and the position of the motif in the alignment is known, then we have labeled training data. In other words, it is possible determine from the alignment which state is associated with each symbol in each sequence. In that case, all we need to do is determine the number of Match states in the Profile HMM, set up the topology, and calculate the parameter values from the labeled data.

Given unlabeled sequences that are known to have a common pattern, we can use the Profile HMM to discover the pattern using the Baum-Welch algorithm to infer the values of the parameters. Once the parameters have been estimated, we use the Profile HMM to label the data. Finally, we construct a multiple sequence alignment by planning symbols with the same label in the same column of the alignment. We give an example of each case below.

Constructing a profile HMM for a variable length motif with labeled data: Profile HMMs, like the one show in Fig. 5.14, can be used to model variable length motifs. We illustrate this process with this example:

```
VG--H
V---N
VE--D
IAADN
```

The length of the Profile HMM should correspond to the typical length of the motif. A rule of thumb is to use the average of the length of the training sequences. The lengths of the above sequences are 3, 2, 3, and 5 (before the gaps were inserted), respectively, yielding an average length of 3.25. This suggests that a Profile HMM of length 3 is appropriate for modeling this pattern. Our HMM will have a silent Start state M_0, Match states M_1, M_2, M_3, Insertion states, I_0, I_1, I_2, I_3, Deletion states, D_1, D_2, D_3, and a silent End state, M_4.

The Insertion and Match states emit the 20 amino acids (for protein motifs), or the four nucleotides (for DNA and RNA motifs). Deletion states emit the indel symbol, e.g. "–". For our Profile HMM, the emission probabilities of the Insertion and Deletion states might look like this:

$$e_{D_j}(\sigma) = 0, \quad e_{D_j}(-) = 1$$

$$e_{I_j}(\sigma) = p(\sigma), \ \forall I_j$$

where $p(\sigma)$ is the background probability of residue σ. In order to estimate the parameters for the Match states, we assign labels to the data using the multiple alignment as a guide. Columns in the alignment that have gaps in less than half of the rows correspond to Match states. Those with more gaps in than half of the rows correspond to Insertion states:

V	G	-	-	H
V	-	-	-	N
V	E	-	-	D
I	A	A	D	N
M_1	M_2	I_2	I_2	M_3

This yields the following labeled sequences:

V	G	H
M_1	M_2	M_3

V	_	H
M_1	D_2	M_3

V	E	D
M_1	M_2	M_3

I	A	A	D	N
M_1	M_2	I_2	I_2	M_3

Note that when a gap ('_') appears in a Match column (M_i), it is labeled as a deletion (D_i). For example, the first gap in the second sequence is labeled D_i.

From these labeled sequences, we can estimate the emission and transition probabilities from Equations 5.6 and 5.7. For example, using $b = 1$ as a pseudocount, we obtain

$$e_{M_1}(V) = \frac{3+1}{4+20}.$$

Similarly, the probability of a transition from M_2 to I_2 is

$$a_{M_2 I_2} = \frac{1+1}{(2+1)+(1+1)+(0+1)} .$$

The three sums in parentheses in the denominator correspond to all possible transitions out of state M_2. The first term in each sum in the number of transitions observed in the training data; the second term is a pseudocount. In the training sequences in our example, there are two transitions from M_2 to M_3, one transition from M_2 to I_2 and no transitions from M_2 to D_3. The other emission and transition probabilities are calculated the same way. The model always starts in M_0, so $\pi_{M_j} = 0$, when $j>0$.

Modeling unlabeled data with a Profile HMM: To discover a pattern in unlabeled data requires the following steps:

1. **Estimating the length:** Given a set of unaligned sequences, where each sequence is an instance of the pattern, we set the length of the HMM (i.e., the number of non-silent Match states) to L, where L is the average sequence length. An example of this type of input would be sequences approximately 50 residues long, where each sequence corresponds to an instance of the Ig domain.
   Alternatively, we might be given sequences that contain a pattern, but are much longer than the pattern. In this case, we must rely on biological knowledge to obtain an initial estimate of the pattern length. The initial estimate of the pattern length can be adjusted later using model surgery (Step 6).
   An example of this type of input would be a set of protein sequences, typically several hundred residues in length, each of which contains an instance of an unknown domain. In this case, you might estimate the length of the pattern to be approximately 100, since that is the length of a typical protein domain.

2. **The topology:** Construct a Profile HMM with $L+2$ Match states, $L+1$ Insertion states, and L Deletion states. M_0 and M_{L+1} are silent states corresponding to the Start state and the End state.

3. **Parameter estimation:** Guess "good" initial parameters (e.g., $a_{M_i\, M_j} \gg a_{M_i\, I_j}$ and $a_{M_i\, M_j} \gg a_{M_i D_j}$) and train the model using the Baum Welch algorithm.

4. **Determining the motif:** Use the Viterbi algorithm or posterior decoding to infer the state path that emitted each sequence. The Viterbi recurrence can be greatly simplified and expressed in terms of log odds for the special case of Profile HMMs (Durbin, pp 108-110). The log odds formulation avoids underflow and reduces length effects. This was not covered in class and you will not be tested on it. Note the

similarity to the dynamic programming algorithm for pairwise alignment.

5. **Multiple Sequence Alignment:** The most likely paths for each sequence obtained from decoding can be used to obtain a multiple alignment of the input sequences. If symbols O_t^c and O_u^d were emitted by the same Match state, then align position t in sequence O^c with position u in sequence O^d. See Ewens and Grant, p 337 - 339 for a discussion and example of multiple sequence alignment using Profile HMMs.

6. **Model surgery:** The topology of the model can be iteratively refined. If more than half of the sequences enter the Delete state, D_j, then remove M_j, D_j, and I_j from the topology. If more than half of the sequences enter the Insertion state, I_j, then add Match, Insertion and Deletion states between positions j and $j+1$.

7. **Re-estimate the parameters:** If the states change due to model surgery, the parameters must be re-estimated. Label the multiple alignment with the new states and calculate the transition and emission probabilities as described above for labeled data. If the number of states that changed is a substantial fraction of the entire HMM, then you may obtain better results by retraining with the Baum Welch algorithm.

Compared with the exact dynamic programming algorithm for multiple sequence alignment, which runs in exponential time, this approach can align many sequences quickly.

Pattern recognition with profile HMMs: Once you have constructed your Profile HMM, how do you determine whether a new, unlabeled sequence, O, contains the motif?

If you have a model for a suitable null hypothesis, H_0, you can obtain a log odds ratio,

$$\log \frac{\Pr(O|H_A)}{\Pr(O|H_0)} ,$$

using the Forward algorithm to determine the probability of the sequence for each model. Typically, H_A would be represented by a profile HMM and H_0 by a background model such as the one shown in Fig. 5.12. This gives a score, but does not infer the location of the motif.

Alternatively, you can find the most likely path using the Viterbi algorithm or posterior decoding. The location of the motif corresponds to the symbols emitted by the Match states. If no symbols were emitted by Match states, then the motif is not present in O.

Chapter 6

Searching Sequence Databases

The goal of a database search is to find all "high-scoring" local alignments (*i.e.*, local alignments with a score above a given threshold) that are significantly more similar than expected by chance. A database search can be used to find homologous sequences[1] for structural and functional predictions and evolutionary analyses. Another application is to compare a protein or a cDNA sequence with genomic DNA, *e.g.* to find gene location or identify intron/exon boundaries. The Basic Alignment Local Search Tool (Blast) is the most widely used approach to searching sequence databases. Blast has two components: a fast heuristic for searching for similar sequences, discussed in Section 6.1, and a statistical framework for evaluating the results of the search, discussed in Sections 6.2 and 6.3.

6.1 The Blast Heuristic

Database searching is essentially a local alignment problem. In theory, dynamic programming could be used to search a database for sequences that are similar to a query sequence. However, the running time for dynamic programing is $O(mn)$, where m is the length of the query sequence and n is the length of the database.

For large databases, the complexity of this approach is prohibitive. The "typical" amino acid query sequence is 250 to 300 residues long, although query sequences can be much longer. For example, the genomic sequence of the BRCA2 gene, including introns and exons, is 84,193 base pairs long. The length of the transcribed BRCA2 mRNA is 11,386 base pairs, including untranslated regions. The BRCA2 amino acid sequence contains 3418 residues. Currently, the GenBank nr database, which includes non-redundant coding sequence translations and sequences from PDB, SwissProt, PIR and PRF (excluding environmental samples from whole genome sequencing projects), contains more than 200 billion base pairs and almost 64 billion amino acids. Thus, in a typical search, the search

[1] Sequences that share common ancestry.

	Non-redundant (nr) sequence database	
	Nucleic Acid Sequences	Amino Acid Sequences
Date:	Nov 24, 2023 7:56 AM	Nov 22, 2023 2:48 AM
Letters:	1,429,214,902,961	249,059,528,165
Sequences:	100,888,011	636,004,760

space size mn is 19 trillion for amino acids or 200 trillion for nucleotides. Instead of dynamic programming, heuristics are used to search databases of this size.

Blast is a heuristic that searches a database for significant local alignments in less than $O(mn)$ time. Blast takes as input a query sequence, Q, of length m and a database, D, of length n. The database is a series of concatenated nucleic or amino acid sequences $D_1, D_2, \ldots D_j \ldots$.

The Blast literature uses the following terminology:

- A *Segment Pair* is an ungapped local alignment.
- A *Maximal Segment Pair (MSP)* is a segment pair whose score cannot be improved by extending or shortening the alignment.
- A *High scoring Segment Pair (HSP)* is a maximal segment pair with score $\mathcal{S} \geq \mathcal{S}_T$, where $\mathcal{S}_T$ is a similarity score threshold (typically defined by the user).

The original Blast heuristic, called "Blast-90" here because it was published in 1990[2], does not handle alignments with gaps. Gapped Blast, published in 1997[3], includes a provision for gapped alignments, as well as several modifications for improved efficiency.

6.1.1 Blast-90

Given a query sequence, Q, of length m and a database, D, of length n, Blast attempts to find all database sequences that contain a maximal segment pair with a score above the reporting threshold, $\mathcal{S}_T$. Instead of comparing the query to every sequence in the database, the Blast heuristic restricts the search for high scoring ungapped alignments to regions of the database that are "promising"; i.e., that are likely to contain an HSP. This strategy requires a fast method for predicting which regions contain an HSP and which do not. Blast does this by using a fast scan to find tiny, high-scoring matches, called "hits," and then extending the hit to obtain HSPs with scores at least $\mathcal{S}_T$. A *hit* is an ungapped alignment

[2]Karlin and Altschul, Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes, 1990. PNAS, 87:2264-2268.

[3]Altschul, *et al.* Gapped Blast and PSI-Blast: a new generation of protein database search programs, 1997. Nucleic Acids Res., 25(17): 3389-3402.

of a word in the query sequence and a word in the database that has a score of at least T, where T is a Blast performance parameter. A *word* or w-mer is a string of w consecutive letters. Typically, w is small (less than 10 residues).

The original Blast-90 heuristic has three main steps:

1. Construct a list, L, containing high scoring words derived from the query sequence.

   DNA: L contains the $m - w + 1$ words of length w that are subsequences of Q.

   Proteins: For each w-mer, z, in Q, add to L all w-mers that have a similarity score $\geq T$, when aligned with z using a suitable substitution matrix. The values of T and w are pre-specified parameters.

   The entries in L are stored in a hash table for fast retrieval. Note that some words in Q will not appear in L. Specifically, any word with a score less than T when aligned with itself will not appear in L.

2. Find hits by scanning the database for w-mers that correspond to a word in L.

   Note that one could also make a list of the high scoring words in the database and compare each w-mer in the query sequence with all words in that list. Intuitively this might seem like a better option because the same hash table could be used for every query and would only have to be rebuilt when the database was updated. However, this would result in a much bigger hash table. In addition, this approach incurs a disk access performance penalty because it requires that the database be accessed randomly rather than scanned sequentially.

3. Extend hits to obtain HSPs with scores at least $\mathcal{S}_T$.

   The time spent on this step is reduced by using a score cutoff. If the score of the extended alignment is lower than the best score seen so far by the amount of the cutoff, then Blast stops extending the alignment in that direction.

The underlying assumption of the Blast heuristic is that most HSPs will contain a hit and that hits that are not contained in an HSP are rare. If a region contains an ungapped alignment with score at least $\mathcal{S}_T$, but there is no word in that alignment with score at least T, then Blast will not discover and report this HSP, resulting in a false negative. On the other hand, if extending a hit does not lead to an ungapped alignment with score at least $\mathcal{S}_T$, then the time spent on this extension is wasted. The trick is to select w and T to obtain a good balance between false negatives and unnecessary extensions. These scenarios are shown graphically in Fig. 6.1.

Three parameters influence the precision, recall, and speed of the heuristic. Increasing the value of $\mathcal{S}_T$ (i.e., making the threshold more stringent) will result in fewer false positives and more false negatives. For a given $\mathcal{S}_T$, the values of w and T are selected to minimize the number of false negatives and the search time. Steps 1 and 2 in the heuristic are relatively

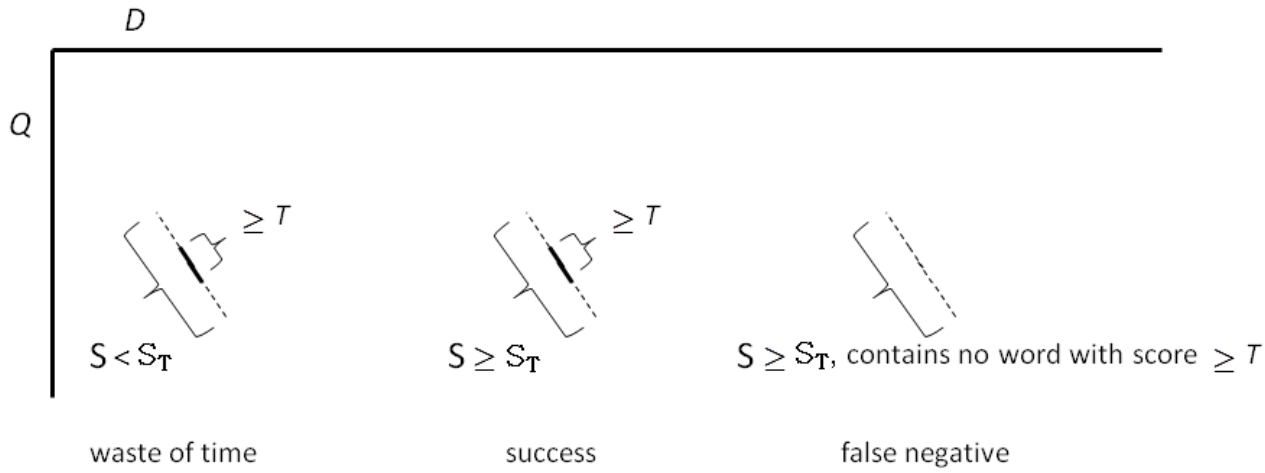


Figure 6.1: An unnecessary extension (left), a successful identification of a matching sequence (middle), and a false negative (right).

fast. Step 3 is slow. Therefore, the goal is to select values for the parameters w and T that limit the number of hits that must be extended in Step 3 without incurring too many false negatives. If the hit threshold, T, is increased, the number of hits - and therefore the number of extensions - will decrease. However, the number of regions that contain a local, ungapped alignment with score greater than $\mathcal{S}_T$, but do not contain a hit, will also increase, resulting in more false negatives. Similarly, decreasing w will decrease the running time and increase the number of false negatives. Note that only $\mathcal{S}_T$ influences the false positive rate, but all three parameters contribute to the false negative rate. A false negative can occur because the heuristic fails to find a related sequence. This is determined by the values of w and T. A false negative also occurs when the search returns a related sequence, but it is not reported because it has a score below the reporting threshold, $\mathcal{S}_T$.

Altschul and his colleagues used simulation studies to estimate the probability, for a given set of parameter values, that hits found in the database would in fact be contained in local, ungapped alignments with score at least $\mathcal{S}_T$. This is discussed in detail in Altschul *et al.*, 1990, which is on electronic reserve. Briefly, they used a statistical approach to minimize the probability of unnecessarily attempting to extend a hit. They determined empirically that a choice of $w = 4$ and $T = 17$ offered a good compromise between the false negative rate and the running time. Note that the values of w and T have been changed since 1990. Different values are used today.

6.1.2 Gapped and Two-Hit Blast

In 1997, three innovations to the Blast algorithm were introduced[2] to address issues in sensitivity and running time:

- Gapped extensions
- Two hit Blast
- Position-Specific Iterative (PSI) Blast

We discuss the first two innovations in detail below.

Despite the early success of the Blast heuristic as a sequence database search tool, the exponential growth of sequence databases created a need for a faster heuristic. By 1997, parameters had been reduced to $w = 3$ and $T = 13$, resulting in many more attempted ungapped extensions in Step 3 of the heuristic on page 127. Since ninety percent of the running time was expended in the third step of the procedure, improving efficiency required reducing the number of extensions without loss of sensitivity.

A second difficulty with the Blast-90 heuristic is that it only finds ungapped alignments. A simple solution might be to find two (or more) MSPs and merge them later. For this to work, however, the heuristic must be able to identify the individual MSPs so that they can be merged. This, in turn, requires that each MSP contain a hit (a word match with score at least T). The probability of finding both MSPs can be increased by decreasing the word score threshold from $T = 13$ to $T = 11$, but this will increase the number of hits found in Step 2, the number of unnecessary extensions in Step 3, and the running time.

Instead, Gapped Blast uses a two phase protocol for selecting candidate regions for a full, gapped alignment: first, ungapped extensions are attempted in those regions that contain a word with score at least $T = 13$. To limit the computational cost, the ungapped extension is terminated if the alignment score drops below X_u, the ungapped extension cutoff. If the score of the resulting MSP exceeds a preset minimum score, then a gapped extension (using dynamic programming) is attempted. Again, to limit the computational cost of this step, the extension is terminated if the alignment score drops below X_g, the gapped extension cutoff. If the score of this gapped alignment exceeds $\mathcal{S}_T$, the resulting match is reported.

This innovation delivered both gapped alignments and higher sensitivity, yet still achieved an improvement in running time. By increasing T from 11 to 13, the number of ungapped extensions was reduced by two thirds. Using the ungapped extensions as a filter for identifying candidates for gapped extension resulted in one gapped extension per 4000 ungapped extensions. Although the computational cost of gapped extensions is 500 times the cost of ungapped extensions, the total running time was reduced by more than a factor of 2.

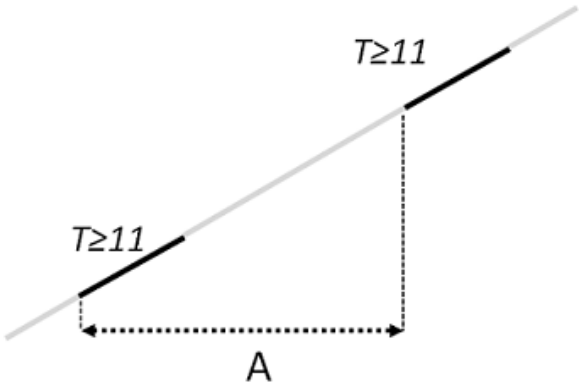


Figure 6.2: In Two-Hit Blast, an extension is triggered if a pair of hits is found on the same diagonal within a distance of A.

The second innovation, Two-Hit Blast, delivered further performance improvement without unduly compromising sensitivity. The underlying rationale is that an HSP will typically contain more than one hit. Better specificity, resulting in fewer extensions, can be obtained by reducing the threshold, T, to obtain more hits, but requiring two hits on the same diagonal in close proximity in order to trigger an ungapped extension (Fig. 6.2).

In Two-Hit Blast, the hit score threshold was reduced to $T = 11$. Ungapped alignments are attempted only when two hits are found on the same diagonal that are separated by a distance no greater than $A = 40$. This modification resulted in 3.2 times as many hits, but decreased the number of extensions by 0.14, yielding an additional two-fold speedup. An example showing the reduction in the number of extensions with Two-Hit Blast is shown in Fig. 6.3.

Combining these two innovations results in the following procedure:

1. Find hits of length w with a similarity score of at least T.
2. If database sequence D_j contains two hits on same diagonal separated by a distance of at most A, perform an ungapped extension to obtain an MSP using cutoff, X_u.
3. If the MSP score in Step 2 exceeds $\mathcal{S}_g$, perform a gapped extension using dynamic programming with cutoff, X_g.
4. If the gapped local alignment score obtained in Step 3 exceeds $\mathcal{S}_T$, report matching sequence, D_j.

6.1.3 PSI-Blast

PSI-Blast, the third innovation, yields improved sensitivity by constructing a Position Specific Scoring Matrix (PSSM) of sequences with significant similarity to the query

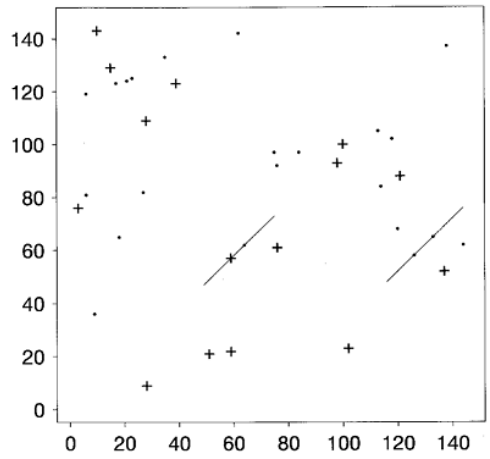


Figure 6.3: Hits with T=11 (.) and T=13 (+) in an alignment of Broad bean leghemoglobin and Horse beta globin, reproduced from Altschul *et al.* (1997). This alignment contains 37 hits when T=11, but only two pairs satisfy the requirements for an extension. In contrast, 15 hits are obtained when T=13, which would result in 15 extensions with the original 1990 Blast algorithm.

sequence. PSI-Blast constructs a PSSM using an iterative process. In the first iteration, PSI-Blast behaves like regular Blast: the query sequence is compared with sequences in the database and a set of HSPs is retrieved. A PSSM is constructed from a local multiple alignment of these matches. The resulting PSSM captures the distribution of amino acid sequences observed at each conserved position in the local multiple alignment obtained from the HSPs. In the next iteration, rather than just searching with the query sequence, candidate matching sequences are scored based on their similarity to the PSSM. Since the PSSM contains more information than a single sequence, some sequences that were not significantly similar to the query sequence alone may be significantly similar to the PSSM. A new PSSM is constructed that incorporates these new matching sequences obtained in the second iteration and the process is repeated. In each subsequent iteration, candidate sequences are compared to the PSSM from the previous iteration, until no further improvement in sensitivity is obtained.

6.2 Blast Statistics

In the previous section, we discussed the Blast search heuristic. Here, Blast statistics, including (1) the statistical significance of finding a match with score $\mathcal{S}$, and (2) properties of sequence statistics and substitution matrices that influence the specificity and sensitivity of database searches.

Given a query sequence, Q, of length m and a database, D, of length n, where D is

a series of concatenated amino acid sequences $D_1, D_2, D_3, \ldots$, Blast attempts to find all matching sequences D_j that have an MSP with score $\mathcal{S} \geq \mathcal{S}_T$, where $\mathcal{S}_T$ is a similarity score threshold (typically user defined). The score of the MSP between Q and D_j is the sum of the scores of the aligned residues

$$\mathcal{S} = \sum_i S^N[\sigma(i), \tau(i)], \tag{6.1}$$

where σ and τ are the subsequences of the query and the matching sequence that participate in the MSP and S^N is a suitable scoring matrix with evolutionary divergence, N. Fig. 6.4 shows an MSP of length l between query sequence, Q and matching sequence, D_j.

The challenge we face in searching a sequence database is to distinguish between sequences that are similar to the query due to shared ancestry and sequences that are similar by chance. We have already introduced a probabilistic framework with log-odds scoring matrices in which $S^N[x, y]$ reflects the relative probabilities of observing x aligned with y under the alternate hypothesis that Q and D_j are related with divergence N and the null hypothesis of chance similarity. However, while the use of a log odds scoring matrix provides a hypothesis testing framework for scoring, it does not account for chance in the context of a database search. To consider whether the sequences retrieved in a database search are significant, we must also consider the length of the query sequence and the size of the database. This is analogous to searching for four-leaf clover. If you search a square yard of lawn for four-leaf clover and find five clover sprigs with four leaflets, you may truly have the luck of the Irish. However, if you search for four-leaf clovers in a square mile, finding five clover sprigs with four leaflets may be unremarkable.

An approach to assessing the statistical significance of similarity scores in a database search was derived in a series of papers by Sam Karlin and Stephen Altschul. Here, we give a sketch of the approach taken to derive these statistics. The details of this derivation are outside the scope of this course. A good presentation is given in the textbook by Ewens and Grant[4].

[4]Statistical Methods in Bioinformatics: An Introduction. Ewens, W., Grant, G. Springer; 2nd edition

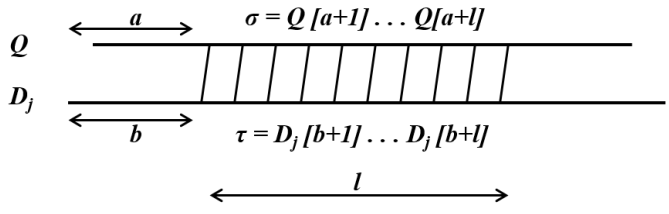


Figure 6.4: An MSP between query sequence, Q and matching sequence, D_j. The MSP is l residues long and starts at residue $a+1$ in Q and at residue $b+1$ in D_j. We use σ and τ, respectively, to designate the subsequences of Q and D_j that participate in the alignment.

To estimate the statistical significance of matching sequences, Karlin and Altschul defined a null hypothesis for database searches and then estimated the distribution of scores of MSPs given that null hypothesis. The null hypothesis is that Q is a sequence of m amino acids that are randomly sampled such that amino acid, x, occurs with the background probability p_x. Similarly, the database D is a randomly sampled sequence of length n, in which x occurs with probability p_x. The background probabilities are the amino acid frequencies observed in typical proteins sequences; for example, the amino acid frequencies in GenBank.

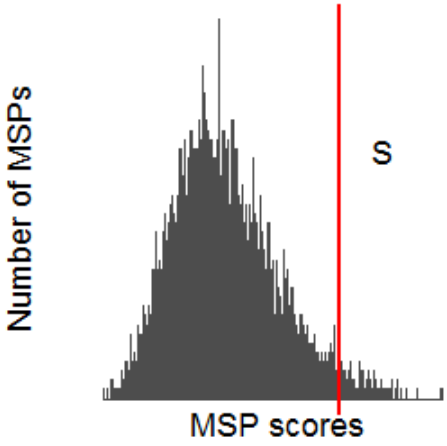


Figure 6.5: A histogram of hypothetical MSP scores under the null hypothesis that the query and matching sequences are unrelated. The E-value of a match with score $\mathcal{S}$ is the number of chance MSPs with a score at least $\mathcal{S}$, i.e., the number of points to the right of the red bar.

The significance of a matching sequence with score $\mathcal{S}$ retrieved in a Blast search is expressed as an "E value". E is defined as the expected number of MSPs with score at least $\mathcal{S}$ under the null hypothesis. Informally, we can think of the E value as the number of chance matches with score at least $\mathcal{S}$ that we would expect to see in a search of a database of size n with a query of length m. In the histogram of hypothetical MSP scores in Fig. 6.5, E corresponds to the sum of the bars to the right of the red line.

Karlin and Altschul estimate the distribution of MSPs by modeling an alignment under the null hypothesis as a random walk. A simple example of this is an ungapped alignment of two nucleic acid sequences, where matches are assigned a score of $+1$ and mismatches are assigned a score of -1. In the random walk corresponding to an ungapped alignment under this simple scoring scheme, our drunk makes a step in the positive direction for every match

(2005)

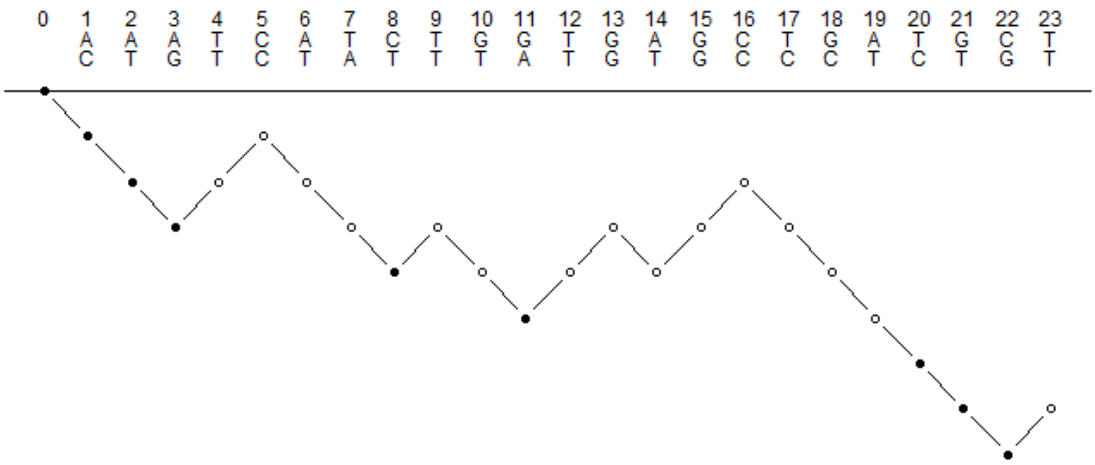


Figure 6.6: An alignment of nucleotide sequences and the corresponding random walk. Ladder points shown in black.

in the alignment and a step in the negative direction for every mismatch. The cumulative alignment score for the first i positions in the alignment corresponds to the position of the drunk after i steps in the random walk. Fig. 6.6 shows the trajectory of the drunk for a pair of aligned nucleotide sequences. An amino acid alignment can also be represented as a random walk. Unlike the nucleic acid case, the step sizes with amino acids are not uniform. Rather, the size of the step corresponding to site i is $S^N[\sigma[i], \tau[i]]$, the score of the pair of residues aligned at that site.

We define a *ladder point*, L, to be a position in the random walk where a new low occurs. The ladder points in Fig. 6.6 are shown in black. Let L_j and L_{j+1} be a pair of successive ladder points that are not immediately adjacent to each other. We define the random variable Y_j to be the position where the cumulative score achieves the maximum value between L_j and L_{j+1}. The segment of the walk from a ladder point L_j to the next maximum Y_j is called an *excursion*. In the example in Fig. 6.6, there are four excursions of length greater than zero: from positions 3 to 5, 8 to 9, 11 to 16, and 22 to 23.

Each excursion in the random walk corresponds to a maximal segment pair. For example, the excursion that starts at position 11 corresponds to the ungapped alignment

```
TGAGC
TGTGC
```

which has a score of 3. Note that the ladder point itself is not included in alignment; since ladder points are new lows by definition, they must correspond to mismatches. The score of an ungapped local alignment that includes the ladder point can always be improved

by removing the ladder point. The distribution of excursion lengths in random walks can therefore be used to model the null distribution of MSP scores. Karlin and Altschul applied known theory about the distribution of excursions in random walks to derive the statistics of HSPs of score at least $\mathcal{S}$ under the null hypothesis. This theoretical development required the following assumptions:

1. The scoring system must allow for at least one positive step and one negative step; i.e., there exists some x and y in Σ, such that $S^N[x, y] \geq 0$ and there exists some z and w such that $S^N[z, w] < 0$.
2. The expected step size is negative; i.e., $\sum_{x,y} p_x p_y \, S^N[x, y] < 0$.

Note that these requirements are very similar to the requirements for local alignment scoring that we proposed based on intuitive arguments at the beginning of the semester. The PAM and BLOSUM matrices both satisfy these conditions. They contain both positive and negative entries and the expected score is negative.

Under these assumptions, Karlin and Altschul derived an expression for the expected number of MSPs with score at least $\mathcal{S}$ under the null hypothesis:

$$E = Km'n'e^{-\lambda \mathcal{S}}, \tag{6.2}$$

where K and λ are constants that depend on the scoring matrix, S^N, and m' and n' are the effective lengths of Q and D after they have been adjusted for edge effects. The correction for edge effects reflects the fact that an alignment that starts near the end of the query sequence will not be long enough to achieve a score of at least $\mathcal{S}$. For example, a segment pair that starts at the very last position in Q will be one residue long and can have a score no higher than the largest value in $\mathcal{S}^N$. The effective length m' is the last possible starting position in Q for an ungapped alignment with an expected score of $\mathcal{S}_T$. Similarly, n' is the effective length of the database, accounting for the fact that an HSP cannot start at the end of a database sequence, D_j. If the database is the concatenation of a series of unrelated protein sequences, then the effective length must also account for the fact that a match cannot begin at the end of one sequence and continue on into the beginning of the next one.

Equation 6.2 makes intuitive sense. First, the expected number of false positives, E, is proportional to the size of the search space, $m'n'$. This is reasonable. If we search a bigger space, we expect to find more matching sequences by chance. Second, E decreases exponentially with $\mathcal{S}$. This is also reasonable. The higher the score, $\mathcal{S}$, the lower the probability of finding a match with a score as least as high as $\mathcal{S}$ by chance.

In the Blast web interface, the user specifies an E value threshold, E_T, that corresponds to the expected number of chance matches that the user is willing to tolerate. The current default is $E_T = 0.05$. The minimum score threshold, $\mathcal{S}_T$, corresponding to E_T, is calculated from Equation 6.2 "behind the scenes" and used to limit the scope of the search.

Normalized bit scores: The score, $\mathcal{S}$, in Equation 6.2 is the *raw* score, obtained by summing the scores of each pair of amino acids in the alignment (Equation 6.1). The current implementation of Blast does not report the raw score of an MSP. Instead, it reports the normalized *bit score*, which is a linear transformation of the raw score:

$$\mathcal{S}_b = \frac{\lambda \mathcal{S} - lnK}{ln2}. \tag{6.3}$$

This gives an expression relating the E value to the bit score that is independent of the parameters, K and λ:

$$E = m'n'2^{-\mathcal{S}_b}. \tag{6.4}$$

Bit scores have several advantages. First, Equation 6.4 is simpler than Equation 6.2. Second, with normalized bit scores, E values from database searches with different scoring matrices can be compared since all dependence on the scoring matrix (*i.e.*, the parameters K and λ) is included in the bit score. Finally, bit scores are in units of bits (not surprisingly), which will be convenient when we discuss the information content of alignments in Section 6.3.2.

6.3 Limitations on retrieval accuracy

Another question of importance is the extent of the statistical power available in a database search to retrieve related sequences, while excluding unrelated matches. A database search can be viewed as a classification problem, in which we attempt to assign scores to database sequences in such a way that related sequences have higher scores than unrelated sequences. If there is no overlap between the score distributions of related and unrelated sequences, then it is possible to achieve perfect precision and recall. (In Information Retrieval, *precision* is the number of items retrieved that are true positives; in this case, sequences that are truly related to the query. *Recall* is the number of true positives in the database that are retrieved by the search.) Failing that, we would like the overlap in the score distributions to be as small as possible (see Fig. 6.7). Several factors contribute to the size of this overlap, including the length of the query sequence, the matrix used to calculate MSP scores, and the frequency of amino acid pairs found in the alignments of related sequences.

6.3.1 Target frequencies

The similarity between the query sequence and a given database sequence is represented by the score of the highest scoring maximal segment pair between the two sequences. Therefore, discrimination between related and chance matches depends on the distribution of related and chance MSP scores. The greater the overlap between these distributions, the more difficult it is to distinguish related matches from chance. MSP scores, in turn, depend on the difference between the amino acid pair frequencies in related and chance MSPs and the substitution matrix used to score those pairs. For any given query sequence, Q, there exists

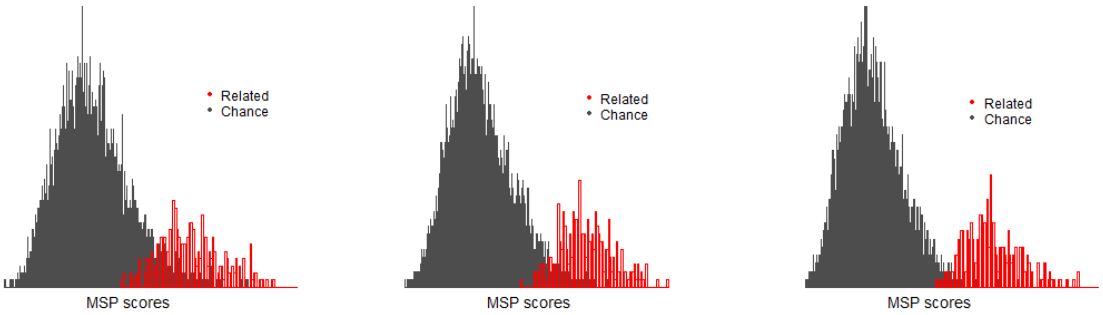


Figure 6.7: The degree of overlap between distributions of the scores for related (red)and chance (black) MSPs represents a fundamental trade off between false positives and false negatives in a database search. The best accuracy that can be obtained is highest in the hypothetical scenario on the right where the overlap is small and lowest in the scenario on the left where overlap is big.

a set of characteristic amino acid pair frequencies, q^Q_{xy}, corresponding to the frequency of x aligned with y in alignments of Q with other proteins in the same family. Altschul calls these "target frequencies."

Recall that the PAM and BLOSUM matrices were both constructed in an explicit log-odds framework, with entries of the form

$$S^N[x,y] = c\ log_2 \frac{q^N_{xy}}{p_x p_y},$$

where the denominator, $p_x p_y$, is the frequency with which the pair (x,y) will occur if amino acids are sampled according to their background frequencies and the numerator q^N_{xy} is the frequency of the amino acid pair (x,y) in alignments of related sequences with evolutionary divergence, N.

In the construction of the PAM and BLOSUM matrices, the values of the q^N_{xy} were estimated from training data. However, any scoring matrix that satisfies the assumptions for Karlin Altschul statistics stated on page 135 implicitly defines a set of characteristic target frequencies, regardless of how the matrix was derived. Given a scoring matrix S, the *theoretical* target frequencies are specified by the equation

$$q^S_{xy} = p_x p_y e^{S[x,y]}. \tag{6.5}$$

The characteristic target frequencies of any arbitrary scoring matrix of, S, can also be determined empirically using the following simulation strategy:

$\forall_x, \forall_y, A_{xy} = 0$
`repeat` {

```
    Generate two amino acid sequences, s1 and s2, with background frequencies, px
    Find the highest scoring MSP between s1 and s2 using matrix, S
    For each aligned pair (x, y) in the MSP, increment Axy.
}
```

This procedure tabulates amino acid pairs that appear in MSPs that receive high scores when scored with matrix S. In other words, these are the amino acid pairs that S "prefers" and the relative frequencies of various pairs will correspond to the ratio of their scores in S. An empirical estimate of the target frequencies can be calculated from the tabulated pair counts:

$$q_{xy}^{S} = \frac{A_{xy}}{\sum_h \sum_i A_{hi}}.$$

For a sufficiently large number of iterations of the simulation strategy, the resulting *empirical* target frequencies will converge to the theoretical frequencies specified in Equation 6.5.

What is the relationship between substitution matrices, target frequencies, and the accuracy of Blast searches? Karlin and Altschul[5] assert that the scoring matrix that corresponds to q_{jk}^{Q}, the target frequencies of Q

$$S^N[x, y] = \log \frac{q_{xy}^{Q}}{p_x p_y}$$

best discriminates between alignments of sequences related to Q and alignments of unrelated sequences with chance similarity. To see why this is true, consider what happens if we assume the opposite. Suppose that S^* is the matrix that best distinguishes chance alignments from related alignments, but that the theoretical target frequencies specified by S^*

$$q_{xy}^{*} = p_x p_y e^{\lambda S^*[x,y]}$$

differ from q_{xy}^{Q}. Since the two sets of frequencies differ, there must be some pair of amino acids a and b such that $q_{ab}^{Q} > q_{ab}^{*}$, and some c and d such that $q_{cd}^{Q} < q_{cd}^{*}$. Thus, we can construct a new substitution matrix by increasing the score of (a, b) pairs relative to $S^*[a, b]$ and decreasing the score of (c, d) pairs relative to $S^*[c, d]$. Using this new substitution matrix will increase the scores of MSPs in alignments of related sequences and therefore yield greater discriminatory power. But that means that $S^*[x, y]$ does not have the best discriminatory power, leading to a contradiction.

The best discriminatory power is obtained by using the substitution matrix with theoretical target frequencies that are identical to the observed amino acid pair frequencies in the alignment of the query and matching sequence. However, when we start the

[5]Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Karlin, S. and Altschul, S. 1990, PNAS, 87:2264-2268.

search the matching sequence(s) is unknown. Moreover, in any given search, we expect to retrieve multiple matching sequences spanning a range of evolutionary divergences, so which substitution matrix should we use? Fortunately, Blast will give reasonable accuracy as long as the observed pair frequencies in the alignments of interest do not deviate too far from the theoretical target frequencies given in Equation 6.5. For many queries, a search with the BLOSUM62 matrix will be sufficient. Greater accuracy can be achieved over the entire range of sequence divergence by searching with the same query two or three times using different matrices corresponding to different levels of divergence, *e.g.*, PAM30, BLOSUM62, and BLOSUM45 (see Table 2 in Altschul 1991[6] and Figure 1 in Altschul 1993[7]). For any given database sequence that is related to the query, one of these matrices will have target frequencies that are reasonably close to the amino acid pair frequencies that are characteristic of the divergence between that sequence and the query.

6.3.2 Information content of substitution matrices

We can also approach the problem of discrimination between related and chance alignments from an information theoretic perspective. As a warm up, let us first consider the problem of determining whether a coin is biased. Each time the coin is tossed, there are two possible outcomes: Heads (H) or Tails (T). According to the alternate hypothesis, H_A, the coin is biased; the probability of observing heads is $q \neq 0.5$. The null hypothesis, H_0, says that the coin is fair; the probability of observing heads is $p = 0.5$. How many tosses do we need to observe before we are ready to decide whether the coin is biased or not?

For a single coin toss, the information available to discriminate between H_A and H_0 is given by the *Relative Entropy*,

$$\mathcal{H} = \sum_{i \in \{H,T\}} q_i \log_2 \frac{q_i}{p_i}.$$

The first term, q_i, is the probability of outcome i given H_A. The second term is the log-odds ratio of the probabilities of outcome i under the alternate and null hypotheses. $\mathcal{H}$, the number of bits of information available to distinguish between H_A and H_0 in a single toss, increases as the deviation between q_i and p_i increases. This makes intuitive sense. If the probability of observing heads with a biased coin is 0.8, only a few tosses will be required to convince ourselves that the coin is, in fact, biased. If the probability of observing heads with a biased coin is 0.51, a much longer sequence of trials will be needed.

The information content of a substitution matrix can be defined analogously. Instead of considering a sequence of coin tosses, we consider a sequence of aligned amino acid pairs

[6] Amino acid substitution matrices from an information theoretic perspective, Altschul, S. 1991, J Mol Biol, 219:555-565.

[7] A protein alignment scoring system sensitive at all evolutionary distances, Altschul, S. 1993, J Mol Evol, 36:290-300.

in an MSP. In this case, there are 210 possible outcomes, corresponding to all possible combinations of two (possibly identical) amino acids, given that we do not distinguish between xy and yx. The amount of information, per position, available to distinguish between chance alignments and alignments in related sequences with divergence N is the relative entropy,

$$\mathcal{H}^N = \sum_{x,y} q_{xy}^N \log_2 \frac{q_{xy}^N}{p_x p_y}, \tag{6.6}$$

where the sum is over all 210 amino acid pairs. The first term is the probability of seeing x aligned with y in related sequences. The second term is the log odds ratio of the probabilities of observing x aligned with y in related sequences and in randomly sampled amino acid pairs. As above, the more q_{xy}^N deviates from $p_x p_y$, the greater the information per position available to determine whether a matching sequence is truly related to the query sequence. Again, this makes intuitive sense. If the amino acid pairs commonly observed in alignments of related sequences are only rarely observed by chance, then even a short alignment is sufficient to convince us that a pair of aligned sequences is truly related. If the amino acid pair frequencies in related and chance alignments are similar, we may need to see a very long alignment before we can decide whether the sequences are related or not.

Since the log odds term in Equation 6.6 is proportional to $S^N[x, y]$, the right hand side can be rewritten to yield

$$\mathcal{H}^N = \sum_{x,y} q_{xy}^N S^N[x, y].$$

In other words, $\mathcal{H}^N$ is the expected score for each position in an MSP in related sequences with divergence N. Thus, we can think of $\mathcal{H}^N$ as the relative entropy or information content of matrix S^N. That is, $\mathcal{H}^N$ is the number of bits available to discriminate between related and chance alignments in each position of an MSP with divergence N, when scored with matrix $S^N[x, y]$. The relative entropies of selected substitution matrices are given in Table 6.1.

BLOSUM	bits/site	PAM	bits/site	% identity
		30	2.57	
		60	2.00	63%
90	1.18	100	1.18	43%
80	0.99	120	0.98	38%
60	0.66	160	0.70	30%
56	0.52	200	0.51	25%
45	0.38	250	0.36	20%

Table 6.1: Relative entropies

6.3.3 Information content of alignments

In the previous section, we discussed the amount of information available to distinguish between related and chance alignments, *per position*. Ultimately, the success of a database search depends the total amount of discriminatory information available in alignments between the query sequence and matching database sequences. Thus, the choice of a substitution matrix for a specific search has practical implications for how the query length may limit the amount of discriminatory information available to find related matches. The discriminatory information available in an alignment depends on the number of bits per position associated with the scoring matrix used in the search. The lower the information per position, the longer the minimum alignment length required to distinguish between related and chance alignments. How long does a sequence have to be in order to find a statistically significant match at a given evolutionary divergence?

We approach this question by first asking how many bits are needed to assert with confidence that the query and matching sequence are not similar by chance. To obtain a rough estimate, we solve Equation 6.4 to obtain an expression for $\mathcal{S}_b$ in terms of m', n', and the E value threshold, E_T:

$$\mathcal{S}_b = \log_2 \left(\frac{m'n'}{E_T} \right). \tag{6.7}$$

This is the minimum score, in bits, required to identify MSPs in related sequences at our specified E value threshold in a search space of size $m'\ n'$. For mathematical convenience, let us consider an E value threshold of $E_T = 1$, which is more permissive than the current default, but not an unreasonable choice. Then

$$\mathcal{S}_b = \log_2(m'n'),$$

is equivalent to the logarithm base 2 of the size of the search space. We can interpret this as the number of bits required to specify the starting positions of the alignment in the query and the database. To see this, note that the indices required to specify the starting position of any alignment in a search space of size $m'n'$, in binary, would require $\log_2 m'$ bits to specify the starting position in the query, and $\log_2 n'$ bits to specify the starting position in the database. Given a matrix with an information content of $\mathcal{H}^N$ bits per position, a rough estimate of the minimum length of an alignment required to obtain at least $\log_2(m'n')$ bits of information is

$$\frac{\log_2(m'n')}{\mathcal{H}^N}. \tag{6.8}$$

Since the alignments found in a given search can never be longer than the query length, Equation 6.8 can be used to estimate an upper bound on the maximum evolutionary divergence of sequences we can expect to retrieve with a particular substitution matrix, given the lengths of the query and the database.

Let's try an example. How many bits are required to find meaningful alignments in a database of 1 billion residues? For a typical amino acid sequence of effective length $m' = 250$ and a database of size $n' = 10^9$,

$$\begin{aligned} log_2(m'n') &= log_2(2.5 \times 10^{11}) \\ &\approx 38 \text{ bits} \end{aligned}$$

are required to distinguish significant MSPs from chance. Since an MSP cannot be longer than the query sequence, this suggests that in a database of length $n = 10^9$, a query sequence must be at least $38/\mathcal{H}^n$ residues long to distinguish significant HSPs from chance. The PAM30 matrix has 2.57 bits per position (Table 6.1), so if the PAM30 matrix is used to score alignments in matching sequences, then the shortest alignment for which significance can be reliably determined is 38/2.57 or 15 residues long. Recall that the PAM30 matrix only yields 2.57 bits per position if σ and τ are truly separated by 30 PAMs. At the other extreme, at least $38/0.36 = 105$ residues are required to find significant alignments with a PAM250 matrix. Again, this is assuming that the divergence between the aligned regions of the query and the match is, in fact, 250 PAMs. If it is not, the number of bits provided by each position of the alignment would be lower than 0.36 and an even longer alignment would be required.

This has implications for searches with short query sequences. Suppose that you wish to find sequences related to a query sequence of 28 residues. In order to obtain the 38 bits required to find significant matches in a database with 1 billion residues, you will need to search with a matrix with a relative entropy that is fairly high. (How many bits per position will you need?) If you have reason to believe that your query sequence is a member of a highly conserved gene family, then you are in luck! The PAM30 matrix will provide enough information to find matches in a database of 1 billion residues, and this matrix is suitable for a conserved family.

If, on the other hand, you have reason to believe that your query sequence is a member of a highly diverged gene family, you have a problem. We know from the Karlin-Altschul "theorem" that we will obtain the best sensitivity with a matrix that corresponds to the evolutionary divergence of the matches we seek. If the family is highly diverged, the best sensitivity will be obtained with the PAM250 matrix, but an alignment of length 28 will only give you $28 \times 0.36 \approx 9$ bits of information, when the sequences have 250 PAMs divergence.

In this case, you could try to find a longer query sequence. Or, you could consider searching a smaller database, which requires fewer bits. (Why?) Perhaps restricting your search to, say, mammals to find related sequences would be sufficient for your study.

Chapter 7

Multiple Sequence Alignment

In multiple sequence alignment, the goal is to align k sequences, so that residues in each column share a property of interest, typically descent from a common ancestor or a shared structural or functional role. Applications of multiple sequence alignment include identifying functionally important mutations, predicting RNA secondary structure, and constructing phylogenetic trees.

Given sequences s_1,...,s_k of lengths n_1,...,n_k, $\alpha = \{s'_1,...,s'_k\}$ is an alignment of s_1,...,s_k if and only if

- $s'_a \in (\Sigma')^*$, for $1 \leq a \leq k$
- $|s'_a| = l$, for $1 \leq a \leq k$, where $l \geq \max(n_1, \ldots, n_k)$
- s_a is the sequence obtained by removing gaps from s'_a
- No column contains all gaps

7.0.1 Scoring a multiple alignment

As with pairwise alignment, multiple sequence alignments (MSAs) are typically scored by assigning a score to each column and summing over the columns. The most common approach to scoring individual columns in a multiple alignment is to calculate a score for each pair of symbols in the column, and then sum over the pair scores. This is called sum-of-pairs or SP-scoring. For global multiple sequence alignment, SP-scoring can be used with either a distance metric or a similarity scoring function. The sum-of-pairs similarity score of an alignment of k sequences is

$$S_{sp}(s'_1, \ldots, s'_k) = \Sigma_{i=1}^{l} \Sigma_{a=1}^{k} \Sigma_{b>a}\ p(s'_a[i], s'_b[i]), \tag{7.1}$$

where l is the length of the alignment. As before, $p(x, y)$ is a numerical score that represents the similarity of x and y and $p(x, \_)$ is the gap score. Further, we define $p(\_, \_)$ to be zero. In pairwise alignment there is no need to assign a value to $p(\_, \_)$, because the definition of a pairwise alignment specifies that no column may contain two gaps. However, in a multiple alignment, two aligned sequences can have a gap in the same column (i.e., $s'_a[i] = s'_b[i] = \_$), as long as there exists at least one sequence in the MSA that does not have a gap in that column.

As an example, let us calculate the SP-score for the alignment of three sequences shown below:

```
s1   A_TT
s2   A_T_
s3   ACAT
```

We can calculate the SP-score for each column separately:

```
            A_TT
            A_T_
            ACAT
          ------
s1, s2    MOMg
s1, s3    MgmM
s2, s3    Mgmg
```

Note that the second column contains two gaps and that these are assigned a score of zero. The total SP-score is $5M + 2m + 4g$. (Is this alignment optimal? If not, how could you improve it?)

We can also use sum-of-pairs with distance scoring for global multiple alignment. This is how we would score the same alignment using unweighted edit distance:

```
            A_TT
            A_T_
            ACAT
          ------
s1, s2    0001
s1, s3    0110
s2, s3    0111
```

The sum-of-pairs edit distance for this alignment is 6.

Sum-of-pairs scoring tends to overestimate the number of mutations required to explain the data. For example, a single mutation is required to explain the column `(A, A, A, G, G)`, when scored on the tree in Fig. 7.1a. In contrast, SP-scoring assigns this column a score of six (Fig 7.1b), because SP-scoring is based on the implicit assumption that each pair of symbols is independent of all other pairs.

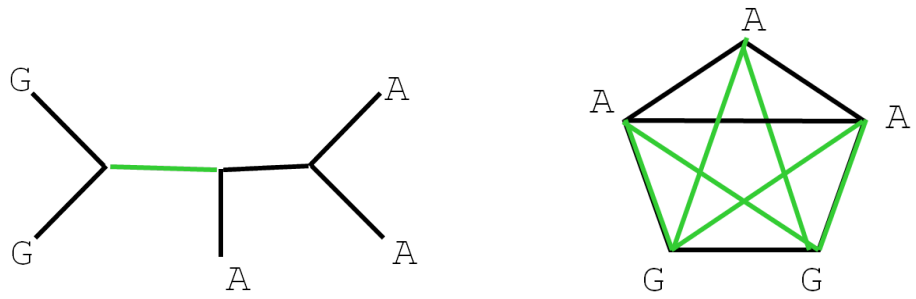


Figure 7.1: Two ways of scoring the column (`A, A, A, G, G`) in a multiple alignment. Green edges represent mismatches. **(a)** Scoring mutations on a tree. **(b)** Sum-of-pairs scoring

Scoring an alignment on a tree, also known as tree alignment, is based on the assumption that the residues in the columns of the multiple sequence alignment share an evolutionary history and that this history can be expressed as a single tree for all columns.

Given a known tree topology as input, the k extant sequences are associated with the k leaves of the tree. Sequences for the internal nodes are selected such that the sum of edge costs, i.e. the total number of mutations required along the branches of the tree, is minimized. Under this model, the cost of an edge (X_i, X_j) in the tree is the minimum number of mutations required to transform sequence X_i into sequence X_j.

In order to use this approach, several issues must be resolved. First, a tree topology is needed. In general, the underlying tree is not known. In fact, multiple sequence alignments are generally used to estimate evolutionary trees and not vice versa. Second, tree alignment methods are often based on the assumption that every column in the alignment has the same underlying tree topology. For many sequences, such as those that that have undergone domain shuffling, this is not the case. Third, in order to compute the branch costs of the tree, we must infer the ancestral sequences associated with the internal nodes. Tree alignment has historically been based exclusively upon the parsimony criterion; that is, on the assumption that mutations are rare and the minimum number of evolutionary steps required to explain the data is the best evolutionary hypothesis. Data that does not happen to be parsimonious can favor the wrong tree model. In addition, column-oriented optimization approaches to MSA usually assume that sequence positions are independent and identically distributed. These assumptions frequently do not hold for biological sequence data.

Finally, for some data sets, a tree may not be a suitable model for describing the relationship between residues in each column, for example, when property of interest is functional or structural. When alignment is used to study function or structure, residues in a column do not always share a common ancestor. Although residues that share a functional or structural role often also share an evolutionary history, this is not the case

when functional or structural roles migrate to neighboring residues. For all of these reasons, tree alignment is rarely used in practice.

Given two sequences s_a and s_b in a multiple alignment, the pairwise alignment of s_a and s_b induced by the MSA is the alignment obtained by deleting the other sequences in the MSA and then removing any column that contains two gaps. For example, in the multiple alignment below,

```
AC_T_G
A_GT_G
ACGTAG
```

the induced alignment of the first two sequences is

```
AC_TG
A_GTG.
```

Further, the pairwise alignment induced by the optimal multiple alignment is not necessarily the optimal pairwise alignment. In this example, the optimal pairwise alignment is

```
ACTG
AGTG.
```

Although the optimal pairwise alignment may have a better score, the induced pairwise alignment may be a biologically more realistic alignment because it reflects properties of the family as a whole.

7.0.2 A dynamic programming algorithm for multiple alignment

The dynamic programming algorithm used for finding the optimal global alignment of two sequences can be extended to the problem of global alignment of k sequences. First, let us consider a dynamic program to align three sequences using a sum-of-pairs similarity score.

Global alignment of three sequences with similarity scoring:

Input:

Sequences s_1, s_2, and s_3 of lengths n_1, n_2, and n_3, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[i_1, 0, 0] &= \mathcal{A}[i_1 - 1, 0, 0] + 2g \\ \mathcal{A}[0, i_2, 0] &= \mathcal{A}[0, i_2 - 1, 0] + 2g \\ \mathcal{A}[0, 0, i_3] &= \mathcal{A}[0, 0, i_3 - 1] + 2g \end{aligned}$$

Recurrence:

$$\mathcal{A}[i_1, i_2, i_3] = \max \begin{cases} \mathcal{A}[i_1 - 1, i_2, i_3] + 2g \\ \mathcal{A}[i_1, i_2 - 1, i_3] + 2g \\ \mathcal{A}[i_1, i_2, i_3 - 1] + 2g \\ \mathcal{A}[i_1 - 1, i_2 - 1, i_3] + 2g + p(s_1[i_1], s_2[i_2]) \\ \mathcal{A}[i_1 - 1, i_2, i_3 - 1] + 2g + p(s_1[i_1], s_3[i_3]) \\ \mathcal{A}[i_1, i_2 - 1, i_3 - 1] + 2g + p(s_2[i_2], s_3[i_3]) \\ \mathcal{A}[i_1 - 1, i_2 - 1, i_3 - 1] + p(s_1[i_1], s_2[i_2]) + p(s_1[i_1], s_3[i_3]) + p(s_2[i_2], s_3[i_3]) \end{cases}$$

Store the indices of the entry in $\mathcal{A}$ that maximize the right hand side of the recurrence in a trace-back matrix, $\mathcal{T}$.

Trace back:

From $\mathcal{T}[n_1, n_2, n_3]$ to $\mathcal{T}[0, 0, 0]$ to obtain the optimal alignment.

Output:

The optimal alignment score, $\mathcal{A}[n_1, n_2, n_3]$.

The optimal alignment of s_1, s_2, and s_3 with respect to similarity function, $\mathcal{S}$.

The dynamic program for multiple sequence alignment has the same structure as the algorithms for pairwise sequence alignment, but the initiation and recurrence steps are more complex. Since the alignment matrix, $\mathcal{A}$, is a 3-dimensional matrix, the first row in each of the three dimensions must be initialized. The algorithm calculates the entries in $\mathcal{A}$ according to the recurrence proceeding diagonally from $\mathcal{A}[0, 0, 0]$ to $\mathcal{A}[n_1, n_2, n_3]$. As in the pairwise case, a trace-back matrix, $\mathcal{T}$, is used to record the indices that gave the optimal score for each i_1, i_2, i_3 prefix. Once the entire matrix has been filled in, the optimal score is found in $\mathcal{A}[n_1, n_2, n_3]$.

It is straightforward, if messy, to generalize the dynamic program for three sequences to a dynamic program for k sequences. To convince yourself that you understand how this works, try writing down the algorithm for four sequences. For three sequences, the recurrence has seven entries. How many entries will there be in the recurrence when $k = 4$? How many entries will there be for arbitrary k?

The computational complexity of the dynamic programming algorithm to align k sequences is $O(n^k 2^k k^2)$. To see this, note that for k sequences, the alignment matrix has $O(n^k)$ entries. For each entry in $\mathcal{A}$, the recurrence relation considers $O(2^k)$ neighboring cells. Calculating the SP-score for each of those neighbors requires $O(k^2)$ time. (Why?) Thus,

the time complexity of the dynamic program for multiple sequence alignment is exponential in the number of sequences. Given 10 sequences of length at most 500, it is possible to calculate the optimal alignment using dynamic programming. For larger problem instances, a heuristic is typically used.

7.0.3 Heuristics for global multiple alignment

The dynamic programming approach to global multiple sequence alignment is framed as an optimization problem. In this approach, we design an optimization criterion over the set of all possible MSAs and then seek the MSA that optimizes this criterion using dynamic programming. The advantage of this approach is that the optimization criterion makes explicit the assumptions upon which the optimization is based. Because they are explicit, these assumptions are open to scrutiny and falsification.

However, this formal optimization approach has disadvantages as well. One, as we have already seen, is that the computational complexity is exponential in the number of sequences. A second problem is the selection of an optimization criterion. In computational biology, the optimization criterion must follow a specific biological model relating the data to the evolutionary, structural, or functional question at hand. If the optimization criterion is not directly linked to a biological model, then the optimal solution may not reflect biological relationships. As we have seen, both sum-of-pairs and tree alignment have limitations in how well they capture the underlying biology.

In practice, the most widely used multiple alignment programs are based on heuristic methods, not only because of the exponential running time of the exact algorithm, but also because heuristics often give MSAs that are more convincing biologically, even though they do not guarantee mathematically optimal alignments. A survey of multiple alignment software based on heuristic methods, *Protein multiple sequence alignment* by Do and Katoh, 2008, is posted in the "Optional reading" section of the course syllabus.

The performance of MSA programs is typically evaluated empirically using curated or automated structural alignments. BALiBase (http://www.lbgi.fr/balibase/), a collection of "high quality, manually refined, reference alignments based on 3D structural superpositions"[1], is one of the most widely used benchmarks. The BALiBase reference data sets are designed to mimic properties of different types of data sets encountered in practice, especially those that are challenging to align. Examples of challenging data sets include highly divergent sequences that are variable in length and have less than 50% identity, related sequences combined with several outlier, or "orphan", sequences, and related sequences that differ due to large insertions, deletions or terminal extensions.

One of the most commonly used heuristic strategies is *progressive alignment*. This approach is used in a number of programs, including the widely-used CLUSTAL family

[1] "BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark." Thompson JD, Koehl P, Ripp R, Poch O., Proteins. 2005 Oct 1;61(1):127-36.

of multiple alignment programs. Given k sequences, $s_1, \ldots, s_k$, of lengths $n_1, \ldots, n_k$, progressive methods construct an alignment as follows:

1. Construct pairwise alignments for all pairs of sequences.
2. Compute $\mathcal{D}$, the matrix of pairwise distances, where $\mathcal{D}[a, b]$ is the distance between sequences s_a and s_b. Note that $\mathcal{D}$ is a symmetric matrix with zeros on the diagonal.
3. Construct a "guide tree", T, from $\mathcal{D}$. T is a rooted tree with k leaves corresponding to the k sequences.
4. Construct an MSA by repeatedly merging intermediate multiple alignments to obtain progressively larger alignments, until all k sequences have been incorporated in the alignment. The order of merging is determined by the guide tree, T.

The merge operation in Step 4 takes as input two multiple alignments of k_1 sequences and k_2 sequences and returns a multiple alignment of $k_1 + k_2$ sequences. This is repeated until all k sequences are incorporated into the alignment. The order in which sequences are merged is determined by a bottom up traversal of the guide tree. For example, if the tree in Fig. 7.2 were the guide tree, then the pairwise alignment of s_1 and s_2 would be merged with the pairwise alignment of s_3 and s_4, yielding an intermediate MSA of four sequences. A similar merging of two pairwise alignments would result in the MSA of s_5, s_6, s_7 and s_8. Finally, these two alignments, of four sequences each, would be merged to obtain a full alignment of eight sequences.

The actual merge operation is carried out using the pairwise global alignment algorithm to align the two alignments, where the input alignments are treated as sequences over an expanded alphabet. Durbin calls this a "profile." Two aligned sequences can be viewed as a sequence over the alphabet $\Sigma' \times \Sigma' \setminus \{(\_\ \_)\}$. For example, when $\Sigma = \{A, C, G, T\}$, this alphabet contains 24 symbols ($\{AA, AC, AG, AT, A\_, CA, \ldots \_T\}$).

We illustrate profile alignment with the case where a multiple alignment of three sequences is obtained by aligning a profile t with a single sequence s. The elements in $s[i]$ are of the form $\binom{x}{y}$, $\binom{x}{-}$, or $\binom{-}{y}$, where x and y are symbols in Σ. The score for aligning $t[i]$ with $s[j]$ is the sum of the scores for aligning the first character in $t[i]$ with $s[j]$ and the second character in $t[i]$ with $s[j]$. For example, when similarity scoring is used and $s[i]$ is of the form, $\binom{x}{y}$, the recurrence relation for calculating the alignment matrix $\mathcal{A}[i, j]$ is

$$\mathcal{A}[i,j] = \max \left\{ \begin{array}{l} \mathcal{A}[i-1, j-1] + p(x, s[j]) + p(y, s[j]), \\ \mathcal{A}[i, j-1] + 2g, \\ \mathcal{A}[i-1, j] + 2g \end{array} \right\}. \tag{7.2}$$

When t[i] contains an indel (i.e., $t[i]$ is of the form $\binom{x}{-}$), the recurrence relation for calculating $\mathcal{A}[i, j]$ is

$$\mathcal{A}[i,j] = \max \left\{ \begin{array}{l} \mathcal{A}[i-1, j-1] + p(x, s[j]) + g, \\ \mathcal{A}[i, j-1] + 2g, \\ \mathcal{A}[i-1, j] + g \end{array} \right\}. \tag{7.3}$$

Note that $p(x, y)$, the similarity of x and y, does not appear in the right hand side of Equation 7.2. Similarly, the gap score for the alignment of $\frac{x}{-}$ does not appear in the recurrence in Equation 7.3. This is because the two symbols in $s[i]$ were compared and scored during the pairwise alignment in a previous step in the progressive alignment procedure.

A key aspect of the merging operation is that we are not allowed to modify the profiles being aligned. For the example above, that means we cannot change juxtaposition of the two symbols in $t[i]$, even if modifying the alignment in t would result in a better alignment with s. This is called the "once a gap, always a gap" rule (although it also applies to mismatches). A consequence of this rule is that if a bad decision is made with regard to the placement of gaps early in the procedure, then that bad decision will propagate through subsequent iterations and cannot be corrected. It is this rule that makes progressive alignment a fast heuristic; that is, this rule underlies both the improvement in running time and the possibility that the result may be suboptimal.

The complexity of progressive alignment is $O(k^2n^2)$, where $n = \max\{n_a\}, 1 \leq a \leq k$. Calculating the distance matrix in Step 2 requires $O(k^2)$ pairwise alignments. The cost of each pairwise alignment is $O(n^2)$. The merging process also requires $O(k^2n^2)$ time. The computational complexity of merging depends on the number of profile alignments required, the number of cells in the alignment matrix for each profile alignment, and number of comparisons required for each cell. The size of alignment matrix is $O(n^2)$ in all profile alignments. The number of comparisons for a single cell in an alignment of two profiles with k_1 and k_2 sequences, respectively, is $O(k_1 \cdot k_2)$.

The number of profile alignments and the values of k_1 and k_2 depend on the shape of the guide tree. At one extreme, we have a completely unbalanced guide tree with k sequences (e.g., Fig. 7.3, where $k = 8$). In this case, each merge represents an alignment of a single sequence ($k_1 = 1$) with a profile of size $k_2 = h$, $1 \leq h \leq k - 1$, resulting in a complexity of

$$\sum_{h=1}^{k-1} n^2 h,$$

which is $O(k^2n^2)$.

At the other extreme, we have a balanced guide tree with k leaves (e.g., Fig. 7.2). We consider the case when k is a power of 2. We leave the case where k is not a power of two to the masochistic reader. In a balanced guide tree, there are $k/2^h$ merges at height, h. Each of these represents an alignment of two profiles, each comprising 2^{h-1} sequences. The computational complexity is the sum of the complexity of the merges at height h, $1 \leq h \leq \log_2 k$, or

$$\sum_{h=1}^{log_2 k} n^2 \cdot (2^{h-1})^2 \cdot \frac{k}{2^h}.$$

This reduces to

$$kn^2 \cdot \sum_{h=1}^{log_2 k} 2^{h-2}. \tag{7.4}$$

It is easy to verify that the series

$$\sum_{i=1}^{N} 2^{(i-1)} = 2^N - 1. \tag{7.5}$$

Substituting the right hand side of Equation 7.5 into Equation 7.4, where $N = \log_2 k$, we obtain

$$\frac{1}{2}\, kn^2(2^{(\log_2 k)} - 1).$$

This reduces to $(k-1)kn^2/2$, so we again obtain a computational complexity of $O(k^2n^2)$.

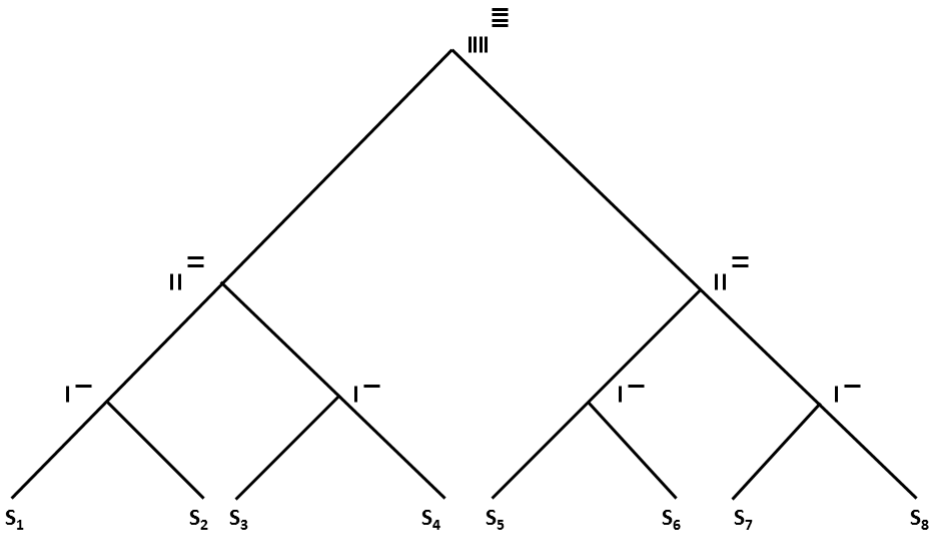


Figure 7.2: A balanced guide tree for 8 sequences.

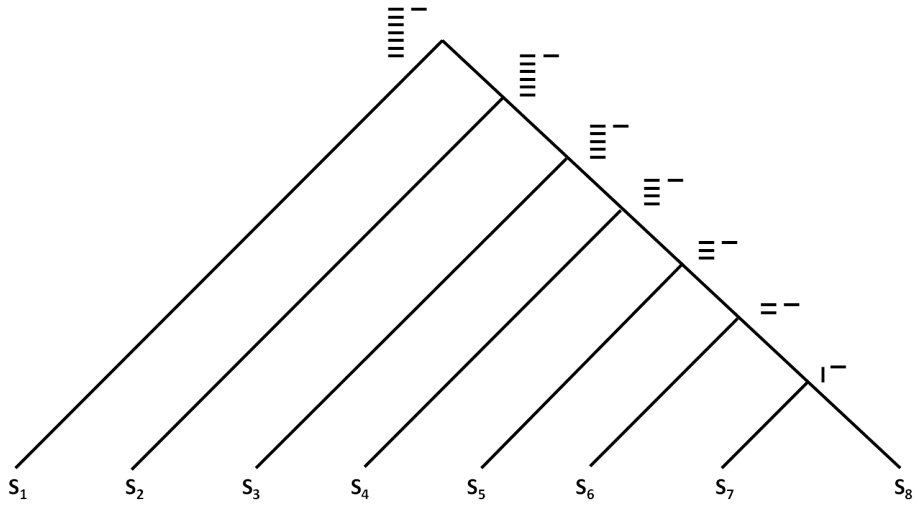


Figure 7.3: An unbalanced guide tree for 8 sequences.

Bibliography