y	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
7	0.990					
8	0.682					
9	0.249	0.999				
10	0.069	0.824				
11	0.018	0.353	0.995			
12	0.004	0.102	0.733			
13	0.001	0.028	0.281			
14		0.007	0.079	0.487		
15		0.002	0.020	0.154		
16			0.005	0.041	0.224	
17			0.002	0.010	0.061	
18				0.003	0.016	0.074
19				0.001	0.004	0.019
20				0.000	0.001	0.005
21						0.001
22						0.000

Table 6.6. Approximate P-values for the maximum of 75,000 iid generalized geometric random variables, for various values of k and y, calculated using equations (3.51), (6.5) and (6.6). $p = \frac{1}{4}$.

cult, and we do not pursue further topics here. Recent research results may be found, for example, in Bailey and Gribskov (1998), Jonassen, Collins, and Higgins (1995), Karlin and Brendel (1992), Neuwald and Green (1994), and Rigoutsos and Floratos (1998).

6.4 Alignment Algorithms for Two Sequences

6.4.1 Introduction

One way to discriminate between good and bad alignments is to use a scoring scheme. A simple example of a scoring scheme is

$$\text{(the number of matches)} - \text{(the number of mismatches and indels)}. \quad (6.7)$$

Scoring schemes used for aligning DNA are often not much different from this simple scheme. For protein sequences, however, a more complex scoring scheme is appropriate. Commonly used scoring schemes are developed using statistical analysis of existing data, and we discuss the statistical theory behind these scoring schemes in Section 6.5. For now, we assume that we have assigned a score to each alignment in a meaningful way that reflects the likelihood that this alignment was produced as a consequence of divergence from a common ancestor. Then we can consider the alignments with the

"best" score, and we can define the score of the sequence pair to be this best score. What "best" means here depends on whether high scores in the scoring scheme are more indicative of relatedness (so the "best" score is the maximum over all alignments), or whether low scores are more indicative (so "best" is the minimum).

This mathematical framework allows a statistical analysis where we make inferences about the relatedness of the sequences. We can investigate the hypothesis that they did indeed diverge from a common ancestor by considering the probability of the observed score (or one more extreme) arising by chance, under some appropriate model of evolution. If the two sequences are judged to be related, we can use their alignment to discover common patterns in the sequences. This is useful in particular for finding functional domains. Finally, by comparing scores among several different species we can get information to help reconstruct the phylogenetic tree that relates them all.

Scores of alignments consist of two main types: *similarity scores* and *distance scores* (also commonly called *distance measures*). In similarity scores the higher the score, the more closely related are the two aligned sequences; in the distance measures the opposite is the case. In the remainder of this section we use similarity scores. These are usually computed as the sum of individual scores, one for each aligned pair of residues, together with a score for each gap. We will denote by $s(X, Y)$ the score assigned to the aligned pair consisting of the residues X and Y. This score reflects how conservative the substitution represented by the alignment of X with Y is. For example, it is much less likely that the amino acid W (tryptophan) will be substituted for V (valine) in a functional domain than it is that W will be substituted for R (arginine). (This is not only an empirically observed fact, but also makes sense in terms of the chemical properties involved.) Thus the score $s(W, V)$ assigned to an alignment of the two symbols W and V is lower than $s(W, R)$, the score assigned to an alignment of the two symbols W and R. The score assigned to a gap of length ℓ is usually a function of ℓ, which we denote by $\delta(\ell)$. It represents the cost of having a gap of length ℓ and is therefore zero or negative. The simplest gap penalty model is a *linear gap model*, where $\delta(\ell) = -\ell d$ for some non-negative constant d, called the *linear gap penalty*. Therefore, in the linear gap model, each indel in a gap is weighted in the same way, namely by a penalty of d.

Thus if the alphabet has size N ($N = 4$ for nucleotides and $N = 20$ for amino acids), a scoring scheme consists of an $N \times N$ matrix S and a gap cost function δ. The matrix S is called a *substitution matrix* and the entry in its ith row and jth column is the score of the alignment of the ith and jth symbols in the alphabet.

Example. Consider the comparison of two nucleotide sequences with a simple scoring scheme that assigns $+1$ to each match, -1 to each mismatch, and a linear gap score with $d = 2$. Then the score for the following alignment of

the two sequences *cttagg* and *catgagaa* is $1-1+1-2+1-2+1-4=-5$:

$$\begin{array}{cccccccccc} c & t & t & a & g & - & g & - & - \\ c & a & t & - & g & a & g & a & a \end{array}$$

One of the main aims of the statistical theory is to find for nucleotides, and more importantly for amino acids, what an optimal scoring scheme should be. This matter is taken up in detail in Section 6.5 and Chapter 10.

6.4.2 Gapped Global Comparisons and Dynamic Programming Algorithms

Suppose that we are given a scoring scheme made up of a substitution matrix and a linear gap penalty. Our aim is to find, of the possible global alignments of two sequences (with gaps allowed), the one (or those ones) with the highest score. One method in principle for doing this is to list exhaustively all possible alignments and their scores, and then note the highest-scoring alignment(s). However, when the sequences are long, this is not computationally feasible, and more efficient algorithms are needed. We describe one such algorithm below, but first we justify our assertion that the exhaustive search illustrated above is indeed not efficient, by getting a sense of how large the number of global alignments between a sequence $\boldsymbol{x} = X_1X_2\ldots X_m$ of length m and a sequence $\boldsymbol{y} = Y_1Y_2\ldots Y_n$ of length n is. We will denote this number by $c(m,n)$. Since there is no point in matching two deletions, no alignments of one indel with another are allowed.

Let $g(m,n)$ be the number of groups obtained by grouping together those alignments that have the same combination of aligned residue pairs ignoring the indels. Then $g(m,n) < c(m,n)$, and this provides a lower bound for $c(m,n)$. We can compute $g(m,n)$ as follows.

The number k of aligned residues for two sequences of lengths m and n is between 0 and $\min\{m,n\}$. Moreover, for each such k there are $\binom{m}{k}$ ways of choosing the residues of $\boldsymbol{x}$ that align with residues of $\boldsymbol{y}$, and $\binom{n}{k}$ ways of choosing the residues of $\boldsymbol{y}$ that align with residues of $\boldsymbol{x}$. So there are $\binom{m}{k}\binom{n}{k}$ alignments with k aligned residues. Therefore,

$$g(m,n) = \sum_{k=0}^{\min\{m,n\}} \binom{m}{k}\binom{n}{k}. \tag{6.8}$$

From the result of Problem 6.1 below, it follows that

$$g(m,n) = \binom{m+n}{n}. \tag{6.9}$$

In particular, when $m=n$,

$$g(n,n) = \binom{2n}{n}.$$

This number grows quite fast with n. Stirling's approximation (B.4), and even more directly (B.5), shows that

$$\binom{2n}{n} \sim \frac{2^{2n}}{\sqrt{\pi n}}. \tag{6.10}$$

Thus the number $c(1,000, 1,000)$ of global alignments between two sequences each of length 1,000 satisfies

$$c(1,000, 1,000) \geq g(1,000, 1,000) \cong \frac{2^{2,000}}{\sqrt{1,000\pi}} \cong 10^{600}.$$

This shows why it is not feasible to examine all possible alignments. This motivates the search for algorithms that can compute the best score efficiently and an alignment with this score, without having to examine all possibilities. One such algorithm is the Needleman–Wunsch algorithm (1970), and we discuss a version of this procedure introduced by Gotoh (1982). These are examples of dynamic programming algorithms, and we use them to illustrate the general concept of dynamic programming.

The input consists of two sequences,

$$\boldsymbol{x} = X_1X_2 \dots X_m \text{ and } \boldsymbol{y} = Y_1Y_2 \dots Y_n,$$

of lengths m and n, respectively, whose elements belong to some alphabet of N symbols (for DNA or RNA sequences $N = 4$, for proteins $N = 20$). We assume that we are given a substitution matrix S and a linear gap penalty d. The output consists of the highest score over all alignments between $\boldsymbol{x}$ and $\boldsymbol{y}$ and a highest-scoring global alignment between $\boldsymbol{x}$ and $\boldsymbol{y}$.

The broad approach is to break the problem into sub-problems of the same kind and build the final solution using the solutions for the sub-problems: This is the basic idea behind any dynamic programming algorithm. In this problem we find a highest-scoring alignment using previous solutions for highest-scoring alignments of smaller sub-sequences of $\boldsymbol{x}$ and $\boldsymbol{y}$. We denote by $\boldsymbol{x}_{1,i}$ the initial segment of $\boldsymbol{x}$ given by $X_1X_2 \cdots X_i$ and similarly we denote by $\boldsymbol{y}_{1,j}$ the initial segment of $\boldsymbol{y}$ given by $Y_1Y_2 \cdots Y_j$. For $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$, we denote by $B(i, j)$ the score of a highest-scoring alignment between $\boldsymbol{x}_{1,i}$ and $\boldsymbol{y}_{1,j}$. For $i = 1, 2, \dots, m$, we denote by $B(i, 0)$ the score of an alignment where $\boldsymbol{x}_{1,i}$ is aligned to a gap of length i, so $B(i, 0) = -id$. Similarly, for $j = 1, 2, \dots, n$, we denote by $B(0, j)$ the score of an alignment where $\boldsymbol{y}_{1,j}$ is aligned to a gap of length j, so $B(0, j) = -jd$. Finally, we initialize $B(0, 0) = 0$. These calculations lead to an $(m+1) \times (n+1)$ matrix B. The entry in the last row and in the last column of B, namely $B(m, n)$, is the score of a highest-scoring alignment between our two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$, and it is one of the things we want our algorithm to output.

The essence of the procedure is to fill in the elements of the matrix B recursively. We already have the values of B at $(0, 0)$, $(i, 0)$, and $(0, j)$, for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. Now we proceed from top left to

bottom right by noting that a highest-scoring alignment between $\boldsymbol{x}_{1,i}$ and $\boldsymbol{y}_{1,j}$ could terminate in one of three possible ways, namely, with

$$\begin{matrix} X_i \\ Y_j \end{matrix}, \quad \begin{matrix} X_i \\ - \end{matrix}, \quad \text{or} \quad \begin{matrix} - \\ Y_j \end{matrix}.$$

In the first case, $B(i,j)$ is equal to the sum of the score for a highest-scoring alignment between $\boldsymbol{x}_{1,i-1}$ and $\boldsymbol{y}_{1,j-1}$ together with the extra term $s(i,j)$ to account for the match between X_i and Y_j; that is, $B(i,j) = B(i-1,j-1) + s(i,j)$. In the second case, $B(i,j)$ is equal to the sum of the score for a highest-scoring alignment between $\boldsymbol{x}_{1,i-1}$ and $\boldsymbol{y}_{1,j}$ together with an extra term $-d$ to account for the indel to which X_i is aligned, (i.e., $B(i,j) = B(i-1,j)-d$). Similarly, in the third case, $B(i,j) = B(i,j-1)-d$. These are all the possible options, and hence $B(i,j)$ is the highest of the three. In other words,

$$B(i,j) = \max\{B(i-1,j-1)+s(i,j), B(i-1,j)-d, B(i,j-1)-d\}. \quad (6.11)$$

In this way we recursively fill in every cell in the matrix B and determine the value of $B(m,n)$, which is the desired maximum score. The running time of this algorithm is clearly $O(mn)$. To find an alignment that has this score we must keep track, at each step of the recursion, of one of the three choices giving the value of the maximum. Although there could be more than one choice giving the maximum, if we are interested in finding only one alignment, we choose one and keep a pointer to it. Once $B(m,n)$ is obtained, by tracing back through the pointers, we can reconstruct an alignment with the highest score. We now illustrate this procedure with an example.

Example. Let $\boldsymbol{x} = gaatct$ and $\boldsymbol{y} = catt$, so that $m = 6$ and $n = 4$. Using the same scoring scheme as in the example in Section 6.4.1, B is given in Figure 6.1, where we have used arrows to denote where each cell came from. The best score for an alignment is given by the element in the bottom rightmost cell, which is -2. Tracing back along the bold arrows, we get the highest-scoring alignment

$$\begin{matrix} g & a & a & t & c & t \\ c & - & a & t & - & t \end{matrix}.$$

By making different choices of arrows in the traceback procedure we can get the following other alignments, which are also highest-scoring, (i.e., which also have a score of -2):

$$\begin{matrix} g & a & a & t & c & t \\ c & a & - & t & - & t \end{matrix} \quad \text{and} \quad \begin{matrix} g & a & a & t & c & t \\ - & c & a & t & - & t \end{matrix}.$$

We next consider modifications of the Needleman–Wunsch algorithm, which can be used to address other kinds of pairwise alignment problems.

	–		c		a		t		t
–	0		−2		−4		−6		−8
		↖		↖		↖		↖	
g	−2		−1	←	−3	←	−5	←	−7
		↖	↑	↖					
a	−4		−3		0	←	−2	←	−4
		↖	↑	↖	↑	↖		↖	
a	−6		−5		−2		−1	←	−3
		↖	↑		↑	↖		↖	
t	−8		−7		−4		−1		0
		↖			↑		↑	↖	↑
c	−10		−7		−6		−3		−2
			↑	↖	↑	↖	↑	↖	
t	−12		−9		−8		−5		−2

Figure 6.1.

6.4.3 Fitting One Sequence into Another Using a Linear Gap Model

In this section we address the following problem: Given two sequences, a longer and a shorter one, find the sub-sequence(s) of the longer one that can be best aligned with the shorter sequence, where gaps are allowed. This procedure is relevant when one is interested in locating a specified pattern within a sequence.

Let $\boldsymbol{x} = X_1X_2 \dots X_m$ and $\boldsymbol{y} = Y_1Y_2 \dots Y_n$ be two sequences with $n \geq m$. For $1 \leq k \leq j \leq n$, denote by $\boldsymbol{y}_{k,j}$ the sub-sequence of $\boldsymbol{y}$ given by $Y_kY_{k+1} \dots Y_j$. For two sequences $\boldsymbol{u}$ and $\boldsymbol{v}$, denote by $B(\boldsymbol{u}, \boldsymbol{v})$ the score of a highest-scoring (global) alignment between $\boldsymbol{u}$ and $\boldsymbol{v}$. Our aim is to find

$$\max\{B(\boldsymbol{x}, \boldsymbol{y}_{k,j}) : 1 \leq k \leq j \leq n\}. \tag{6.12}$$

For each choice of k and j the running time of the Needleman–Wunsch algorithm, giving the value of $B(\boldsymbol{x}, \boldsymbol{y}_{k,j})$, is $O(m(j-k))$. Thus if we used this algorithm for all possible choices of k and j, and then took the maximum over all such choices, the total running time would be $O(mn^3)$, since there are $\binom{n}{2}$ possible choices for j and k. We now illustrate another approach with a better running time, namely an $O(mn)$ running time.

For $1 \leq i \leq m$ and $1 \leq j \leq n$, let $F(i, j)$ be the maximum of the scores $B(\boldsymbol{x}_{1,i}, \boldsymbol{y}_{k,j})$ over the values of k between 1 and j. That is, of all the possible scores for highest-scoring alignments between the initial segment of $\boldsymbol{x}$ up to x_i and the segments of $\boldsymbol{y}$ ending at y_j and beginning at some k we take $F(i, j)$ to be the greatest of such scores. The value of (6.12) is the maximum of $F(m, j)$ over all values of j between 1 and n. To find this, we initialize $F(i, 0) = -id$ for $1 \leq i \leq m$ and initialize $F(0, j) = 0$ for $0 \leq j \leq n$, since deletions of the beginning of $\boldsymbol{y}$ should clearly be without

penalty. Then we fill in the matrix F recursively by

$$F(i,j) = \max\{F(i-1,j-1) + s(i,j), F(i,j-1) - d, F(i-1,j) - d\},$$

where the reasoning behind this formula is analogous to that behind (6.11). Note that there might be more than one value of j giving the maximum score. In order to recover the highest-scoring alignments of $\boldsymbol{x}$ to sub-sequences of $\boldsymbol{y}$ we can keep pointers, as in the Needleman–Wunsch algorithm.

6.4.4 Local Alignments with a Linear Gap Model

Another interesting alignment problem is to find, given two sequences, which respective sub-sequences have the highest-scoring alignment(s) (with gaps allowed). This is called a local alignment problem, and it is appropriate when one is seeking common patterns/domains in two sequences.

In the following we make the assumption that the scoring scheme we use is such that the expected (or mean) score for a random alignment is negative. If this assumption did not hold, then long matches between sub-sequences could score highly just because of their lengths, so that two long unrelated sub-sequences could give a highest-scoring alignment. Clearly, we do not want this to occur.

For $1 \le h \le i \le m$ we denote by $\boldsymbol{x}_{h,i}$ the sub-sequence of $\boldsymbol{x}$ given by $X_h X_{h+1} \dots X_i$. With the notation as in the previous section, we want to find

$$\max\{B(\boldsymbol{x}_{h,i}, \boldsymbol{y}_{k,j}) : 1 \le h \le i \le m, 1 \le k \le j \le n\}, \tag{6.13}$$

when this is non-negative. There are $\binom{m}{2}\binom{n}{2}$ pairs of sub-sequences of $\boldsymbol{x}$ and $\boldsymbol{y}$, one for each choice of h and i among m possible values and of k and j among n possible values. Thus computing a highest-scoring alignment for each pair, using the Needleman–Wunsch algorithm, requires a total running time of $O(m^3 n^3)$. Clearly, we want to give a more efficient approach to this problem. Such an approach is provided by the Smith–Waterman algorithm (Smith and Waterman (1981)) which computes (6.13) in $O(mn)$ time. The procedure is as follows.

For each $1 \le i \le m$ and $1 \le j \le n$, we define $L(i,j)$ to be the maximum of 0 and the maximum of all possible scores for alignments between a sub-sequence of $\boldsymbol{x}$ ending at X_i and one of $\boldsymbol{y}$ ending at Y_j. That is,

$$L(i,j) = \max\{0, B(\boldsymbol{x}_{h,i}, \boldsymbol{y}_{k,j}) : 1 \le h \le i, 1 \le k \le j\}.$$

The reason we want $L(i,j) = 0$ when the max of the $B(\boldsymbol{x}_{h,i}, \boldsymbol{y}_{k,j})$'s is negative is because it is sensible to always remove the first part of an alignment if this part has a negative score, as it will just decrease the overall score of the alignment. Then the maximum of 0 and (6.13) is the maximum of $L(i,j)$ over all values of i between 1 and m and of j between 1 and n. To

determine this maximum we again use dynamic programming, by initializing $L(i,0) = 0 = L(0,j)$ for $0 \le i \le m$ and $0 \le j \le n$ (since deletions at the beginning or end of our two sequences should not be penalized), and by computing

$$L(i,j) = \max\{0, L(i-1,j-1) + s(i,j), L(i-1,j) - d, L(i,j-1) - d\}.$$

We then calculate the maximum of $L(i,j)$ over all values of i and j. As in the previous maximizing procedures there might be more than one highest-scoring local alignment. To find a highest-scoring alignment, we follow the traceback procedure previously described. However, for this algorithm, we stop this process when we encounter a 0.

Figure 6.2 shows an example of an $L(i,j)$ matrix arising in locally aligning two sequences of lengths 7 and 10. In this example, the score of an

		Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}
	0	0	0	0	0	0	0	0	0	0	0
X_1	0	0	0	0	0	0	0	0	0	0	0
X_2	0	0	0	5	0	5	0	0	0	0	0
X_3	0	0	0	0	2	0	20	12	4	0	0
X_4	0	10	2	0	0	0	12	18	22	14	6
X_5	0	2	16	8	0	0	4	10	18	28	20
X_6	0	0	8	21	13	5	0	4	10	20	27
X_7	0	0	6	13	18	12	4	0	4	16	26

Figure 6.2.

optimal local alignment of the two sequences is 28, and there is only one alignment of sub-sequences giving this score, the one indicated by the bold arrows, which is

$$\begin{array}{cccccc} X_2 & X_3 & - & X_4 & X_5 \\ Y_5 & Y_6 & Y_7 & Y_8 & Y_9 \end{array}.$$

6.4.5 Other Gap Models

There are many variants and extensions of the algorithms discussed above. For example, while the linear gap model used above is appealing in its simplicity, it is often not appropriate for biological sequences, since often