Using an additive scoring scheme corresponds to an assumption that we can consider mutations at different sites in a sequence to have occurred independently (treating a gap of arbitrary length as a single mutation). All the algorithms in this chapter for finding optimal alignments depend on such a scoring scheme. The assumption of independence appears to be a reasonable approximation for DNA and protein sequences, although we know that interactions between residues play a very critical role in determining protein structure. However, it is seriously inaccurate for structural RNAs, where base pairing introduces very important long-range dependencies. It is possible to take these dependencies into account, but doing so gives rise to significant computational complexities; we will delay the subject of RNA alignment until the end of the book (Chapter 10).

Substitution matrices

We need score terms for each aligned residue pair. A biologist with a good intuition for proteins could invent a set of 210 scoring terms for all possible pairs of amino acids, but it is extremely useful to have a guiding theory for what the scores mean. We will derive substitution scores from a probabilistic model.

First, let us establish some notation. We will be considering a pair of sequences, x and y, of lengths n and m, respectively. Let x_i be the ith symbol in x and y_j be the jth symbol of y. These symbols will come from some alphabet $\mathcal{A}$; in the case of DNA this will be the four bases {A, G, C, T}, and in the case of proteins the twenty amino acids. We denote symbols from this alphabet by lower-case letters like a, b. For now we will only consider ungapped global pairwise alignments: that is, two completely aligned equal-length sequences as in Figure 2.1a.

Given a pair of aligned sequences, we want to assign a score to the alignment that gives a measure of the relative likelihood that the sequences are related as opposed to being unrelated. We do this by having models that assign a probability to the alignment in each of the two cases; we then consider the ratio of the two probabilities.

The unrelated or *random* model R is simplest. It assumes that letter a occurs independently with some frequency q_a, and hence the probability of the two sequences is just the product of the probabilities of each amino acid:

$$P(x, y|R) = \prod_i q_{x_i} \prod_j q_{y_j}. \tag{2.1}$$

In the alternative *match* model M, aligned pairs of residues occur with a joint probability p_{ab}. This value p_{ab} can be thought of as the probability that the residues a and b have each independently been derived from some unknown original residue c in their common ancestor (c might be the same as a and/or b). This

gives a probability for the whole alignment of

$$P(x,y|M) = \prod_i p_{x_i y_i}.$$

The ratio of these two likelihoods is known as the *odds ratio*:

$$\frac{P(x,y|M)}{P(x,y|R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}.$$

In order to arrive at an additive scoring system, we take the logarithm of this ratio, known as the *log-odds ratio*:

$$S = \sum_i s(x_i, y_i), \tag{2.2}$$

where

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right) \tag{2.3}$$

is the log likelihood ratio of the residue pair (a,b) occurring as an aligned pair, as opposed to an unaligned pair.

As we wanted, equation (2.2) is a sum of individual scores $s(a,b)$ for each aligned pair of residues. The $s(a,b)$ scores can be arranged in a matrix. For proteins, for instance, they form a 20×20 matrix, with $s(a_i,a_j)$ in position i,j in the matrix, where a_i,a_j are the ith and jth amino acids (in some numbering). This is known as a *score matrix* or a *substitution matrix*. An example of a substitution matrix derived essentially as above is the BLOSUM50 matrix, shown in Figure 2.2. We can use these values to score Figure 2.1a and get a score of 130. Another commonly used set of substitution matrices are called the PAM matrices. A detailed description of the way that the BLOSUM and PAM matrices are derived is given at the end of the chapter.

An important result is that even if an intuitive biologist were to write down an *ad hoc* substitution matrix, the substitution matrix implies 'target frequencies' p_{ab} according to the above theory [Altschul 1991]. Any substitution matrix is making a statement about the probability of observing ab pairs in real alignments.

Exercise

2.1 Amino acids D, E and K are all charged; V, I and L are all hydrophobic. What is the average BLOSUM50 score within the charged group of three? Within the hydrophobic group? Between the two groups? Suggest reasons for the pattern observed.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Figure 2.2 *The* BLOSUM50 *substitution matrix. The log-odds values have been scaled and rounded to the nearest integer for purposes of computational efficiency. Entries on the main diagonal for identical residue pairs are highlighted in bold.*

Gap penalties

We expect to penalise gaps. The standard cost associated with a gap of length g is given either by a linear score

$$\gamma(g) = -gd \tag{2.4}$$

or an affine score

$$\gamma(g) = -d - (g-1)e \tag{2.5}$$

where d is called the *gap-open* penalty and e is called the *gap-extension* penalty. The gap-extension penalty e is usually set to something less than the gap-open penalty d, allowing long insertions and deletions to be penalised less than they would be by the linear gap cost. This is desirable when gaps of a few residues are expected almost as frequently as gaps of a single residue.

Gap penalties also correspond to a probabilistic model of alignment, although this is less widely recognised than the probabilistic basis of substitution matrices. We assume that the probability of a gap occurring at a particular site in a given sequence is the product of a function $f(g)$ of the length of the gap, and the

combined probability of the set of inserted residues,

$$P(\text{gap}) = f(g) \prod_{i \text{ in gap}} q_{x_i}. \tag{2.6}$$

The form of (2.6) as a product of $f(g)$ with the q_{x_i} terms corresponds to an assumption that the length of a gap is not correlated to the residues it contains.

The natural values for the q_a probabilities here are the same as those used in the random model, because they both correspond to unmatched independent residues. In this case, when we divide by the probability of this region according to the random model to form the odds ratio, the q_{x_i} terms cancel out, so we are left only with a term dependent on length $\gamma(g) = \log(f(g))$; gap penalties correspond to the log probability of a gap of that length.

On the other hand, if there is evidence for a different distribution of residues in gap regions then there should be residue-specific scores for the unaligned residues in gap regions, equal to the logs of the ratio of their frequencies in gapped versus aligned regions. This might happen if, for example, it is expected that polar amino acids are more likely to occur in gaps in protein alignments than indicated by their average frequency in protein sequences, because the gaps are more likely to be in loops on the surface of the protein structure than in the buried core.

Exercises

2.2 Show that the probability distributions $f(g)$ that correspond to the linear and affine gap schemes given in equations (2.4) and (2.5) are both geometric distributions, of the form $f(g) = ke^{-\lambda g}$.

2.3 Typical gap penalties used in practice are $d = 8$ for the linear case, or $d = 12, e = 2$ for the affine case, both expressed in half bits. A *bit* is the unit obtained when one takes log base 2 of a probability, so in natural log units these correspond to $d = (8\log 2)/2$ and $d = (12\log 2)/2$, $e = (2\log 2)/2$ respectively. What are the corresponding probabilities of a gap (of any length) starting at some position, and the distributions of gap length given that there is a gap?

2.4 Using the BLOSUM50 matrix in Figure 2.2 and an affine gap penalty of $d = 12$, $e = 2$, calculate the scores of the alignments in Figure 2.1b and Figure 2.1c.

2.3 Alignment algorithms

Given a scoring system, we need to have an algorithm for finding an optimal alignment for a pair of sequences. Where both sequences have the same length n, there is only one possible global alignment of the complete sequences, but things

become more complicated once gaps are allowed (or once we start looking for local alignments between subsequences of two sequences). There are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}} \tag{2.7}$$

possible global alignments between two sequences of length n. It is clearly not computationally feasible to enumerate all these, even for moderate values of n.

The algorithm for finding optimal alignments given an additive alignment score of the type we have described is called *dynamic programming*. Dynamic programming algorithms are central to computational sequence analysis. All the remaining chapters in this book except the last, which covers mathematical methods, make use of dynamic programming algorithms. The simplest dynamic programming alignment algorithms to understand are pairwise sequence alignment algorithms. The reader should be sure to understand this section, because it lays an important foundation for the book as a whole. Dynamic programming algorithms are guaranteed to find the optimal scoring alignment or set of alignments. In most cases heuristic methods have also been developed to perform the same type of search. These can be very fast, but they make additional assumptions and will miss the best match for some sequence pairs. We will briefly discuss a few approaches to heuristic searching later in the chapter.

Because we introduced the scoring scheme as a log-odds ratio, better alignments will have higher scores, and so we want to maximise the score to find the optimal alignment. Sometimes scores are assigned by other means and interpreted as *costs* or *edit distances*, in which case we would seek to minimise the cost of an alignment. Both approaches have been used in the biological sequence comparison literature. Dynamic programming algorithms apply to either case; the differences are trivial exchanges of 'min' for 'max'.

We introduce four basic types of alignment. The type of alignment that we want to look for depends on the source of the sequences that we want to align. For each alignment type there is a slightly different dynamic programming algorithm. In this section, we will only describe pairwise alignment for linear gap scores, with cost d per gap residue. However, the algorithms we introduce here easily extend to more complex gap models, as we will see later in the chapter.

We will use two short amino acid sequences to illustrate the various alignment methods, HEAGAWGHEE and PAWHEAE. To score the alignments, we use the BLOSUM50 score matrix, and a gap cost per unaligned residue of $d = -8$. Figure 2.3 shows a matrix s_{ij} of the local score $s(x_i, y_j)$ of aligning each residue pair from the two example sequences. Identical or conserved residue pairs are highlighted in bold. Informally, the goal of an alignment algorithm is to incorporate as many of these positively scoring pairs as possible into the alignment, while minimising the cost from unconserved residue pairs, gaps, and other constraints.

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

Figure 2.3 *The two example sequences we will use for illustrating dynamic programming alignment algorithms, arranged to show a matrix of corresponding* BLOSUM50 *values per aligned residue pair. Positive scores are in bold.*

Exercises

2.5 Show that the number of ways of intercalating two sequences of lengths n and m to give a single sequence of length $n+m$, while preserving the order of the symbols in each, is $\binom{n+m}{m}$.

2.6 By taking alternating symbols from the upper and lower sequences in an alignment, then discarding the gap characters, show that there is a one-to-one correspondence between gapped alignments of the two sequences and intercalated sequences of the type described in the previous exercise. Hence derive the first part of equation (2.7).

2.7 Use Stirling's formula ($x! \simeq \sqrt{2\pi}x^{x+\frac{1}{2}}e^{-x}$) to prove the second part of equation (2.7).

Global alignment: Needleman–Wunsch algorithm

The first problem we consider is that of obtaining the optimal global alignment between two sequences, allowing gaps. The dynamic programming algorithm for solving this problem is known in biological sequence analysis as the Needleman–Wunsch algorithm [Needleman & Wunsch 1970], but the more efficient version that we describe was introduced by Gotoh [1982].

The idea is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. We construct a matrix F indexed by i and j, one index for each sequence, where the value $F(i,j)$ is the score of the best alignment between the initial segment $x_{1\ldots i}$ of x up to x_i and the initial segment $y_{1\ldots j}$ of y up to y_j. We can build $F(i,j)$ recursively. We begin by initialising $F(0,0) = 0$. We then proceed to fill the matrix from top left to bottom right. If $F(i-1,j-1)$, $F(i-1,j)$ and $F(i,j-1)$ are known, it is possible to calculate $F(i,j)$. There are three possible ways that the best score

```
I G A x_i        A I G A x_i        G A x_i - -
L G V y_j        G V y_j - -        S L G V y_j
```

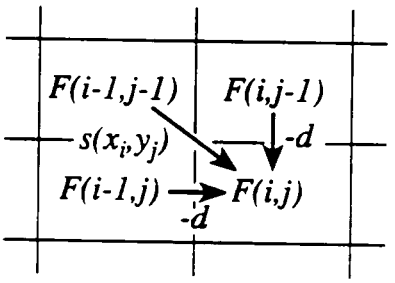
Figure 2.4 *The three ways an alignment can be extended up to (i,j): x_i aligned to y_j, x_i aligned to a gap, and y_j aligned to a gap.*

$F(i,j)$ of an alignment up to x_i, y_j could be obtained: x_i could be aligned to y_j, in which case $F(i,j) = F(i-1,j-1) + s(x_i, y_j)$; or x_i is aligned to a gap, in which case $F(i,j) = F(i-1,j) - d$; or y_j is aligned to a gap, in which case $F(i,j) = F(i,j-1) - d$ (see Figure 2.4). The best score up to (i,j) will be the largest of these three options.

Therefore, we have

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases} \tag{2.8}$$

This equation is applied repeatedly to fill in the matrix of $F(i,j)$ values, calculating the value in the bottom right-hand corner of each square of four cells from one of the other three values (above-left, left, or above) as in the following figure.



As we fill in the $F(i,j)$ values, we also keep a pointer in each cell back to the cell from which its $F(i,j)$ was derived, as shown in the example of the full dynamic programming matrix in Figure 2.5.

To complete our specification of the algorithm, we must deal with some boundary conditions. Along the top row, where $j = 0$, the values $F(i,j-1)$ and $F(i-1,j-1)$ are not defined so the values $F(i,0)$ must be handled specially. The values $F(i,0)$ represent alignments of a prefix of x to all gaps in y, so we can define $F(i,0) = -id$. Likewise down the left column $F(0,j) = -jd$.

The value in the final cell of the matrix, $F(n,m)$, is by definition the best score for an alignment of $x_{1\ldots n}$ to $y_{1\ldots m}$, which is what we want: the score of the best global alignment of x to y. To find the alignment itself, we must find the path of choices from (2.8) that led to this final value. The procedure for doing this is known as a *traceback*. It works by building the alignment in reverse, starting from the final cell, and following the pointers that we stored when building the

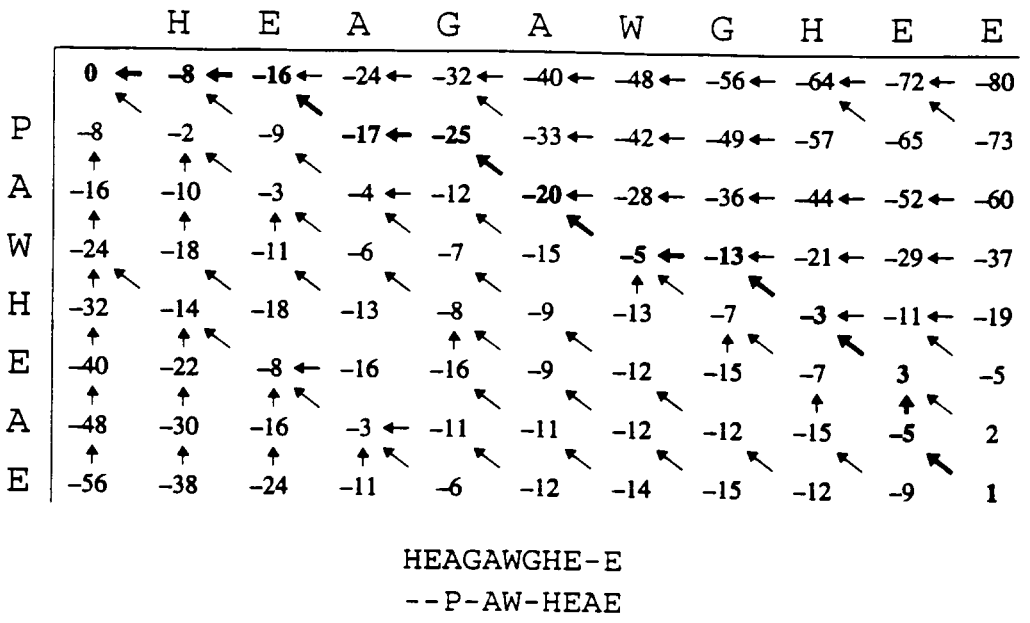


Figure 2.5 *Above, the global dynamic programming matrix for our example sequences, with arrows indicating traceback pointers; values on the optimal alignment path are shown in bold. Below, a corresponding optimal alignment, which has total score* 1.

matrix. At each step in the traceback process we move back from the current cell (i, j) to the one of the cells $(i-1, j-1)$, $(i-1, j)$ or $(i, j-1)$ from which the value $F(i, j)$ was derived. At the same time, we add a pair of symbols onto the front of the current alignment: x_i and y_j if the step was to $(i-1, j-1)$, x_i and the gap character '–' if the step was to $(i-1, j)$, or '–' and y_j if the step was to $(i, j-1)$. At the end we will reach the start of the matrix, $i = j = 0$. An example of this procedure is shown in Figure 2.5.

Note that in fact the traceback procedure described here finds just one alignment with the optimal score; if at any point two of the derivations are equal, an arbitrary choice is made between equal options. The traceback algorithm is easily modified to recover more than one equal-scoring optimal alignment. The set of all possible optimal alignments can be described fairly concisely using a sequence graph structure [Altschul & Erickson 1986; Hein 1989a]. We will use sequence graph structures in Chapter 7 where we describe Hein's algorithm for multiple alignment.

The reason that the algorithm works is that the score is made of a sum of independent pieces, so the best score up to some point in the alignment is the best score up to the point one step before, plus the incremental score of the new step.

Big-O notation for algorithmic complexity

It is useful to know how an algorithm's performance in CPU time and required memory storage will scale with the size of the problem. From the algorithm

above, we see that we are storing $(n+1) \times (m+1)$ numbers, and each number costs us a constant number of calculations to compute (three sums and a max). We say that the algorithm takes $O(nm)$ time and $O(nm)$ memory, where n and m are the lengths of the sequences. '$O(nm)$' is a standard notation, called *big-O* notation, meaning 'of order nm', i.e. that the computation time or memory storage required to solve the problem scales as the product of the sequence lengths nm, up to a constant factor. Since n and m are usually comparable, the algorithm is usually said to be $O(n^2)$. The larger the exponent of n, the less practical the method becomes for long sequences. With biological sequences and standard computers, $O(n^2)$ algorithms are feasible but a little slow, while $O(n^3)$ algorithms are only feasible for very short sequences.

Exercises

2.8 Find a second equal-scoring optimal alignment in the dynamic programming matrix in Figure 2.5.

2.9 Calculate the dynamic programming matrix and an optimal alignment for the DNA sequences GAATTC and GATTA, scoring +2 for a match, −1 for a mismatch, and with a linear gap penalty of $d = 2$.

Local alignment: Smith–Waterman algorithm

So far we have assumed that we know which sequences we want to align, and that we are looking for the best match between them from one end to the other. A much more common situation is where we are looking for the best alignment between *subsequences* of x and y. This arises for example when it is suspected that two protein sequences may share a common domain, or when comparing extended sections of genomic DNA sequence. It is also usually the most sensitive way to detect similarity when comparing two very highly diverged sequences, even when they may have a shared evolutionary origin along their entire length. This is because usually in such cases only part of the sequence has been under strong enough selection to preserve detectable similarity; the rest of the sequence will have accumulated so much noise through mutation that it is no longer alignable. The highest scoring alignment of subsequences of x and y is called the best *local* alignment.

The algorithm for finding optimal local alignments is closely related to that described in the previous section for global alignments. There are two differences. First, in each cell in the table, an extra possibility is added to (2.8), allowing $F(i,j)$ to take the value 0 if all other options have value less than 0:

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases} \tag{2.9}$$

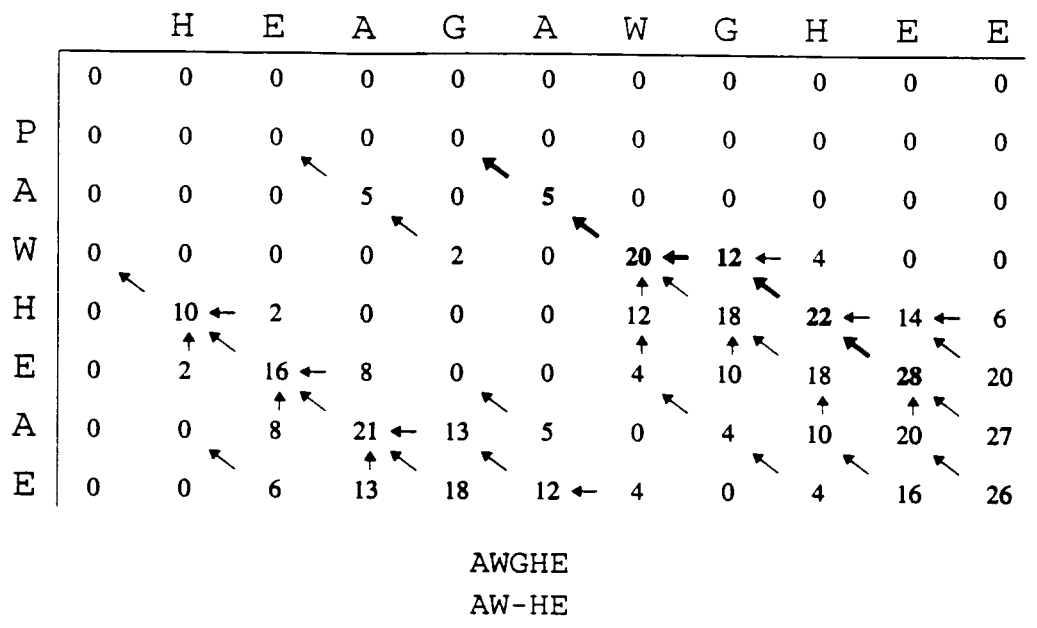


Figure 2.6 *Above, the local dynamic programming matrix for the example sequences. Below, the optimal local alignment, with score* 28.

Taking the option 0 corresponds to starting a new alignment. If the best alignment up to some point has a negative score, it is better to start a new one, rather than extend the old one. Note that a consequence of the 0 is that the top row and left column will now be filled with 0s, not $-id$ and $-jd$ as for global alignment.

The second change is that now an alignment can end anywhere in the matrix, so instead of taking the value in the bottom right corner, $F(n,m)$, for the best score, we look for the highest value of $F(i,j)$ over the whole matrix, and start the traceback from there. The traceback ends when we meet a cell with value 0, which corresponds to the start of the alignment. An example is given in Figure 2.6, which shows the best local alignment of the same two sequences whose best global alignment was found in Figure 2.5. In this case the local alignment is a subset of the global alignment, but that is not always the case.

For the local alignment algorithm to work, the expected score for a random match must be negative. If that is not true, then long matches between entirely unrelated sequences will have high scores, just based on their length. As a consequence, although the algorithm is local, the maximal scoring alignments would be global or nearly global. A true subsequence alignment would be likely to be masked by a longer but incorrect alignment, just because of its length. Similarly, there must be some $s(a,b)$ greater than 0, otherwise the algorithm won't find any alignment at all (it finds the best score or 0, whichever is higher).

What is the precise meaning of the requirement that the expected score of a random match be negative? In the ungapped case, the relevant quantity to consider is the expected value of a fixed length alignment. Because successive posi-

tions are independent, we need only consider a single residue position, giving the condition

$$\sum_{a,b} q_a q_b s(a,b) < 0, \tag{2.10}$$

where q_a is the probability of symbol a at any given position in a sequence. When $s(a,b)$ is derived as a log likelihood ratio, as in the previous section, using the same q_a as for the random model probabilities, then (2.10) is always satisfied. This is because

$$\sum_{a,b} q_a q_b s(a,b) = -\sum_{a,b} q_a q_b \log \frac{q_a q_b}{p_{ab}} = -H(q^2||p)$$

where $H(q^2||p)$ is the relative entropy of distribution $q^2 = q \times q$ with respect to distribution p, which is always positive unless $q^2 = p$ (see Chapter 11). In fact $H(q^2||p)$ is a natural measure of how different the two distributions are. It is also, by definition, a measure of how much information we expect per aligned residue pair in an alignment.

Unfortunately we cannot give an equivalent analysis for optimal gapped alignments. There is no analytical method for predicting what gap scores will result in local versus global alignment behaviour. However, this is a question of practical importance when setting parameter values in the scoring system (the match and gap scores $s(a,b)$ and $\gamma(g)$), and tables have been generated for standard scoring schemes showing local/global behaviour, along with other statistical properties [Altschul & Gish 1996]. We will return to this subject later, when considering the statistical significance of scores.

The local version of the dynamic programming sequence alignment algorithm was developed in the early 1980s. It is frequently known as the *Smith–Waterman* algorithm, after Smith & Waterman [1981]. Gotoh [1982] formulated the efficient affine gap cost version that is normally used (affine gap alignment algorithms are discussed on page 29).

Repeated matches

The procedure in the previous section gave the best single local match between two sequences. If one or both of the sequences are long, it is quite possible that there are many different local alignments with a significant score, and in most cases we would be interested in all of these. An example would be where there are many copies of a repeated domain or motif in a protein. We give here a method for finding such matches. This method is asymmetric: it finds one or more non-overlapping copies of sections of one sequence (e.g. the domain or motif) in the other. There is another widely used approach for finding multiple matches due to Waterman & Eggert [1987], which will be described in Chapter 4.