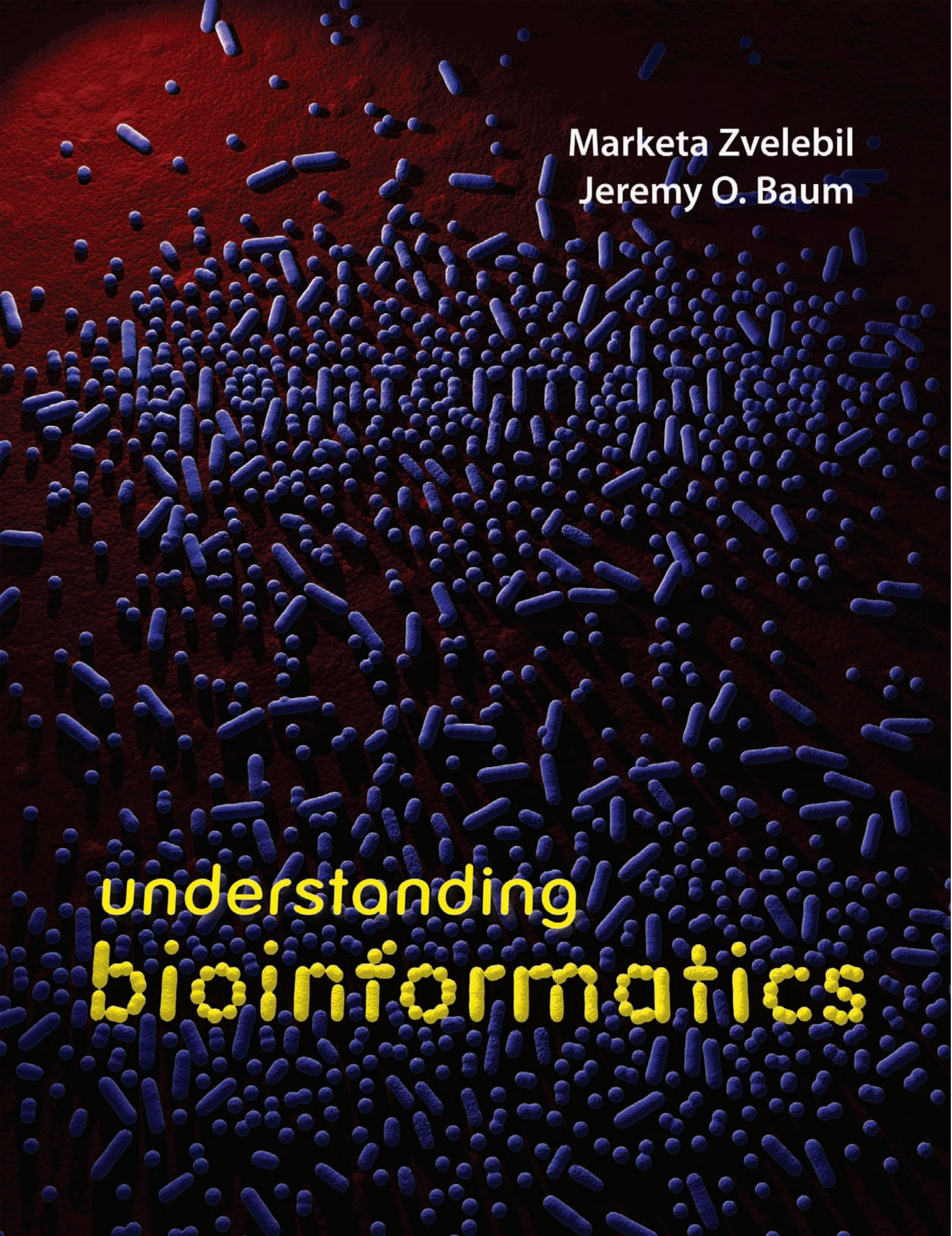
Marketa Zvelebil
Jeremy O. Baum

understanding bioinformatics



4

PRODUCING AND ANALYZING SEQUENCE ALIGNMENTS

APPLICATIONS CHAPTER

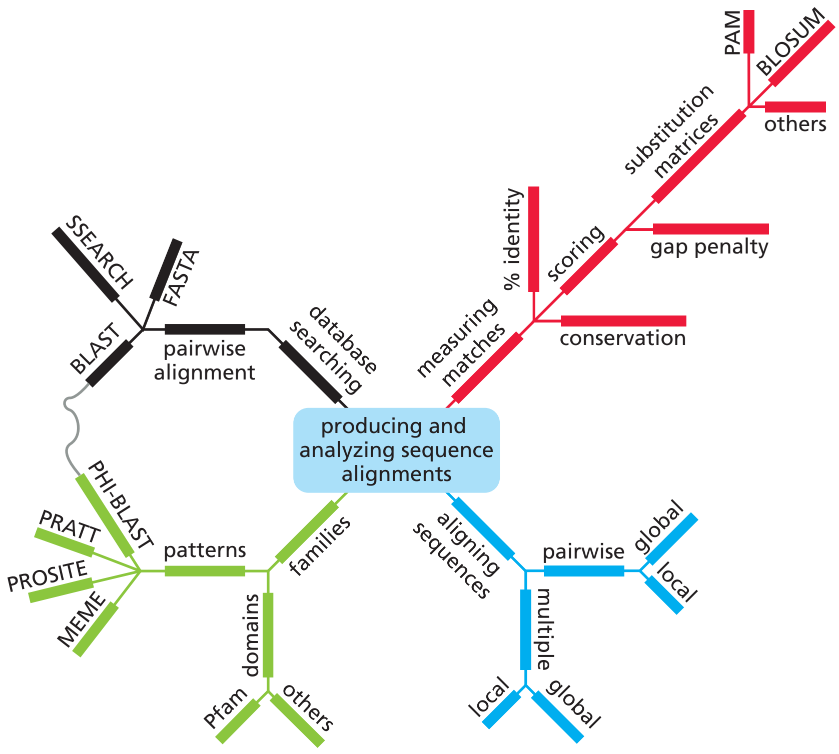
When you have read Chapter 4, you should be able to:

- Determine homology by sequence alignment.
- Describe different uses of protein and DNA sequence alignments.
- Define scoring alignments.
- Make alignments between two sequences.
- Make multiple alignments between many sequences.
- Compare local alignment techniques for finding limited areas of similarity.
- Explain global alignment techniques for matching whole sequences.
- Search databases for homologous sequences.
- Look for patterns and motifs in a protein sequence.
- Use patterns and motifs to locate proteins of similar function.

The revolution in genetic analysis that began with recombinant DNA technology and the invention of DNA sequencing techniques in the 1970s has, 30 years later, filled vast databases with nucleotide and protein sequences from a wide variety of organisms. Genomes that have now been completely sequenced include human, mouse, chimpanzee, the fruit fly *Drosophila*, the nematode *Caenorhabditis*, and the yeast *Saccharomyces*, as well as numerous bacteria, archaea, and viruses. Although entries for nucleotide and protein sequences in databases such as GenBank, dbEST, and UniProt KB now number many millions, nothing is known about the structure or function of the proteins specified by many of them. Converting this sequence information into useful biological knowledge is now the main challenge.

To find out more about a newly determined sequence, it is subjected to the process of sequence analysis. There are many aspects to this, depending on the source of the sequence and what you ultimately want to find out about it. In this chapter, we will focus on one of the key stages in most sequence analyses: the alignment of different sequences to detect homology and the comparison of a novel sequence with those in the databases to see whether there is any similarity between them. The practical use of techniques and programs for general alignment, database searching, and pattern searching will be described in this chapter, with the main focus on the alignment and analysis of protein sequences. The theory underlying programs for pairwise alignment is described in Chapter 5 and that dealing with multiple alignments in Chapter 6, for both nucleic acid and protein sequences. Techniques and programs for detecting genes and other sequence features in genomic DNA are dealt with in Chapters 9 and 10.

Mind Map 4.1
A mind map of the four major sections relating to sequence analysis and alignment: aligning sequences, searching through databases, measuring how well sequences match, and looking for families of proteins.



The identification of similar sequences has a multitude of applications. For raw, uncharacterized genomic DNA sequences, comparison with sequences in a database can often tell you whether the sequence is likely to contain, or be part of, a protein-coding gene. The similarity search may retrieve a known gene or family of genes with a strong similarity to the new sequence. This will provide the first clues to the type of protein the new gene encodes and its possible function. Similarities in sequence can also help in making predictions about a protein's structure (see Chapters 11–14). Sequences of proteins or DNAs from different organisms can also be compared in order to construct phylogenetic trees, which trace the evolutionary relationships between species or within a family of proteins (see Chapters 7 and 8).

As well as many general and specialized databases of DNA and protein sequences, the fully sequenced genomes of various organisms are now available (see Chapter 3), providing vast amounts of information for comparison. It is, however, important to remember that although many newly discovered sequences will share some or considerable similarity to sequences in the databases, there will still be many that are unique.

4.1 Principles of Sequence Alignment

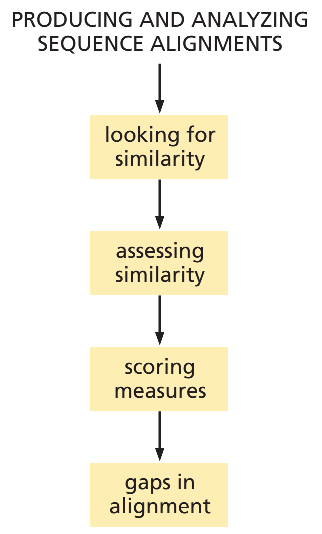
Devising ways of comparing sequences has never been straightforward, not just because of the vast amounts of information now available for searching. The difficulties arise because of the many ways DNA and protein sequences can change during evolution. Mutation and selection over millions of years can result in considerable divergence between present-day sequences derived from the same ancestral gene. Bases at originally corresponding positions, and the amino acids they encode, can change as a result of point mutation, and the sequence lengths can be quite different as a result of insertions and deletions. Even more dramatic changes may have occurred; for example, the fusion of sequences from two different genes. Gene

Box 4.1 Genes and pseudogenes

Pseudogenes are sequences in genomic DNA that have a similar sequence to known protein-coding genes but do not produce a functional protein. They are assumed to arise after gene duplication, when one of the gene copies undergoes mutation that either prevents its transcription or disrupts its protein-coding sequence. The human genome is estimated to contain up to 20,000 pseudogenes. As the pseudogene sequence is no longer under selection to retain protein function, it will generally accumulate further mutations at a higher rate than the functional gene. Despite this, many pseudogenes retain considerable sequence similarity to their active counterparts. One case has even been found in which the RNA from a transcribed pseudogene regulates the expression of the corresponding functional gene.

duplications are common in eukaryotic genomes, and in many cases mutation has disabled one copy of a gene so that it is either no longer expressed or, if transcribed, does not produce a functional protein. Such genes are called pseudogenes (see Box 4.1) and can be found in homology searches.

On superficial inspection, such changes in gene sequence and length can effectively mask any underlying sequence similarity. To reveal it, the sequences have to be aligned with each other to maximize their similarities. This crucial step in sequence comparison is the main topic of the first half of this chapter (Sections 4.1 to 4.5). Alignment methods are at the core of many of the software tools used to search the databases, and in the second half of the chapter we will describe some of these tools and how they can be used to retrieve similar sequences from the databases (Sections 4.6 to 4.10). The first steps to consider are shown in Flow Diagram 4.1.



Flow Diagram 4.1
The key concept introduced in these first four sections is that in order to assess the similarity of two sequences it is necessary to have a quantitative measure of their alignment, which includes the degree of similarity of two aligned residues as well as accounting for insertions and deletions.

Alignment is the task of locating equivalent regions of two or more sequences to maximize their similarity

As the result of mutation, even the sequences of the same protein or gene from two closely related species are rarely identical. Ideally, what we want to achieve when comparing sequences is to line them up in such a way that, when they do derive from a common ancestor, bases or amino acids derived from the same ancestral base are aligned. Without information to the contrary, this is best achieved by maximizing the similarity of aligned regions.

To illustrate the general principle, take the two hypothetical amino acid sequences THISSEQUENCE and THATSEQUENCE. If we align them so that as many identical letters as possible pair up we get

```
T H I S S E Q U E N C E
| |     | | | | | | | |
T H A T S E Q U E N C E
```

where the letters in red type are identical. As we can easily see with such short and similar sequences, this alignment clearly identifies their strong similarity to each other.

So far so good, but when sequences become more different from each other, they become more difficult to compare. How would we go about comparing the two sequences THATSEQUENCE and THISISASEQUENCE, in which a mutation has led to the insertion of the three amino acids I, S, A into one of the original sequences? Simply lining them up from the beginning loses much of the similarity we can see exists. More subtly, because of the difference in length, it also creates false matches between noncorresponding positions.

```
T  H  A  T  S  E  Q  U  E  N  C  E
|  |                    |        |
T  H  I  S  I  S  A  S  E  Q  U  E  N  C  E
```

To get round this problem, **gaps** are introduced into one or both of the sequences so that maximum similarity is preserved.

```
T  H  I  S  I  S  A  -  S  E  Q  U  E  N  C  E
|  |              |     |  |  |  |  |  |  |  |
T  H  -  -  -  -  A  T  S  E  Q  U  E  N  C  E
```

There is never just one possible alignment between any two sequences, and the best one is not always obvious, especially when the sequences are not very similar to each other. At the heart of sequence-comparison and database-searching methods are algorithms for testing the fit of each alignment generated, giving it a quantitative score, and filtering out the unsatisfactory ones according to preset criteria.

Alignment can reveal homology between sequences

In all methods of sequence comparison, the fundamental question is whether the similarities perceived between two sequences are due to chance, and are thus of little biological significance, or whether they are due to the derivation of the sequences from a common ancestral sequence, and are thus homologous. The terms “homology” and “similarity” are sometimes used interchangeably, but each has a distinct meaning. **Similarity** is simply a descriptive term telling you that the sequences in question show some degree of match. Homology, in contrast, has distinct evolutionary and biological implications. In the molecular biological context, it is generally defined as referring specifically to similarity in sequence or structure due to descent from a common ancestor. Homologous genes are therefore genes derived from the same ancestral gene. During their evolutionary history they will have diverged in sequence as a result of accumulating different mutations.

Because homology implies a common ancestor, it can also imply a common function or structure for two homologous proteins, which can be a useful pointer to function if one of the proteins is known only from its sequence. The operation of natural selection tends to result in the acceptance of mutations that preserve the folding and function of a protein, whereas those that destroy folding or function will be eliminated. However, similar or identical aligned residues may simply be due to relatively recent divergence of the two sequences, and so care must be taken not to overestimate their functional importance. Moreover, mutation and selection can generate proteins with new functions but relatively little change in sequence. Therefore, sequence similarity does not always imply a common function.

Conversely, there are proteins with very little sequence similarity to each other but in which a common protein fold and function are preserved. Consequently, low sequence similarity does not necessarily rule out common function or homology. Such cases require extra information, such as structural or biochemical knowledge, to demonstrate their true relationship.

Sequences can also be significantly similar to each other, and yet not be evolutionarily homologous, as a result of **convergent evolution** for similar function (see Box 4.2). In this case, identical or very similar aligned residues can be argued to have an important functional role. Convergent evolution does not, however, usually produce highly similar sequences of any great length.

All these considerations have to be taken into account when analyzing the results of a database search. An alignment of two sequences is, in effect, a hypothesis about which pairs of residues have evolved from the same ancestral residue. But an alignment in itself does not imply an evolutionary order of events, so that the two

Box 4.2 **Convergent and divergent evolution**

Convergent evolution is the evolutionary process in which organs, proteins, or DNA sequences that are unrelated in their evolutionary origin independently acquire the same structure or function. This usually reflects a response to similar environmental and selective pressures. Convergent evolution is contrasted with the process of **divergent evolution**, which produces different structures or sequences from a common ancestor. An example of convergent evolution for function can be seen in the wings of insects and bats. Although adapted to the same function—that of flight—insect wings and bat wings do not derive from the same ancestral structure.

(A)

(B)

Figure B4.1
(A) Bat wings and (B) butterfly wings. (A, courtesy of Ron Austing/Science Photo Library.)

alternatives of homology and convergent evolution cannot usually be distinguished without additional information.

Sequence comparison methods have to take account of such factors as the types of mutation that occur over evolutionary time, differences in the physicochemical properties of amino acids and their role in determining protein structure and function, and the selective pressures that result in some mutations being accepted and others being eliminated. One has to consider the evolutionary processes that are responsible for sequence divergence and find a way to include the salient features in practicable schemes for testing the goodness of fit of the alignment. These must be quantitative and hence involve a score. Such **scoring schemes** can then be incorporated in algorithms designed to generate the best possible alignments. Finally, ways must be found to discriminate between fortuitously good alignments and those due to a real evolutionary relationship.

As we shall see in this chapter, all computational methods of sequence comparison take account of these factors in some way.

It is easier to detect homology when comparing protein sequences than when comparing nucleic acid sequences

For most purposes, comparisons of protein sequences show up homology more easily than comparisons of the corresponding DNA sequences. There are many reasons for this greater sensitivity. First, there are only four letters in the DNA alphabet compared to the 20 letters in the protein alphabet, and so a DNA sequence, of necessity, provides less information at each sequence position than does a protein sequence. In other words, there is a much greater probability that a match at any one position between two DNA sequences will have occurred by chance. Therefore, the degree of similarity, as judged by some appropriate quantitative score, needs to be greater between DNA sequences than between protein sequences for the alignment to be of importance. As we shall see later in this chapter, ways have been devised of determining the likelihood that one amino acid can be substituted for another during evolution, and this provides additional information beyond simple identity for scoring an alignment and determining homology.

Second, as we saw in Chapter 1, the genetic code is redundant; that is, there are two or more different codons for most amino acids (see Table 1.1). This means that identical amino acid sequences can be encoded by different nucleotide sequences. Finally, the complex three-dimensional structure of a protein, and hence its function, is determined by the amino acid sequence. The importance of maintaining protein function usually leads to amino acid sequences changing less over evolutionary time than homologous DNA sequences. In this chapter we will concentrate for the most part on protein sequence analysis.

There are many circumstances, however, in which it is necessary to compare DNA sequences: when searching for promoters and other regulatory sequences, for example, or in whole-genome comparisons. DNA alignment is also performed, to some extent, as part of gene identification (see Chapters 9 and 10).

4.2 Scoring Alignments

The quality of an alignment is measured by giving it a quantitative score

Two homologous sequences are often so different that a correct or best alignment is not obvious by visual inspection. Furthermore, the large numbers of sequences that can be examined for similarity nowadays oblige us to use automated computational methods to judge the quality of an alignment, at least as an initial filter.

Because it is possible for two sequences to be aligned in a variety of different ways, including the insertion of gaps to improve the number of matched positions, how does one objectively determine which is the best possible alignment for any given pair of sequences? In practice, this is done by calculating a numerical value or **score** for the overall similarity of each possible alignment so that the alignments can be ranked in some order.

We can then work on the basis that alignments of related sequences will give good scores compared with alignments of randomly chosen sequences, and that the correct alignment of two related sequences will ideally be the one that gives the best score. The alignment giving the best score is referred to as the **optimal alignment**, while others with only slightly worse scores are often called **suboptimal alignments**. No one has yet devised a scoring scheme that perfectly models the evolutionary process, which is so complex that it defies any practical method of modeling. The implication of this is that the best-scoring alignment will not necessarily be the correct one, and conversely, that the correct alignment will not necessarily have the best score. However, the scoring schemes now in common use, and which are described in this chapter, are generally reliable and useful in most circumstances, as long as the results are treated with due caution and regard for biological plausibility. In principle, a scoring scheme can either measure similarity or difference, the best score being a maximum in the former case and a minimum in the latter.

The simplest way of quantifying similarity between two sequences is percentage identity

Identity describes the degree to which two or more sequences are actually identical at each position, and is simply measured by counting the number of identical bases or amino acids matched between the aligned sequences. Identity is an objective measure and can be precisely defined. **Percentage** or **percent identity** is obtained by dividing the number of identical matches by the total length of the aligned region and multiplying by 100. For the THATSEQUENCE/THISISASEQUENCE comparison, for example, the alignment given on page 74 is the best that can be achieved, and has a percentage identity score of 68.75% (11 matches over a total length of 16 positions, including the gaps).

One might think that an alignment of completely unrelated sequences would have a percentage identity of zero. However, as there are only four different nucleotides in nucleic acid sequences, and only 20 different amino acids in protein sequences, there is always a small but finite probability for any aligned sequences that identical residues will be matched at some positions. Because there are often hundreds of residues in a protein sequence and thousands in a nucleotide sequence, unrelated sequences are expected to align matches at several positions. The length of the sequence matters: a 30% identity over a long alignment is less likely to have arisen by chance than a 30% identity over a very short alignment. Statistically rigorous methods have been devised to measure the significance of an alignment, which will be discussed later in connection with database searches and in Section 5.4.

The dot-plot gives a visual assessment of similarity based on identity

A dot matrix or **dot-plot** is one of the simplest ways to compare sequence similarity graphically, and can be used for both nucleotide and protein sequences. To compare two sequences X and Y, one sequence is written out vertically, with each residue in the sequence represented by a row, while the other is written horizontally, with each residue represented by a column. Each residue of X is compared to each residue of Y (row to column comparison) and a dot is placed where the residues are identical. In the simplest scoring system, identical residues are scored as 1 and nonidentical residues as 0, and dots are placed at all positions that contain a 1. For example, if we take the pair THISSEQUENCE/THISISASEQUENCE pair, then a simple dot-plot will look like that illustrated in Figure 4.1. The dots in red, which form diagonal lines, represent runs of matched residues. The pink dots scattered either side of the diagonals are the same residues found elsewhere in the sequence. The diagonals are interrupted by a few cells, where a gap has been inserted.

Dot-plots can be useful for identifying intrasequence repeats in either proteins or nucleic acids. However, dot-plots suffer from background noise. To distinguish dot-patterns arising from background noise from significant dot-patterns it is usually necessary to apply a filter. The most widely used filtering procedure uses

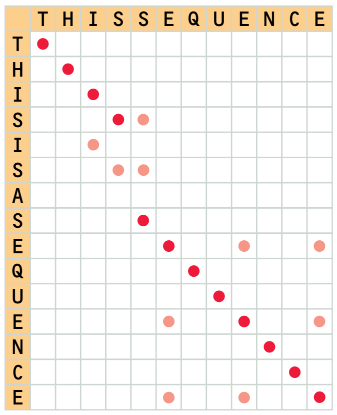
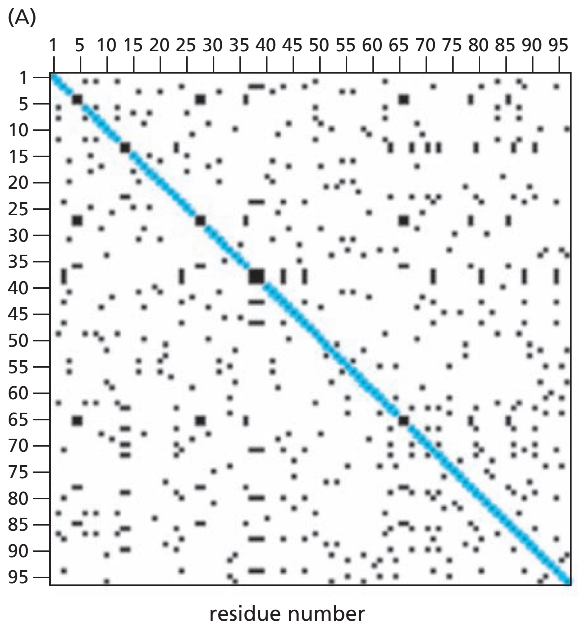
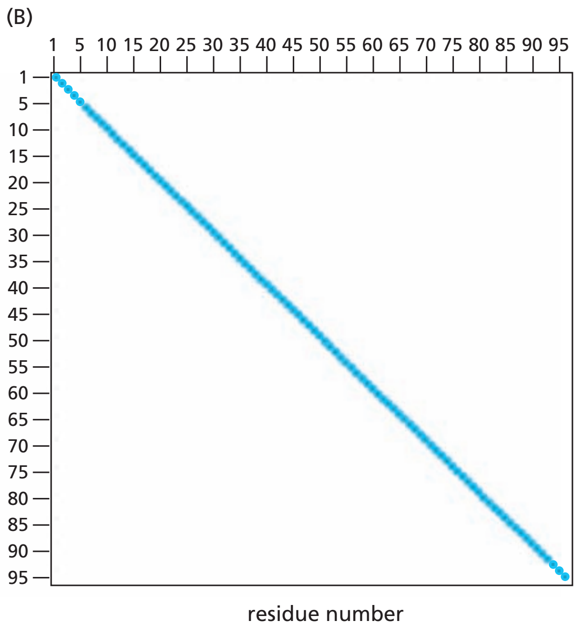


Figure 4.1
Dot-plot representations. A dot-plot matrix of the THISSEQUENCE/ THISISASEQUENCE example where red dots represent identities that are due to true matching of identical residue-pairs and pink dots represent identities that are due to noise; that is, matching of random identical residue-pairs.

Figure 4.2
Two views of dot-plot representations of an SH2 sequence compared with itself. (A) Unfiltered dot-plot (window length = 1 residue). The identity between the two sequences is shown by the unbroken identity diagonal. Nevertheless, there is still background noise. (B) Dot-plot of the same sequence comparison with a window of 10 residues and a minimum identity score within each window set to 3. The background noise has all been removed, leaving only the identity diagonal.





overlapping fixed-length windows and requires that the comparison achieve some minimum identity score summed over that window before being considered; that is, only diagonals of a certain length will survive the filter. Figure 4.2 shows a dot-plot between two identical SH2 sequences (see Box 4.3).

Figure 4.2A has a window length of 1; in other words, every residue is considered individually. Although the diagonal line indicating matched identical residues is clear and unbroken, as one would expect from a comparison of two identical sequences, there is still a certain amount of background noise detracting from the result, as most types of amino acid occur more than once in the sequence. Figure 4.2B shows the same comparison with a window of 10 residues and a minimum score for each window set to 3. Only the main diagonal is now seen, representing the one-to-one matching of the identical sequences.

Most dot-plot software provides a default window length and this is sufficient for an initial analysis. But one can use the window length to greater effect by varying it depending on what one is searching for. Window length can be set, for example, to the length of an exon when comparing coding sequences, or to the size of an average secondary structure within a protein when looking for structural motifs. When searching for internal repeats, the length of the repeat can be used to cut out background noise. In addition, rather than using 0 and 1 as the scores for nonidentical and identical residues, other values can be used and the score can be varied depending on the type of residues involved.

Figure 4.3 illustrates how a dot-plot can be used to identify repeats within a sequence. It shows two dot-plot calculations on the protein BRCA2 encoded by the breast cancer susceptibility gene *BRCA2*. This protein contains eight repeats of a short sequence of around 39 amino acids, called the BRC repeat (see Box 4.4). Figure 4.3A shows an unfiltered version of a self-comparison dot-plot of a region of BRCA2 containing two BRC repeats. The background noise is so strong that it is very difficult to pick out the repeats. Figure 4.3B shows a highly filtered dot-plot of the same comparison in which a diagonal line is now visible. This is the identity diagonal, where the one-to-one alignment of the sequence with itself is highlighted. But two other runs of dots are now also visible; these represent the internal BRC repeats.

Box 4.3 **The SH2 protein-interaction domain**

The SH2 or Src-homology 2 domain is a small domain of about 100 residues found in many proteins involved in intracellular signaling in mammalian cells. It gets its name from the protein tyrosine kinase Src, where it was first found. It is one of numerous protein-interaction domains found in signaling proteins, which recognize and bind to particular features on other proteins to help pass the signal onward. SH2 domains bind specifically to phosphotyrosines on proteins; these are formed by the phosphorylation—the modification by covalent addition of a phosphate group—of tyrosine residues in specific peptide motifs by protein tyrosine kinases. This type of kinase is often part of, or associated with, cell-surface receptors, and is activated in response to an extracellular signal. The phosphotyrosine-binding site on SH2 domains consists of two pockets. One is conserved and binds the phosphotyrosine residue (pY); the other is more variable in sequence between different SH2 domains and binds residues located downstream from the pY, thereby conferring specificity on the protein–protein interaction. Because of its role in intracellular signaling, the SH2 domain is an important potential drug target for a number of diseases, including cancer and osteoporosis.

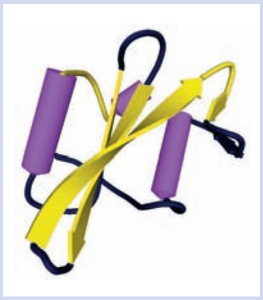


Figure B4.2
A ribbon representation of an SH2 domain.

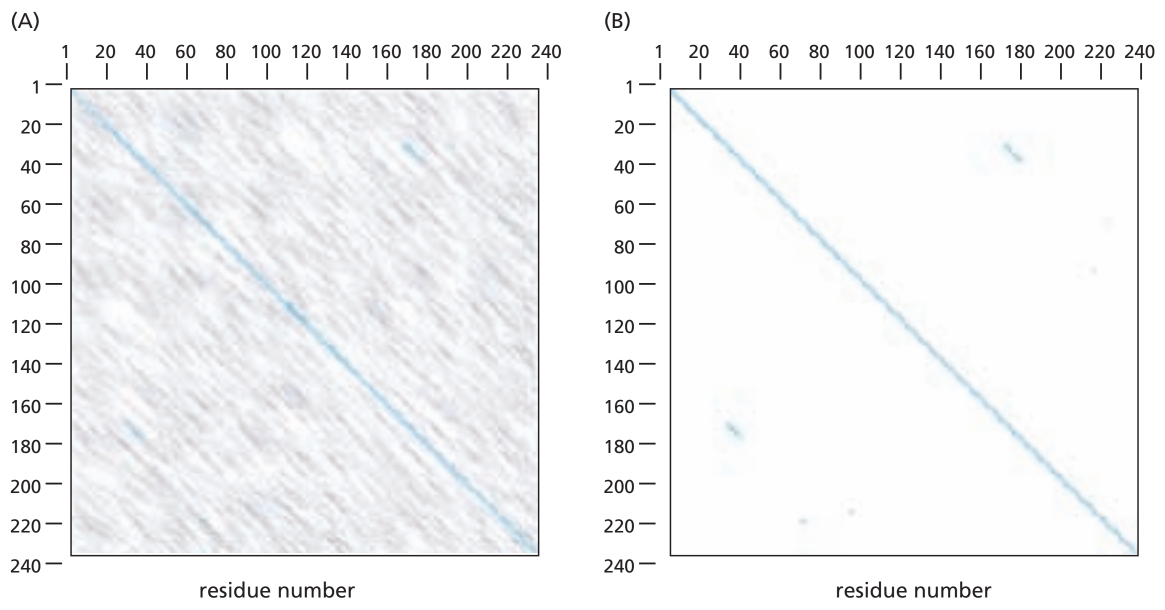


Figure 4.3
Two dot-plots involving the breast cancer susceptibility gene protein BRCA2, which contains the small BRCA2 repeat domain. (A) An unfiltered self-comparison dot-plot of part of the human BRCA2 sequence containing two BRCA2 repeats (the first and second BRCA repeat in the sequence). The background noise is so strong that it is very difficult to pick out the repeats. (B) The same dot-plot with a window length of 30 and a minimum score of 5. In addition to the identity diagonal there are two other clear diagonal runs of dots that represent the two internal BRCA2 repeats.

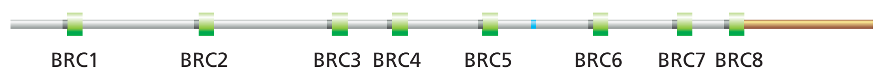
Genuine matches do not have to be identical

Although it is the simplest alignment score to obtain, and can be very useful as a quick test of the quality of an alignment, percentage identity is a relatively crude measure and does not give a complete picture of the degree of similarity of two sequences to each other, especially in regard to protein sequences. For example,

Box 4.4 The breast cancer susceptibility genes *BRCA1* and *BRCA2*

Two genes that confer increased susceptibility to breast cancer have been identified: the *BRCA1* gene on chromosome 17 in 1994 and the *BRCA2* gene on chromosome 13 in 1995. Women with a mutation in either *BRCA1* or *BRCA2* are at increased risk of developing breast, ovarian, and some other cancers by a given age than those without a mutation. The normal role of the BRCA1 and BRCA2 proteins, which are not structurally related, is to associate with the protein RAD51, a protein essential for the repair of double-strand breaks in DNA. Mutations in *BRCA1* or *BRCA2* can thus partly disable this repair mechanism, leading to more errors in DNA repair than usual, an increased mutation rate, and, ultimately, a greater risk of tumorigenesis. The BRCA2 protein has a number of repeats of 39 amino acids, the BRC repeats. Eight BRC repeats in BRCA2 are defined in the Pfam database, of which six are highly conserved and are involved in binding RAD51.

Figure B4.3
BRC repeats of the BRCA2 protein as defined by the Pfam database.



Some typical BRC repeats

```
xFxTASxKxIxVSxxxxxxKxKxFFxD
xFxxAxGxxxxVSxxxLxKxKxLFkD
```

simply scoring identical matches as 1 and mismatches as 0 ignores the fact that the type of amino acid involved is highly significant. In particular, certain nonidentical amino acids are very likely to be present in the same functional position in two related sequences, and thus are likely to represent genuine matches. This is chiefly because certain amino acids resemble each other closely in their physical and/or chemical properties (see Figure 2.3) and can thus substitute functionally for each other. Mutational changes that replace one amino acid with another having similar physicochemical properties are therefore more likely to have been accepted during evolution. So pairs of amino acids with similar properties will often represent genuine matches rather than matches occurring randomly.

The simplest way of taking this into account is simply to count such similar pairs of amino acids as matches, and to refer to the score as **percent similarity**. In the now familiar example sequences below, red is used to indicate residues that are similar but not identical. Here the sequences have been realigned to take into account similarity as well as identity. Isoleucine (I) and alanine (A) are similar as they are both hydrophobic, whereas serine (S) and threonine (T) both have an -OH group in their side chain and are polar.

```
T  H  I  S  I  S  A  S  E  Q  U  E  N  C  E
|  |                 |  |  |  |  |  |  |  |
T  H  A  T  -  -  -  S  E  Q  U  E  N  C  E
```

Not all similar amino acid pairs are equally likely to occur, however, and more sophisticated measures of assessing similarity are more commonly used. In these, each aligned pair of amino acids is given a numerical score based on the probability of the relevant change occurring during evolution. In such scoring schemes, pairs of identical amino acids are assigned the highest score; then, pairs of amino acids with similar properties (such as isoleucine and leucine) score more highly than those with quite different properties (such as isoleucine and lysine), which are rarely found in corresponding positions in known homologous protein sequences.

Other properties of amino acids can be added into scoring schemes for greater accuracy. For example, the type of residue involved should be taken into account. Many cysteine residues are highly conserved because of their important structural role in forming disulfide bonds, and tryptophan residues are usually key components of the hydrophobic cores of proteins. To mimic this, the scores for matching residues can be varied according to the type, with pairs of cysteines and tryptophans, for example, being assigned particularly high values. When aligned amino acid pairs are given varying scores in this way, summing the values at all positions gives the **overall alignment score**.

Most currently used alignment-scoring schemes for protein sequences measure the relative likelihood of an evolutionary relationship compared to chance. The theory behind such assessments is explained further in Section 5.1. With such schemes, the higher the alignment score, the more likely it is that the aligned sequences are homologous.

Ideally, it would be possible to decide unequivocally whether two sequences are homologous by simply looking at their best alignment score. This turns out to be more difficult than might be imagined, as the significance of the score will depend on the length of the sequences, their amino acid composition, and the number of sequences being compared—for example when searching a large database. We shall return to this topic later in the chapter.

The concept of similarity, rather than identity, has little relevance to comparisons of nucleotide sequences, especially in generating alignments. Purines tend to mutate to purines ($A \leftrightarrow G$) and pyrimidines to pyrimidines ($C \leftrightarrow T$). This information can be used to help construct phylogenetic trees (see Sections 7.2 and 8.1), but is not helpful for sequence alignment. In the case of an alignment of nucleotide sequences, the scoring scheme is almost always very simple. For example, in the database-searching program FASTA, which is discussed later and in Section 5.3, a score of +5 for matching bases and –4 for mismatches has been found to be effective for DNA database searches. This simpler scoring scheme is sufficiently sensitive to be useful in part because of the much higher percentage identity expected if there is significant homology between the sequences, since there are only four types of bases as compared to 20 amino acids.

There is a minimum percentage identity that can be accepted as significant

What is the minimum percentage identity that can reasonably be accepted as significant? Burkhard Rost analyzed more than a million alignments of pairs of protein sequences for which structural information was available to find a cut-off for the level of sequence identity below which alignment becomes unreliable as a measure of homology. He found that 90% of sequence pairs with identity at or greater than 30% over their whole length were pairs of structurally similar proteins. Given both sequence and structural similarity, one can usually be confident that two sequences are homologous, so 30% sequence identity is generally taken as the threshold for an initial presumption of homology. Below about 25% sequence identity, however, Rost found that only 10% of the aligned pairs represented structural similarity. The region between 30% and 20% sequence identity has been called the **twilight zone,** where homology may exist but cannot be reliably assumed in the absence of other evidence. Even lower sequence identity (<20%) is referred to as the **midnight zone.**

There are many different ways of scoring an alignment

The function of an alignment score is to provide a single numerical value for the degree of similarity or difference between two sequences. Most current applications measure similarity, and in this case the highest scores are best. A few applications, particularly those used for generating phylogenetic trees (see Chapters 7 and 8), use a score related to sequence difference, usually known as a **distance**, in which case the most closely related sequences give alignments with the lowest scores. The measure of difference between two homologous sequences from different species is sometimes called the **genetic** or **evolutionary distance**.

There is no a priori reason why residue pair alignment scores cannot be negative, for example to represent especially unlikely alignments. In fact, some of the popular techniques require scores that can be negative, and most commonly used schemes have both positive and negative scores for pairs of residues.

Scoring schemes have to represent two salient features of an alignment. On the one hand, they must reflect the degree of similarity of each pair of residues; that is, the likelihood that both are derived from the same residue in the presumed common ancestral sequence. On the other hand, they must assess the validity of inserted gaps. Ways of quantifying these two features will be described separately here, although in fact they are used together to arrive at the final score. We will first go through the ways of assessing the degree of similarity for pairs of aligned residues.

4.3 Substitution Matrices

Substitution matrices are used to assign individual scores to aligned sequence positions

For alignments of protein sequences, the score is assigned to each aligned pair of amino acids is generally determined by reference to a **substitution matrix**, which defines values for all possible pairs of residues. Various types of substitution matrices have been used over the years. Some were based on theoretical considerations, such as the number of mutations that are needed to convert one amino acid into another, or similarities in physicochemical properties. The most successful, however, use actual evidence of what has happened during evolution, and are based on analysis of alignments of numerous homologs of well-studied proteins from many different species.

The choice of which substitution matrix to use is not trivial because there is no one correct scoring scheme for all circumstances. There is a wide range of variation in the similarity of sequences, from almost complete identity to a few percent. On one occasion we may need to align and score closely related sequences, whereas on another we may want to identify very distant relationships reliably. In the first case, the scoring scheme should be strongly biased toward giving high values to perfect matches and highly conserved substitutions. In the second case, a wider range of substitutions should be treated favorably.

Most scoring schemes for amino acid sequences use as reference a 20×20 substitution matrix, representing the 20 amino acids found in proteins. Each cell of the matrix is occupied by a score representing the likelihood that that particular pair of amino acids will occupy the same position through true homology, compared to the likelihood of their occurring as a random match. The most important scoring matrices will be described below, with general guidance as to which one to use when. A more comprehensive description of the theory underlying the scoring schemes discussed here is given in Section 5.1.

When an alignment is made, each aligned amino acid pair is given a score from the substitution matrix. These scores are then summed to give the overall score (S) of the alignment. For example, using the BLOSUM-62 matrix (see Figure 4.4A) we would score our example alignment as follows (in this case "U" represents an unknown residue; that is, a residue that could not be identified by sequencing techniques and is thus not given a score).

Seq1:	T	H	I	S	S	E	Q	U	E	N	C	E
Seq2:	T	H	A	T	S	E	Q	U	E	N	C	E
Score:	5	8	–1	1	4	5	5	0	5	6	9	5

Therefore the overall score S for this alignment equals 52. The BLOSUM matrices are described in more detail below.

The PAM substitution matrices use substitution frequencies derived from sets of closely related protein sequences

A commonly used set of substitution matrices is based on the observed amino acid substitution frequencies in alignments of homologous protein sequences. These matrices were first developed by Margaret Dayhoff and her co-workers in the 1960s and 1970s, and have been found to be superior to substitution schemes that use only the physicochemical similarities of amino acids, as they use real data to model the evolutionary process. The sequences used to generate these matrices were all very similar, allowing the alignment to be made with confidence. In addition, the high similarity meant that there was a high probability that amino acid differences at an alignment position were due to just a single mutation event, over a short period of time, since it is unlikely that more than one mutation would occur at the same site. A phylogenetic tree (see Section 7.1) was constructed for the protein sequences, from which the individual mutations that had occurred could be deduced. From this tree, the researchers calculated the ratio of the number of changes undergone by each type of amino acid to the total number of occurrences of that amino acid in the sequence set.

From these ratios it was possible to calculate the probabilities that any one amino acid would mutate into any other over a given period of evolutionary time. The final matrix of substitution scores is a logarithmic matrix of the mutation probabilities. Probabilities are converted to logarithms so that the final alignment score can be calculated by summation of the individual scores from aligned pairs of amino acids, rather than by multiplication of probabilities.

(A)

C	9																			
S	−1	4																		
T	−1	1	5																	
P	−3	−1	−1	7																
A	0	1	0	−1	4															
G	−3	0	−2	−2	0	6														
N	−3	1	0	−2	−2	0	6													
D	−3	0	−1	−1	−2	−1	1	6												
E	−4	0	−1	−1	−1	−2	0	2	5											
Q	−3	0	−1	−1	−1	−2	0	0	2	5										
H	−3	−1	−2	−2	−2	−2	1	−1	0	0	8									
R	−3	−1	−1	−2	−1	−2	0	−2	0	1	0	5								
K	−3	0	−1	−1	−1	−2	0	−1	1	1	−1	2	5							
M	−1	−1	−1	−2	−1	−3	−2	−3	−2	0	−2	−1	−1	5						
I	−1	−2	−1	−3	−1	−4	−3	−3	−3	−3	−3	−3	−3	1	4					
L	−1	−2	−1	−3	−1	−4	−3	−4	−3	−2	−3	−2	−2	2	2	4				
V	−1	−2	0	−2	0	−3	−3	−3	−2	−2	−3	3	2	1	3	1	4			
F	−2	−2	−2	−4	−2	−3	−3	−3	−3	−3	−1	−3	−3	0	0	0	−1	6		
Y	−2	−2	−2	−3	−2	−3	−2	−3	−2	−1	2	−2	−2	−1	−1	−1	−1	3	7	
W	−2	−3	−2	−4	−3	−2	−4	−4	−3	−2	−2	−3	−3	−1	−3	−2	−3	1	2	11
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W

(B)

C	9																			
S	−1	3																		
T	−3	2	4																	
P	−3	1	−1	6																
A	−3	1	1	1	3															
G	−5	1	−1	−2	1	5														
N	−5	1	0	−2	0	0	4													
D	−7	0	−1	−2	0	0	2	5												
E	−7	−1	−2	−1	0	−1	1	3	5											
Q	−7	−2	−2	0	−1	−3	0	1	2	6										
H	−4	−2	−3	−1	−3	−4	2	0	−1	3	7									
R	−4	−1	−2	−1	−3	−4	−1	−3	−3	1	1	6								
K	−7	−1	−1	−2	−2	−3	1	−1	−1	0	−2	2	5							
M	−6	−2	−1	−3	−2	−4	−3	−4	−4	−1	−4	−1	0	8						
I	−3	−2	0	−3	−1	−4	−2	−3	−3	−3	−4	−2	−2	1	6					
L	−7	−4	−3	−3	−3	−5	−4	−5	−4	−2	−3	−4	−4	3	1	5				
V	−2	−2	0	−2	0	−2	−3	−3	−3	−3	−3	−3	−4	1	3	1	5			
F	−6	−3	−4	−5	−4	−5	−4	−7	−6	−6	−2	−4	−6	−1	0	0	−3	8		
Y	−1	−3	−3	−6	−4	−6	−2	−5	−4	−5	−1	−6	−6	−4	−2	−3	−3	4	8	
W	−8	−2	−6	−7	−7	−8	−5	−8	−8	−6	−5	1	−5	−7	−7	−5	−8	−1	−1	12
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W

Figure 4.4
Amino acid substitution scoring matrices. (A) The BLOSUM-62 matrix and (B) the PAM120 substitution matrix. Each cell represents the score given to a residue paired with another residue (row × column). The values are given in half-bits, as discussed in Section 5.1. The colored shading indicates different physicochemical properties of the residues (see Figure 2.3): small and polar, yellow; small and nonpolar, white; polar or acidic, red; basic, blue; large and hydrophobic, green; aromatic, orange.

There is more than one such matrix and each matrix corresponds to a particular quantity of **accepted mutations** — mutations that have been retained in the sequence. This quantity is measured in PAM units, where PAM stands for Point Accepted Mutations (accepted point mutations per 100 residues), and these matrices are generally called **PAM matrices**. One of the more frequently used substitution matrices corresponds to 250 PAM, which means that 250 mutations have been fixed on average per 100 residues; that is, many residues have been subject to more than one mutation. The matrix itself is called PAM250. This amount of change is near the limit of detection of distant relationships. Other matrices, such as PAM120, correspond to a smaller amount of mutation (see Figure 4.4B)

The currently used PAM matrices, also known as Dayhoff mutation data matrices (MDMs), were originally created in 1978. More recent matrices have also been constructed using newer and larger data sets. The PET91 matrix, for example, represents a new generation of Dayhoff-type matrices.

The BLOSUM substitution matrices use mutation data from highly conserved local regions of sequence

The **BLOSUM matrix** is another very commonly used amino acid substitution matrix that depends on data from actual substitutions. It was derived much more recently than the Dayhoff matrices, in the early 1990s, using local multiple alignments rather than global alignments. First, a large set of aligned highly conserved short regions was generated from analysis of the protein-sequence database SWISS-PROT. The sequences were then clustered into groups according to similarity, so that sequences were grouped together if they exceeded a specified threshold for percentage identity. Substitution frequencies for all possible pairs of amino acids were then calculated between the clustered groups (without the construction of phylogenetic trees) and used to compute BLOSUM (BLOck SUbstitution Matrix) scores. Various BLOSUM matrices are obtained by varying the percentage cut-off for clustering into similarity groups. For example, the commonly used BLOSUM-62 matrix was derived using a threshold of 62% identity (see Figure 4.4).

The choice of substitution matrix depends on the problem to be solved

With many scoring matrices available, it is hard to know which one to use. Within a group of matrices such as the PAM or BLOSUM series, different ones, for example PAM250 versus PAM120 or BLOSUM-50 versus BLOSUM-80, are more suitable for different types of problem. The PAM matrix number indicates evolutionary distance whereas the BLOSUM matrix number refers to percentage identity. When aligning sequences that are anticipated to be very distantly related, matrices such as PAM250 and BLOSUM-50 may therefore be preferable. PAM120 and BLOSUM-80 may perform better for more closely related sequences.

Some matrices have been derived using additional information; the STR matrix, for example, includes information from known protein structures. Because protein structure is more conserved than sequence, more distantly related proteins can be compared using such methods, even when sequence alignment alone would not pick up any significant relationship.

Some scoring matrices have been designed to work well in special situations. For example, the matrices SLIM (ScoreMatrix Leading to Intra-Membrane) and PHAT (Predicted Hydrophobic And Transmembrane matrix) are especially designed for membrane proteins, where the characteristic amino acid composition and the selective forces for acceptable mutations are different from those for soluble proteins. In 2006, there were 94 matrices collected in a database list called AAINDEX and searchable at GenomeNet.

As well as the degree of evolutionary distance, the length of the sequences to be aligned must be taken into account when choosing a suitable matrix. This is especially relevant when searching databases against a query sequence, as the length of the sequence is taken into account when assessing the significance of the score: the shorter the sequence, the higher the score needs to be in order to be judged significant. Short sequences need to use matrices designed for short evolutionary time scales, such as PAM40 or BLOSUM-80. Longer sequences of 100 residues or more can use matrices intended for use with longer evolutionary time scales (such as PAM250 and BLOSUM-50). The reasons why the significance of a score depends on the length of the sequences to be aligned are discussed in more detail in Section 5.4.

4.4 Inserting Gaps

Gaps inserted in a sequence to maximize similarity with another require a scoring penalty

Homologous sequences are often of different lengths as the result of insertions and deletions (**indels**) that have occurred in the sequences as they diverged from the ancestral sequence. Their alignment is generally dealt with by inserting **gaps** in the sequences to achieve as correct a match as possible. To signify that an insertion or deletion has occurred, a letter or stretch of letters in one sequence is paired up with blank spaces (usually indicated by hyphens) inserted into the other sequence to achieve a better match.

Gaps must be introduced judiciously: forcing two sequences to match up simply by inserting large numbers of gaps will not reflect reality and will produce a meaningless alignment. To place limits on the introduction of gaps, alignment programs use a **gap penalty**: each time a gap is introduced, the penalty is subtracted from the score, decreasing the overall score of the alignment. Structural analysis has shown that fewer insertions and deletions occur in sequences of structural importance, and that insertions tend to be several residues long rather than just a single residue long. This information can be included in the scoring scheme by placing a smaller penalty on lengthening an existing gap (**gap extension penalty**) than on introducing a new gap, thus penalizing single-residue gaps relatively more. The best alignment is thus the one that returns the maximum score for the smallest number of introduced gaps.

Gap penalties can usually be varied in an alignment program, so the user has to decide what gap penalty to use. It should be kept in mind that the insertion of a gap must improve the quality of the alignment and therefore the maximum-match value. If a gap penalty is set high, then fewer gaps will be inserted into the alignment, as their inclusion will radically decrease the maximum-match value. If a low gap penalty is chosen, then more and larger gaps will be inserted. Therefore, if you are searching for sequences that are a strict match for your query sequence, the gap penalty should be set high. This will often retrieve a region, or regions, of very closely related sequence. If you are searching for similarity between distantly related sequences, the gap penalty should be set low. Note that suitable gap-penalty values may be different with different substitution matrices. It is advisable to start, when possible, with a combination of matrix and gap penalties that have been reported to give optimal performance.

In some alignment programs, a gap score depends on the type of residue with which the gap is aligned. Some types of residues are more likely to be conserved than others because their side chains tend to be more important in determining structure or function. An example is tryptophan, and so a gap aligned with a tryptophan will exact a larger gap penalty than a gap aligned with a glycine, for example.

(A)

```
Bovine PI-3Kinase p110a         LNWENPDIMSELLFQNNEIIFKNGDDLRQDMLTLQIIRIMENIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIMQIQCKGGLKGAL
cAMP-dependent protein kinase   --WENPAQNTAHLDQFERIKTLGTGSFGRVMLVKHMETGNHYAMKILDKQKVVKLKQIEHTLNEKRILQAVNFPFLVKLEFSFKDNSNLY

Bovine PI-3Kinase p110a         QFNSHTLHQWLKDKNKGEIYDAAIDLFTRSCAGYCVATFILGIGDRHNSNIMVKDDGQLFHIDFGHFLDHKKKFGYKRERVPFVLTQDF
cAMP-dependent protein kinase   MVMEYVPGGEMFSHLRRIGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCGTPEYLAP

Bovine PI-3Kinase p110a         LIVISKGAQECTKTREFERFQEMCYKAYLAIRQHANLFINLFSMMLGSGMPELQSFDDIAYIRKTLALDKTEQEALEYFMKQMNDAHHGG
cAMP-dependent protein kinase   EIILSKGYNKAVDWWALGVLIYEMAAGYPPFFADQPIQIYEKIVSGKVRFPSHFSSDLKDLLRNLLQVDLTKRFGNLKNGVNDIKNHKWF

Bovine PI-3Kinase p110a         WTTKMDWIFHTIKQHALN-----------------------------------
cAMP-dependent protein kinase   ATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVXINEKCGKEFSEF
```

(B)

```
Bovine PI-3Kinase p110a         LNWENPDIMSELLFQNNEIIFKNGDDLRQDMLTLQIIRIMENIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIMQIQCKGGLKGAL
cAMP-dependent protein kinase   ?-WENPAQNTAHLDQFERIKTLGTGSFGRVMLVKHM--ETGNHYAMKILDKQKV-VKLKQIEHTLNEKRILQAVNFPFLVKLEFSFKDN-

Bovine PI-3Kinase p110a         QFNSHTLHQWLKDKNKGEIYDAAIDLFTRSCAGYCVATFILGIGDRHNSNIMVKD-DGQLFHIDFGHFLDHKKKKFGYKRERVPFVL--T
cAMP-dependent protein kinase   -SNLYMVMEYVPGGEMFSHLRR-IGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCGT

Bovine PI-3Kinase p110a         QDFL---IVISKGAQECTKTREFERF-QEMC--YKAYLAIRQHANLFINLFSMMLGSGMPELQSFDDIAYIRKTLALDKTEQEALEYFMK
cAMP-dependent protein kinase   PEYLAPEIILSKGYNKAVDWWALGVLIYEMAAGYPPFFA-DQPIQIYEKIVSGKVRF--PSHFSSDLKDLLRNLLQVDLTKR--FGNLKN

Bovine PI-3Kinase p110a         QMNDAHHGGWTTKMDWI-----------------------FHTIKQHAL----N----------
cAMP-dependent protein kinase   GVNDIKNHKWFATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVXINEKCGKEFSEF
```

Figure 4.5
Pairwise alignments of the PI3-kinase p110α and a cAMP-dependent protein kinase. Note that the protein kinase sequence is considerably longer than the p110α sequence. (A) An alignment where the gap penalty has been set very high. Gaps have therefore only been inserted at the beginning and end of the sequences. The percentage identity of this alignment is 10%. (B) An alignment with a very low gap penalty. Many more gaps have been inserted to maximize the number of matched residues. Especially apparent is the lone matched pair of asparagine (N) residues in the carboxy-terminal region. The percentage identity of this alignment is 18%. Green shading, identical amino acids.

It is best to start with the default values given by the program you are using and then raise or lower the penalty to obtain a desired alignment. However, the number of gaps should always be kept to the minimum possible. Figure 4.5 shows two pairwise alignments of a **phosphatidylinositol-3-OH** kinase sequence (from bovine PI3-kinase p110α) and a **protein kinase** sequence from a cyclic AMP (cAMP)-dependent protein kinase (see Box 4.5), which have only limited similarity to each other.

In the first alignment (see Figure 4.5A) the gap penalty was set very high; therefore the program inserts as few gaps as possible. Any inserted gaps are found at the ends of the sequence, as often, unless there is an obvious relationship between the terminal amino acids, end gaps are not penalized. In the second alignment (see Figure 4.5B) the gap penalty was set very low; the effect is that many more gaps are inserted and the number of matched amino acids is increased (identities are shown in green). Although there are more matched residues in the alignment with low gap penalties, this does not necessarily mean that it is more accurate. In sequences that share such low homology as these, expert knowledge, such as the location of active-site residues, has to be used to decide if the alignment is accurate.

Dynamic programming algorithms can determine the optimal introduction of gaps

In practice, it is nearly always necessary to insert gaps into sequences when aligning them. The most obvious way of finding the best alignment with gaps would be to generate all possible gapped alignments, find the score for each, and select the highest-scoring alignment. This would be enormously time consuming,

Box 4.5 **Protein kinases and phospholipid kinases**

Phosphorylation is one of the commonest ways of rapidly altering a protein's activity. The enzymes that phosphorylate proteins are known as protein kinases and add phosphate groups to specific amino acid residues in the protein. Most, such as the cAMP-dependent protein kinases, phosphorylate serine or threonine residues, whereas others phosphorylate tyrosine residues. The effect of protein phosphorylation can be reversed by phosphoprotein phosphatases, which specifically remove the phosphate group. Because of their important roles as regulators of cellular activity and behavior, the activity of protein kinases is, in general, tightly controlled. The cAMP-dependent protein kinases, for example, are activated by binding the intracellular second messenger cAMP, which is specifically generated in response to a variety of extracellular signals acting at cell-surface receptors.

The phosphatidylinositol-3-OH kinases (PI3-kinases) phosphorylate inositol phospholipids in the cytoplasmic surface of the cell membrane, adding a phosphate group to position 3 on the inositol ring. Other members of this family, the PI4-kinases, phosphorylate the inositol ring on position 4. The phosphorylated lipids then specifically bind and activate other proteins, such as protein kinases, to initiate intracellular signal transduction cascades. PI3-kinases are involved in initiating the pathway by which the hormone insulin controls carbohydrate metabolism. PI3-kinases and protein kinases have very little sequence similarity to each other except in the enzymatic kinase domain.

however. For example, approximately 10^{75} alignments would need to be generated for a sequence of only 100 residues. It only became practicable to incorporate gaps into an alignment with the development of **dynamic programming algorithms**. These avoid unnecessary exploration of the bulk of alignments that can be shown to be nonoptimal. The name "dynamic programming" reflects the fact that the precise behavior of the algorithm is established only when it runs (in other words, dynamically) because it depends on the sequences being aligned.

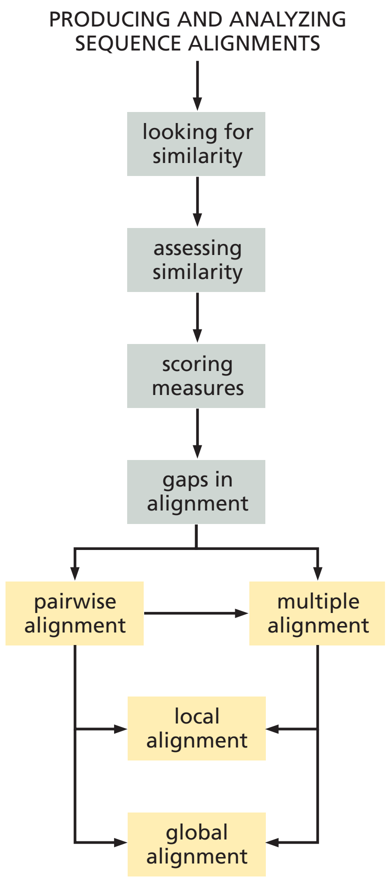
The first algorithm to use dynamic programming for sequence comparison was that of S. B. Needleman and C. D. Wunsch, published in 1970. Their technique is still the core of many present-day alignment and sequence-searching methods. In their method, gaps, regardless of length, have an associated penalty score; newer methods use more complicated gap penalties. The actual values of the gap scores can be varied depending on the type of scoring matrix being used. One rule always followed is that gaps can never be aligned with each other.

The basic concept of a Needleman–Wunsch-type algorithm is that comparisons are made on the basis of all possible pairs of amino acids that could be made between the two sequences. All possible pairs are represented as a two-dimensional matrix, in which one of the sequences to be aligned runs down the vertical axis and the other along the horizontal axis. All possible comparisons between any number of pairs are given by pathways through the array, each of which can be scored. The principles and method of the algorithm are dealt with in detail in Section 5.2. The general idea is to grow the alignment from the amino or carboxy terminus, at each step rejecting all possible alignments except that with the best score.

4.5 Types of Alignment

Different kinds of alignments are useful in different circumstances

The general principles outlined in the previous sections can be used to make different types of alignment (see Flow Diagram 4.2). Two closely related homologous sequences will generally be of approximately the same length, so that their alignment



Flow Diagram 4.2
The key concept introduced in this section is that there are several different types of sequence alignment, one of which will be the most appropriate for a particular problem.

will cover the full range of each sequence. This is referred to as a **global alignment**, and is generally the appropriate one to use when you want to compare or find closely related sequences that are similar over their whole length.

On the other hand, there are many cases where only parts of sequences are related. A simple example is the amino acid sequences of two proteins each consisting of two domains, with only one domain common to both proteins and the other domains completely unrelated. In this case, the only meaningful alignment will be a **local alignment** of the shared domain. Looking only at global alignments may not reveal the limited but important similarity between the sequences. This is particularly the case for comparisons between multidomain proteins, such as PI3-kinases, which consist of a number of small protein domains strung together (see Figure 4.6). Local alignment programs are therefore useful for detecting shared domains in such proteins.

When searching through a sequence database with a query sequence from an unknown protein, local alignment is a very useful tool to use initially. Once sequences with regions of high similarity are found using local alignment, global alignment can be used to align the rest of the sequence that is not so similar. Local alignment is also a good tool for identifying particular functional sites from which sequence patterns and motifs can be derived.

A widely used local alignment algorithm is the Smith–Waterman algorithm, which is a modification of the Needleman–Wunsch algorithm. Instead of looking at each sequence in its entirety, which is what the Needleman–Wunsch algorithm does, the Smith–Waterman method compares segments of all possible lengths and chooses the segment that optimizes the similarity measure. The scoring matrix used must include both positive and negative scores, and only alignments with a positive total score are considered. Therefore, if on extending the alignment at a particular step none of the possible alignments has a positive score, all previous alignments are

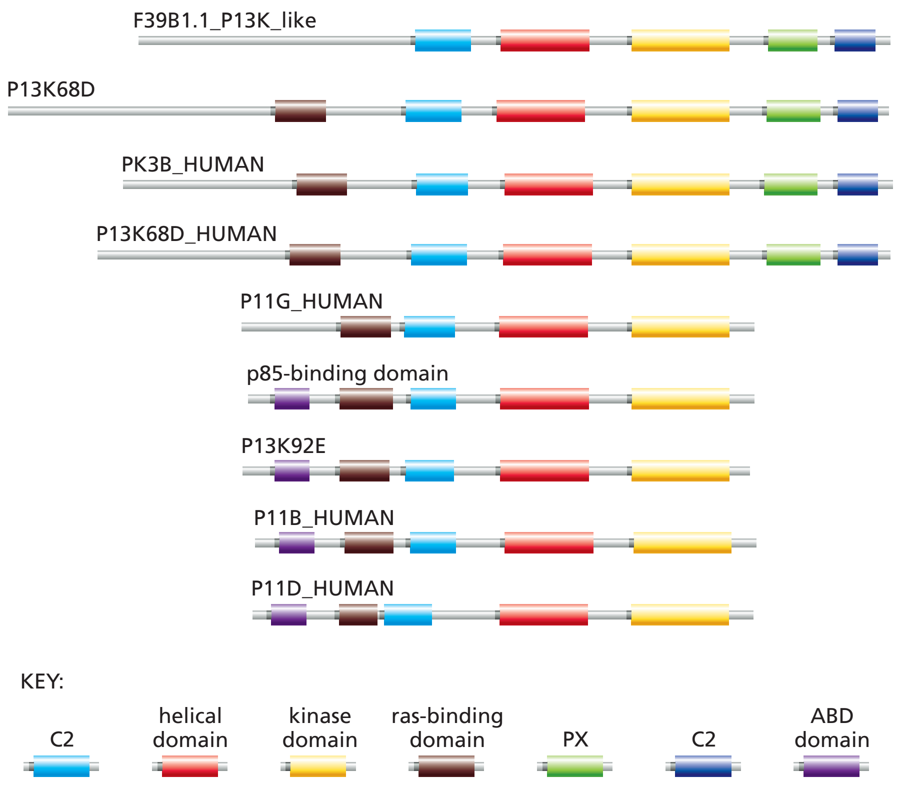
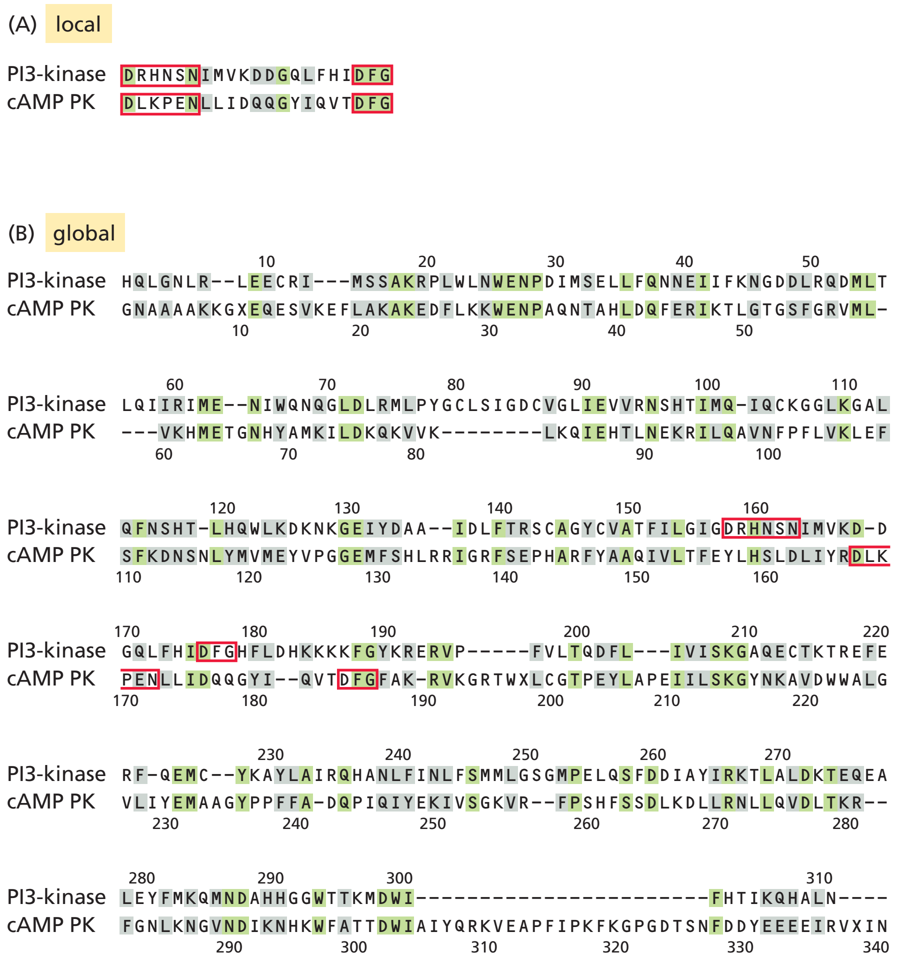


Figure 4.6
PI3-kinase is a multidomain protein. One possible output from a search of the Pfam database with the p100α PI3-kinase catalytic domain (yellow bar) is shown here. The figure also shows the complete domain structure of the protein family comprising the PI3-kinases and the related PI4-kinases, which catalyze phosphorylation of position 4 of the inositol ring of inositol phospholipids. The other domains and their arrangement are represented by the other colored bars.

rejected, and new ones are considered starting from that point. This makes the calculation sensitive to the precise match and mismatch scores and gap penalties. Section 5.2 describes the algorithm in detail.

Figure 4.7 shows an example of local versus global alignment of the complete protein sequences of the bovine PI3-kinase p110α and the cAMP-dependent protein kinase shown in Figure 4.5, using the Web-based programs ALIGN (global) and LALIGN (local). Although these proteins share structural homology within the core kinase catalytic domain, there is very little sequence homology. Figure 4.7A shows that local alignment of the catalytic domains has identified one important conserved region, out of five regions that were aligned. This region is involved in catalysis and also contains the three-residue motif DFG, which is conserved between many kinases. Figure 4.7B shows that, in this case, a global alignment fails to identify this region. The percentage sequence identity for these two sequences is very low (17.8%), well into the midnight zone of sequence alignment.

For both global and local alignments, methods exist for making **pairwise alignments**, that is, the alignment of just two sequences, and for making **multiple alignments**, in which more than two sequences are aligned with each other. In this part of the chapter, we have mainly used examples of pairwise alignments to illustrate the general principles of alignment scoring and quality assessment. Multiple alignment introduces yet another dimension to the computational problems of alignment. The theory is dealt with in detail in Chapter 6, but a few general points are described here.



```
(A) local

PI3-kinase  DRHNSNIMVKDDGQLFHIDFG
cAMP PK     DLKPENLLIDQQGYIQVTDFG

(B) global

                     10             20          30          40          50
PI3-kinase  HQLGNLR--LEECRI---MSSAKRPLWLNWENPDIMSELLFQNNEIIFKNGDDLRQDMLT
cAMP PK     GNAAAAKKGXEQESVKEFLAKAKEDFLKKWENPAQNTAHLDQFERIKTLGTGSFGRVML-
                   10         20          30          40          50

                60           70          80          90          100         110
PI3-kinase  LQIIRIME--NIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIMQ-IQCKGGLKGAL
cAMP PK     ---VKHMETGNHYAMKILDKQKVVK--------LKQIEHTLNEKRILQAVNFPFLVKLEF
               60         70          80                90          100

                  120         130         140         150         160
PI3-kinase  QFNSHT-LHQWLKDKNKGEIYDAA--IDLFTRSCAGYCVATFILGIGDRHNSNIMVKD-D
cAMP PK     SFKDNSNLYMVMEYVPGGEMFSHLRRIGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLK
            110        120         130         140         150         160

            170        180         190              200         210         220
PI3-kinase  GQLFHIDFGHFLDHKKKKFGYKRERVP-----FVLTQDFL---IVISKGAQECTKTREFE
cAMP PK     PENLLIDQQGYI--QVTDFGFAK-RVKGRTWXLCGTPEYLAPEIILSKGYNKAVDWWALG
            170        180              190         200         210         220

                        230         240         250         260         270
PI3-kinase  RF-QEMC--YKAYLAIRQHANLFINLFSMMLGSGMPELQSFDDIAYIRKTLALDKTEQEA
cAMP PK     VLIYEMAAGYPPFFA-DQPIQIYEKIVSGKVR--FPSHFSSDLKDLLRNLLQVDLTKR--
                230         240         250         260         270         280

              280         290         300                                 310
PI3-kinase  LEYFMKQMNDAHHGGWTTKMDWI-----------------------FHTIKQHALN----
cAMP PK     FGNLKNGVNDIKNHKWFATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVXIN
                      290         300         310         320         330         340
```

Figure 4.7
Local and global alignments. The complete sequences of PI3-kinase p110α and the cAMP-dependent protein kinase (cAMP PK) shown in Figure 4.5 were compared. (A) Local alignment using the program LALIGN (a subset of the FASTA package) has matched a short conserved region in the kinase domains that contains the functionally important residues D and N in the DLKPEN sequence and the DFG repeat common to nearly all kinases. (B) Because of the low overall sequence similarity, a standard global alignment of these two sequences using the program ClustalW has not matched these functionally important residues (boxed in each sequence). Green shading, identical amino acids; gray shading, similar amino acids.

Multiple sequence alignments enable the simultaneous comparison of a set of similar sequences

Multiple alignments can be used to find interesting patterns characteristic of specific protein families, to build phylogenetic trees, to detect homology between new sequences and existing families, and to help predict the secondary and tertiary structures of new sequences, as we shall see in more detail in Chapters 11 to 14.

In general, the alignment of multiple sequences will give a more reliable assessment of similarity than a pairwise alignment. The reason for this is that ambiguities in a pairwise comparison can often be resolved when further sequences are compared. Multiple alignment provides more information than pairwise alignment on the individual amino acid positions, such as the overall similarity and evolutionary relationships. This is especially important when using sequence-comparison methods to construct taxonomic phylogenetic trees. Multiple alignment is especially useful for illustrating sequence conservation throughout the aligned sequences. Such conservation over many sequences can identify amino acids that are important for function or for the structural integrity of the protein fold.

Multiple alignments can be constructed by several different techniques

A number of methods are available for generating multiple alignments. One of these is an extension of the dynamic programming method, so that instead of a two-dimensional matrix for a pair of sequences, an alignment of *n* protein sequences uses an *n*-dimensional matrix. However, this is limited by the prohibitively large computational requirement of the algorithm, and none of the examples discussed below uses this technique.

Other methods, while often using dynamic programming to align pairs of sequences, use other techniques to combine these together into one multiple alignment. Tree or hierarchical methods of multiple alignment are widely used, for example in the multiple alignment program ClustalW. This method first compares all the sequences in a pairwise fashion, then performs a **cluster analysis** on the pairwise data to generate a hierarchy of sequences in order of their similarity (see Figure 4.8A). The hierarchy is a simple phylogenetic tree and is often referred to as the **guide tree**. A multiple alignment is then built based on the guide tree by first aligning the most similar pairs, then aligning the other sequences with these pairs until all the sequences have been aligned (Figure 4.8B). However accurate this method is, there are problems with it in that any errors in the initial alignments cannot be corrected later as new information from other sequences is added. This

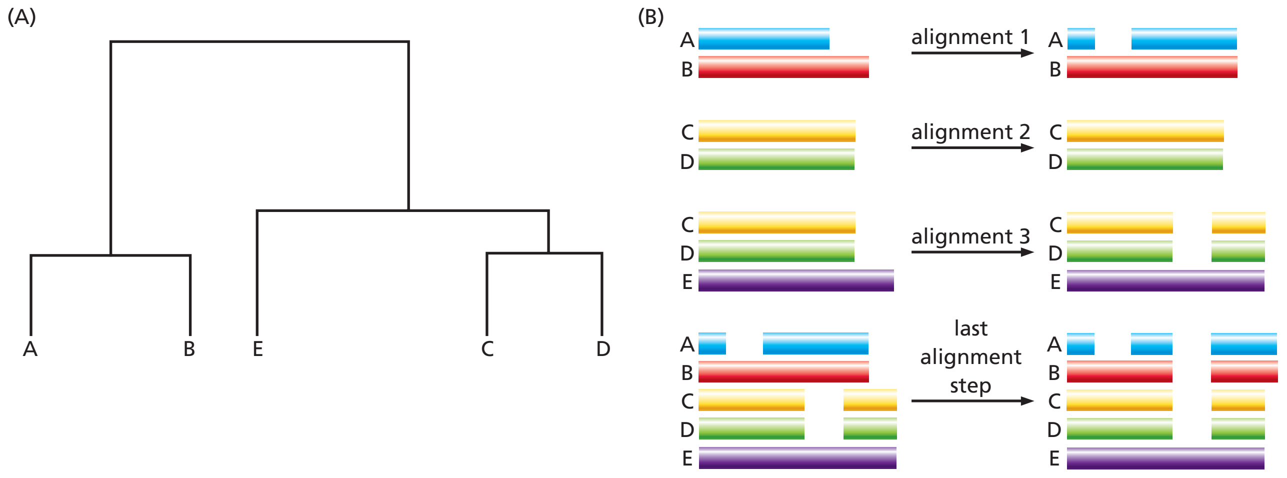


Figure 4.8
The tree method for the multiple alignment of sequences A, B, C, D, and E. Pairwise alignments are first made between all possible pairs of sequences—that is, AB, AC, AD, and so on—to determine their relative similarity to each other (not shown). (A) A cluster analysis is performed on this preliminary round of alignments, and the individual sequences are ranked in a tree according to their similarity to each other. (B) In the next step, the most similar sequences are aligned in pairs as far as possible. These are then aligned to the next closest sequence. This is repeated until all sequences or groups of sequences are aligned.

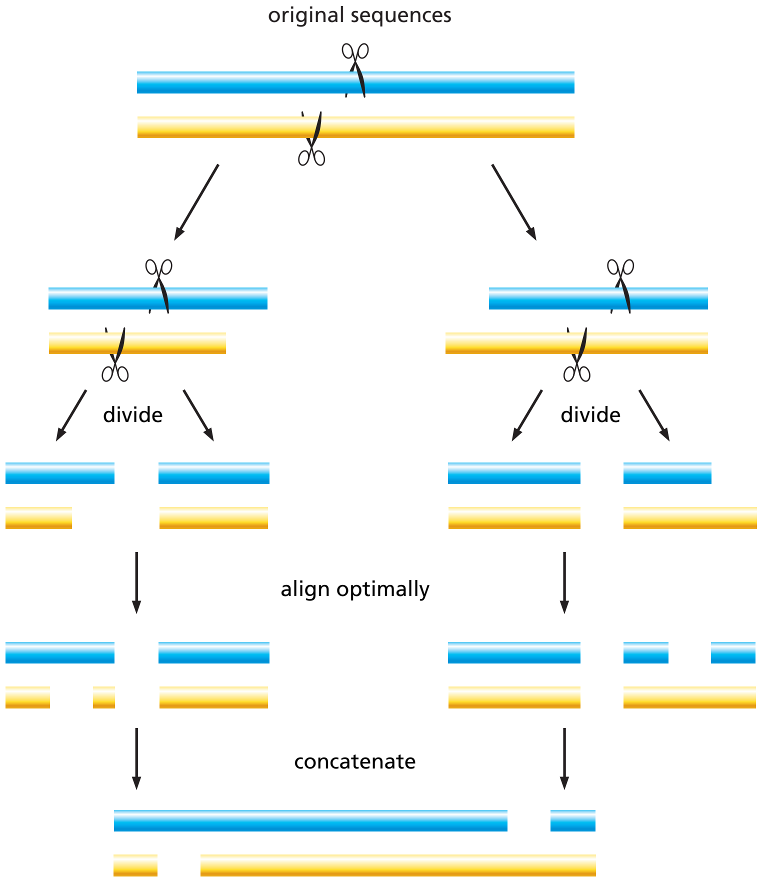


Figure 4.9
The divide-and-conquer method of multiple alignment. The sequences to be aligned are divided into two regions, then into four, and so on until the segments are considered small enough for accurate optimal alignment. The segments are then aligned and in the last step the alignments are concatenated to form the final complete multiple alignment.

difficulty has been avoided in iterative or stochastic sampling procedures as in the Barton and Sternberg program (see Chapter 6).

Other methods for building multiple alignments include the segment method, the consensus method, and the **divide-and-conquer method**. In the divide-and-conquer alignment, the sequences are first cut several times to reduce the length of the sequences to be aligned, the cut sequences are then aligned, and they are finally concatenated into a multiple alignment (see Figure 4.9). Initially, each sequence is divided into two segments at a suitable cut-position somewhere close to the midpoint of the sequences. This procedure is repeated until the sequences are shorter than a predetermined size, which is set as a parameter of the divide-and-conquer algorithm. Therefore the problem of aligning one family of long sequences is divided into several smaller alignment tasks. The segments are then aligned. The last step concatenates the short alignments, giving a multiple alignment of the original sequences.

Multiple alignments can improve the accuracy of alignment for sequences of low similarity

The same proteins with which we illustrated local versus global alignment—a cAMP-dependent protein kinase and a PI3-kinase—will be used to illustrate the improvement multiple alignment can make to the alignment of sequences of low similarity. Figure 4.10A shows part of a pairwise alignment between the protein kinase and the PI3-kinase. The active-site region and the DFG pattern are not aligned. Figure 4.10B shows the result of a multiple alignment between five different PI3-kinases and the protein kinase made using the program ClustalW with the default settings. The effect of the multiple alignment is to give added weight to

```
(A) p110α          TFILGIGDRHNSNIMVKDDG-QLFHIDFGHFLDHKKKKFGYKRERVPFVLT--QDFLIVI 142
    cAMP-kinase    QIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCGTPEYLAPE 179

(B) p110β          SYVLGIG----------DRHSDNINVKKTGQLFHIDFGHILGNFKSKFGIKRERVPFILT 136
    p110δ          TYVLGIG----------DRHSDNIMIRESGQLFHIDFGHFLGNFKTKFGINRERVPFILT 136
    p110α          TFILGIG----------DRHNSNIMVKDDGQLFHIDFGHFLDHKKKKFGYKRERVPFVLT 135
    p110γ          TFVLGIG----------DRHNDNIMITETGNLFHIDFGHILGNYKSFLGINKERVPFVLT 135
    p110_dicti     TYVLGIG----------DRHNDNLMVTKGGRLFHIDFGHFLGNYKKKFGFKRERAPFVFT 135
    cAMP-kinase    QIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCG--TPEYLA 177
```

Figure 4.10
Pairwise and multiple alignments of part of the catalytic domains of five PI3-kinases and a cAMP-dependent protein kinase. (A) Pairwise alignment of PI3-kinase p110α and the protein kinase does not align the important active-site residues and the DFG motif (in green). (B) Multiple alignment of the protein kinase with a set of five PI3-kinases (which have considerable overall homology to each other) has the effect of forcing the best-conserved regions to be matched. Here the DFG motif and the important N and D (green) residues are aligned correctly in all the sequences. In addition it is apparent that a G (green) is also totally conserved (identical) and that three more residues are conserved in their physicochemical properties (blue).

the conserved residues within the PI3-kinases, resulting in a better alignment for that region of the kinase domain.

ClustalW can make global multiple alignments of both DNA and protein sequences

ClustalW uses a tree method of multiple alignment as described briefly above. The program is easy to use with the default settings and can be accessed from a number of Web sites. To use it, one must have collected a set of sequences, perhaps from a database search. Either protein or DNA sequences can be used. The sequences are cut and pasted into a dialog box; you can then run the program immediately with the default settings (for gap penalties and type of scoring matrix, for example). All the settings can be changed if required.

Multiple alignments can be made by combining a series of local alignments

DIALIGN is a relatively recent method for multiple alignment developed by Burkhard Morgenstern and colleagues. Whereas standard alignment programs such as ClustalW compare residues one pair at a time and impose gap penalties, DIALIGN constructs pairwise and multiple alignments by comparing whole ungapped segments several residues long. The alignment is then constructed from pairs of equal-length gap-free segments, which are termed diagonals because they would show up as diagonal lines in the respective pairwise comparison matrices. The segment length varies between diagonals. Many diagonals overlap, and the program has to find a set that can be combined into one consistent alignment (see Section 6.5). As the segments are gap-free there is no need to use a gap-penalty parameter. Every diagonal is given a weight reflecting the degree of similarity between the two segments involved. The overall score of an alignment is the sum of the weights of all the diagonals, and the program finds the alignment with the maximum score. A threshold can be set so that diagonals are considered only if their weights exceed this threshold, so that regions of lower similarity are ignored. As DIALIGN is a local alignment method it may not align the whole sequence, and may align several blocks of residues with unaligned regions between them.

Figure 4.11 illustrates the alignment of five SH2 domain sequences using ClustalW, DIALIGN, and the divide-and-conquer algorithm (DCA) methods compared with the structural/functional alignment from BAliBase, which can be considered accurate. All three methods fail to some extent to align the residues of the first helix correctly, inserting a gap. ClustalW does slightly worse in this region by splitting the helix, but is better in conserving the integrity of the second core block around the FLVR region important for binding. DCA does not align the last helix as well as ClustalW or DIALIGN. However, all the alignment programs are generally good and useful in that they often produce alignments very close to the correct ones based on extra information, such as those found in BALiBase.

(A) structural/functional alignment from BAliBase

```
1csy   SHEKMPWFHGKISREESEQIVLIGSKTNGKFLIRARD--NNGSYALCLLHEGKVLHYRIDKDKTGKLSIPEGK-KFDTLWQLVEHYSYKA------DGLLRVL-TVPCQK
1gri   EMKPHPWFFGKIPRAKAEEML-SKQRHDGAFLIRESES-APGDFSLSVKFGNDVQHFKVLRDGAGKYFL-WVV-KFNSLNELVDYHRSTS-VSRNQQIFLRDIEQVPQQ-
1aya   ---MRRWFHPNITGVEAENLLLTRG-VDGSFLARPSKS-NPGDFTLSVRRNGAVTHIKIQN--TGDYYDLYGGEKFATLAELVQYYMEHHGQLKEKNGDVIEL-KYPLN-
2pna   -LQDAEWYWGDISREEVNEKLRDT--ADGTFLVRDASTKMHGDYTLTLRKGGNNKLIKIFH-RDGKYGFSDPL-TFNSVVELINHYRNES-LAQYNPKLDVKL-LYPVS-
1bfi   HHDEKTWNVGSSNRNKAENLLRGK--RDGTFLVRESS--KQGCYACSVVVDGEVKHCVINKTATG-YGFAEPYNLYSSLKELVLHYQHTS-LVQHNDSLNVTL-AYPVYA
```

(B) DIALIGN multiple sequence alignment

```
1csy   SHEKMPWFHGKISREESEQIVLIGSKT-NGKFLIRAR-DN--NGSYALCLLHEGKVLHYRIDKDKTGKLSIPEGKK-FDTLWQLVEHYSYKA-------DGLLRVLT-VPCQK
1gri   EMKPHPWFFGKIPRAKAEEML--SKQRHDGAFLIRESESA--PGDFSLSVKFGNDVQHFKVLRDGAGKYFLWVV-K-FNSLNELVDYHRST--SVSRNQQIFLRDIEQVPQQ-
1aya   M---RRWFHPNITGVEAENLLLTRGV--DGSFLARPSKSN--PGDFTLSVRRNGAVTHIKIQNTGDYYDLYG-GEK-FATLAELVQYYMEHHGQLKEKNGDV-IELK-YPLN-
2pna   LQDAE-WYWGDISREEVNEKL--RDTA-DGTFLVRDA-STKMHGDYTLTLRKGGNNKLIKIFHRDGKYGFSD-PLT-FNSVVELINHYRNE--SLAQYNPKLDVKLL-YPVS-
1bfi   HHDEKTWNVGSSNRNKAENLL--RGKR-DGTFLVRES-SK--QGCYACSVVVDGEVKHCVINKTATGYGFAE-PYNLYSSLKELVLHYQHT--SLVQHNDSLNVTLA-YPVYA
```

(C) ClustalW multiple sequence alignment

```
1csy   SHEKMPWFHGKISREESEQIVLIGSKTNGKFLIRARDN--NGSYALCLLHEGKVLHYRIDKDKTGKLSIPEGKKFD-TLWQLVEHYSYK------ADGLLRVLTVPCQK
1gri   EMKPHPWFFGKIPRAKAEE-MLSKQRHDGAFLIRESES-APGDFSLSVKFGNDVQHFKVLRDGAGKY-FLWVVKFN-SLNELVDYHRSTS-VSRNQQIFLRDIEQVPQQ
1aya   ---MRRWFHPNITGVEAEN-LLLTRGVDGSFLARPSKS-NPGDFTLSVRRNGAVTHIKIQNT-GDYYDLYGGEKFA-TLAELVQYYMEHHGQLKEKNGDVIELKYPLN-
2pna   -LQDAEWYWGDISREEVN--EKLRDTADGTFLVRDASTKMHGDYTLTLRKGGNNKLIKIFHR-DGKYGFSDPLTFN-SVVELINHYRNES-LAQYNPKLDVKLLYPVS-
1bfi   HHDEKTWNVGSSNRNKAE--NLLRGKRDGTFLVRESSK--QGCYACSVVVDGEVKHCVINKT-ATGYGFAEPYNLYSSLKELVLHYQHTS-LVQHNDSLNVTLAYPVYA
```

(D) divide-and-conquer multiple sequence alignment

```
1csy   SHEKMPWFHGKISREESEQIVLIGSKTNGKFLIRA-RDNN-GSYALCLLHEGKVLHYRIDKDKTGKLSIPEGKK-FDTLWQLVEHY-SY----KADGLLRV-L-TVPCQK
1gri   EMKPHPWFFGKIPRAKAEEMLS-KQRHDGAFLIRE-SESAPGDFSLSVKFGNDVQHFKVLRDGAGK-YFLWVVK-FNSLNELVDYH-RSTSVSRNQQIFLRDIEQVPQQ-
1aya   ---MRRWFHPNITGVEAENLLL-TRGVDGSFLARP-SKSNPGDFTLSVRRNGAVTHIKIQNTGDYY-DLYGGEK-FATLAELVQYYMEHHGQLKEKNGDVIEL-KYPLN-
2pna   -LQDAEWYWGDISREEVNEKL--RDTADGTFLVRDASTKMHGDYTLTLRKGGNNKLIKIFHRDGKY-GFSDPLT-FNSVVELINHY-RNESLAQYNPKLDVKL-LYPVS-
1bfi   HHDEKTWNVGSSNRNKAENLL--RGKRDGTFLVRE-SSKQ-GCYACSVVVDGEVKHCVINKTATGY-GFAEPYNLYSSLKELVLHY-QHTSLVQHNDSLNVTL-AYPVYA
```

Figure 4.11
Known structural alignments can be useful in checking sequence alignments. (A) Multiple alignment of the sequences of five SH2 domains according to their sequence/structure alignment in BAliBase. α-Helices are shown in red and β-strands in yellow. (B) Multiple alignment for the same set of sequences obtained by DIALIGN. (C) Alignment obtained by ClustalW. (D) Alignment obtained by the divide-and-conquer method. There is not much difference in performance between the algorithms (all were run with the default settings), although some alignment programs break the secondary structure element indicated by dashes. The coded names of the domains on the left are their identification numbers in the Protein Data Bank (PDB).

Once a satisfactory alignment has been obtained, there are now numerous programs available through the Web that allow you to view, analyze, and even edit alignments. AMAS (Analyze Multiply Aligned Sequences), CINEMA (Colour Interactive Editor for Multiple Alignments), and ESPript (Easy Sequencing in Postscript) are but a few.

Alignment can be improved by incorporating additional information

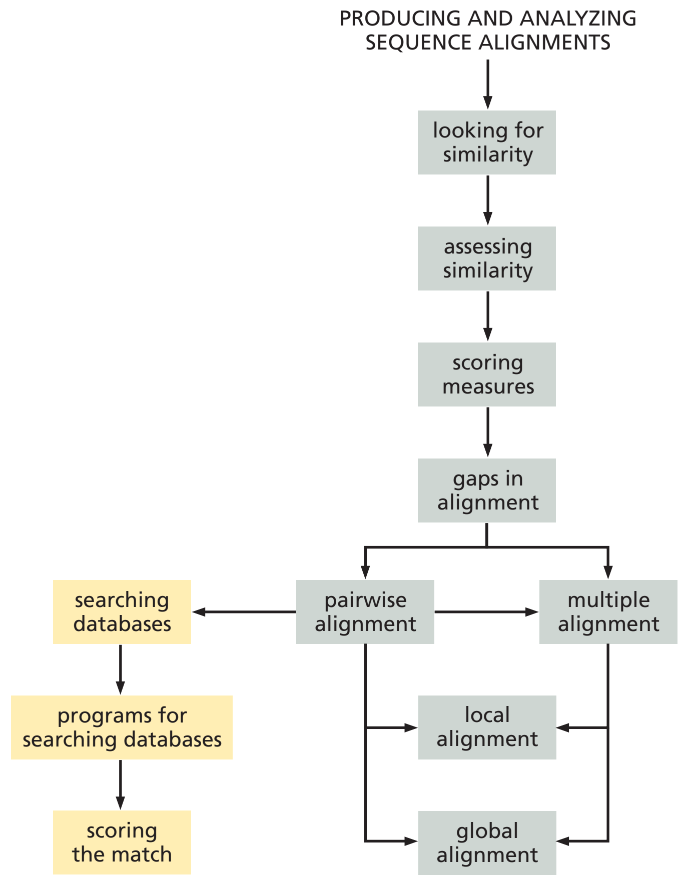
The alignment of two or more sequences can be improved by incorporating expert knowledge such as known structural properties of one or more sequences. For example, if the structure of one of the proteins to be aligned is known, then the gap penalty can be increased for regions of known secondary structures such as α-helices or β-strands, as these regions are less likely to suffer insertions or deletions. This will mean that few or no gaps are introduced into these regions. On the other hand, gap penalties can be decreased for loop regions, in which insertions and deletions are better tolerated.

Often the results of an automatic alignment program benefit from manual final adjustment. For example, if specific residues are known to be important for structure, function, or ligand binding, then manual realignment may be necessary to match these residues.

4.6 Searching Databases

Searching sequence databases now has a part to play in nearly every branch of molecular biology, and is crucial for making sense of the sequence data becoming available from the genome projects. For example, one may wish to search the database with a DNA sequence to locate and identify a gene in a new genome. When a protein sequence is available, then searching through the database can be used to identify the potential function. Sometimes one wishes to find the gene for

Flow Diagram 4.3
The key concept introduced in this and the following section is that applications have been designed to overcome the problems associated with searching a database for sequences that are similar to a query sequence, including the need to pay special attention to the statistical significance of the alignment scores obtained.



a particular protein in a genome, which can be done by searching with a homologous protein or DNA sequence.

We will now discuss the practical task of searching sequence databases to find sequences that are similar to the query sequence or search sequence that we submit to them (Flow Diagram 4.3). When searching a database with a newly determined DNA or protein query sequence, one does not usually know whether an expected similarity might span the entire query sequence or just part of it; similarly, one does not know if the match will extend along the full length of a database sequence or only part of it. Therefore, one initially needs to look for local alignments between the query sequence and any sequence in the database. The top-scoring database sequences are then candidates for further analysis.

Database searching needs to be both sensitive, in order to detect distantly related homologs and avoid **false-negative** searches, and also specific, in order to reject unrelated sequences with fortuitous similarity (**false-positive** hits). This is not an easy balance to achieve, and search results should be scrutinized with care.

In general, it is not possible to decide from a visual inspection of the alignment whether the database and query sequences are truly homologous. However, analysis of the score statistics has provided us with useful measures to estimate the validity of a hit. This important aspect of database searching, which is required to interpret any database search correctly, is discussed later in this chapter and in more detail in Section 5.4.

Fast yet accurate search algorithms have been developed

The sequence databases are now extremely large and growing daily. This means that aligning a query sequence with sequences in a database requires considerable computer resources. In the past, this exceeded the available computing power and so great effort was put into developing fast yet accurate alignment methods. Almost all database search programs currently in use are modifications of the rigorous methods discussed earlier. The Needleman–Wunsch and Smith–Waterman methods are rigorous in the sense that given a scoring scheme they are guaranteed to find the best-scoring alignments between two sequences. Two suites of programs are in common use for database searching: FASTA and BLAST. These use dynamic programming, but only for database entries that have a segment sufficiently similar to the query sequences. The methods used to find these entries are purely heuristic; that is, not rigorous.

FASTA is a fast database-search method based on matching short identical segments

FASTA is a popular database-searching program that increases the speed of a search at the expense of some sensitivity. It speeds up the searching process by using **k-tuples**, short stretches of *k* contiguous residues. In protein searches *k* can equal 1 or 2, while 6 is a typical value for DNA. The program makes up a dictionary of all possible k-tuples within the query sequence. Each entry contains a list of numbers that describe the location of the k-tuple in the query sequence. This is called **hashing**, and the theory behind it is described in Section 5.3. Therefore, for each k-tuple in the searched sequences, FASTA only has to consult the dictionary to find out if it occurs in the query sequence. However, sensitivity is reduced because a partial match of a k-tuple (for example, AC to AG in DNA) is ignored. Therefore, although speed increases with the length of a k-tuple, sensitivity will decrease.

In the first step of the FASTA method all possible pairwise k-tuples are identified: these can be considered as diagonals in a set of dot-plots. In the second step, alignments of these diagonals are rescored using a scoring matrix such as one of those described above. In this step, the k-tuple regions are also extended without including gaps, and only those that score above a given threshold are retained. In the third step, the program checks to see if some of the highest-scoring diagonals can be joined together. Finally, the search sequences with the highest scores are aligned to the query sequence using dynamic programming. The final alignment score ranks the database entries and the highest-scoring set is reported.

BLAST is based on finding very similar short segments

BLAST (Basic Local Alignment Search Tool) or Wu-BLAST (a version of BLAST developed at Washington University, St Louis) is one of the most widely used database-search program suites. It relies on finding core similarity, which is defined by a window of preset size (called a "word") with a certain minimum density of matches (for DNA) or with an amino-acid similarity score above a given threshold (for proteins). Note that these amino acid word-matches do not only include identities and that they are scored with a standard substitution matrix. In the first step, all suitable matches are located in each database sequence. Subsequently, matches are extended without including gaps, and on this basis the database sequences are ranked. The highest-scoring sequences are then subjected to dynamic programming to obtain the final alignments and scores. BLAST and Wu-BLAST can be run with or without the use of gaps. The gapped setting of BLAST, which is usually the default setting, reports the best local alignments and is suitable for most applications. Both the FASTA and BLAST methods are described in detail in Section 5.3.

Different versions of BLAST and FASTA are used for different problems

Many of the search algorithms can be used to search either nucleic acid or protein sequences, or even to search a protein-sequence database using a nucleic acid

Table 4.1
The various algorithms within the FASTA package are given with descriptions of their uses. Equivalent programs in the BLAST package are highlighted. The ktup parameter of fasta defines the length of a k-tuple, as explained in Section 5.3.

Program	Description	BLAST equivalent
fasta	Protein compared to protein database or DNA to DNA database. For protein, ktup = 2 by default (ktup = 1 is more sensitive); default for DNA is 6; 4 or 3 is more sensitive. 1 should be used for short DNA stretches.	blastp/blastn
ssearch	Uses Smith–Waterman algorithm. Can search protein to protein or DNA to DNA. Can be more sensitive than fasta with protein sequences.	
fastx/fasty	DNA compared to protein database. DNA translated into all three frames. fasty slower than fastx but better. Used to see if DNA encodes a protein.	blastx
tfastx/tfasta	Protein compared to DNA database. Mainly used to identify EST sequences. This is preferred over fastx as protein comparison is more sensitive than DNA.	tblastn (tblastx compares translated DNA to translated DNA database)
fastf	Mixed peptide sequence (such as obtained by Edman degradation) compared to protein database.	
tfastf	Mixed peptide sequence compared to DNA database.	

sequence and vice versa. However, you need to choose the correct program for the required type of search. In BLAST, for example, one can choose among blastp, which compares an amino acid query sequence against a protein-sequence database; blastn, which compares a nucleotide query sequence against a nucleic acid sequence database; blastx, which compares a nucleotide query sequence translated in all reading frames against a protein-sequence database; tblastn, which compares a protein query sequence against a nucleotide-sequence database dynamically translated in all reading frames; and finally, tblastx, which compares the six possible translations of a nucleotide query sequence against the six frame translations of a nucleotide-sequence database. The FASTA suite has similar versions of these search programs (see Table 4.1).

PSI-BLAST enables profile-based database searches

Variations of BLAST such as PSI-BLAST (Position-Specific Iterative BLAST) have been devised. This suite of programs makes use of features characteristic of a particular protein family to identify related sequences in a protein database, and can identify related sequences that are too dissimilar to be found in a straightforward BLAST search. In PSI-BLAST, a **profile**, or **position-specific scoring matrix (PSSM)**, of a set of sequences is constructed from a multiple alignment of the highest-scoring hits returned in an initial BLAST search (see Section 6.1). The PSSM is created by calculating new scores for each position in this alignment. A highly conserved residue at a particular position is assigned a high positive score, while other residues at that position are assigned high negative scores. At positions that are weakly conserved throughout the alignment, all residues are given scores near zero. The profile generated is used to replace the substitution matrix in a subsequent BLAST search. This process can be repeated many times; each time, the results from the search are used to refine the profile. This type of iterative searching results in increased sensitivity and has been used to good effect in protein-fold recognition programs such as 3D-PSSM (see Chapter 13).

Ways of extracting more distantly related homologous sequences and finding links between known families are now being explored. Such methods include, for example, the use of **intermediate sequences**; that is, sequences that are found in more than one family. Suppose we submit an unknown sequence A to a database search and among the significant hits there is a protein called, for example, mediator protein. We then submit an unknown sequence B to the same database search, and this also returns mediator protein with a significant score. Then, especially if more than one such intermediate sequence is found, we can deduce that sequences A and B are homologous, as their families are related. Such ideas can be automated for ease of application.

SSEARCH is a rigorous alignment method

Despite the computational requirements, some programs have been written that use rigorous methods to search databases. SSEARCH is a search program based on the Smith–Waterman algorithm and is therefore slower than either BLAST or FASTA. SSEARCH performs a rigorous search for similarity between a query sequence and the database. Other search algorithms based on the Smith–Waterman method have been written and are gaining in popularity as computer power increases.

4.7 Searching with Nucleic Acid or Protein Sequences

DNA or RNA sequences can be used either directly or after translation

In general, nucleic acid sequence searches are more difficult to handle and analyze than protein sequence searches. However, most primary data will be in the form of nucleic acid sequences. If you have an untranslated DNA or RNA sequence and you want to know if the DNA codes for a protein, you can use fasta, ssearch, or blastn (see Table 4.1) to search the EST (expressed sequence tag), EMBL, or nr (nonredundant) databases, or one of the species-specific genome EST databases, such as EST-Rodent. The results may well be confusing, in that a lot of partial sequence matches will be found. Many retrieved sequences will also be unknown sequences. An easier search can be made using fastx/fasty (or blastx), which will translate the DNA in all three reading frames on both strands—six translations in all—and search a protein database of choice. More details and examples of dealing with DNA sequences can be found in Chapter 9.

The quality of a database match has to be tested to ensure that it could not have arisen by chance

How good is an alignment and how believable are the results of a database search? These vital questions must be answered before any further use can be made of the results. Every alignment reported will have been selected on the basis of its score. What we need to know is whether the score is greater than we would expect from the alignment of the sequence with a random (unrelated) sequence. However, there is a complex relationship between the score and the significance of the sequence similarity. For one thing, as each pair of aligned residues contributes to the score, longer sequences are expected to give higher alignment scores, assuming the same degree of similarity.

If a large number of random sequences are generated and aligned with the query sequence, the resulting alignment scores will follow a particular distribution. Because we always choose the best-scoring alignment, the distribution will be related to the extreme-value distribution (see Section 5.4). Through application of

Figure 4.12
The results of a search of the SWISS-PROT protein sequence database using BLAST with PI3-kinase p100α as the query sequence. (A) Output from a standard BLAST search. Each line reports a separate database sequence. The penultimate column gives the alignment score, and the last column the *E*-value. Hits before the arrow are significant, while below the arrow the hit does not have a significant score. (B) A BLAST search on one month's new sequences, using the same query sequence as in (A), finds only two matches. One is a PI4-kinase, which has most of its sequence aligned to the query sequence (magenta line). The other has only a small region aligned (black line) and a borderline score. (C) Output from a Conserved Domain Database (CDD) search.

this distribution it is possible to estimate the probability of two random sequences aligning with a score greater than or equal to *S*. This is usually reported as an **expectation value** or ***E*-value**, and is used to order the database search results.

The programs BLAST and FASTA calculate an *E*-value, which is the number of alignments with a score of at least *S* that would be expected by chance alone in searching a complete database of *n* sequences. These *E*-values can vary from 0 to *n*. For example, by chance alone, you would expect to find three sequence alignments with an *E*-value of 3.0 or less in a database search, so an *E*-value of 3.0 suggests that the database sequence is not related to the query sequence. Quite closely related sequences often give very small *E*-values of 10^{-20} or less, and such scores clearly indicate a significant similarity of the database and query sequences. However, we really need to know how large an *E*-value can be while still reliably indicating a significant sequence similarity. It is important to remember that the *E*-value depends on the sequence length and the number of sequences in the database as well as on the alignment score.

In general, the smaller the *E*-value the better the alignment, and the higher the percentage identity the more secure the assessment of the significance of the similarity between the database sequence and the query sequence. The default *E*-value threshold in many search packages is set to either 0.01 or 0.001. However, most programs permit the user to set the *E*-value threshold, and matches above that threshold will not be included in the output.

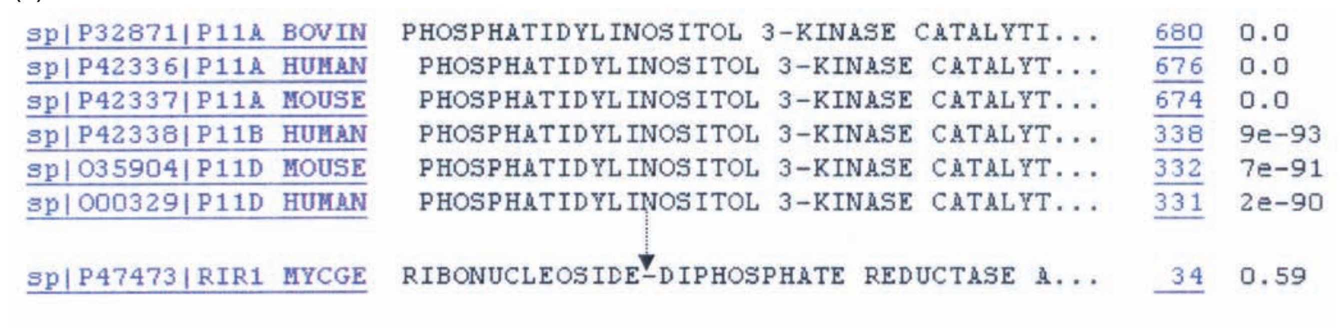
To test new or existing sequence-alignment programs and their scoring schemes one can compare the alignment obtained by the program against carefully constructed alignments that are based on known structural features or biological function. There are databases of such accurately aligned sequences, such as BAliBase.

Choosing an appropriate *E*-value threshold helps to limit a database search

To illustrate some of the possible sequence searches, alignments, and analyses that can be carried out via the Web, we will use two examples: the catalytic domain from a PI3-kinase and the protein-interaction domain SH2. Structural information and an accurate alignment in the BAliBase database are available for the family of SH2 domains.

The human *Syk* tyrosine kinase carboxy-terminal SH2 domain is the first query sequence. Protein searches with BLAST through the SWISS-PROT database gave 149 sequences below the default *E*-value cut-off. All these were SH2-related domains. That is a lot of information to cope with. All the *E*-values were very low, indicating that all the hits were significant. This is a case of result overload. Decreasing the *E*-value cut-off would have no effect in this case, as all the hits were far below the threshold used.

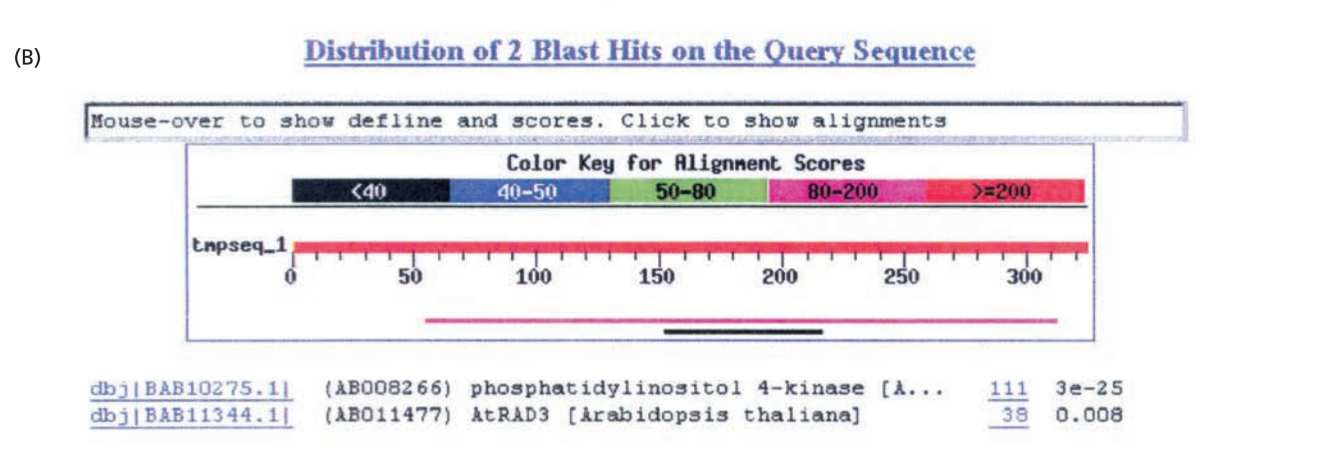
(A)



```
sp|P32871|P11A BOVIN   PHOSPHATIDYLINOSITOL 3-KINASE CATALYTI...   680  0.0
sp|P42336|P11A HUMAN   PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...    676  0.0
sp|P42337|P11A MOUSE   PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...    674  0.0
sp|P42338|P11B HUMAN   PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...    338  9e-93
sp|O35904|P11D MOUSE   PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...    332  7e-91
sp|O00329|P11D HUMAN   PHOSPHATIDYLINOSITOL 3-KINASE CATALYT...    331  2e-90

sp|P47473|RIR1 MYCGE   RIBONUCLEOSIDE-DIPHOSPHATE REDUCTASE A...    34  0.59
```

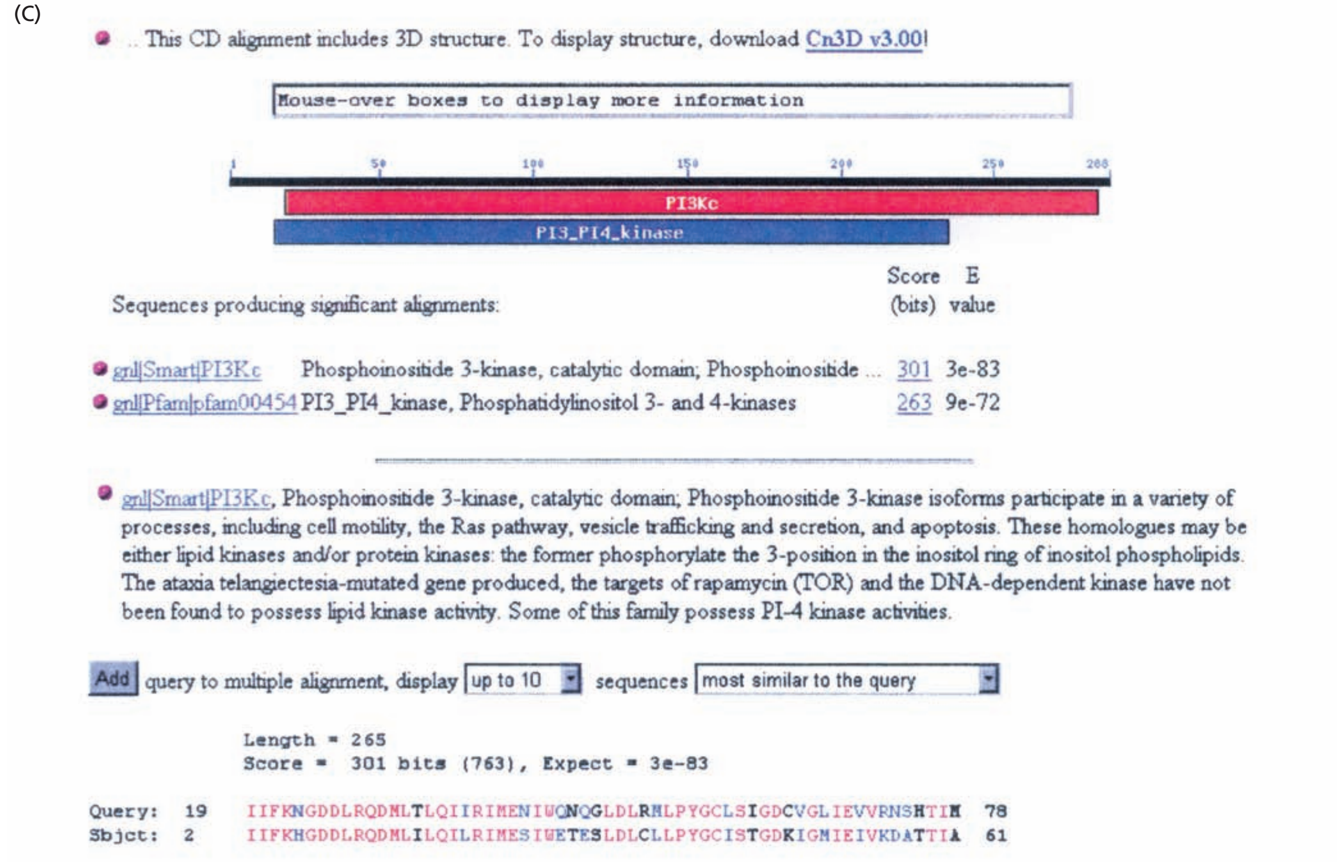
(B)



Distribution of 2 Blast Hits on the Query Sequence
Mouse-over to show defline and scores. Click to show alignments
Color Key for Alignment Scores
<40
40-50
50-80
80-200
>=200
tmpseq_1
0
50
100
150
200
250
300

```
dbj|BAB10275.1|   (AB008266) phosphatidylinositol 4-kinase [A...   111  3e-25
dbj|BAB11344.1|   (AB011477) AtRAD3 [Arabidopsis thaliana]          38  0.008
```

(C)



.. This CD alignment includes 3D structure. To display structure, download Cn3D v3.00!

Sequences producing significant alignments:		Score (bits)	E value
gnl|Smart|PI3Kc	Phosphoinositide 3-kinase, catalytic domain; Phosphoinositide ...	301	3e-83
gnl|Pfam|pfam00454	PI3_PI4_kinase, Phosphatidylinositol 3- and 4-kinases	263	9e-72

gnl|Smart|PI3Kc, Phosphoinositide 3-kinase, catalytic domain; Phosphoinositide 3-kinase isoforms participate in a variety of processes, including cell motility, the Ras pathway, vesicle trafficking and secretion, and apoptosis. These homologues may be either lipid kinases and/or protein kinases: the former phosphorylate the 3-position in the inositol ring of inositol phospholipids. The ataxia telangiectesia-mutated gene produced, the targets of rapamycin (TOR) and the DNA-dependent kinase have not been found to possess lipid kinase activity. Some of this family possess PI-4 kinase activities.

Add query to multiple alignment, display up to 10 sequences most similar to the query

```
          Length = 265
          Score =  301 bits (763), Expect = 3e-83

Query:  19    IIFKNGDDLRQDMLTLQIIRIMENIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIM  78
Sbjct:  2     IIFKHGDDLRQDMLILQILRIMESIWETESLDLCLLPYGCISTGDKIGMIEIVKDATTIA  61
```

To reduce the large number of hits one could search a subset of the data, for example only the newest sequences in the database (the “month” option; that is, those deposited in the last month) or a specific genome database. A search through sequences released in the past month detected eight sequences, all with significant scores, of which three had not been identified. A search through the *Drosophila* genome data yielded three hits, all of which are unknown. Taking one of the regions that matched our SH2 (a section of the *Drosophila* 3R chromosome arm) and searching with this sequence through SWISS-PROT yielded a highly significant hit to an SH2 domain of a rat protein. So we may have identified a previously unknown *Drosophila* sequence as containing an SH2 domain.

This example illustrates a search through the database with a family that is very well represented and shows the problems that can arise. We will now look at a family that is not so well represented—the PI3-kinases.

First the SWISS-PROT database was searched using the catalytic domain protein sequence from the PI3-kinase p110α using BLAST with the *E*-value cut-off set to 1. Thirty-two hits were found. In this list there are three near-identical isoforms of p110α which have an *E*-value of 0.0; that is, the chance of obtaining such a match with random sequence is taken to be zero. There is one match that is not significant: ribonucleoside-diphosphate reductase, with an *E*-value of 0.59 (see Figure 4.12A). From the assigned function this is clearly a different enzyme, but the enzymatic reactions of both this reductase and the kinases involve a nucleotide, which might have led to some small degree of similarity between the sequences. Any such speculation would need further and more thorough investigation. If we rerun the search with the *E*-value cut-off set to 0.01 (the advised setting) only the significant matches are retrieved.

Searching with BLAST through a subset of sequences such as those that have only been released in one month found two hits: one is a homolog of PI3-kinase, a PI4-kinase with a significant *E*-value, and the other is a segment of an *Arabidopsis thaliana* protein, atRad3, with an *E*-value score of borderline significance. From the length of the matched sequence illustrated in the search output (see Figure 4.12B) the segment seems far too short to be of interest; compare the length of the matched PI4-kinase, in magenta. For this reason the hit can now be reclassified as not significant.

Another useful option available within the BLAST search server is a concurrent search of the Conserved Domain Database (CDD) entries. Figure 4.12C shows the results of using this option on the PI3-kinase sequence. Proteins often contain several domains, and the program CD-Search can potentially identify domains present in a protein sequence. CDD contains domains derived mainly from the SMART and Pfam protein-family databases. To identify conserved domains in a protein sequence, the CD method uses the BLAST algorithm where the query sequence is matched with a PSSM designed from the conserved domain alignments. Matches are shown as either a pairwise alignment of the query sequence to a representative domain sequence or as a multiple alignment.

A FASTA search with the p110α sequence through SWISS-PROT with default settings (k-tuple = 2) yielded 36 hits, of which eight had a nonsignificant score (see Figure 4.13). Of these eight, ribonucleoside-diphosphate reductase was also found by BLAST. Although both FASTA and BLAST report an *E*-value, the actual values are different, which reflects subtle differences in the methods used. An SSEARCH search of the SWISS-PROT database with default settings found 29 significant hits. SSEARCH, a more rigorous method, found fewer hits than BLAST or FASTA.

Low-complexity regions can complicate homology searches

Among the many features that can complicate a sequence-similarity search is the occurrence of **low-complexity regions** in protein sequences. These are regions with a highly biased amino acid composition, often runs of prolines or acidic amino

```
the best scores are:                                                      E(86391)
SW:P11A BOVIN P32871 PHOSPHATIDYLINOSITOL 3-KINAS  (1068) 2228 493 1.2e-138
SW:P11A HUMAN P42336 PHOSPHATIDYLINOSITOL 3-KINAS  (1068) 2216 490 7.4e-138
SW:P11A MOUSE P42337 PHOSPHATIDYLINOSITOL 3-KINAS  (1068) 2204 488 4.5e-137
SW:P11B HUMAN P42338 PHOSPHATIDYLINOSITOL 3-KINAS  (1070) 1126 254 1.1e-66

                                   ↓ other sequences

SW:ESR1 YEAST P38111 ESR1 PROTEIN.                 (2368)  144  41     0.028
SW:PRA2 USTMA P31303 PHEROMONE RECEPTOR 2.         ( 346)  116  35      0.35
SW:TEL1 YEAST P38110 TELOMER LENGTH REGULATION PR  (2787)  127  37      0.42
SW:YA51 METJA Q58451 HYPOTHETICAL PROTEIN MJ1051.  ( 513)  112  34      0.91
SW:RIR1 MYCGE P47473 RIBONUCLEOSIDE-DIPHOSPHATE R  ( 721)  106  33         3
SW:YAY1 SCHPO Q10209 HYPOTHETICAL 44.8 KDA PROTEI  ( 392)   99  31       5.1
SW:PAFA CAVPO P70683 PLATELET-ACTIVATING FACTOR A  ( 436)   96  30       8.8
SW:KC47 ORYSA P29620 CDC2+/CDC28-RELATED PROTEIN   ( 424)   95  30       9.9
```

Figure 4.13
Output from a search of the SWISS-PROT protein sequence database using FASTA with PI3-kinase p110α as the query sequence. Thirty-six hits were obtained. Eight of these have a nonsignificant score (below the arrow). One of these, ribonucleoside-diphosphate reductase, was also found by BLAST. The *E*-values in FASTA are different from those in BLAST.

acids. In some cases, self-comparison dot-plots (see page 77) can identify low-complexity regions in a protein sequence. Alignments of such regions in different proteins can achieve high scores, but these can be misleading and can obscure the biologically significant hits. It is better to exclude low-complexity regions when constructing the alignment. By default, the BLAST program filters the query sequence for low-complexity regions. In the BLAST output file, an X marks regions that have been filtered out (using SEG for proteins and DUST for DNA) (see Box 5.2).

Figure 4.14A shows a self-comparison dot-plot of human prion protein precursor (PrP), an abnormal form of which is found in large amounts in the brains of people with neurodegenerative diseases such as Creutzfeldt–Jakob disease (CJD) and kuru (see Box 4.6). It has several low-complexity regions, which are seen as dark diagonal lines (apart from the main identity diagonal) and the ordered dark-shaded regions. Figure 4.14B shows a search for homologs of the human PrP. The extensive low-complexity regions have been filtered out in the query sequence (as indicated by the strings of Xs). A BLAST search of SWISS-PROT with human PrP with the low-complexity filter turned on gave approximately 40 hits, all prion proteins. One of

Box 4.6 Prions: Proteins that can exist in different conformations

Scrapie in sheep, bovine spongiform encephalopathy (BSE) in cattle, and Creutzfeldt–Jakob disease (CJD), fatal familial insomnia, and kuru in humans are rare, fatal, transmissible, neurodegenerative diseases known generally as the transmissible spongiform encephalopathies, after the characteristic damage they do to the brain. They can arise sporadically, or as a result of the inheritance of a faulty gene, or can be transmitted by ingestion of infected material. Kuru, which was relatively common in people in the Eastern Highlands of Papua New Guinea in the 1950s and 1960s, was found to be caused by the ritual custom of eating the brains of dead relatives, while a variant form of CJD (vCJD), which has appeared only recently, is thought to be caused by people having eaten BSE-infected meat products.

The causal agent in the spongiform encephalopathies is believed to be an infectious protein, a prion, rather than a DNA or RNA virus. Prions are normal proteins that have the property of being able to convert into an alternative stable conformation, which is associated with disease, although the mechanism by which prions cause cell death and neurodegeneration is not yet fully understood. The normal form of the prion protein (PrP^{c}) is a monomer with a structure consisting mainly of α-helices, and is mainly found at the cell surface, whereas the abnormal form (PrP^{Sc}), is mainly β-sheet and has a tendency to aggregate into clumps. PrP^{Sc} itself appears to be able to induce the conversion of PrP^{c} into PrP^{Sc}. The prion protein is an example of a metastable protein, where the same or similar sequences can exist in different stable structural forms.

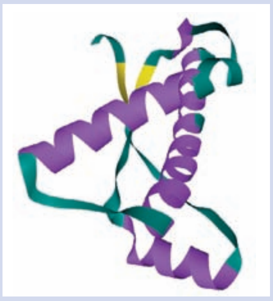
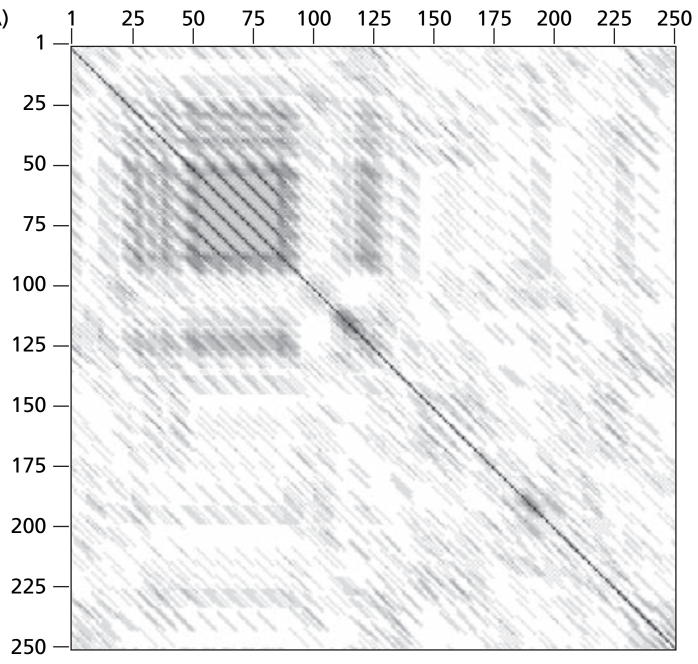


Figure B4.4
A ribbon representation of the normal form of prion protein, PrP^{c}.

(A)



(B)

```
>sp[P04156]PRIO HUMAN  MAJOR PRION PROTEIN PRECURSOR (PRP) (PRP27-30) (PRP33-35C) (ASCR)
          Length = 253

Score =  312 bits (792), Expect = 5e-85
Identities = 154/236 (65%), Positives = 154/236 (65%)

Query: 64   MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRYXXXXXXXXXXX 123
            MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRY
Sbjct: 1    MANLGCWMLVLFVATWSDLGLCKKRPKPGGWNTGGSRYPGQGSPGGNRYPPQGGGGWGQP 60

Query: 124  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTHSQWNKPSKPKTNMKHMXXXXXXXX 183
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTHSQWNKPSKPKTNMKHM
Sbjct: 1    HGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQGGGTHSQWNKPSKPKTNMKHMAGAAAAGA 120

Query: 184  XXXXXXXXXXXXXXXRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV 243
                           RPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV
Sbjct: 121  VVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDEYSNQNNFVHDCV 180

Query: 244  NITIKQHXXXXXXXXXXXXXXDVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS 299
            NITIKQH              DVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS
Sbjct: 181  NITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQRGSSMVLFS 236
```

Figure 4.14
Dealing with low-complexity regions of sequence. (A) The low-complexity regions are clearly visible on a self-comparison dot-plot of a human prion precursor protein (PrP). They are indicated by the black diagonal lines on either side of the identity diagonal and by the ordered dark-shaded regions. (B) Results of a database search with PrP from which sequences of low complexity have been filtered out by application of the program SEG, which marks them with Xs (top row of the alignment).

these is shown aligned with the query sequence in the figure. When the filter was turned off, the number of hits rose to 220, most of which were not homologous.

Sometimes one may wish to study low-complexity regions in particular. For example, in the case of the tubulin and actin gene clusters it is thought that amplification of the protein-coding genes may be related to these regions. There are options in BLAST that allow you to select these regions for study.

Different databases can be used to solve particular problems

To some extent, the choice of which database to search will depend on which databases are provided by the site that runs the search algorithms. Most sites contain a selection of the most popular databases, such as GenBank for DNA sequences, SWISS-PROT for annotated protein sequences, TrEMBL, a translated EMBL DNA-sequence database, and PDB, a database of protein structures with sequences (see Chapter 3). Some sites also provide access to expressed sequence tag (EST) databases, such as dbEST, and genome-sequence databases from some of the fully sequenced genomes

In general, a first pass should be run on a generic protein- or nucleic acid sequence database. You can also carry out a search on the PDB to see if your query sequence has a homolog with known structure. More specific searches can be performed to answer particular questions. For example, if it is suspected that the query sequence belongs to a family of immune-system proteins, the search could be carried out on the Kabat database, which contains sequences of immunological interest. If the sequence originates from a mouse, you may want to know if a homolog exists in the rat, *Drosophila*, or human genomes, and should therefore search the databases containing sequences from the appropriate species. You also need to check that you are searching a database that is up to date; sites such as those at NCBI and EBI are regularly updated.

If no match is found to the query sequence, it does not necessarily mean that there is no homolog in the databases, just that the similarity is too weak to be picked up by existing techniques. Techniques are continually being improved and the amount of sequence data continues to increase; you should therefore periodically resubmit your sequence.

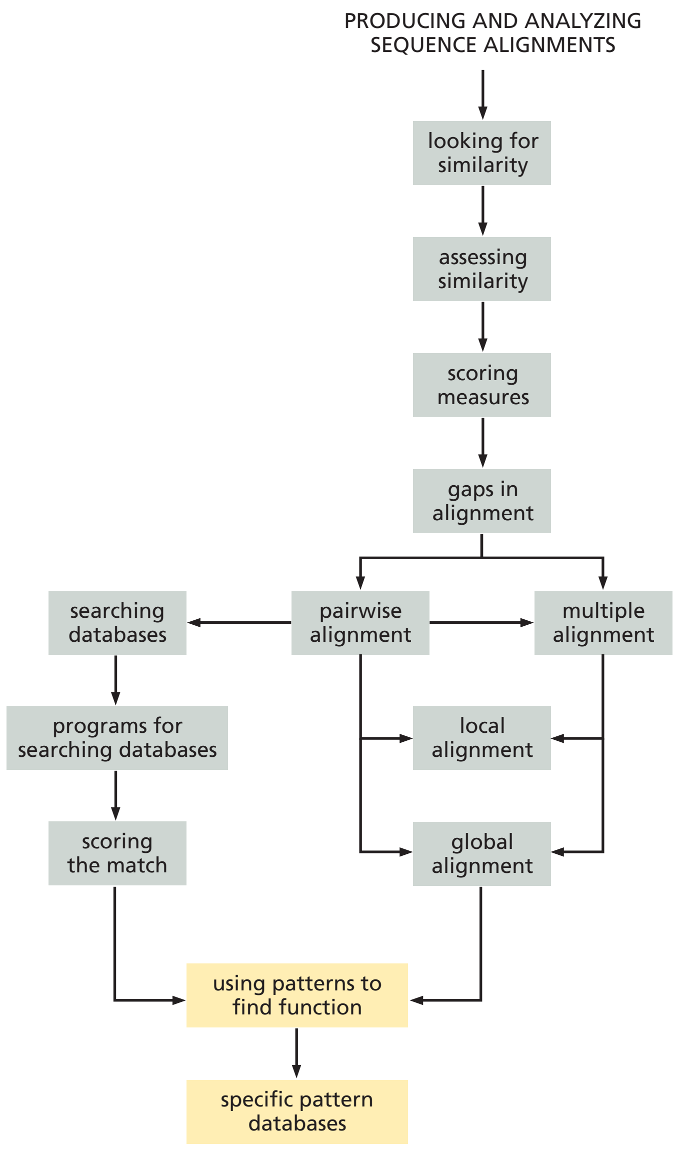
Many other sequence-related databases can usefully be searched and provide additional information. For example the Sequences Annotated by Structure (SAS) server is a collection of programs and data that can help identify a protein sequence by using structural features that are the result of sequence searches of annotated PDB sequences. Residues in the sequences of known structures are colored according to selected structural properties, such as residue similarity, and are displayed using a Web browser. SAS will perform a FASTA alignment of the query sequence against sequences in the PDB database and return a multiple alignment of all hits. Each of the hits is annotated with structural and functional features. That information can be used to annotate the unknown protein sequence. Further links are provided to the separate PDB files. Databases such as Clusters of Orthologous Groups (COGs) and UniGene can help in gene discovery, gene-mapping projects, and large-scale expression analysis. Sites such as Ensembl provide convenient access for gene searches in many different annotated eukaryotic genomes and useful associated information.

4.8 Protein Sequence Motifs or Patterns

If the similarity between an unknown sequence and a sequence of known function is limited to a few critical residues, then standard alignment searches using BLAST or FASTA against general sequence databases such as GenBank, dbEST, or SWISS-PROT will fail to pick up this relationship. What is required is a method of searching for the occurrence of short sequence patterns, or **motifs** (see Flow Diagram 4.4). A motif, in general, is any conserved element of a sequence alignment, whether composed of a short sequence of contiguous residues or a more distributed pattern. Functionally related sequences will share similar distribution patterns of critical functional residues that are not necessarily contiguous. For example, conserved amino acid residues comprising an enzyme's active site may be distant from each other in the protein sequence but will still occur in a recognizable pattern because of the constraints imposed by the requirement for them to come together in a particular spatial configuration to form the active site in the three-dimensional structure.

There are three different types of activity associated with pattern searching. A query sequence can be searched for patterns (from a patterns database) that could help suggest functional activity. A sequence database can be searched with a specific pattern, for example to determine how many gene products in a genome have a

Flow Diagram 4.4
The key concepts introduced in this and the following two sections are that sequence patterns can be very useful in identifying protein function and that special pattern databases and search programs have been designed to assist in identifying patterns in a query sequence.



specific function. Lastly, we may want to define a new pattern specific for a selected set of sequences.

In searches with new sequences, the whole database is searched and expert knowledge, such as the known function of a homologous protein, is then used to extrapolate the function of the new sequence. In contrast, when new patterns and motifs are used to search a database, the expert knowledge is needed right at the beginning to construct the motifs that are intended to identify the specific function or any other physicochemical property.

Pattern and motif searches are mostly used with protein sequences rather than nucleotide sequences, as the greater number of different amino acids makes protein patterns more efficient in discriminating truly significant hits. In addition, many of the patterns identify biological function, which is mediated at the protein level. There are, however, particular problems in DNA- and genome-sequence analysis where searching for motifs is useful (see Chapters 9 and 10).

Creation of pattern databases requires expert knowledge

The construction of patterns or motifs is of prime importance in characterizing a protein family, and much time and energy has gone into constructing pattern and motif databases that one can search with an unknown sequence. One of the most important steps is careful selection of the sequences used to define the pattern. If these do not all share the same biological properties for which you wish to define a pattern, you will almost certainly encounter problems later. Thus, experimental evidence of function or clear homology is necessary for all the sequences used.

Some pattern databases have been constructed by hand by inspection of large amounts of data. This is very time consuming, but necessary, as the task of extracting a pattern is a complex one, depending on expert knowledge of the structures and/or functions of the sequences involved. For example, analysis of the X-ray structure of a protein can delineate the functional residues involved in an enzyme active site or a regulatory binding site, and an initial pattern can be generated. This pattern can then be refined by multiple alignment of sequences of other members of the same structural or functional protein family. If no structural data are available, multiple sequence alignment of short regions of similarity, assessed alone or in conjunction with experimental data on biochemical and cellular function, can be used to extract a pattern.

The simplest method of constructing a pattern or motif is the consensus method, in which the most similar regions in a global multiple sequence alignment are used to construct a pattern. Those positions in the alignment that are all occupied by the same residue (or a limited subset of residues) are used to define the pattern at these positions, by specifying just the allowed residues at each position. More sophisticated patterns can be generated using scoring tables to assess the similarity of the matched amino acids. In this case, instead of just defining the pattern as requiring, for example, a glutamic or aspartic acid at a given position, different residues at this position have associated scores.

The BLOCKS database contains automatically compiled short blocks of conserved multiply aligned protein sequences

Sequence motifs can also be defined automatically from the multiple alignment of a specified set of sequences. Blocks are multiple alignments of ungapped segments of protein sequence corresponding to the most highly conserved regions of the proteins. The blocks for the BLOCKS database are compiled automatically by looking for the most highly conserved regions in groups of proteins documented in the PROSITE database. The blocks are then calibrated against the SWISS-PROT database to obtain a measure of the chance distribution of matches. The calibrated blocks make up the BLOCKS database, against which a new sequence can be searched. Both protein and DNA sequences (automatically translated into a protein sequence) can be submitted to search the BLOCKS database. The BLOCKS Web site, in addition to providing the BLOCKS database, will align your set of sequences and automatically design a block with which you can search SWISS-PROT. Generating blocks from your set of sequences and searching with them can find sequences that have very weak sequence similarity but are, nevertheless, functionally related. Generating patterns and/or blocks is also a useful method to search for hints to function within an unknown sequence.

Another program that will analyze a set of sequences for similarities and produce a motif for each pattern it discovers is MEME (Multiple Expectation maximization for Motif Elicitation). MEME characterizes motifs as position-dependent probability matrices. The probability of each possible letter occurring at each possible position in the pattern is given in the matrices. Single MEME motifs do not contain gaps and therefore patterns with gaps will be divided by the program into two or more separate motifs.

The program takes the group of DNA or protein sequences provided by the user and creates a number of motifs. The user can choose the number of motifs that MEME will produce. MEME uses statistical techniques to choose the best width,

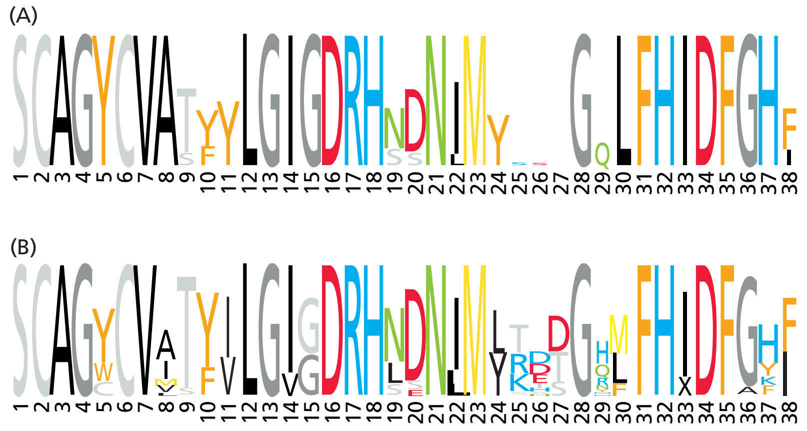


Figure 4.15
Residues that contribute to one of the blocks returned by the BLOCKS database after submission of the PI3-kinase p100α sequence. (A) A block for four homologous sequences, and (B) for 31 homologous sequences. These representations are called logos, and are computed using a position-specific scoring matrix. This block contains the active-site amino acids and the DFG kinase motif. The size of the letters indicates the level of conservation and the colors indicate physicochemical properties of the residues: acidic, red; basic, blue; small and polar, white; asparagine and glutamine, green; sulfur-containing amino acids, yellow; hydrophobic, black; proline, purple; glycine, gray; aromatic, orange.

number of occurrences, and description for each motif (see Section 6.6). The motifs found by MEME can also be given in BLOCKS format to allow further analysis as described below.

If we submit the PI3-kinase p110α sequence and four homologs to the BLOCKS program it creates six separate blocks of high similarity. Figure 4.15A illustrates the block that contains residues important in PI3-kinase catalysis and ATP binding. Letters that are large and occupy the whole position represent identities in the multiple sequence alignment (see Section 6.1 for further details on this sequence representation). The SCAGY, DRH, and DFG motifs that are the markers for PI3-kinases are identified by the BLOCKS program and form part of the conserved regions. If more distant sequences are submitted, fewer residues will form the highly conserved regions with the largest residues, as shown in Figure 4.15B where 31 sequences were aligned.

The six blocks can now be submitted to a database search using the program LAMA (Local Alignment of Multiple Alignments), which compares multiple alignments of protein sequences with each other. The program searches the BLOCKS database, the PRINTS database (see below), or your own target data, to see if similar blocks or patterns already exist. This is a sensitive search technique, detecting weak sequence relationships between protein families. The LAMA search of the BLOCKS database has identified seven blocks, of which three are significant: these are PI3/4-kinase signatures.

The blocks can also be submitted to a MAST (Motif Alignment and Search Tool) search of one of the online nucleotide- or protein-sequence databases. MAST is a program that searches for motifs—highly conserved regions or blocks. Here we submit the six PI3-kinase blocks to a MAST search of SWISS-PROT (to use this program you need to have an e-mail address to receive the results). Twenty-four sequences were found with significant scores, with the PI3-kinase sequences all scoring more highly than the PI4-kinases.

The same set of PI3-kinase p110α sequences was submitted to the MEME motif-generating program. The number of motifs to be generated was set to six (the same number found by BLOCKS). The top-scoring motif (see Figure 4.16A) describes similar residues as the BLOCK motif described above (see Figure 4.15). The MEME motif starts at the end of the SCAGY motif (Y), contains the active site D, N, and DFG residues, and extends a bit further than the BLOCKS motif. A nice feature of the MEME program is that it generates a figure containing a summary of all motifs (see Figure 4.16B). This illustrates where the motifs are located with respect to each other within all the sequences (only three are shown for clarity). Submitting the MEME motifs to a search through SWISS-PROT finds 21 matches. Matches with significant

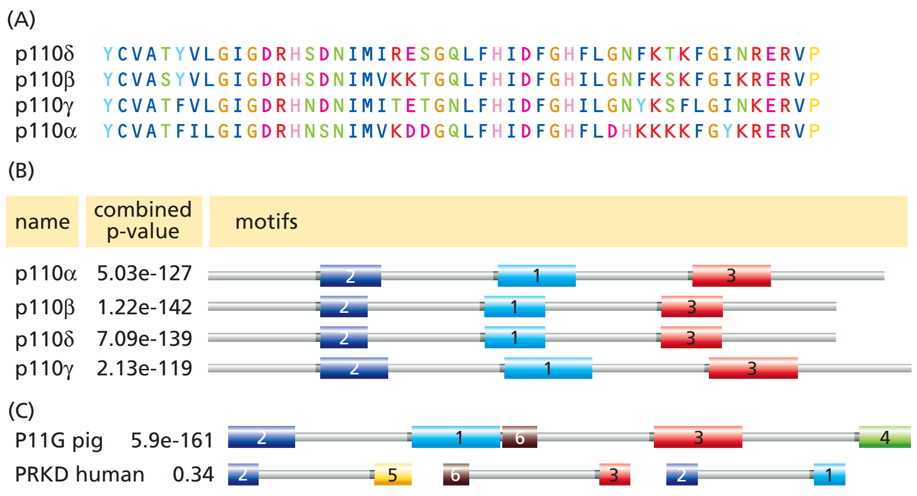

(A)
p110δ YCVATYVLGIGDRHSDNIMIRESGQLFHIDFGHFLGNFKTKFGINRERVP
p110β YCVASYVLGIGDRHSDNIMVKKTGQLFHIDFGHILGNFKSKFGIKRERVP
p110γ YCVATFVLGIGDRHNDNIMITETGNLFHIDFGHILGNYKSFLGINKERVP
p110α YCVATFILGIGDRHNSNIMVKDDGQLFHIDFGHFLDHKKKKFGYKRERVP
(B)
name
combined p-value
motifs
p110α 5.03e-127
p110β 1.22e-142
p110δ 7.09e-139
p110γ 2.13e-119
(C)
P11G pig 5.9e-161
PRKD human 0.34
2
1
3
6
4
5

Figure 4.16
MEME generates motifs. (A) The top-scoring patterns are color coded according to the physicochemical properties of the amino acid side chains: dark blue is used for the residues ACFILVM; green for NQST; magenta to indicate DE; red color is used for residues KR; pink for H; orange for G; yellow for P; and light blue shows Y. (B) Summary motif information where each motif is represented by a colored block. The number in a block gives the scored position of the motif. The light blue block, number 1, contains the motif described in (A). The combined p-value of a sequence measures the strength of the match of the sequence to all the motifs. (C) Illustration of how lower-scoring motif matches can still find interesting and true homologs. The distances between the motif blocks are not representative.

scores are all PI3-kinases and PI4-kinases. The significant scores usually match most, if not all, of the motifs submitted. However, lower scores can be informative as well; distant relationships can be found if only a subset of the motifs matches.

For example, a search using the motifs of PI3-kinases finds the DNA-dependent protein kinase catalytic subunit (PRKD), which has shared kinase activity with the PI3-kinases. Four of the six motifs are matched (Figure 4.16C) and some are repeated within the DNA-dependent kinase. Simple sequence-alignment searches through the sequence databases may not pick up this type of relationship, although in this case a blastp search with p110α through SWISS-PROT matches PRKD with a score that would be considered significant, and a search with FASTA gives a borderline score.

4.9 Searching Using Motifs and Patterns

The PROSITE database can be searched for protein motifs and patterns

The PROSITE database is a compilation of motifs and patterns extracted from protein sequences and compiled by inspection of protein families. This database can be searched with an unknown protein sequence to obtain a list of hits to possible patterns or protein signatures. It is also possible to create your own pattern in the manner of a PROSITE pattern to search another sequence database. The syntax of a PROSITE pattern consists of amino acid residues interspersed with characters that denote the rules for the pattern, such as distances between residues, and so on (see Table 4.2).

For example, a pattern for the kinase active site, starting from the conserved DRH and making use of the very conserved DFG region, can be created manually from the 31 sequences used in the BLOCKS example.

D-R-[KH]-X-[DE]-N-[IL]-[MILV](2)-X(3)-G-X-[LI]-X(3)-D-F-G

Inputting this pattern into the ScanProsite Web page and running it against the SWISS-PROT database of protein sequences obtained 92 hits; all were PI3 (PI4)-kinases or protein kinases. If, on the other hand, we submit the catalytic domain of the PI3-kinase p110α sequence to be scanned through the PROSITE database to see if there are any existing patterns, the search retrieves two signature

Code	Explanation	Example explanation	Examples
One-letter codes	Standard amino acids		G-L-L-M-S-A-D-F-F-F
-	All positions must be separated by -		G-L-L-M-S-A-D-F-F-F
X	Any amino acid	Any amino acid allowed in second place	G-X-L-M-S-A-D-F-F-F
[]	Two or more possible amino acids	L or I allowed in second place	G-[LI]-L-M-S-A-D-F-F-F
{}	Disallowed amino acids	R or K not allowed in sixth place	G-[LI]-L-M-S-A-{RK}-F-F-F
(*n*) *n* = number	Repetition can be indicated by a number in brackets after the amino acid	F repeated three times	G-[LI]-L-M-S-A-{RK}-F(3)
(*n*,*m*)	A range: only allowed with X	One to three positions with any amino acids (X) allowed	G-[LI]-L-M-S-A-{RK}-X(1,3)
<	Pattern at amino-terminal of sequence		
>	Pattern at carboxy-terminal of sequence		

Table 4.2
The various codes used to define a PROSITE protein pattern for a search through a sequence database.

sequences for PI3- and PI4-kinases. These give us a much more specific search signature for the PI3/4-kinase family, but do not tell us, for example, that this kinase family is also similar to the protein kinase family. The patterns for the signatures are:

(1) [LIVMFAC]-K-X(1,3)-[DEA]-[DE]-[LIVMC]-R-Q-[DE]-X(4)-Q

(2) [GS]-X-[AV]-X(3)-[LIVM]-X(2)-[FYH]-[LIVM](2)-X-[LIVMF]-X- $\underline{\text{D-R-H-X(2)-N}}$

The second signature pattern contains at its right-hand end (underlined) the start of the kinase pattern we created above to scan SWISS-PROT. Pattern 1 and the rest of pattern 2 contain conserved regions within the PI3/4-kinase families that are amino-terminal to our created pattern.

The pattern-based program PHI-BLAST searches for both homology and matching motifs

The BLAST set of programs also has a version that uses motifs in the query sequence as a pattern. PHI-BLAST (Pattern Hit Initiated BLAST) uses the PROSITE pattern syntax shown in Table 4.2 to describe the query protein motif. The specified pattern need not be in the PROSITE database and can be user generated. PHI-BLAST looks for sequences that not only contain the query-specific pattern but are also homologous to the query sequence near the designated pattern. Because PHI-BLAST uses homology as well as motif matching, it generally filters out those sequences where the pattern may have occurred at random. On the NCBI Web server, PHI-BLAST is integrated with PSI-BLAST, enabling one or more subsequent PSI-BLAST database searches using the PHI-BLAST results.

Patterns can be generated from multiple sequences using PRATT

The program PRATT can be used to extract patterns conserved in sets of unaligned protein sequences. The patterns are described using the PROSITE syntax. The power of PRATT is that it requires no knowledge of possible existing patterns in a set of sequences. Figure 4.17 shows the results for the PI3-kinase p110α family. The pattern illustrated in the figure contains the DFG motif which is highlighted in the second PROSITE pattern.

```
PRATT output :

   p110-a:   qlfhi DFGHFLDhkKkkFGykRERVPFVLTqDFLiViskGaQE ctktr
   p110-b:   qlfhi DFGHILGnfKskFGikRERVPFILTyDFIhViqqGkTG ntekf
   p110-d:   qlfhi DFGHFLGnfKtkFGinRERVPFILTyDFVhViqqGkTN nsekf
   p110-g:   nlfhi DFGHILGnyKsfLGinKERVPFVLTpDFLfVm--GtSG kktsp

D-F-G-H-[FI]-L-[DG]-x(2)-K-x(2)-[FL]-G-x(2)-[KR]-E-R-V-P-F-[IV]-L-T-x-D-F-[ILV]-x-V-x(1,3)-G-x-[QST]-[EGN]
```

Figure 4.17
PRATT pattern search. Sequences of the four types of PI3-kinase sequences (α, β, γ, and δ) have been submitted to PRATT to automatically create PROSITE-like patterns from a multiple alignment. This figure shows the alignment block and a PRATT-generated PROSITE pattern of the region that contains the DFG motif (shaded in green and boxed in red).

The PRINTS database consists of fingerprints representing sets of conserved motifs that describe a protein family

The PRINTS database is a next-generation pattern database consisting of fingerprints representing sets of conserved motifs that describe a protein family. The fingerprint is used to predict the occurrence of similar motifs, either in an individual sequence or in a database. The fingerprints were refined by iterative scanning of the OWL composite sequence database: a composite, nonredundant database assembled from sources including SWISS-PROT, sequences extracted from NBRF/PIR protein sequence database, translated sequences from GenBank, and the PDB structural database. A composite, or multiple-motif, fingerprint contains a number of aligned motifs taken from different parts of a multiple alignment. True family members are then easy to identify by virtue of possessing all the elements of the fingerprint; possession of only part of the fingerprint may identify subfamily members. A search of the PRINTS database with our PI3-kinase sequence found no statistically significant results.

The Pfam database defines profiles of protein families

Pfam is a collection of protein families described in a more complex way than is allowed by PROSITE's pattern syntax. It contains a large collection of multiple sequence alignments of protein domains or conserved regions. Hidden Markov model (HMM)-based profiles (see Section 6.2) are used to represent these Pfam families and to construct their multiple alignments. Searching the Pfam database involves scanning the query sequence against each of these HMM profiles. Using these methods, a new protein can often be assigned to a protein family even if the sequence homology is weak. Pfam includes a high proportion of extracellular protein domains. In contrast, the PROSITE collection emphasizes domains in intracellular proteins—proteins involved in signal transduction, DNA repair, cell-cycle regulation, and apoptosis—although there is some overlap. A search of the Pfam database allows you to look at multiple alignments of the matched family, view protein domain organization (see Figure 4.6), follow links to other databases by clicking on the boxed areas, and view known protein structures.

A search in Pfam using the sequence of the PI3-kinase p110α catalytic domain will find the PI3/4-kinase family. You can then retrieve the multiple alignment that has been used to define the family and obtain a diagram of the domain structure of the whole family. (Clicking on a domain will call up another Web page of detailed information.) Figure 4.6 shows a snapshot of the interactive diagram; the yellow boxed area is the catalytic domain upon which the search was based.

Only the most commonly used pattern and profile databases have been described here; links to others are given on the Publisher's Web page.

4.10 Patterns and Protein Function

Searches can be made for particular functional sites in proteins

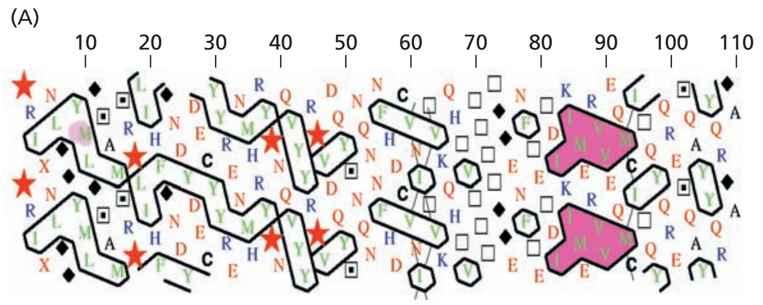
There are techniques other than simple sequence comparison that can identify functional sites in protein sequences. In contrast to the methods discussed above, which tend to cover a very wide range of biological functions, these techniques are usually made available in programs which predict only one specific functional site.

For example, signals from the environment are transmitted to the inside of the cell where they induce biochemical reaction cascades called signal transduction pathways. These result in responses such as cell division, proliferation, and differentiation and, if not properly regulated, cancer. During signal transduction, cellular components are chemically modified, often transiently. One of the key modifications used in these pathways is the addition and removal of phosphate groups. Sites susceptible to such modification can be predicted by the NetPhos server, which uses neural network methods to predict serine, threonine, and tyrosine phosphorylation sites in eukaryotic proteins. PROSITE also has patterns describing sites for phosphorylation and other posttranslational modifications, but specific programs such as NetPhos are expected to be more accurate.

Sequence comparison is not the only way of analyzing protein sequences

Apart from sequence comparison and alignment methods, there are various other ways of analyzing protein sequences to detect possible functional features. These techniques can be useful either when you have found a homolog in a database search and want to analyze it further, or when you have failed to find any similar sequence homolog and have no other avenue open. The physicochemical properties of amino acids, such as polarity, can be useful indicators of structural and functional features (see Chapter 2). There are programs available on the Web that plot hydrophobicity profiles, the percentage of residues predicted to be buried, some secondary-structure prediction (see Chapters 11 and 12), and percentage accessibility. ProtScale is one easy-to-use Web site that allows many of the above protein properties to be plotted.

Hydrophobic cluster analysis (HCA) is a protein-sequence comparison method based on α-helical representations of the sequences, where the size, shape, and orientation of the clusters of hydrophobic residues are compared. Hydrophobic cluster analysis can be useful for comparing possible functions of proteins of very low sequence similarity. It can also be used to align protein sequences. The patterns generated by HCA via the online tool drawhca can be compared with any other sequences one is interested in. It has been suggested that the effectiveness of HCA



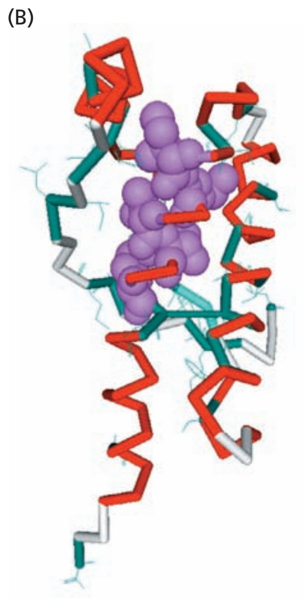


Figure 4.18
(A) Hydrophobic cluster analysis (HCA) of the prion protein using drawhca. Hydrophobic residues are in green, acidic in red, and basic in blue. A star indicates proline, a diamond glycine, an open box threonine, and a box with a dot serine. The same types of residues tend to cluster together, forming hydrophobic or charged patches. One such patch is highlighted in magenta. (B) X-ray structure of the same protein, with the same residues highlighted in magenta. As shown by the X-ray structure, the patch found by HCA forms a hydrophobic core in the interior of the protein.

Box 4.7 Protein localization signals

Proteins are all synthesized on ribosomes in the cytosol but, in eukaryotic cells in particular, have numerous final destinations: the cell membrane, particular organelles, or secretion from the cell. Intrinsic localization signals in the protein itself help to direct it to its destination and these can often be detected by their sequence characteristics. Proteins sorted through the endoplasmic reticulum (ER) for secretion or delivery to the cell membrane and some other organelles usually have a characteristic signal sequence at the amino-terminal end. This interacts with transport machinery in the ER membrane, which delivers the protein into the ER membrane or into the lumen. The signal sequence is often subsequently removed. Signal sequences are characterized by an amino-terminal basic region and a central hydrophobic region, and these features are used to predict their presence.

for comparison originates from its ability to focus on the residues forming the hydrophobic cores of globular proteins. Figure 4.18 shows the prion protein patterns that were generated using the program drawhca.

Information about the possible location of proteins in the cell can sometimes be obtained by sequence analysis. Membrane proteins and proteins destined for organelles such as the endoplasmic reticulum and nucleus contain intrinsic sequence motifs that are involved in their localization. Most secreted proteins, for example and other proteins that enter the endoplasmic reticulum protein-sorting pathway, contain sequences known as signal sequences when they are newly synthesized (see Box 4.7). The PSORT group of programs predicts the presence of signal sequences by looking for a basic region at the amino-terminal end of the protein sequence followed by a hydrophobic region. A score is calculated on the basis of the length of the hydrophobic region, its peak value, and the net charge of the amino-terminal region. A large positive score means that there is a high possibility that the protein contains a signal sequence. More methods of analyzing protein sequences to deduce structure and function are described in Chapters 11 to 14.

Summary

The comparison of different DNA or protein sequences to detect sequence similarity and evolutionary homology is carried out by a process known as sequence alignment. This involves lining up two or more sequences in such a way that the similarities between them are optimized, and then measuring the degree of matching. Alignment is used to find known genes or proteins with sequence similarity to a novel uncharacterized sequence, and forms the basis of programs such as BLAST and FASTA that are used to search sequence databases. Similarities in sequence can help make predictions about a protein's structure and function. Sequences of proteins or DNAs from different organisms can be compared to construct phylogenetic trees, which trace the evolutionary relationships between species or within a family of proteins.

The degree of matching in an alignment is measured by giving the alignment a numerical score, which can be arrived at in several different ways. The simplest scoring method is percentage identity, which counts only the number of matched identical residues, but this relatively crude score will not pick up sequences that are only distantly related. Other scoring methods for protein sequences take into account the likelihood that a given type of amino acid will be substituted for another during evolution, and these methods give pairs of aligned amino acids numerical scores which are summed to obtain a score for the alignment. The probabilities are obtained from reference substitution matrices, which have been compiled from the analysis of alignments of known homologous proteins. Because insertions or deletions often occur as two sequences diverge during evolution, gaps must usually be inserted in either sequence to maximize matching, and scoring schemes exact a penalty for each gap. As there are 20 amino acids, compared to only four different nucleotides, it is easier to detect homology in alignments of protein sequences than in nucleic acid sequences since chance matches are less likely.

There are several different types of alignment. Global alignments estimate the similarity over the whole sequence, whereas local alignments look for short regions of similarity. Local alignments are particularly useful when comparing multi-domain proteins, which may have only one domain in common. Multiple alignments compare a set of similar sequences simultaneously and are more accurate and more powerful than pairwise alignments in detecting proteins with only distant homology to each other.

Algorithms that automate the alignment and scoring process have been devised and are incorporated into various programs. Once an alignment has been scored, the significance of the score has to be tested to determine the likelihood of its arising by chance. Factors such as the length of the alignment and the total number of sequences in the database are taken into account. In database search programs such as BLAST and FASTA, potential matches are evaluated automatically and given a significance score, the *E*-value.

Databases may also be searched to find proteins of similar structure or function by looking for conserved short sequence motifs or discontinuous patterns of residues. These are likely to relate to a functional feature, such as an active site, a binding site, or to structural features. When sufficient members of a protein family have been sequenced, a characteristic profile of the family, summarizing the most typical sequence features, can be derived and can be used to search for additional family members. Database searches can also be widened to include structural information, where available. This is useful for finding homologs which have diverged so much in sequence that their sequence similarity can no longer be detected, but which retain the same overall structure.

Further Reading

4.1 Principles of Sequence Alignment

Bignall G, Micklem G, Stratton MR et al. (1997) The BRC repeats are conserved in mammalian BRCA2 proteins. *Hum. Mol. Genet.* 6, 53–58.

Brenner SE, Chothia C & Hubbard TJP (1998) Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proc. Natl Acad. Sci. USA* 95, 6073–6078.

Durbin R, Eddy S, Krogh A & Mitchison G (1998) Biological Sequence Analysis. Cambridge: Cambridge University Press.

Higgins D & Taylor W (eds) (2000) Bioinformatics. Sequence, Structure and Databanks, chapters 3–5, 7. Oxford: Oxford University Press.

Shivji MKK, Davies OR, Savill JM et al. (2006) A region of human BRCA2 containing multiple BRC repeats promotes RAD51-mediated strand exchange. *Nucleic Acids Res.* 34, 4000–4011.

4.2 Scoring Alignments

Twilight zone and midnight zone

Doolittle RF (1986) Of URFs and ORFs: A Primer on How to Analyze Derived Amino Acid Sequences. Mill Valley, CA: University Science Books.

Doolittle RF (1994) Convergent evolution: the need to be explicit. *Trends Biochem. Sci.* 19, 15–18.

Rost B (1999) Twilight zone of protein sequence alignments. *Protein Eng.* 12, 85–94.

4.3 Substitution Matrices

Dayhoff MO, Schwartz RM & Orcutt BC (1978) A model of evolutionary change in proteins. In Atlas of Protein Sequence and Structure (MO Dayhoff ed.), vol. 5, suppl. 3, pp. 345–352. Washington, DC: National Biomedical Research Foundation.

Henikoff S & Henikoff JG (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA* 89, 10915–10919.

4.4 Inserting Gaps

Goonesekere NCW & Lee B (2004) Frequency of gaps observed in a structurally aligned protein pair database suggests a simple gap penalty function. *Nucleic Acids Res.* 32, 2838–2843.

4.5 Types of Alignment

Gotoh O (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162, 705–708.

Needleman SB & Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48, 443–453.

Smith TF & Waterman MS (1981) Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195–197.

ClustalW

Higgins DW, Thompson JD & Gibson TJ (1996) Using CLUSTAL for multiple sequence alignments. *Methods Enzymol.* 266, 383–402.

DIALIGN

Morgenstern B (1999) DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics* 15, 211–218.

Morgenstern B, Frech K, Dress A & Werner T (1998) DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics* 14, 290–294.

4.6 Searching Databases

BLAST

Altschul SF, Gish W, Miller W et al. (1990) Basic Local Alignment Search Tool. *J. Mol. Biol.* 215, 403–410.

FASTA

Pearson WR & Lipman DJ (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* 85, 2444–2448.

PSI-BLAST

Altschul SF, Madden TL, Schäffer AA et al. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25, 3389–3402.

4.8 Protein Sequence Motifs or Patterns

MEME

Bailey TL & Elkan C (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Mach. Learn.* 21, 51–80.

4.9 Searching Using Motifs and Patterns

MOTIF

Smith HO, Annau TM & Chandrasegaran S (1990) Finding sequence motifs in groups of functionally related proteins. *Proc. Natl Acad. Sci. USA* 87, 826–830.

PRATT

Jonassen I (1997) Efficient discovery of conserved patterns using a pattern graph. *Comput. Appl. Biosci.* 13, 509–522.

Jonassen I, Collins JF & Higgins DG (1995) Finding flexible patterns in unaligned protein sequences. *Protein Sci.* 4, 1587–1595.

4.10 Patterns and Protein Function

HCA

Lemesle-Varloot L, Henrissat B, Gaboriaud C et al. (1990) Hydrophobic cluster analysis: procedures to derive structural and functional information from 2-D-representation of protein sequences. *Biochimie* 72, 555–574.

This page is intentionally left blank.

PAIRWISE SEQUENCE ALIGNMENT AND DATABASE SEARCHING

THEORY CHAPTER

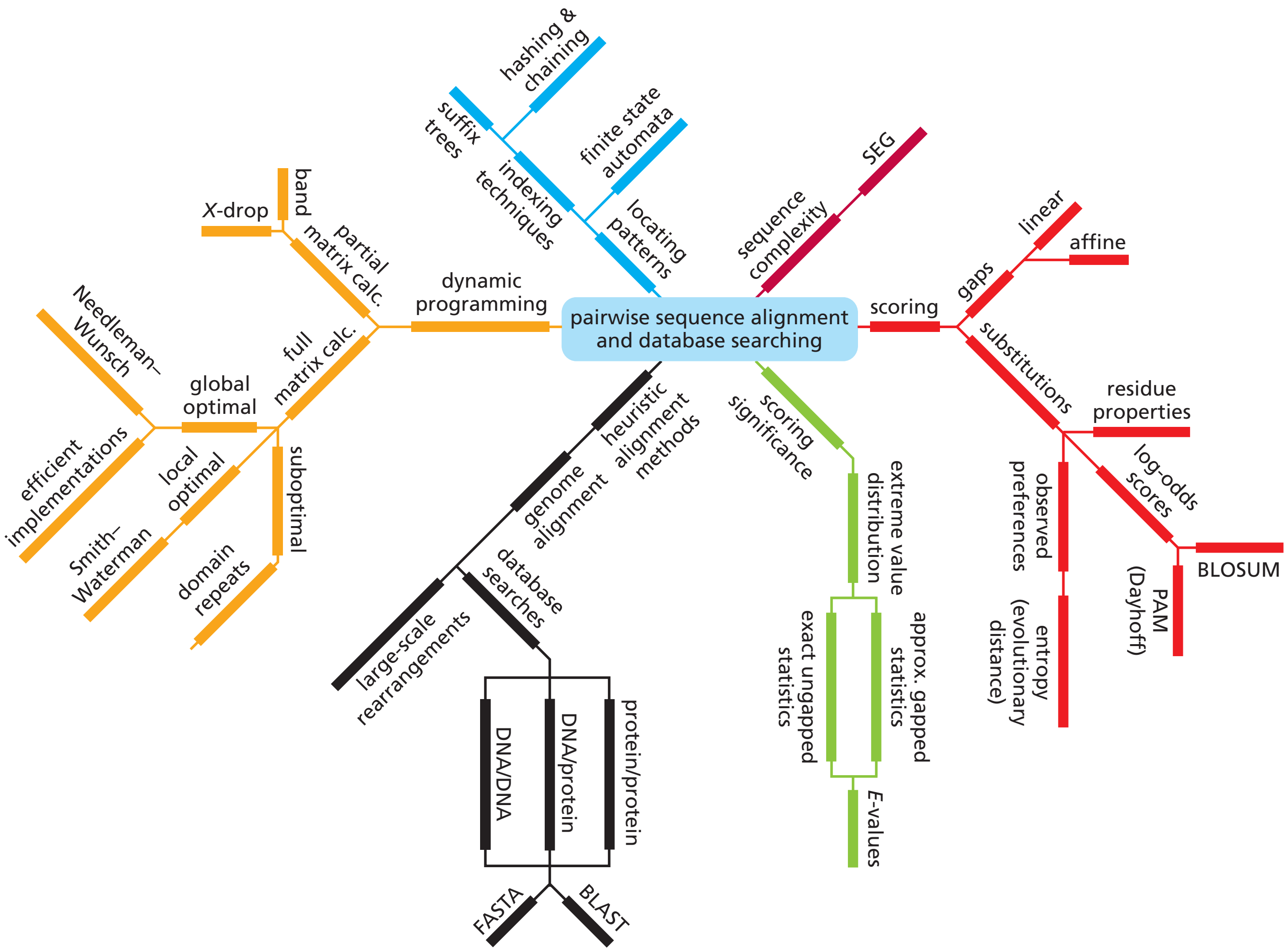
When you have read Chapter 5, you should be able to:

- Compare and contrast different scoring schemes.
- Summarize the techniques for obtaining the best-scoring alignments of a given type.
- Describe ways to reduce the computational resources required.
- Speed up database searches using index techniques.
- Evaluate approximations used in common database search programs.
- Summarize techniques for aligning DNA and protein sequences together.
- Identify sequences of low complexity.
- Identify significant alignments on the basis of their score.
- Summarize the techniques for alignments involving complete genome sequences.

The identification of homologous sequences and their optimal alignment is one of the most fundamental tasks in bioinformatics. As will be seen in many other chapters of this book, there are very few topics in bioinformatics which do not at some stage involve these techniques. A large part of Chapter 4 was devoted to an introduction to some practical aspects of sequence comparison and alignment. In this chapter we will focus in considerable detail on these methods and the science that lies behind them, but will restrict our attention solely to methods of aligning two sequences. The problems of obtaining multiple alignments and profiles and of identifying patterns will be discussed in Chapter 6.

Given two sequences, and allowing gaps to be inserted as was described in Section 4.4, it is possible to construct a very large number of alignments. Of these, there will be an optimal alignment, which in the ideal case perfectly identifies the true equivalences between the sequences. However, there will be many alternative alignments with varying degrees of error that could potentially be seriously misleading. Furthermore, the fact that an alignment can be constructed for any two sequences, even ones with no meaningful equivalences, has the potential to be even more misleading. Therefore, all useful methods of sequence alignment must not only generate alignments but also be able to compare them in a meaningful way and to provide an assessment of their significance.

Both the comparison of alignments and the assessment of their significance require a method of scoring. As discussed in Chapter 4, by considering the evolutionary processes that are responsible for sequence divergence it is possible to find ways of including their salient features in an alignment scoring system. If the scoring



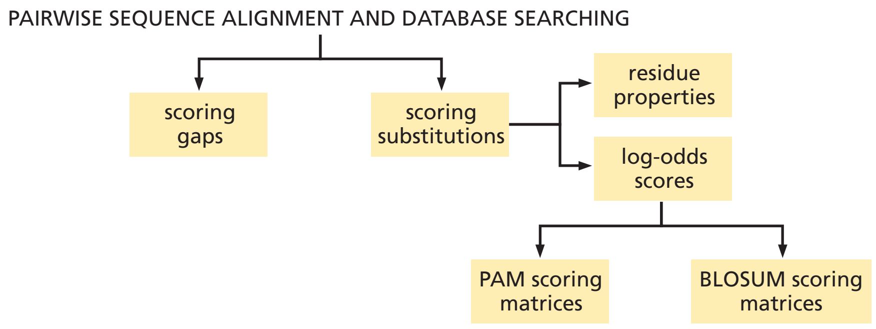
Mind Map 5.1
The theory of pairwise alignment and searching the database depicted in a mind map.

scheme is accurate and appropriate, all meaningful alignments should have optimal scores, i.e., they will have a better score when compared to any alternative. In Section 5.1 some of the best scoring schemes will be presented.

The number of alternative alignments is so great, however, that efficient methods are required to determine those with optimal scores. Fortunately, algorithms have been derived that can be guaranteed to identify the optimal alignment between two sequences for a given scoring scheme. These methods are described in detail in Section 5.2, although it should be noted that the emphasis is mostly on the scientific rather than the computer science aspects. As long as only single proteins or genes, or small segments of genomes, are aligned, these methods can be applied with ease on today's computers. When searching for alignments of a query sequence with a whole database of sequences it is usual practice to use more approximate methods that speed up the search. These are described in Section 5.3.

Finding the best-scoring alignment between two sequences does not guarantee the alignment has any scientific validity. Ways must be found to discriminate between fortuitously good alignments and those due to a real evolutionary relationship. Section 5.4 presents some of the concepts behind the theory of assessing the statistical significance of alignment scores.

The large number of complete genome sequences has led to increased interest in aligning very long sequences such as whole genomes and chromosomes. As described in Section 5.5, these applications require a number of approximations and techniques to increase the speed and reduce the storage requirements. In



Flow Diagram 5.1

The key concept introduced in this section is that if alignments of two sequences are assigned a quantitative score based on evolutionary principles then meaningful comparisons can be made. Several alternative approaches have been suggested, resulting in a number of different scoring schemes including those which account for insertions and deletions.

addition, the presence of large-scale rearrangements in these sequences has required the development of new algorithms.

5.1 Substitution Matrices and Scoring

As discussed in Chapter 4, the aim of an alignment score is to provide a scale to measure the degree of similarity (or difference) between two sequences and thus make it possible to quickly distinguish among the many subtly different alignments that can be generated for any two sequences. Scoring schemes contain two separate elements: the first assigns a value to a pair of aligned residues, while the second deals with the presence of insertions or deletions (indels) in one sequence relative to the other. For protein sequence alignments, reference substitution matrices (see Section 4.3) are used to give a score to each pair of aligned residues. Indels necessitate the introduction of gaps in the alignment, which also have to be scored. The total score S of an alignment is given by summing the scores for individual alignment positions. Special scoring techniques that are applicable only to multiple alignments will be dealt with in Sections 6.1 and 6.4.

One might think that a relatively straightforward way of assessing the probability of the replacement of one amino acid by another would be to use the minimum number of nucleotide base changes required to convert between the codons for the two residues. However, most evolutionary selection occurs at the level of protein function and thus gives rise to significant bias in the mutations that are accepted. Therefore the number of base changes required cannot be expected to be a good measure of the likelihood of substitution. Currently used reference substitution matrices are based on the frequency of particular amino acid substitutions observed in multiple alignments that represent the evolutionary divergence of a given protein. The substitution frequencies obtained thus automatically take account of evolutionary bias.

Two methods that have been used in deriving substitution matrices from multiple sequence alignments will be described here. These have provided two sets of matrices in common use: the PAM and the BLOSUM series. Both these matrices can be related to a probabilistic model, which will be covered first. The key concepts involved in deriving scoring schemes for sequence alignments are outlined in Flow Diagram 5.1.

Alignment scores attempt to measure the likelihood of a common evolutionary ancestor

The theoretical background of alignment scoring is based on a simple probabilistic approach. Two alternative mechanisms could give rise to differences in DNA or protein sequences: a random model and a nonrandom (evolutionary) model. By generating the probability of occurrence of a particular alignment for each model, an assessment can be made about which mechanism is more likely to have given rise to that alignment.

In the random model, there is no such process as evolution; nor are there any structural or functional processes that place constraints on the sequence and thus cause similarities. All sequences can be assumed to be random selections from a given pool of residues, with every position in the sequence totally independent of every other. Thus for a protein sequence, if the proportion of amino acid type a in the pool is p_a, this fraction will be reproduced in the amino acid composition of the protein.

The nonrandom model, on the other hand, proposes that sequences are related—in other words, that some kind of evolutionary process has been at work—and hence that there is a high correlation between aligned residues. The probability of occurrence of particular residues thus depends not on the pool of available residues, but on the residue at the equivalent position in the sequence of the common ancestor; that is, the sequence from which both of the sequences being aligned have evolved. In this model the probability of finding a particular pair of residues of types a and b aligned is written $q_{a,b}$. The actual values of $q_{a,b}$ will depend on the properties of the evolutionary process.

Suppose that in one position in a sequence alignment, two residues, one type a and the other type b, are aligned. The random model would give the probability of this occurrence as $p_a p_b$, a product, as the two residues are seen as independent. The equivalent value for the nonrandom model would be $q_{a,b}$. These two models can be compared by taking the ratios of the probabilities, called the **odds ratio**, that is $q_{a,b}/p_a p_b$. If this ratio is greater than 1 (that is, $q_{a,b} > p_a p_b$) the nonrandom model is more likely to have produced the alignment of these residues. However, a single model is required that will explain the complete sequence alignment, so all the individual terms for the pairs of aligned residues must be combined.

In practice, there is often a correlation between adjacent residues in a sequence; for example, when they are in a hydrophobic stretch of a protein such as a transmembrane helix (see Chapter 2). This type of correlation is ignored in both of these models, so that each position in an alignment will be regarded as independent. In that case, the odds ratios for the different positions can be multiplied together to give the odds ratio for the entire alignment:

$$\prod_u \left(\frac{q_{a,b}}{p_a p_b} \right)_u \qquad \text{(EQ5.1)}$$

where the product is over all positions u of the alignment.

It is frequently more practical to deal with sums rather than products, especially when small numbers are involved. This can easily be arranged by taking logarithms of the odds ratio to give the **log-odds ratio**. This ratio can be summed over all positions of the alignment to give S, the score for the alignment:

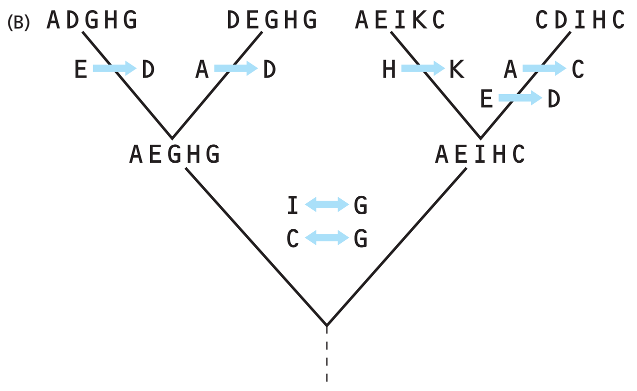
$$S = \sum_u \log \left(\frac{q_{a,b}}{p_a p_b} \right)_u = \sum_u \left(s_{a,b} \right)_u \qquad \text{(EQ5.2)}$$

where $s_{a,b}$ is the score (that is, the **substitution matrix** element) associated with the alignment of residue types a and b. A positive value of $s_{a,b}$ means that the probability of those two residues being aligned is greater in the nonrandom than in the random model. The converse is true for negative $s_{a,b}$ values. S is a measure of the relative likelihood of the whole alignment arising due to the nonrandom model as compared with the random model. However, as discussed later in this chapter, a

(A)

```
DEGHG
ADGHG
CDIHC
AEIKC
```

(B)



(C)

	A	C	D	E	G	H	I	K
A		1	1					
C	1				1			
D	1			2				
E			2					
G		1					1	
H								1
I					1			
K						1		

positive S is not a sufficient test of the alignment's significance. There will be a distribution of values of S for a given set of sequences, which can be used to determine significant scores.

From this discussion one would expect there to be both positive and negative $s_{a,b}$ values. In practice this is not always the case, because each $q_{a,b}/p_a p_b$ term can be multiplied by a constant. Multiplication by a constant X will result in a term $\log X$ being added to each $s_{a,b}$. This could result in all $s_{a,b}$ values being positive, for example. The alignment score S is shifted by $L_{aln} \log X$ for an alignment of length L_{aln}. Similarly, all the $s_{a,b}$ values can be multiplied by a constant. In both cases, scores of alternative alignments of the same length retain the same relative relationship to each other. However, local alignments discussed below involve comparing alignments of different lengths, in which case adding a constant $\log X$ to each $s_{a,b}$ will have an effect on the relative scores.

Note that the link between substitution matrices and the log-odds ratio described by Equation EQ 5.2 may exist even if the matrix was derived without any reference to this theory. Firstly, note that the **expected score** for aligning two residues can be written

$$E\left(s_{a,b}\right) = \sum_{a,b} p_a p_b s_{a,b} \qquad \text{(EQ5.3)}$$

If this is negative, and there is at least one positive score, one can in principle perform the reverse procedure to obtain the $q_{a,b}$ given the $s_{a,b}$ and the p_a. The procedure is not entirely straightforward, and interested readers should refer to the Further Reading at the end of the chapter.

Figure 5.1
The identification of accepted point mutations in the derivation of the PAM/MDM amino acid substitution scoring matrices. (A) Part of an alignment that might have occurred in the protein sequence data used to obtain the Dayhoff PAM matrices. (B) The phylogenetic tree derived from this alignment. Note that this tree hypothesizes the smallest number of mutations required to explain the observed sequences, but does not necessarily represent the actual evolutionary history. See Chapters 7 and 8 for further details on phylogenetic trees. The direction of some mutations cannot be resolved due to lack of knowledge of the oldest ancestral sequence. This is also the case for the other ancestral sequences shown, which are specified here for clarity only. (C) The matrix of accepted substitutions derived from this tree. Note that all observed mutations have been assumed to be equally likely in either direction, and hence are counted in both directions.

The PAM (MDM) substitution scoring matrices were designed to trace the evolutionary origins of proteins

As we saw in Section 4.3, one commonly used type of matrix for protein sequences is the point accepted mutations (PAM) matrix, also known as the mutation data matrix (MDM), derived by Margaret Dayhoff and colleagues from the analysis of multiple alignments of families of proteins, including cytochrome *c*, α- and β-globin (the protein chains of which hemoglobin is composed), insulin A and B chains, and ferredoxin.

These raw data are biased by the amino acid composition of the sequences, the differing rates of mutation in different protein families, and sequence length. The first step in an attempt to remove this bias is to calculate, for each alignment, the exposure of each type of amino acid residue to mutation. This is defined as the compositional fraction of this residue type in the alignment multiplied by the total number of mutations (of any kind) that have occurred per 100 alignment positions.

See below for a description of how the number of mutations is calculated. The value for each residue type is summed over all the alignments in the data set to give the total exposure of that residue type. The mutability of a specific residue type *a* is defined as the ratio of the total number of mutations in the data that involve residue *a* divided by the total exposure of residue *a*. Usually this is reported relative to alanine as a standard, and is referred to as the **relative mutability**, m_a. A few residues have higher mutability than alanine, but more notable are those that are less likely to change, especially cysteine and tryptophan. The mutability of these residues is approximately one-fifth that of alanine.

Phylogenetic trees are constructed for each alignment by a method that infers the most likely ancestral sequence at each internal node and hence postulates all the mutations that have occurred. The observed substitutions are tabulated in a matrix, ***A*** (**accepted point mutation matrix**) according to the residue types involved (see Figure 5.1). It is assumed that a mutation from residue type *a* to type *b* was as likely as a mutation from *b* to *a*, so all observed mutations are included in the count for both directions. Where there is uncertainty as to which mutations have occurred, all possibilities are treated as equally likely, and fractional mutations are added to the matrix of accepted substitutions. The matrix element values are written $A_{a,b}$. Dayhoff's 1978 dataset contained 1572 observed substitutions, but even so, 35 types of mutations had not been observed, for example tryptophan (W) → alanine (A). This was due to the relatively small dataset of highly similar sequences that was used.

From this information a **mutation probability matrix**, ***M***, can be defined. Each element $M_{a,b}$ gives the probability that a residue of type *b* will be replaced by one of type *a* after a given period of evolutionary time. The residue *b* has a likelihood of mutating that is proportional to m_b. The expected fraction of mutations of *b* into residue *a* can be obtained from the accepted point mutation matrix (see Figure 5.1C) with elements $A_{a,b}$. Thus the off-diagonal ($a \neq b$) terms of *M* are given by the formula

$$M_{a,b} = \frac{\Lambda m_b A_{a,b}}{\sum_a A_{a,b}} \qquad \text{(EQ5.4)}$$

where Λ is a constant that accounts in part for the proportionality constant of m_b and in part for the unspecified evolutionary time period. Note that matrix ***M*** is not symmetrical because of the residue relative mutability m_b. The diagonal terms ($a = b$) of this matrix, corresponding to no residue change, are

$$M_{b,b} = 1 - \Lambda m_b \qquad \text{(EQ5.5)}$$

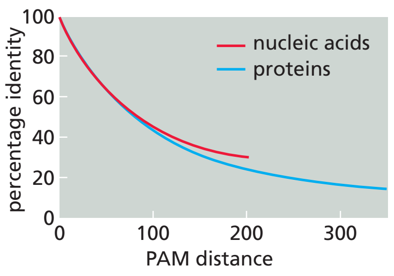
Note that the sum of all off-diagonal elements $M_{a,b}$ involving mutation from a given residue *b* to another type and the element $M_{b,b}$ equals 1, as required for ***M*** to be a probability matrix.

The percentage of amino acids unchanged in a sequence of average composition after the evolutionary change represented by the matrix ***M*** can be calculated as

$$100\sum_b f_b M_{b,b} = 100\sum_b f_b\left(1 - \Lambda m_b\right) \qquad \text{(EQ5.6)}$$

where f_b is approximately the frequency of residue type *b* in the average composition. In fact f_b is the total exposure of residue *b* normalized so that the sum of all values is 1, and is the residue composition weighted by the sequence mutation rate. The value of Λ is selected to determine this percentage of unchanged residues. If Λ is chosen to make this sum 99, the matrix represents an evolutionary change of 1 PAM (that is, one accepted mutation per 100 residues).

Figure 5.2
The relationship between the evolutionary distance in PAMs and the percentage identity between two sequences. This shows that the evolutionary model predicts considerable identity even between very distantly related sequences.



C	11																			
S	1	2																		
T	-1	1	2																	
P	-2	1	1	6																
A	-1	1	2	1	2															
G	-1	1	-1	-1	1	5														
N	-1	1	1	-1	0	0	3													
D	-3	0	-1	-2	0	1	2	5												
E	-4	-1	-1	-2	-1	0	1	4	5											
Q	-3	-1	-1	0	-1	-1	0	1	2	5										
H	0	-1	-1	0	-2	-2	1	0	0	2	6									
R	-1	-1	-1	-1	-1	0	0	-1	0	2	2	5								
K	-3	-1	-1	-2	-1	-1	1	0	1	2	1	4	5							
M	-2	-1	0	-2	-1	-3	-2	-3	-3	-2	-2	-2	-2	6						
I	-2	-1	1	-2	0	-3	-2	-3	-3	-3	-3	-3	-3	3	4					
L	-3	-2	-1	0	-1	-4	-3	-4	-4	-2	-2	-3	-3	3	2	5				
V	-2	-1	0	-1	1	-2	-2	-2	-2	-3	-3	-3	-3	2	4	2	4			
F	0	-2	-2	-3	-3	-5	-3	-5	-5	-4	0	-4	-5	0	0	2	0	8		
Y	2	-1	-3	-3	-3	-4	-1	-2	-4	-2	4	-2	-3	-2	-2	-1	-3	5	9	
W	1	-3	-4	-4	-4	-2	-5	-5	-5	-3	-3	0	-3	-3	-4	-2	-3	-1	0	15
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

Figure 5.3
The PET91 version of the PAM250 substitution matrix. Scores that would be given to identical matched residues are in blue; positive scores for nonidentical matched residues are in red. The latter represent pairs of residues for which substitutions were observed relatively often in the aligned reference sequences.

According to the Dayhoff model of evolution, to obtain the probability matrices for higher percentages of accepted mutations, the 1-PAM matrix is multiplied by itself. This is because the model of evolution proposed was a Markov process (Markov models are described in detail in Section 10.3). If the 1-PAM matrix is squared, it gives a 2-PAM matrix; if cubed, a 3-PAM matrix; and so on. These correspond to evolutionary periods twice and three times as long as the period used to derive the 1-PAM matrix. Similarly, a 250-PAM matrix is obtained by raising the 1-PAM matrix to the 250th power.

These matrices tell us how many mutations have been accepted, but not the percentage of residues that have mutated: some may have mutated more than once, others not at all. For each of these matrices, evaluation of Equation EQ5.6 gives the actual percentage identity to the starting sequence expected after the period of evolution represented by the matrix. The percentage identity does not decrease linearly with time in this model (see Figure 5.2).

So far, a probability matrix has been obtained, but not a scoring matrix. As discussed earlier, such a score should involve the ratio of probabilities derived from nonrandom and random models. The matrix $\boldsymbol{M}$ gives the probability of residue b mutating into residue a if the two sequences are related, that is, if substitution is nonrandom. It already includes a term for the probability of occurrence of residue b in the total exposure term used to calculate m_b. Thus the only term needed for the random-model probability is f_a, the likelihood of residue a occurring by chance. Hence, the scoring matrix was originally derived from $\boldsymbol{M}$ by the formula

$$s_{a,b} = 10\log_{10}\left(\frac{M_{a,b}}{f_a}\right) \qquad \text{(EQ5.7)}$$

and is in fact a symmetrical matrix. The resultant scoring matrices *s* are usually named PAM*n*, where *n* is the number of accepted point mutations per 100 residues in the probability matrix ***M*** from which they are derived. The exact scaling factors and logarithm base are to some degree arbitrary, and are usually chosen to provide integer scores with a suitable number of significant figures. In Figure 4.4B the PAM120 matrix is shown, but a scaling factor of $2\log_2$ has been used instead of $10\log_{10}$, so that the values are in units of half bits (a bit is a measure of information) (see Appendix A).

There was a lack of sequence data available when the original work was done to derive the PAM matrices, and in 1991 the method was applied to a larger dataset, producing the PET91 matrix that is an updated version of the original PAM250 matrix (see Figure 5.3). This matrix shows considerable differences for aligning two tryptophan (W) residues, with a score of 15, as opposed to two alanine residues (A), which score only 2. About one-fifth of the scores for nonidentical residues have positive scores, considerably more than occurs in PAM120 (see Figure 4.4B), reflecting the longer evolutionary period represented by the matrix.

The BLOSUM matrices were designed to find conserved regions of proteins

The BLOCKS database containing large numbers of ungapped multiple local alignments of conserved regions of proteins, compiled by Steven and Jorja Henikoff, became available in 1991 (see Section 4.8). These alignments included distantly related sequences, in which multiple base substitutions at the same position could be observed. The BLOCKS database was soon recognized as a resource from which substitution preferences could be determined, leading to the BLOSUM substitution score matrices.

There are two contrasts with the data analysis used to obtain the PAM matrices. First, the BLOCKS alignments used to derive the BLOSUM matrices include sequences that are much less similar to each other than those used by Dayhoff, but whose evolutionary homology can be confirmed through intermediate sequences. In addition, these alignments are analyzed without creating a phylogenetic tree and are simply compared with each other.

A direct comparison of aligned residues does not model real substitutions, because in reality the sequences have evolved from a common ancestor and not from each other. Nevertheless, as the large sequence variation prevents accurate construction of a tree, there is no alternative. However, if the alignment is correct, then aligned residues will be related by their evolutionary history and therefore their alignment will contain useful information about substitution preferences. Another argument in favor of direct analysis of aligned sequence differences is that often the aim is not to recreate the evolutionary history, but simply to try to align sequences to test them for significant similarity. The intention in producing the BLOSUM matrices was to find scoring schemes that would identify conserved regions of protein sequences.

One of the key aspects of the analysis of the alignment blocks is to weight the sequences to try to reduce any bias in the data. This is necessary because the sequence databases are highly biased toward certain species and types of proteins, which means there are many very similar sequences present. The weighting involves clustering the most similar sequences together, and different matrices are produced according to the threshold *C* used for this clustering. Sequences are clustered together if they have $\geq C\%$ identity, and the substitution statistics are calculated only between clusters, not within them. Weights are determined according to the number of sequences in the cluster. For a cluster of N_{seq} sequences, each sequence is assigned a weight of $1/N_{\text{seq}}$. The weighting scheme was used to obtain a series of substitution matrices by varying the value of *C*, with the matrices named

(A)

	1 2 3 4 5
1	A T C K Q
2	A T C R N
3	A S C K N
4	S S C R N
5	S D C E Q
6	S E C E N
7	T E C R Q

(B)

	q_{QN}	q_{NN}	q_{QQ}	p_{N}	p_{Q}
C=62%	0.114	0.057	0.029	0.114	0.086
C=50%	0.117	0.025	0.058	0.084	0.117
C=40%	–	–	–	–	–

Figure 5.4
Derivation of the BLOSUM amino acid substitution scoring matrices. (A) An example ungapped alignment block with one fully conserved position (cysteine) colored red. In this case, sequences have been clustered if they are at least 50% identical to any other within the cluster. Thus for example, although sequences 1 and 4 are only 20% identical to each other, they are in the same cluster, as they are both 60% identical to sequence 2. Similar clustering of sequence data was used to derive the BLOSUM-50 matrix. If the same data were used to derive the BLOSUM-62 matrix (that is, C = 62%) none of these sequences would cluster together, as no two of them share more than 60% identical residues, leading to very different sequence weights. (B) Values of $q_{a,b}$ and p_a for asparagine (N) and glutamine (Q) residues for three different values of C. For C = 40% all the sequences belong in the same cluster, and since $q_{a,b}$ and p_a measure intercluster alignment statistics, they cannot be calculated in this case.

as BLOSUM-62, for example, in the case where C = 62%. The sequences of all the alignments used to obtain the Dayhoff matrices fall in a single cluster for $C \leq 85\%$, indicating that they were much more similar to each other than those used for BLOSUM.

The derivation of substitution data will be illustrated using the example alignment in Figure 5.4A and the case of C = 50%, which would lead to the BLOSUM-50 matrix. There are three sequence clusters, with four, two, and one sequences, giving weights to their constituent sequences of $^1/_4$, $^1/_2$, and 1, respectively. These weights are applied to the counts of observed aligned residues to produce the weighted frequencies $f_{a,b}$. The observed probability ($q_{a,b}$) of aligning residues of types a and b is given by

$$q_{a,b} = \frac{f_{a,b}}{\sum\limits_{1 \leq b \leq a}^{20} f_{a,b}} \qquad \text{(EQ5.8)}$$

Note that this ignores which sequence the residue a or b has come from, so that $f_{a,b}$ and $f_{b,a}$, and hence $q_{a,b}$ and $q_{b,a}$, are equal.

Consider the calculation of $q_{a,b}$ for asparagine (N) and glutamine (Q) residues, which occur only in column 5 in Figure 5.4A. If C = 62%, then all sequence clusters will contain just one sequence and each sequence will have a weight of 1. (No pair of sequences share more than 60% identical residues.) In this case there are 21 ($7 \times 6/2$) distinct cluster pairs and thus 21 pairs of aligned residues in any single alignment column. Counting these, there are 12 QN pairs ($= f_{\mathrm{Q,N}}$), 3 QQ ($= f_{\mathrm{Q,Q}}$), and 6 NN ($= f_{\mathrm{N,N}}$), making a total of 21 pairs. As all sequence weights are 1, they play no real part in this calculation. As there are five alignment columns, the total number of aligned pairs (the denominator of Equation EQ5.8) is 105. From these data the $q_{a,b}$ can be obtained, as listed in Figure 5.4B. Note that if other columns had contained N and Q, they would also have needed to be included in the calculation.

Considering the case of C = 50%, the sequences separate into three clusters, so the total number of cluster pairs at position 5 will be 3 ($3 \times 2/2$). The top, middle, and bottom clusters can be regarded as being {$^1/_4$Q, $^3/_4$N}, {$^1/_2$Q, $^1/_2$N}, and {Q}, respectively, at this position. Remembering to consider only pairs between clusters and not within them, the weighted number of QN aligned pairs is calculated as

$$f_{\mathrm{Q,N}} = \left(\frac{1}{4} \times \frac{1}{2}\right) + \left(\frac{3}{4} \times \frac{1}{2}\right) + \left(\frac{3}{4} \times 1\right) + \left(\frac{1}{2} \times 1\right) = \frac{14}{8} \qquad \text{(EQ5.9)}$$

where the first term is for Q residues of the top cluster and N residues of the second cluster; the second term is for N residues of the top cluster and Q residues of the

Figure 5.5
The BLOSUM-62 substitution matrix scores in half bits. Scores that would be given to identical matched residues are in blue; positive scores for nonidentical matched residues are in red. The latter represent pairs of residues for which substitutions were observed relatively often in the aligned reference sequences.

C	9																			
S	−1	4																		
T	−1	1	5																	
P	−3	−1	−1	7																
A	0	1	0	−1	4															
G	−3	0	−2	−2	0	6														
N	−3	1	0	−2	−2	0	6													
D	−3	0	−1	−1	−2	−1	1	6												
E	−4	0	−1	−1	−1	−2	0	2	5											
Q	−3	0	−1	−1	−1	−2	0	0	2	5										
H	−3	−1	−2	−2	−2	−2	1	−1	0	0	8									
R	−3	−1	−1	−2	−1	−2	0	−2	0	1	0	5								
K	−3	0	−1	−1	−1	−2	0	−1	1	1	−1	2	5							
M	−1	−1	−1	−2	−1	−3	−2	−3	−2	0	−2	−1	−1	5						
I	−1	−2	−1	−3	−1	−4	−3	−3	−3	−3	−3	−3	−3	1	4					
L	−1	−2	−1	−3	−1	−4	−3	−4	−3	−2	−3	−2	−2	2	2	4				
V	−1	−2	0	−2	0	−3	−3	−3	−2	−2	−3	−3	−2	1	3	1	4			
F	−2	−2	−2	−4	−2	−3	−3	−3	−3	−3	−1	−3	−3	0	0	0	−1	6		
Y	−2	−2	−2	−3	−2	−3	−2	−3	−2	−1	2	−2	−2	−1	−1	−1	−1	3	7	
W	−2	−3	−2	−4	−3	−2	−4	−4	−3	−2	−2	−3	−3	−1	−3	−2	−3	1	2	11
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

second cluster; the third term is for the Q residue of the third cluster; and the fourth term is for the N residues of each of the other clusters. The equivalent values for $f_{\mathrm{N,N}}$ and $f_{\mathrm{Q,Q}}$ are 3/8 and 7/8, respectively. Dividing these $f_{a,b}$ values by 15 (the weighted total number of aligned pairs in the data, three for each alignment position) gives the $q_{a,b}$ values shown in Figure 5.4B. The values for $f_{\mathrm{Q,Q}}$ and $f_{\mathrm{N,N}}$ differ between $C =$ 62% and $C =$ 50%, because in the latter case most of the N residues are in one cluster. Note that in the case of $C =$ 40% there is only one cluster, as each sequence is at least 40% identical to at least one other, so that no intercluster residue pairing exists to be counted! This example shows how the value of C can affect the derived substitution matrix in a complicated manner.

The scores for the residue pairs are obtained using the log-odds approach described earlier. The estimate $e_{a,b}$ of the probability of observing two residues of type a and b aligned by chance is given (as in deriving PAM matrices) by a weighted composition of the sequences, with p_a being approximately the fraction of all residues that is type a. The background residue frequencies p_a are defined by

$$p_a = q_{a,a} + \sum_{a \neq b} \frac{q_{a,b}}{2} \qquad \text{(EQ5.10)}$$

Figure 5.6
Three nucleotide substitution scoring matrices, derived by Chiaromonte and co-workers. Each matrix was obtained by analysis of alignments of distinct regions of the human and mouse genomes with different G+C content. (A) was derived from the CFTR region which is 37% G+C; (B) was derived from the HOXD region which is 47% G+C; (C) was derived from the hum16pter region which is 53% G+C. Each matrix was scaled to give a maximum score of 100.

(A)

	A	**C**	**G**	**T**
A	67	−96	−20	−117
C	−96	100	−79	−20
G	−20	−79	100	−96
T	−117	−20	−96	67

(B)

	A	**C**	**G**	**T**
A	91	−114	−31	−123
C	−114	100	−125	−31
G	−31	−125	100	−114
T	−123	−31	−114	91

(C)

	A	**C**	**G**	**T**
A	100	−123	−28	−109
C	−123	91	−140	−28
G	−28	−140	91	−123
T	−109	−28	−123	100

Figure 5.4B shows how these are affected by the choice of *C*. For identical residues, $e_{a,a}$ is p_a^2, whereas for different residues, $e_{a,b}$ is $2p_ap_b$, because there are two ways in which one can select two different residues. The BLOSUM matrices are defined using information measured in bits, so the formula for a general term is

$$s_{a,b} = \log_2\left(\frac{q_{a,b}}{e_{a,b}}\right) \qquad \text{(EQ5.11)}$$

One of the most commonly used of these matrices is BLOSUM-62, which is shown in Figure 5.5 in units of half bits, obtained by multiplying the $s_{a,b}$ in Equation EQ5.11 by 2.

Scoring matrices for nucleotide sequence alignment can be derived in similar ways

The methods described above for deriving scoring matrices have been illustrated with examples from protein sequences. The same techniques can be applied to nucleotides, although often simple scoring schemes such as +5 for a match and –4 for a mismatch are used. With certain exceptions, such as 16S rRNA and repeat sequences, until quite recently almost all sequences studied were for protein-coding regions, in which case it is usually advantageous to align the protein sequences. This has changed with the sequencing of many genomes and the alignment of long multigene segments or even whole genomes.

Matrices have been reported that are derived from alignments of human and mouse genomic segments (see Figure 5.6). The different matrices were derived from sequence regions with different G+C content, as it is thought that this influences the substitution preferences. This is to be contrasted with the different PAM and BLOSUM matrices, which are based on evolutionary distance. The matrices were derived using a similar approach to that described for the BLOSUM series. However, there is a difference worth noting when dealing with DNA sequences. Any alignment of DNA sequences implies a second alignment of the complementary strand. Thus, every observation of a T/C aligned pair implies in addition an aligned A/G pair.

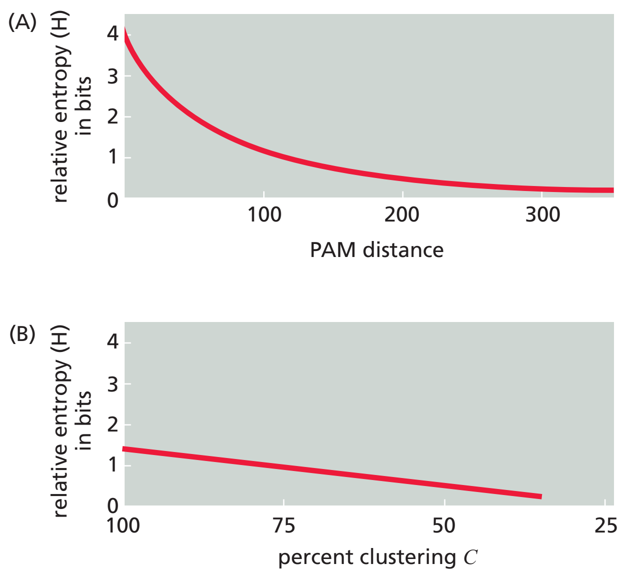


Figure 5.7
Plots of the relative entropy *H* in bits for two different series of substitution matrices. (A) Plot for the PAM matrices according to PAM distance (Data from Altschul, 1991.) (B) Plot for the BLOSUM matrices according to the percentage cut-off *C* used in clustering alignment blocks. (Data from Henikoff and Henikoff, 1992.)

The substitution scoring matrix used must be appropriate to the specific alignment problem

Many other substitution scoring matrices have been derived. Some are based on alternative ways of analyzing the alignment data, while others differ in the dataset used. Some are intended for specialized use, such as aligning transmembrane sequences. The matrices can be compared in three ways, focusing on (1) the relative patterns of scores for different residue types, (2) the actual score values, or (3) the practical application of the matrices.

Cluster analysis of the scores can be used to see if matrices distinguish between the amino acids in different ways. For example, one matrix may be strongly dominated by residue size, another by polarity. This may improve our understanding of the evolutionary driving forces or, alternatively, highlight shortcomings in the sequence data used to derive the matrix, but probably will not assist in determining the relative usefulness of matrices for constructing alignments.

In common with most proposed amino acid substitution matrices, the PAM and BLOSUM matrix series include both positive and negative scores, with the average score being negative. The actual score values can be summarized in two measures that have some practical use. The **relative entropy** (H) of the nonrandom model with respect to the random model is defined as

$$H = \sum_{a,b} q_{a,b} s_{a,b} = \sum_{a,b} q_{a,b} \log\left(\frac{q_{a,b}}{p_a p_b}\right) \qquad \text{(EQ5.12)}$$

The scores $s_{a,b}$ are summed, weighted by $q_{a,b}$, and H is a measure of the average information available at each alignment position to distinguish between the nonrandom and random models, and is always positive. (See Appendix A for further discussion of this measure.) Figure 5.7 shows the variation of H for different PAM and BLOSUM matrices. The shortest local alignment that can have a significant score is in part dependent on the relative entropy of the scoring matrix used, as discussed later in the chapter. The other measure of score values is the expected score, defined in Equation EQ5.3, which is usually—but not necessarily—negative, for example –0.52 for BLOSUM-62. This measure has been found to influence the variation of alignment scores with alignment length.

Perhaps the best way to compare matrices is to see how well they perform with real data. Two different criteria have been used: the ability to discover related sequences in searching a database, and the accuracy of the individual alignments derived. There are many potential difficulties in making a meaningful comparison, including the choice of data to use and the choice of gap penalties. Although some matrices do perform better at certain tasks, often the differences for the commonly used matrices are small enough that their importance is unclear, especially as a poor choice of gap penalties can have a significant effect.

Gaps are scored in a much more heuristic way than substitutions

A scoring scheme is required for insertions and deletions in alignments, as they are common evolutionary events. The simplest method is to assign a gap penalty g on aligning any residue with a gap; that is, a scoring formula $g = -E$, where E is a positive number. If the gap is n_{gap} residues long, then this **linear gap penalty** is defined as

$$g\left(n_{\text{gap}}\right) = -n_{\text{gap}}E \tag{EQ5.13}$$

Usually, no account is taken of the type of residue aligned with the gap, although making the value of E vary with residue type would easily do this. The observed preference for fewer and longer gaps can be modeled by using a higher penalty to initiate a gap [the **gap opening penalty** (GOP), designated I] and then a lower penalty to extend an existing gap [the **gap extension penalty** (GEP), designated E]. This leads to the **affine gap penalty** formula

$$g\left(n_{\text{gap}}\right) = -I - \left(n_{\text{gap}} - 1\right)E \tag{EQ5.14}$$

for a gap of n_{gap} residues. Note that an alternative definition can give rise to the formula

$$g\left(n_{\text{gap}}\right) = -I - n_{\text{gap}}E \tag{EQ5.15}$$

Again, residue preferences can easily be added to this scheme by varying the value of E.

The values of the gap parameters need to be carefully chosen for the specific substitution scoring matrix used. Failure to optimize these parameters can significantly degrade the performance of the overall scoring scheme. This is illustrated by the worked example later in the chapter. Some matrices seem less sensitive to gap parameterization than others. In practice, as optimization is a lengthy process, most workers use previously reported optimal combinations. Typical ranges of the parameters for protein alignment are 7–15 for I and 0.5–2 for E.

5.2 Dynamic Programming Algorithms

For any given pair of sequences, if gaps are allowed there is a large number of possibilities to consider in determining the best-scoring alignment. For example, two sequences of length 1000 have approximately 10^{600} different alignments, vastly more than there are particles in the universe! Given the number and length of known sequences it would seem impossible to explore all these possibilities. Nevertheless, a class of algorithms has been introduced that is able to efficiently explore the full range of alignments under a variety of different constraints. They are known as dynamic programming algorithms, and efficiently avoid needless exploration of the majority of alignments that can be shown to be nonoptimal. There are several variants that produce different kinds of alignments, as outlined in Flow Diagram 5.2.

The key property of dynamic programming is that the problem can be divided into many smaller parts. Consider the following alignment:

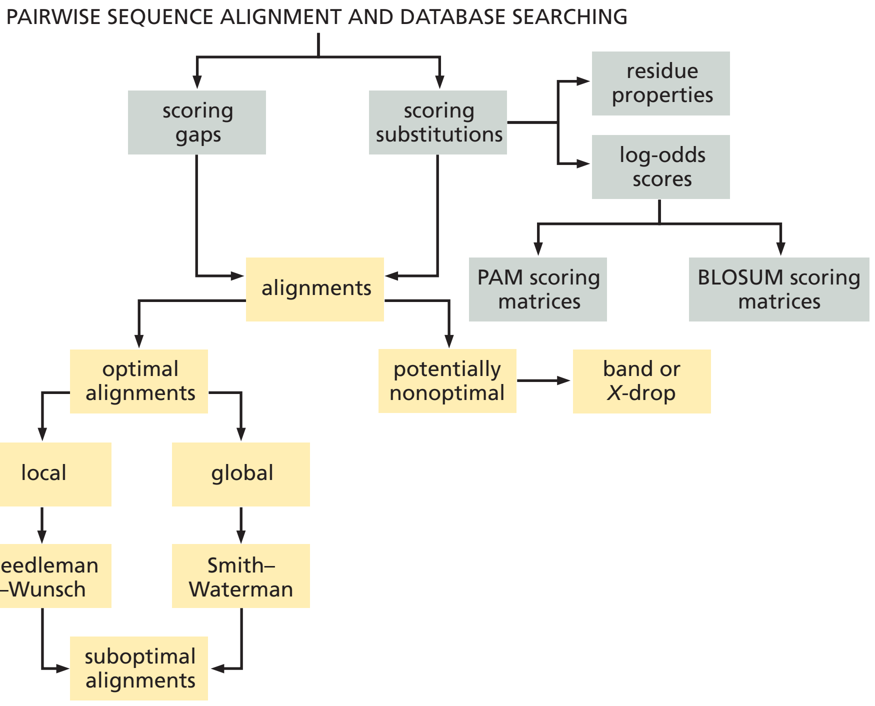
$$X_1 \cdots X_u \; X_{u+1} \cdots X_v \; X_{v+1} \cdots X_L$$

$$Y_1 \cdots Y_u \; Y_{u+1} \cdots Y_v \; Y_{v+1} \cdots Y_L$$

in which the subscripts u, v, etc. refer to alignment positions rather than residue types, so that X_u, Y_v, and so on each correspond to a residue or to a gap. The alignment has been divided into three parts, with positions labeled $1 \rightarrow u$, $u + 1 \rightarrow v$, and

Flow Diagram 5.2

The key concept introduced in this section is that the method of dynamic programming can be applied with minor modifications to solve several related problems of determining optimal and near-optimal pairwise sequence alignments.



$v + 1 \rightarrow L$. Because scores for the individual positions are added together, the score of the whole alignment is the sum of the scores of the three parts; that is, their contributions to the score are independent. Thus, the optimal global alignment can be reduced to the problem of determining the optimal alignments of smaller sections. A corollary to this is that the global optimal alignment will not contain parts that are not themselves optimal. While affine gap penalties require a slightly more sophisticated argument, essentially the same property holds true for them as well.

Starting with sufficiently short sub-sequences, for example the first residue of each sequence, the optimal alignment can easily be determined, allowing for all possible gaps. Subsequently, further residues can be added to this, at most one from each sequence at any step. At each stage, the previously determined optimal subsequence alignment can be assumed to persist, so only the score for adding the next residue needs to be investigated. A worked example later in the section will make this clear. In this way the optimal global alignment can be grown from one end of the sequence. As an alignment of two sequences will consist of pairs of aligned residues, a rectangular matrix can conveniently represent these, with rows corresponding to the residues of one sequence, and columns to those of the other.

Until the global optimal alignment has been obtained, it is not known which actual residues are aligned. All possibilities must be considered or the optimal alignment could be missed. This is not as impossible as it might seem.

Saul Needleman and Christian Wunsch published the original dynamic programming application in this field in 1970, since then many variations and improvements have been made, some of which will be described here. There have been three different motivations for developing these modifications. Firstly, global and local alignments require slightly different algorithms. Secondly, but less commonly, certain gap-penalty functions and the desire to optimize scoring parameters have resulted in further new schemes. Lastly, especially in the past, the computational requirements of the algorithms prevented some general applications. For example, the basic technique in a standard implementation requires computer memory proportional to the product mn for two sequences of length m and n. Some algorithms have been proposed that reduce this demand considerably.

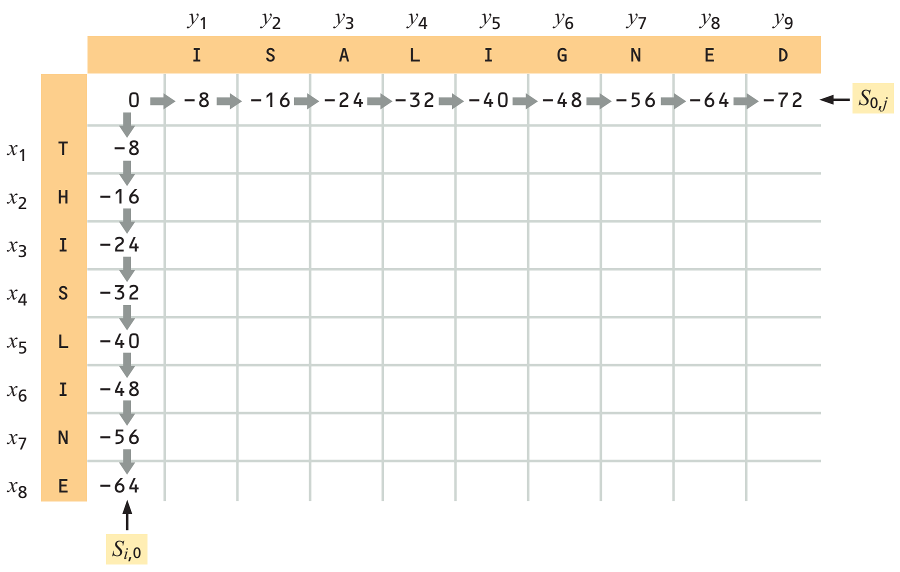


Figure 5.8
The initial stage of filling in the dynamic programming matrix to find the optimal global alignment of the two sequences THISLINE and ISALIGNED. The initial stage of filling in the matrix depends only on the linear gap penalty, defined in Equation EQ5.13 with *E* set to –8. The arrows indicate the cell(s) to which each cell value contributes.

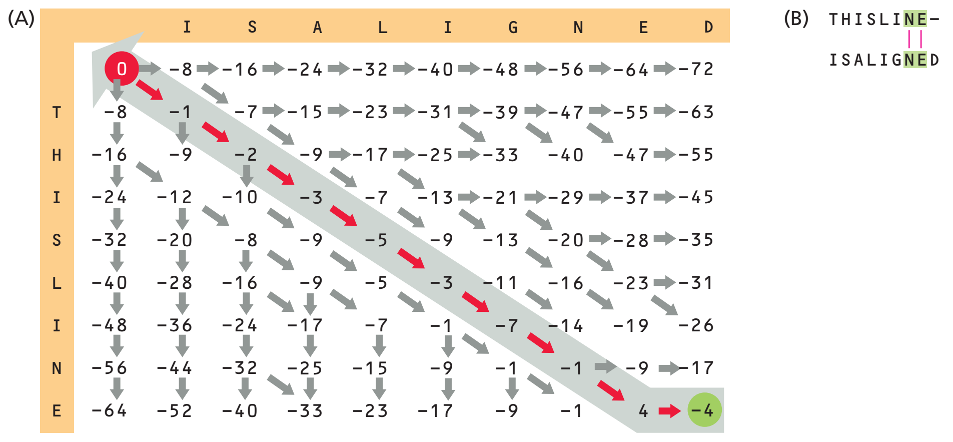
Optimal global alignments are produced using efficient variations of the Needleman–Wunsch algorithm

We will introduce dynamic programming methods by describing their use to find optimal global alignments. Needleman and Wunsch were the first to propose this method, but the algorithm given here is not their original one, because significantly faster methods of achieving the same goal have since been developed. The problem is to align sequence x ($x_1x_2x_{3...}x_m$) and sequence y ($y_1y_2y_{3...}y_n$), finding the best-scoring alignment in which all residues of both sequences are included. The score will be assumed to be a measure of similarity, so that the highest score is desired. Alternatively, the score could be an evolutionary distance (see Section 4.2), in which case the smallest score would be sought, and all uses of "max" in the algorithm would be replaced by "min."

The key concept in all these algorithms is the matrix $\boldsymbol{S}$ of optimal scores of sub-sequence alignments. The matrix has $(m + 1)$ rows labeled $0 \rightarrow m$ and $(n + 1)$ columns labeled $0 \rightarrow n$. The rows correspond to the residues of sequence $\boldsymbol{x}$, and the columns to those of sequence $\boldsymbol{y}$ (see Figure 5.8). We shall use as a working example the alignment of the sequences $\boldsymbol{x}$ = THISLINE and $\boldsymbol{y}$ = ISALIGNED, with the BLOSUM-62 substitution matrix as the scoring matrix (see Figure 5.5). Because the sequences are small they can be aligned manually, and so we can see that the optimal alignment is:

```
TH I S – L I – N E –
   | |   | |   | |
– – I S A L I G N E D
```

This alignment might not produce the optimal score if the gap penalty were set very high relative to the substitution matrix values, but in this case it could be argued that the scoring parameters would then not be appropriate for the problem. In the matrix in Figure 5.8, the element $S_{i,j}$ is used to store the score for the optimal alignment of all residues up to x_i of sequence $\boldsymbol{x}$ with all residues up to y_j of sequence $\boldsymbol{y}$. The sequences ($x_1x_2x_{3...}x_i$) and ($y_1y_2y_{3...}y_j$) with $i < m$ and $j < n$ are called sub-sequences. Column $S_{i,0}$ and row $S_{0,j}$ correspond to the alignment of the first i or j residues with the same number of gaps. Thus, element $S_{0,3}$ is the score for aligning sub-sequence $y_1y_2y_3$ with a gap of length 3.



		I	S	A	L	I	G	N	E	D
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72
T	-8	-1	-7	-15	-23	-31	-39	-47	-55	-63
H	-16	-9	-2	-9	-17	-25	-33	-40	-47	-55
I	-24	-12	-10	-3	-7	-13	-21	-29	-37	-45
S	-32	-20	-8	-9	-5	-9	-13	-20	-28	-35
L	-40	-28	-16	-9	-5	-3	-11	-16	-23	-31
I	-48	-36	-24	-17	-7	-1	-7	-14	-19	-26
N	-56	-44	-32	-25	-15	-9	-1	-1	-9	-17
E	-64	-52	-40	-33	-23	-17	-9	-1	4	-4

Figure 5.9
The dynamic programming matrix (started in Figure 5.8) used to find the optimal global alignment of the two sequences THISLINE and ISALIGNED. (A) The completed matrix using the BLOSUM-62 scoring matrix and a linear gap penalty, defined in Equation EQ5.13 with *E* set to –8. (See text for details of how this was done.) The red arrows indicate steps used in the traceback of the optimal alignment. (B) The optimal alignment returned by these calculations, which has a score of –4.

To fill in this matrix, one starts by aligning the beginning of each sequence; that is, in the extreme upper left-hand corner. We could equally well start at the end of the sequences (extreme bottom right-hand corner), but then the matrix should be labeled $1 \rightarrow m+1$ and $1 \rightarrow n+1$. The elements $S_{i,0}$ and $S_{0,j}$ are easy to fill in, because there is only one possible alignment available. $S_{i,0}$ represents the alignment

$$\begin{array}{llll} a_1 & a_2 & a_3 \cdots & a_i \\ - & - & - \;\cdots & - \end{array}$$

We will start by considering a linear gap penalty *g* of $-8n_{\text{gap}}$ for a gap of n_{gap} residues, giving the scores of $S_{i,0}$ and $S_{0,j}$ as $-8i$ and $-8j$, respectively. This starting point with numerical values inserted into the matrix is illustrated in Figure 5.8.

The other matrix elements are filled in according to simple rules that can be understood by considering a process of adding one position at a time to the alignment. There are only three options for any given position, namely, a pairing of residues from both sequences, and the two possibilities of a residue from one sequence aligning with a gap in the other. These three options can be written as:

$$\begin{array}{lll} \cdots\ \mathrm{x}_i & \cdots\ - & \cdots\ \mathrm{x}_i \\ \cdots\ y_j & \cdots\ y_j & \cdots\ - \end{array}$$

The scores associated with these are $s(x_i,y_j)$, *g*, and *g*, respectively. The value of $s(x_i,y_j)$ is given by the element $s_{a,b}$ of the substitution score matrix, where *a* is the residue type of x_i and *b* is the residue type of y_j. The change in notation is solely to improve the clarity of the following equations.

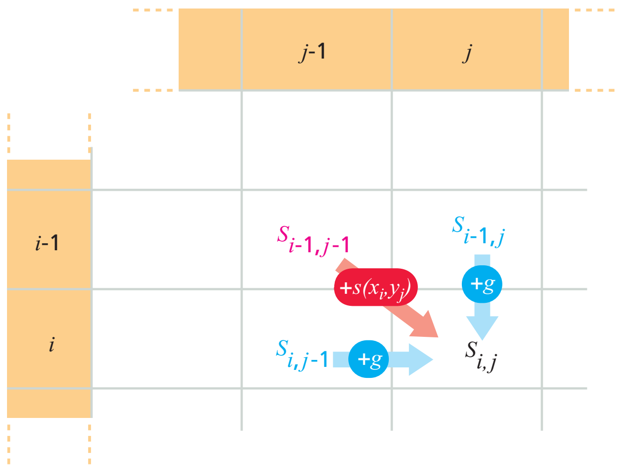


Figure 5.10
Illustration of the application of Equation EQ5.17 to calculate an element of the dynamic programming matrix. Only a small part of the matrix is shown, as only this part contributes directly to the calculation of the element.

Consider the evaluation of element $S_{1,1}$, so that the only residues that appear in the alignment are x_1 and y_1. The left-hand possibility of the three possibilities could only occur starting from $S_{0,0}$, as all other alignments will already contain at least one of these two residues. The middle possibility can only occur from $S_{1,0}$ because it requires an alignment that contains x_1 but not y_1. Similar reasoning shows that the right-hand possibility can only occur from $S_{0,1}$. The three possible alignments have the following scores:

$$\begin{array}{lclclcl} S_{0,0}+s(x_1,y_1) & = & 0+s(\mathrm{I},\mathrm{T}) & = & 0-1 & = & -1 \\ S_{1,0}+g & = & -8-8 & = & -16 & & \\ S_{0,1}+g & = & -8-8 & = & -16 & & \end{array} \qquad \text{(EQ5.16)}$$

where $s(\mathrm{I,T})$ has been obtained from Figure 5.5. Of these alternatives, the optimal one is clearly the first. Hence in Figure 5.9, $S_{1,1} = -1$. Because $S_{1,1}$ has been derived from $S_{0,0}$ an arrow has been drawn linking them in the figure.

An identical argument can be made to construct any element of the matrix from three others, using the formula

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} & + & s(x_i, y_1) \\ S_{i-1,j} & + & g \\ S_{i,j-1} & + & g \end{cases} \qquad \text{(EQ5.17)}$$

The maximum ("max") implies that we are using a similarity score. Figure 5.10 illustrates this formula in the layout of the matrix. Note that it is possible for more than one of the three alternatives to give the same optimal score, in which case arrows are drawn for all optimal alternatives. The completed matrix for the example sequences is given in Figure 5.9A. Note that the number of steps in this algorithm is proportional to the product of the sequence lengths.

We now have a matrix of scores for optimal alignments of many sub-sequences, together with the global sequence alignment score. This is given by the value of $S_{m,n}$, which in this case is $S_{8,9} = -4$. Note that this is not necessarily the highest score in the matrix, which in this case is $S_{8,8} = 4$, but only $S_{m,n}$ includes the information from both complete sequences. For each matrix element we know the element(s) from which it was directly derived. In the figures in this chapter, arrows are used to indicate this information.

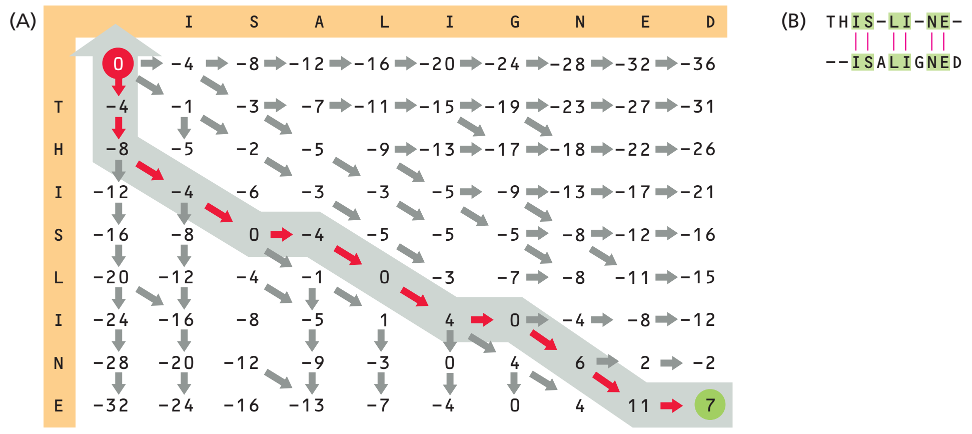

(A)
I S A L I G N E D
0 -4 -8 -12 -16 -20 -24 -28 -32 -36
T -4 -1 -3 -7 -11 -15 -19 -23 -27 -31
H -8 -5 -2 -5 -9 -13 -17 -18 -22 -26
I -12 -4 -6 -3 -3 -5 -9 -13 -17 -21
S -16 -8 0 -4 -5 -5 -5 -8 -12 -16
L -20 -12 -4 -1 0 -3 -7 -8 -11 -15
I -24 -16 -8 -5 1 4 0 -4 -8 -12
N -28 -20 -12 -9 -3 0 4 6 2 -2
E -32 -24 -16 -13 -7 -4 0 4 11 7
(B)
THIS-LI-NE-
|| || ||
--ISALIGNED

Figure 5.11
Optimal global alignment of two sequences, identical to Figure 5.9, except for a change in gap scoring. The linear gap penalty, defined in Equation EQ5.13 using a value of –4 for the parameter *E*. (A) The completed matrix using the BLOSUM-62 scoring matrix. (B) The optimal alignment, which has a score of 7.

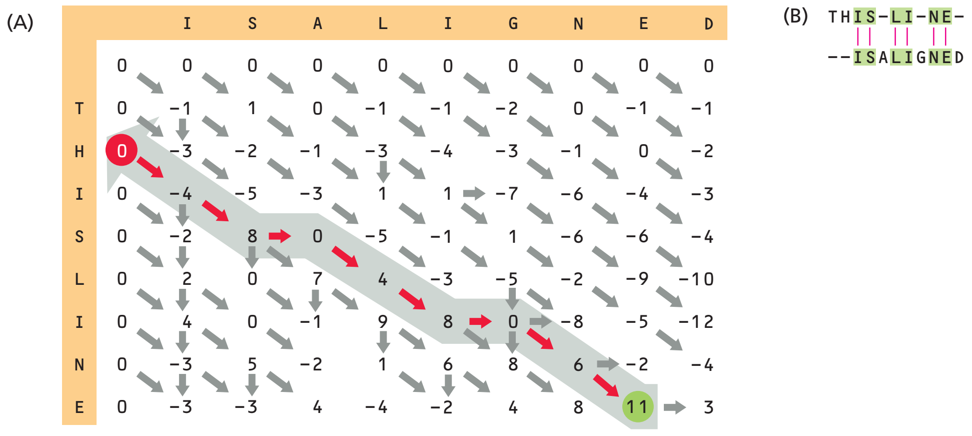


Figure 5.12
Dynamic programming matrix for semiglobal alignment of the same sequences as in Figure 5.9. In this case, end gaps are not penalized. (A) The completed matrix for determining the optimal global alignment of THISLINE and ISALIGNED using the BLOSUM-62 scoring matrix with a linear gap penalty, defined in Equation EQ5.13 with *E* set to –8 and with end gaps not penalized. (B) The optimal alignment, which has a score of 11.

We can use the information on the derivation of each element to obtain the actual global alignment that produced this optimal score by a process called **traceback**. Beginning at $S_{m,n}$ we follow the arrows back through the matrix to the start ($S_{0,0}$). Thus, having filled the matrix elements from the beginning of the sequences, we determine the alignment from the end of the sequences. At each step we can determine which of the three alternatives given in Equation EQ5.17 has been applied, and add it to our alignment. If $S_{i,j}$ has a diagonal arrow from $S_{i-1,j-1}$, that implies the alignment will contain x_i aligned with y_j. Vertical arrows imply a gap in sequence x aligning with a residue in sequence y, and vice versa for horizontal arrows. The traceback arrows involved in the optimal global alignment in Figure 5.9A are shown in red. When tracing back by hand, care must be taken, as it is easy to make mistakes, especially by applying the results to residues x_{i-1} and y_{j-1} instead of x_i and y_j.

The traceback information is often stored efficiently in computer programs, for example using three bits to represent the possible origins of each matrix element. If a bit is set to zero, that path was not used, with a value of one indicating the direction. Such schemes allow all this information to be easily stored and analyzed to obtain the alignment paths.

Note that there may be more than one optimal alignment if at some point along the path during traceback an element is encountered that was derived from more than one of the three possible alternatives. The algorithm does not distinguish between these possible alignments, although there may be reasons for preferring one to the others. Such preference would normally be justified by knowledge of the molecular structure or function. Most programs will arbitrarily report just one single alignment.

The alignment given by the traceback is shown in Figure 5.9B. It is not the one we expected, in that it contains no gaps. The carboxy-terminal aspartic acid residue (D) in sequence y is aligned with a gap only because the two sequences are not the same length. We can readily understand this outcome if we consider our chosen gap penalty of 8 in the light of the BLOSUM-62 substitution matrix. The worst substitution score in this matrix is –4, significantly less than the gap penalty. Also, many of the scores for aligning identical residues are only 4 or 5. This means that if we set such a high gap penalty, a gap is unlikely to be present in an optimal alignment using this scoring matrix. In these circumstances, gaps will occur if the sequences are of different length and also possibly in the presence of particular residues such as tryptophan or cysteine which have higher scores.

If instead we use a linear gap penalty $g(n_{\text{gap}}) = -4n_{\text{gap}}$, the situation changes, as shown in Figure 5.11, which gives the optimal alignment we expected. Because the gap penalty is less severe, gaps are more likely to be introduced, resulting in a different alignment and a different score. In this particular case, four additional gaps occur, two of which occur within the sequences. The overall alignment score is 7, but this alignment would have scored –13 with the original gap penalty of 8.

This example illustrates the need to match the gap penalty to the substitution matrix used. However, care must be taken in matching these parameters, as the performance also depends on the properties of the sequences being aligned. Different parameters may be optimal when looking at long or short sequences, and depending on the expected sequence similarity (see Figure 4.5 for a practical example).

A simple modification of the algorithm allows the sequences to overlap each other at both ends of the alignment without penalty, often called **semiglobal alignment**. In this way, better global alignments can be obtained for sequences that are not the same length. Instead of applying the gap penalty scores to matrix elements $S_{i,0}$ and $S_{0,j}$, we set these to zero. The remaining elements are calculated as before (Equation EQ5.17). However, instead of traceback beginning at $S_{m,n}$, it starts at the highest-scoring element in the bottom row or right-most column. This is illustrated in Figure 5.12 for the gap penalty $g(n_{\text{gap}}) = -8n_{\text{gap}}$. Note that now the expected alignment is obtained, despite the gap penalty being so high. This modified algorithm gives the same optimal alignment with a gap penalty of $g(n_{\text{gap}}) = -4n_{\text{gap}}$.

The methods presented above are for use when scoring gaps with a linear penalty of the form $g(n_{\text{gap}}) = -n_{\text{gap}}E$. If we wish to differentiate between penalties for starting and extending a gap, using a scoring scheme such as $g(n_{\text{gap}}) = -I - (n_{\text{gap}} - 1)E$, a slightly different algorithm is required. The problem is not simply one of ensuring that we know if a gap is just being started or is being extended. In the previous algorithm, the decision for $S_{i,j}$ could be made without knowing the details of the alignment for any of $S_{i-1,j}$, $S_{i-1,j-1}$, or $S_{i,j-1}$. Now we need to know if these alignments ended with gaps, which means knowing details of their derivation, particularly involving $S_{i-2,j}$ and $S_{i,j-2}$.

Consider the general case of using a gap penalty, which is simply written as a function of the gap length n_{gap}; that is, $g(n_{\text{gap}})$. Now, for any matrix element $S_{i,j}$, one must consider the possibility of arriving at that element directly via insertion of a gap of length up to i in sequence ***x*** or j in sequence ***y***. Some of these routes are illustrated in Figure 5.13. The algorithm now has to be modified to

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} & + \quad s(x_i, y_j) \\ [S_{i-n_{\text{gap}1},j} & + \quad g(n_{\text{gap}1})]_{1 \le n_{\text{gap}1} \le i} \\ [S_{i,j-n_{\text{gap}2}} & + \quad g(n_{\text{gap}2})]_{1 \le n_{\text{gap}2} \le j} \end{cases} \qquad \text{(EQ5.18)}$$

The algorithm now has a number of steps proportional to mn^2, where m and n are the sequence lengths with $n > m$. This is a significant increase in requirements over the original, because there are approximately n terms involving gaps used to evaluate each matrix element. However, when we use the specific affine gap penalty formula of Equation EQ5.14 it is possible to reformulate things and obtain the full matrix in mn steps again. Let us define $V_{i,j}$ as

$$V_{i,j} = \max\left\{S_{i-n_{\text{gap}1},j} + g(n_{\text{gap}1})\right\}_{1 \le n_{\text{gap}1} \le i} \qquad \text{(EQ5.19)}$$

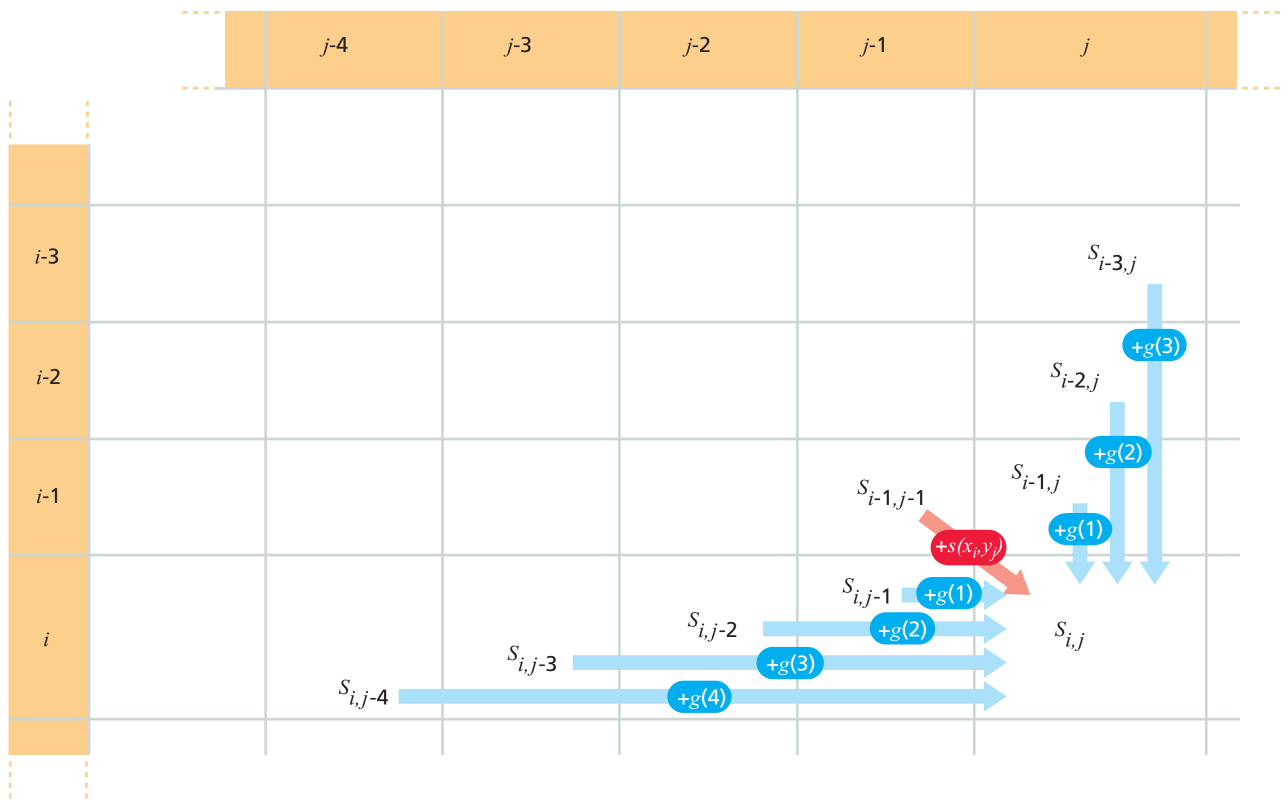


Figure 5.13
Illustration of some of the possible paths that could be involved in calculating the alignment score matrix for a general gap penalty g(n_{gap}). All possible gap sizes must be explored, with $S_{i,j}$ being chosen as the maximum of all these possibilities.

from which

$$V_{i,j} = \max\begin{cases} S_{i-1,j} & + \quad g(1) \\ [S_{i-n_{\text{gap1}},j} & + \quad g(n_{\text{gap1}})]_{2\le n_{\text{gap1}}\le i} \end{cases} \qquad \text{(EQ5.20)}$$

$$= \max\begin{cases} S_{i-1,j} & + \quad g(1) \\ [S_{i-n_{\text{gap1}}-1,j} & + \quad g(n_{\text{gap1}}+1)]_{1\le n_{\text{gap1}}\le i-1} \end{cases} \qquad \text{(EQ5.21)}$$

$$= \max\begin{cases} S_{i-1,j} & - \quad 1 \\ [S_{i-n_{\text{gap1}}-1,j} & + \quad g(n_{\text{gap1}}) - E]_{1\le n_{\text{gap1}}\le i-1} \end{cases} \qquad \text{(EQ5.22)}$$

But from Equation EQ5.19 substituting *i* – 1 for *i*, we can see that

$$V_{i-1,j} = \max\left\{S_{i-n_{\text{gap1}}-1,j} + g(n_{\text{gap1}})\right\}_{1\le n_{\text{gap1}}\le i-1} \qquad \text{(EQ5.23)}$$

so that

$$V_{i,j} = \max\begin{cases} S_{i-1,j} & - \quad I \\ V_{i-1,j} & - \quad E \end{cases} \qquad \text{(EQ5.24)}$$

Thus we have a recursive equation for the elements $V_{i,j}$, involving aligning residues in sequence *x* with gaps in *y*. This can readily be evaluated from a starting point of

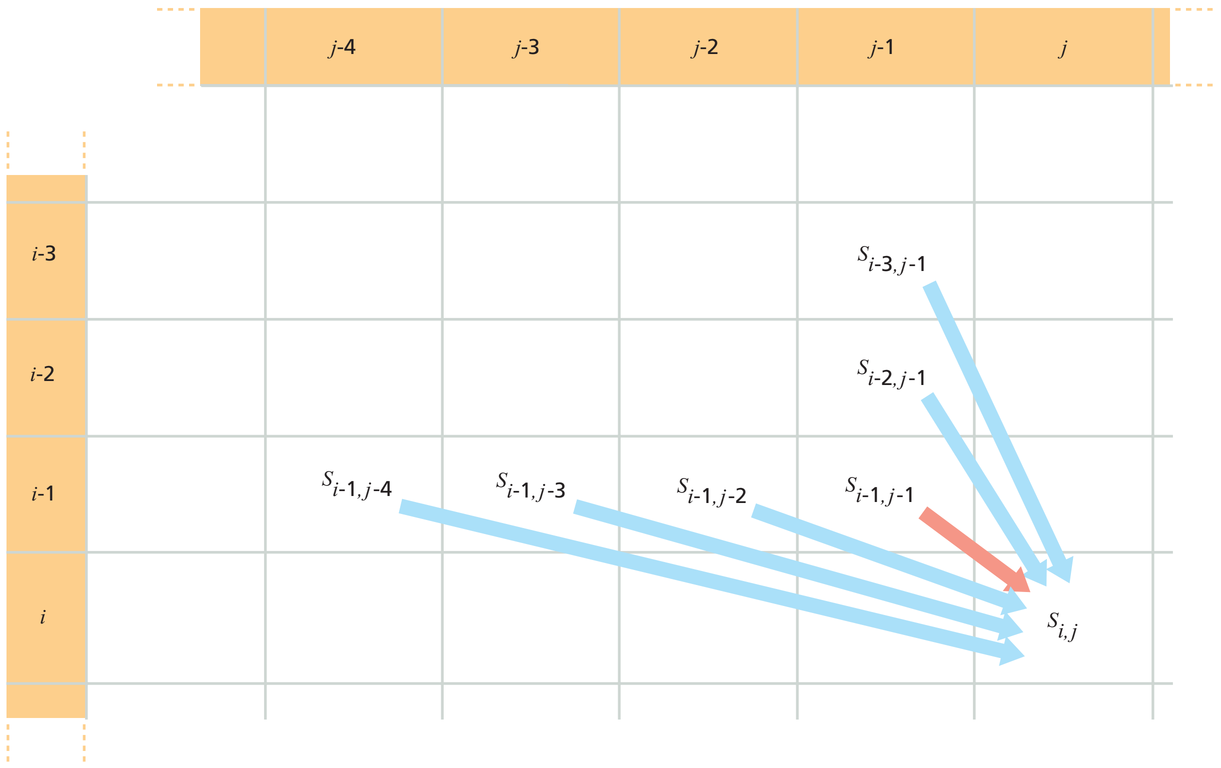

j-4
j-3
j-2
j-1
j
i-3
i-2
i-1
i
$S_{i\text{-}3,j\text{-}1}$
$S_{i\text{-}2,j\text{-}1}$
$S_{i\text{-}1,j\text{-}4}$
$S_{i\text{-}1,j\text{-}3}$
$S_{i\text{-}1,j\text{-}2}$
$S_{i\text{-}1,j\text{-}1}$
$S_{i,j}$

Figure 5.14
Illustration of some of the possible paths that could be involved in calculating the alignment score matrix with the original Needleman and Wunsch algorithm. All possible gap sizes must be explored, with $S_{i,j}$ being chosen as the maximum of all these possibilities. The interpretation differs from the previous matrices in that if a path stops at an element $S_{i,j}$ it indicates that residues x_i and y_j are aligned with each other. Thus the path $S_{i-1,j-2} \rightarrow S_{i,j}$ represents the alignment of x_{i-1}–x_i with $y_{j-2}y_{j-1}y_j$; that is, an insertion in sequence x aligned with y_{j-1}.

$V_{1,j} = S_{0,j} - I$ (Equation EQ5.19 with $i = 1$). In a similar manner, we can define $W_{i,j}$, involving aligning residues in sequence y with gaps in x as

$$W_{i,j} = \max \begin{cases} S_{i,j-1} & - \quad I \\ W_{i,j-1} & - \quad E \end{cases} \tag{EQ5.25}$$

This can readily be evaluated from a starting point of $W_{i,1} = S_{i,0} - I$. These two recursive formulae can be substituted into Equation EQ5.18 to give the faster (nm steps) algorithm

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(x_i, y_i) \\ V_{i,j} \\ W_{i,j} \end{cases} \tag{EQ5.26}$$

It should be noted that the original algorithm of Needleman and Wunsch differs in some key details from those described here. Their method is slower (requiring more computing steps) and is rarely used today. The types of path involved in calculating the matrix elements with their algorithm are shown in Figure 5.14. Note that the interpretation of the paths through the matrix differs from that presented above (see the figure legend for details).

Local and suboptimal alignments can be produced by making small modifications to the dynamic programming algorithm

Often we do not expect the whole of one sequence to align well with the other. For example, the proteins may have just one domain in common, in which case we

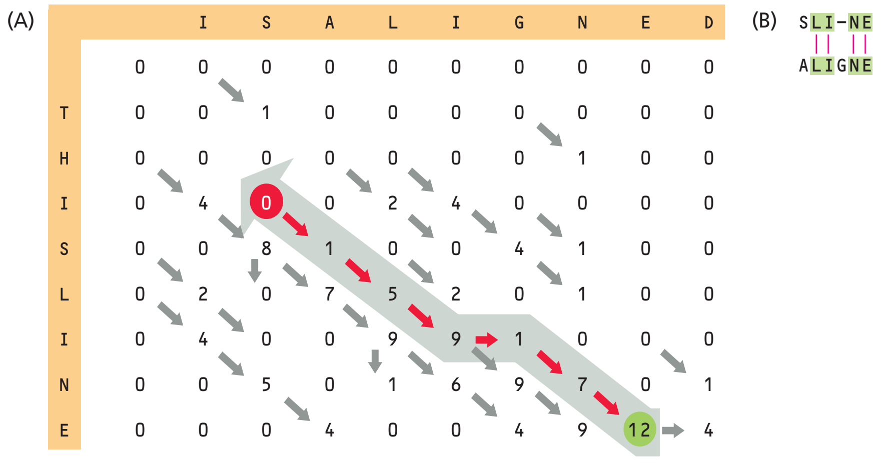


Figure 5.15
The dynamic programming calculation for determining the optimal local alignment of the two sequences THISLINE and ISALIGNED. (A) The completed matrix using the BLOSUM-62 scoring matrix with a linear gap penalty, defined in Equation EQ5.13 with *E* set to –8. (B) The optimal alignment, determined by the highest-scoring element, which has a score of 12.

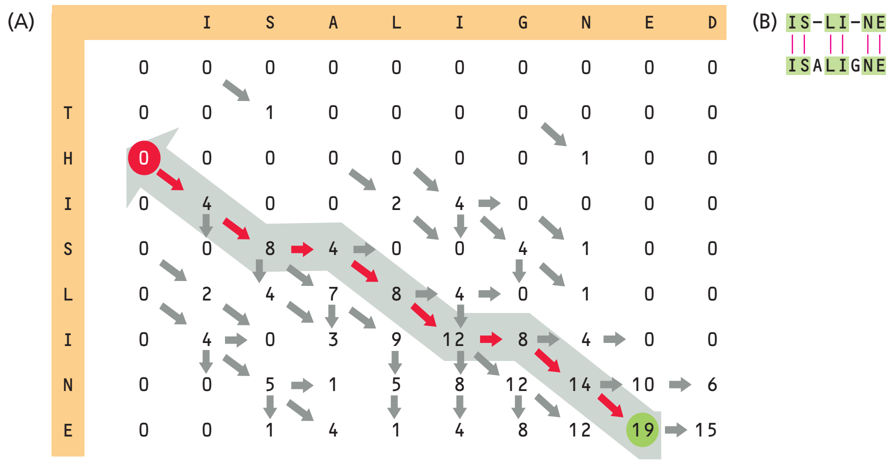


Figure 5.16
Optimal local alignment calculation identical to Figure 5.15, except with a linear gap penalty with *E* set to –4. (A) The completed matrix for determining the optimal local alignment of THISLINE and ISALIGNED using the BLOSUM-62 scoring matrix. (B) The optimal alignment, identified by the highest-scoring element in the entire matrix, which has a score of 19.

want to find this high-scoring zone, referred to as a local alignment (see Section 4.5). In a global alignment, those regions of the sequences that differ substantially will often obscure the good agreement over a limited stretch. The local alignment will identify these stretches while ignoring the weaker alignment scores elsewhere.

It turns out that a very similar dynamic programming algorithm to that described above for global alignments can obtain a local alignment. Smith and Waterman first proposed this method. However, it should be noted that the method presented here requires a similarity-scoring scheme that has an expected negative value for random alignments and positive value for highly similar sequences. Most of the commonly used substitution matrices fulfill this condition. Note that the global alignment schemes have no such restriction, and can have all substitution matrix scores positive. Under such a scheme, scores will grow steadily larger as the alignment gets larger, regardless of the degree of similarity, so that long random alignments will ultimately be indistinguishable by score alone from short significant ones.

The key difference in the local alignment algorithm from the global alignment algorithm set out above is that whenever the score of the optimal sub-sequence alignment is less than zero it is rejected, and that matrix element is set to zero. The scoring scheme must give a positive score for aligning (at least some) identical residues. We would expect to be able to find at least one such match in any alignment worth considering, so that we can be sure that there should be some positive alignment scores. Another algorithmic difference is that we now start traceback from the highest-scoring matrix element wherever it occurs.

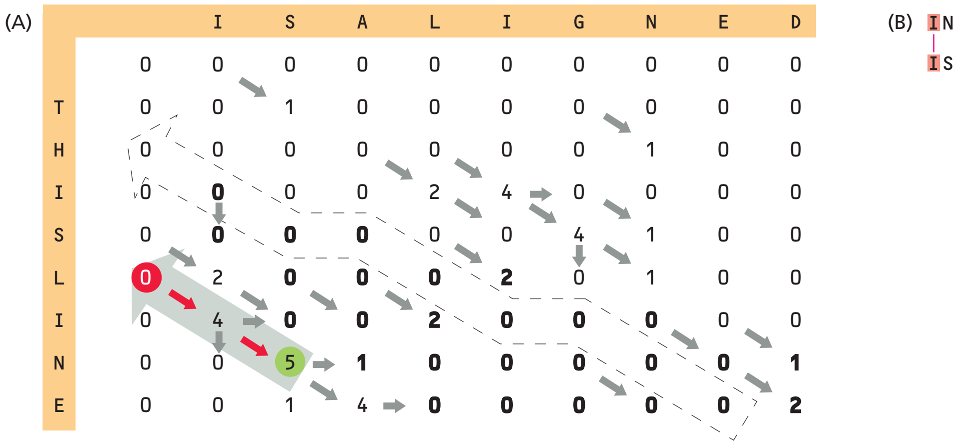


Figure 5.17
The dynamic programming calculation that follows on from the calculation shown in Figure 5.16 to find the best suboptimal local alignment. (A) The completed matrix for determining the best suboptimal local alignment of THISLINE and ISALIGNED using the BLOSUM-62 scoring matrix with a linear gap penalty with *E* set to –4. The matrix elements that were involved in the optimal local alignment have been set to zero and are shown with bold font. The matrix elements that have changed value from Figure 5.16(A) are also shown with bold font, and extend below and to the right of the optimal alignment. (B) The best suboptimal alignment, identified by the highest-scoring element in the entire matrix, has a score of 5.

The extra condition on the matrix elements means that the values of $S_{i,0}$ and $S_{0,j}$ are set to zero, as was the case for global alignments without end gap penalties. The formula for the general matrix element $S_{i,j}$ with a general gap penalty function $g(n_{\text{gap}})$ is

$$S_{i,j} = \max\begin{cases} S_{i-1,j-1} & + \quad s(x_i, y_i) \\ [S_{i-n_{\text{gap}1},j} & + \quad g(n_{\text{gap}1})]_{1\le n_{\text{gap}1}\le i} \\ [S_{i,j-n_{\text{gap}2}} & + \quad g(n_{\text{gap}2})]_{1\le n_{\text{gap}2}\le j} \\ & 0 \end{cases} \qquad \text{(EQ5.27)}$$

which only differs from Equation EQ5.18 by the inclusion of the zero. The same modifications as above can be applied for the cases of linear gap penalty given in Equation EQ5.13 and affine gap penalty given in Equation EQ5.14.

Figures 5.15 and 5.16 show the optimal local alignments for our usual example in the two cases of linear gap penalties $g(n_{\text{gap}}) = -8n_{\text{gap}}$ and $-4n_{\text{gap}}$, respectively. Both result in removal of the differing ends of the sequences. In the first case, the higher gap penalty forces an alignment of serine (S) and alanine (A) in preference to adding a gap to reach the identical IS sub-sequence. Lowering the gap penalty in this instance improves the result to give the local alignment we would expect.

Sometimes it is of interest to find other high-scoring local alignments. A common instance would be the presence of repeats in a sequence. There will usually be a number of alternative local alignments in the vicinity of the optimal one, with only slightly lower scores. These will have a high degree of overlap with the optimal alignment, however, and contain little, if any, extra information beyond that given by the optimal local alignment. Of more interest are those suboptimal local alignments that are quite distinct from the optimal one. Usually their distinctness is defined as not sharing any aligned residue pairs.

An efficient method has been proposed for finding distinct suboptimal local alignments. These are alignments in which no aligned pair of residues is also found aligned in the optimal or other suboptimal alignments. They can be very useful in a variety of situations such as aligning multidomain proteins. Sometimes a pair of proteins has two or more domains in common but other regions with no similarity. In such cases it is useful to obtain separate local alignments for each domain, but only one of these will give the optimal score, the others being suboptimal alignments. The method starts as before by calculating the optimal local alignment. Then, to ensure that any new alignment found does not share any aligned residues

Box 5.1 Saving space by throwing away the intermediate calculations

In this chapter the steps required to identify optimal sequence alignments have been specified, but there has been no discussion of the precise coding used in a program. Such details are in general beyond the scope of this book, although important for the production of efficient and practical tools. The computer science and other specialist texts cited in Further Reading at the end of the chapter should be consulted for details of efficient coding techniques. However, we will briefly examine one algorithmic trick that can substantially increase the range of alignment problems feasible with limited computer resources.

All the methods presented so far calculate the whole of matrix $\boldsymbol{S}$, and by default it might be assumed that the entire matrix was stored for traceback analysis. If the sequences to be compared have lengths of up to a few thousand residues, the matrix can be stored quite easily on current computers. However, particularly in the case of nucleotide sequences, we might wish to align much longer sequences, for example mitochondrial DNA and even whole genomes. Storage of the whole matrix $\boldsymbol{S}$ is often not possible for such long sequences, sometimes requiring gigabytes of memory. Without using an alternative, this memory problem could prevent the use of these dynamic programming methods on such data. However, a neat solution is available by modifying the algorithm to store just two rows of the matrix. Note that further calculation is then needed to recover the actual alignment, as the basic method presented here only provides the optimal alignment score.

The key to this algorithm is to notice that the calculation of any element only requires knowledge of the results for the current and previous row (see Figure 5.10 and Equations EQ5.17 and EQ5.26); we could have chosen to work with columns instead. The two rows will be labeled $R_{0,j}$ and $R_{1,j}$. The scoring scheme used here will involve the linear gap penalty of Equation EQ5.13, but the affine gap penalty scheme can also be used. Similarly, the different variations in the treatment of end gaps can be incorporated.

The steps can be summarized as follows. Firstly, initialize row 0 according to the specific algorithm. For standard Needleman–Wunsch with a linear gap penalty, this means setting $R_{0,j} = -jE$ for $j = 0 \rightarrow$ m. For the 0th column, $R_{1,0}$ is then set to $-E$ or whatever other boundary conditions are required. This is equivalent to the initialization of the full matrix method above. We now step through each residue x_i of sequence x, from x_1 to x_n, calculating all the elements of row $R_{1,j}$, which at each step is the equivalent of the ith row in the full matrix. These elements are labeled with the letter j, and correspond to residue y_j of sequence y. Thus the elements of the $R_{0,j}$ row correspond to the matrix elements $S_{i-1,j}$, and those of the $R_{1,j}$ row to $S_{i,j}$. The other elements of $R_{1,j}$ are assigned a value according to the equation (equivalent to Equation EQ15.17)

$$R_{1,j} = \max \begin{cases} R_{0,j-1} + s\left(x_i, y_j\right) \\ R_{0,j} - E \\ R_{1,j-1} - E \end{cases} \qquad \text{(BEQ5.1)}$$

Once all the elements of the $R_{1,j}$ row have been calculated, the value of each matrix element of $R_{1,j}$ is transferred to $R_{0,j}$, which now represents matrix row 1. In practical programming, pointers would be used, so that this step would take virtually no time. For the 0th column, $R_{1,0}$ is then set to $-2E$ or whatever other boundary conditions are required, with the $R_{1,j}$ elements now representing matrix row 2. For the other columns, $R_{1,j}$ is assigned a value as before. In this way results keep being overwritten, but sufficient are kept to continue the calculation.

Certain scores must be saved, the details differing according to the precise type of alignment sought. Thus for the basic Needleman–Wunsch scheme, only $R_{m,n}$ need be stored, while Smith–Waterman requires the highest-scoring element and the associated values of i and j. The storage requirements are twice the length of the shorter sequence. Because this technique only stores two rows of the matrix, it makes it possible to use dynamic programming to align complete bacterial genomes.

However, the saving in memory required comes at a price, and the traceback procedure is much more complicated than that which can be used if the full matrix calculation has been stored. The traceback now involves considerably more calculation and so production of the overall alignment will require longer computing times. Versions of this technique exist for all the types of alignment mentioned in this chapter. Over the past few years much effort has been devoted to optimizing alignment programs for use with whole genome sequences. See Section 5.5 for more practical methods concerning whole genome sequence alignment. In contrast to the methods above, the methods discussed in that section all involve approximations, but a good argument can be made that such techniques are more appropriate for that type of problem.

with the optimal alignment, it is necessary to set all the matrix elements on this initial path to zero (see Figure 5.17). Subsequently, the matrix needs to be recalculated to include the effect of these zeroed elements. If we recalculate the matrix, only a relatively small number of elements near the optimal local alignment will have new values. The key to the efficiency of the technique is the recognition that these modified elements can only affect elements to their right and below them (see Figure 5.13). Working along a matrix row from the zeroed elements, the new values are monitored until an element is found whose value is unchanged from the original calculation. Further elements on this row will also be unchanged, so calculation can now move to the next row down. In this way the new matrix is obtained with a minimum of work. The suboptimal local alignment is identified by locating the highest matrix element. Further suboptimal alignments can be found by repeating the procedure, zeroing every element involved in a previous alignment. For affine gap penalties the alignment elements in the matrices ***V*** and ***W*** are also forced to zero and must also be unchanged before a row calculation can stop.

In the example shown in Figure 5.17, the first suboptimal local alignment is found for the linear gap penalty $g(n_{\text{gap}}) = -4n_{\text{gap}}$; that is, following on from Figure 5.16. All the matrix elements of the optimal alignment are in bold font, as are those elements that have been recalculated. Only 27 elements needed to be recalculated, all below or to the left of the elements of the optimal alignment. If we had not monitored for unchanged elements we would have had to recalculate 54 elements.

Time can be saved with a loss of rigor by not calculating the whole matrix

High-scoring local alignments indicating significant sequence similarity usually do not have many insertions and deletions. Global alignments may contain large insertions, for example if there is a whole domain inserted in one sequence, but such situations are only rarely encountered. In terms of the matrix, this means that these alignments generally follow a diagonal quite closely. This has led people to try to save time in deriving alignments by only calculating matrix elements within a

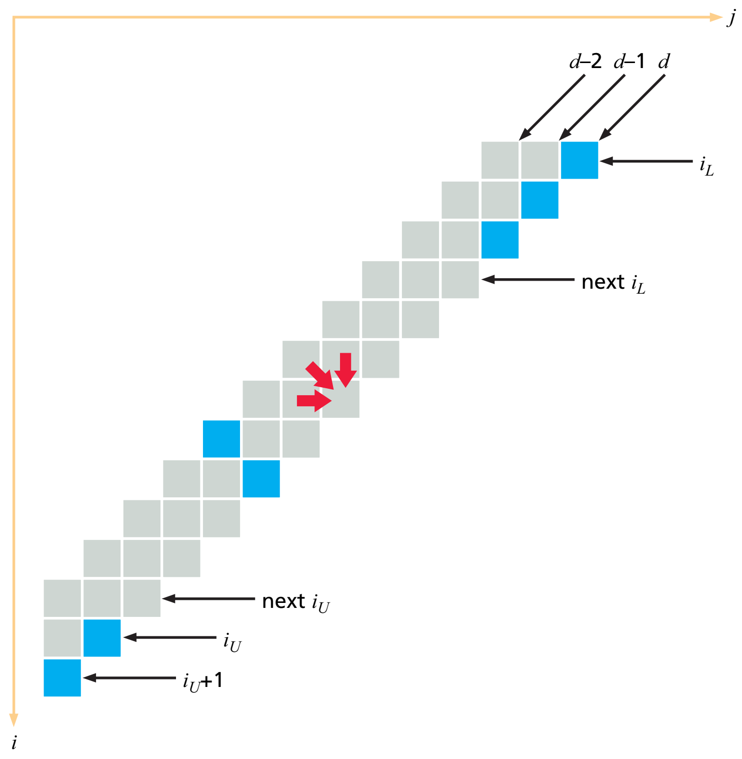


Figure 5.18
The *X*-drop method, proceeding by antidiagonals indexed by *d* = *i* + *j*. The boxes shaded in blue are matrix elements which have a score that falls *X* below the current best score, and therefore have been assigned a value of $-\infty$. The normal three possible paths used to determine the score are shown in one case by red arrows. (Adapted from Z. Zhang et al., A greedy algorithm for aligning DNA sequences, *J. Comput. Biol.* 7 (1–2):203–214, 2000.)

Electronic rights could not be obtained for this image.

Figure 5.19
The matrix elements calculated in the BLAST program using the *X*-drop method for the example of aligning broad bean leghemoglobin I and horse β-globin. The value of X used was 40, with BLOSUM-62 substitution matrix and an affine gap penalty of $g(n_{\text{gap}}) = -10 - n_{\text{gap}}$. The alignment starts at $S_{60,62}$, alanine in both sequences, which is the location determined by the process illustrated in Figure 5.24. Calculated elements are shown black. (From S.F. Altschul et al., Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *N.A.R.* 25 (17):3389–3402, 1997, by permission of Oxford University Press.)

short range of a specified diagonal. Note that such techniques are not guaranteed to find the optimal alignment!

There are two ways of assigning the diagonal around which the alignment will be sought. One can assume that both sequences are similar at a global level, and use the diagonal $S_{i,i}$ of the matrix. Alternatively, especially in the case of database searches, one can use the diagonal of a high-scoring ungapped local alignment found in an initial alignment search, as discussed later.

There are two ways of restricting the coverage of the matrix away from the central diagonal. If a limit is imposed for the maximum difference M_{ins} between the number of insertions in each of the two sequences, the algorithm can simply be set to include only M_{ins} diagonals on either side of the central one; that is $(2M_{\text{ins}} + 1)$ diagonals in total. The database search program FASTA (see Section 4.6 and below) uses this technique, especially for nucleotide sequences, when M_{ins} is frequently set to 15. A second method of restricting the matrix elements examined is to limit the search according to how far the scores fall below the current best score. In this ***X*-drop method**, used in the database searching program BLAST (see Section 4.6 and below), the current best score is monitored. When calculating elements along a row, calculation stops when an element scores a preset value X below this best score, and calculation restarts on the next row. A large value of X results in more of the matrix being evaluated, in which case the true optimal alignment is more likely to be determined.

The X-drop algorithm that follows aligns two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ of length m and n, respectively. The matrix elements are processed for each antidiagonal in turn. The dth antidiagonal is defined by $d = i + j$. Only a restricted region of the dth antidiagonal is evaluated, defined by variables i_L and i_U, as illustrated in Figure 5.18. The current best alignment score is stored in the variable S_{best}.

In some cases, such as occur in the database search methods described in Section 5.3, the algorithm is started at element S_{i_0,j_0}, which has been identified by a preliminary local alignment step. In this instance S_{best} is initially set to the value of element S_{i_0,j_0}, and the other variables are initialized to $d = i_0 + j_0$ and $i_L = i_U = i_0$. Alternatively, the algorithm can be started at element $S_{0,0}$, which has the value zero, so that S_{best} is initially set to zero, as are d, i_L, and i_U. With these initial values, the algorithm proceeds as follows.

Each antidiagonal is evaluated in turn. We will describe the method as proceeding from smaller to larger values of d. The changes required to proceed to decreasing values of d will be discussed afterwards. If there are elements of the antidiagonal to calculate, residues x_i are evaluated in order of increasing i, from i_L to $i_U + 1$. For each value of i the relevant j for this antidiagonal can be obtained using $j = d - i$. This identifies the matrix element $S_{i,d-i}$ under consideration. Initial evaluation is as described previously, for example, Equation EQ5.17 when a linear gap penalty is used.

After this initial evaluation of the antidiagonal elements, a check is first made to see if the score of any of these elements improves on the best score so far, in which case S_{best} is set to this value. The antidiagonal elements are then evaluated to identify any that fall more than X below this current best score, S_{best}. All elements that have such low scores are assigned the value $-\infty$, so that they play no further part in later calculations.

In the next step, i_L and i_U are redetermined. i_L is chosen as the lowest i in the selected region of the current antidiagonal such that $S_{i,j} \neq -\infty$. i_U is set to the highest i in the selected region of the current antidiagonal such that $S_{i,j} \neq -\infty$. Sometimes this will result in a smaller region being defined for the next antidiagonal, the situation illustrated in Figure 5.18. When the elements at the edge of the antidiagonal region do not have the value $-\infty$ the region must be extended. This extension might be based on the calculation of scores for further elements of the antidiagonal, or may be limited to a prespecified number of extra elements. If $i_U + 1 \le i_L$ there are no matrix elements in the selected region, and the calculation is finished. Otherwise, the next antidiagonal, that is, the $(d + 1)$th, is now evaluated.

If the calculation started at an element S_{i_0,j_0}, that is not $S_{0,0}$, it must be carried out in reverse as well to trace the alignment toward the start of the sequences. Note that there is no difference in principle between using dynamic programming to find optimal alignments forward or backward. Some indices need to be changed, however, as for example, element $S_{i,j}$ now depends on the values of $S_{i+1,j+1}$, $S_{i+1,j}$, and $S_{i,j+1}$. The score of such an element $S_{i,j}$ now relates to an alignment starting with residues x_i and y_j and ending at x_u and y_v. The full alignment is found by two trace-backs, one from the forward and one from the backward region of the matrix, both of which end at S_{i_0,j_0}. Figure 5.19 shows an example of the matrix elements calculated for a real alignment by an algorithm like this.

5.3 Indexing Techniques and Algorithmic Approximations

The huge increase in the size of the sequence databases and their daily updating has obliged the general research community to access these databases through central facilities. This makes it easier for people to be confident of searching all known data, but serves to concentrate the demands for similarity searches on a few machines. Even though the power of computers has greatly increased over the years, the methods of full-matrix dynamic programming described above are too demanding for general use in database searches.

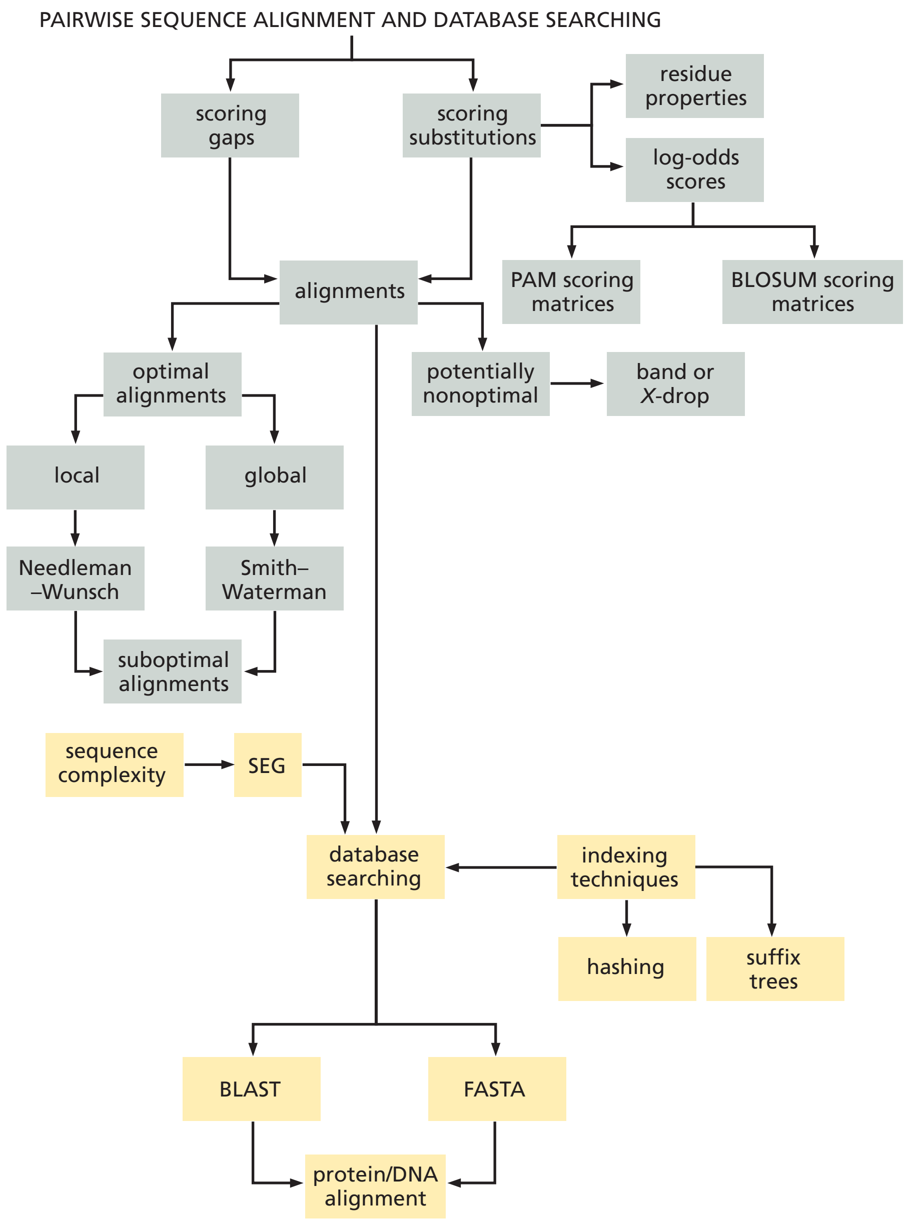
A number of alternative procedures have been developed that are considerably faster, although there is a penalty to pay in that they do not guarantee to find the highest-scoring alignment. The key to their success is the use of **indexing techniques** to locate possible high-scoring short local alignments. These initial local alignments are then extended, subject to certain constraints, to provide scores that are used to rank the database sequences by similarity. In most implementations, modified dynamic programming algorithms are used to examine the best-scoring sequences and to produce final scores and alignments.

In the following sections we will examine in detail the two major methods implemented in the two program suites in widest use today: BLAST and FASTA. These both start by considering very short segments of sequence that they call “words,” “k-tuples,” or “k-mers.” A k-tuple (as used in FASTA) is simply a stretch of *k* residues in the query sequence. A k-mer (as used in BLAST) is a stretch of *k* residues, which when aligned with all *k* residue stretches of the query sequence will score above some threshold value (*T*) at least once. The term “word” is used more generally, meaning simply any short sub-sequence. The initial steps of both methods find high-scoring ungapped local alignments, referred to in BLAST as **high-scoring segment pairs** (**HSPs**), of which the highest-scoring one for a given pair of sequences is the **maximal segment pair** (**MSP**). Flow Diagram 5.3 gives an outline of the steps covered in this section.

Suffix trees locate the positions of repeats and unique sequences

One method of indexing uses a device known as the **suffix tree**. A variant of this technique is used in the BLAST programs. The example given here will be for a nucleotide sequence because an example using a realistic protein sequence would be too complex to illustrate the method. Consider a short segment of a nucleotide sequence ATCCGAGGATATCGA$, where the $ is used to identify the end of the sequence. This has a number of short repeats, such as AT. A suffix tree is a way of

Flow Diagram 5.3
The key concept introduced in this section is that database sequence similarity searches require refinements in the pairwise alignment methods to make them more efficient, although at the cost of decreasing the chance of finding the best-scoring alignment.



representing the sequence that uses these repeats to reduce the space needed for a full description. In addition, the form of the tree makes it very easy to find specific sequences and sequence repeats.

A **suffix** is defined as the shortest sub-sequence starting at a particular position that is unique in the complete sequence and can therefore be used to clearly identify that position. For example, the third base in the sequence above is C. As there are several Cs in the whole sequence, the sub-sequence C is not sufficiently unique to label position 3. However, by including the next base, as in CC, we have found a unique sub-sequence, and the suffix at position 3 is CC.

The suffix tree for the example sequence is given in Figure 5.20. Looking at it you can easily see that the longest repeats are the triplets ATC and CGA, which occur at positions (1,11) and (4,13), respectively. The efficiency of this technique may not be so apparent with this example because the sequence is rather short. Although longer sequences will tend to have longer suffixes, they will also tend to have more repeats, resulting in a more efficient tree.

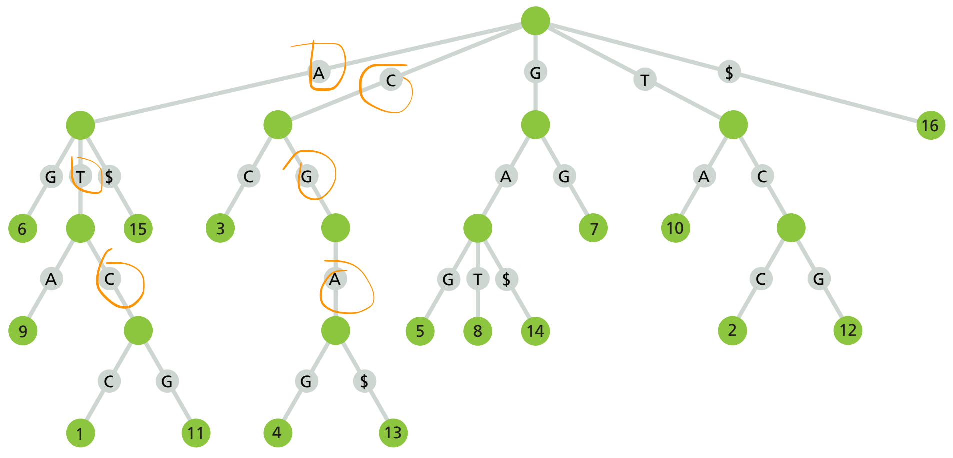


Figure 5.20
The suffix tree for the sequence ATCCGAGGATATCGA$. There are alternative ways of showing and labeling suffix trees, and often they are labeled with the whole suffix, but this form has been used because parallels can then be seen readily with finite-state automata (Figure 5.23). The numbered terminal nodes (leaves) correspond with the numbered positions in the sequence. Thus 6 refers to the sixth position, at which the base is A. The suffix for position 6 is AG. All suffix positions are clustered according to the 5′-terminal part of their sequence (or amino terminal if dealing with a protein sequence), grouping together all suffixes starting with a G for example.

Constructing the tree is straightforward. First, all positions in the sequence are grouped according to their base type; for a protein sequence this would be amino acid residue type. These groups correspond to the first row of nodes from the root. Each of these groups is then regrouped according to the following base to give the second row of nodes. This procedure is continued, stopping for a group when it only contains one sequence position. For a sequence of length L, this method requires a number of steps proportional to $L \log L$. Faster methods, requiring a number of steps proportional to L, are known but are more involved.

Hashing is an indexing technique that lists the starting positions of all k-tuples

The basic aim of hashing is to construct a list of the starting positions of all k-tuples that occur in a query sequence. If we subsequently want to find where a particular k-tuple occurs in the sequence we simply look up the list. This procedure is used in FASTA.

Before construction of the list can begin we need to create a code for each k-tuple. Suppose we are dealing with nucleotide sequences, and k-tuples of length $k = 3$. Because there are four possible bases, there are 4^3 (= 64) possible k-tuples, trinucleotides in this case. We can easily create a number code for these. First, assign each base a number from 0 to 3, for example A = 0, C = 1, G = 2, and T = 3. If we label this variable e, any 3-tuple $x_i x_{i+1} x_{i+2}$ can be assigned a number (c_i) according to the formula

$$c_i = e(x_i)4^2 + e(x_{i+1})4^1 + e(x_{i+2})4^0 \qquad \text{(EQ5.28)}$$

For example, the trinucleotides AAA, CAA, ACA, and AAC would be assigned the numbers 0, 16, 4, and 1, respectively. The c_i will vary from 0 to 63 (= $4^3 - 1$). As an example, the sequence TAAAACTCTAAC has 10 trinucleotides with c_1 to c_{10} given by 48, 0, 0, 1, 7, 31, 55, 28, 48, and 1, respectively. In the case of protein sequences, the amino acids would be numbered from 0 to 19, and instead of using powers of 4, powers of 20 would be used.

If the sequence in question is of length L, there will be $(L - k + 1)$ k-tuples that will be coded into the values of c_1 to c_{n-k+1}. Because we ultimately want to search for particular k-tuples, which means finding particular values of the c_i, we need to sort the c_i into numerical order. For the example given above, the ordering will be c_2, c_3, c_4, c_{10}, c_5, c_8, c_6, c_1, c_9, c_7. There are many textbooks on numerical algorithms that give details of efficient sorting methods, so they will not be discussed here (see Further Reading).

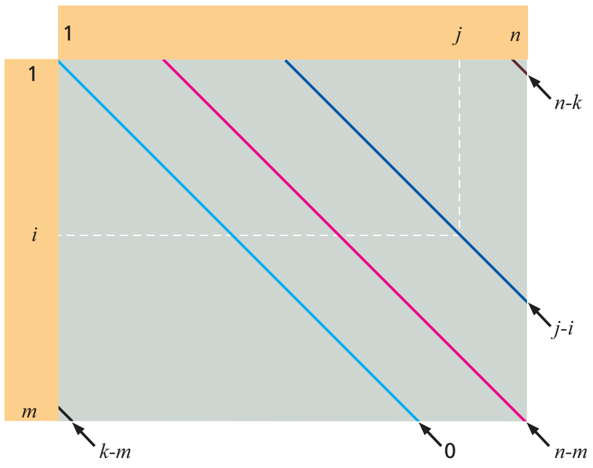


Figure 5.21
Definition of the labeling of matrix diagonals $\mathbf{d}_{j-i}$. A word length of k is used, so the last $(k-1)$ elements of each row and column are not filled. This is why the diagonal labels range from $k-m$ to $n-k$.

For each different k-tuple we need to know whether it occurs, and if so at which base(s) it starts. There are several ways of doing this, of which a particularly efficient one is **chaining**. For sequence of length L, two arrays are required: one, $\boldsymbol{a}$, as long as the number of different possible k-tuples (4^k for nucleotides) and one, $\boldsymbol{b}$, of length $(L-k+1)$. The element a_i contains a pointer to the first base of the first occurrence of the ith k-tuple if it exists in the sequence, or else a value that signifies its absence. Suppose that this pointer is to base x_j. In that case, b_j contains a pointer to x_k, the first base of the second instance of the ith k-tuple, or else a value that signifies there is no second instance. If there is an x_k in b_j, then b_k will contain information about the presence of a further instance of the ith k-tuple.

The first eight elements of $\boldsymbol{a}$ for the example sequence TAAAACTCTAAC will be (2, 4, 0, 0, 0, 0, 0, 5), where a value 0 indicates the absence of that particular 3-tuple. The elements of $\boldsymbol{b}$ are (9, 3, 0, 10, 0, 0, 0, 0, 0, 0). This contains three pointers because three 3-tuples are present more than once in the sequence. It can be seen that no 3-tuple occurs more than twice, because these pointers in $\boldsymbol{b}$ are to elements that contain 0. Hashing and chaining techniques are used to great effect in the FASTA programs, as we discuss next.

The FASTA algorithm uses hashing and chaining for fast database searching

A series of programs have been written by William Pearson and colleagues that allow fast and accurate searching of both protein and nucleic acid databases with both protein and nucleotide query sequences. They are based on a very fast heuristic algorithm that has four distinct steps, which are applied to each database sequence independently. In the first step, local ungapped alignments of k-tuples are located. These are then scored using a standard scoring scheme, and only the highest-scoring aligned regions are retained. Still retaining the ungapped regions, an attempt is then made to join these into a single crude alignment for the pair of sequences, resulting in an initial alignment score. This score is used to rank the database sequences. In the final step, the highest-scoring sequences are aligned using dynamic programming. Depending on the program parameters selected, this may use only a band of the full matrix including the region containing the crude alignment. We will now describe some of these steps in more detail.

FASTA starts by hashing and chaining the query sequence. A parameter, ktup, gives the size of the k-tuples to be hashed, and is usually set to 2 amino acids for proteins and 6 bases for nucleotides. Note that these values produce 400 and 4096 different possible k-tuples, respectively. As the program will search for k-tuples shared by both sequences, using smaller values of the ktup parameter makes the search more

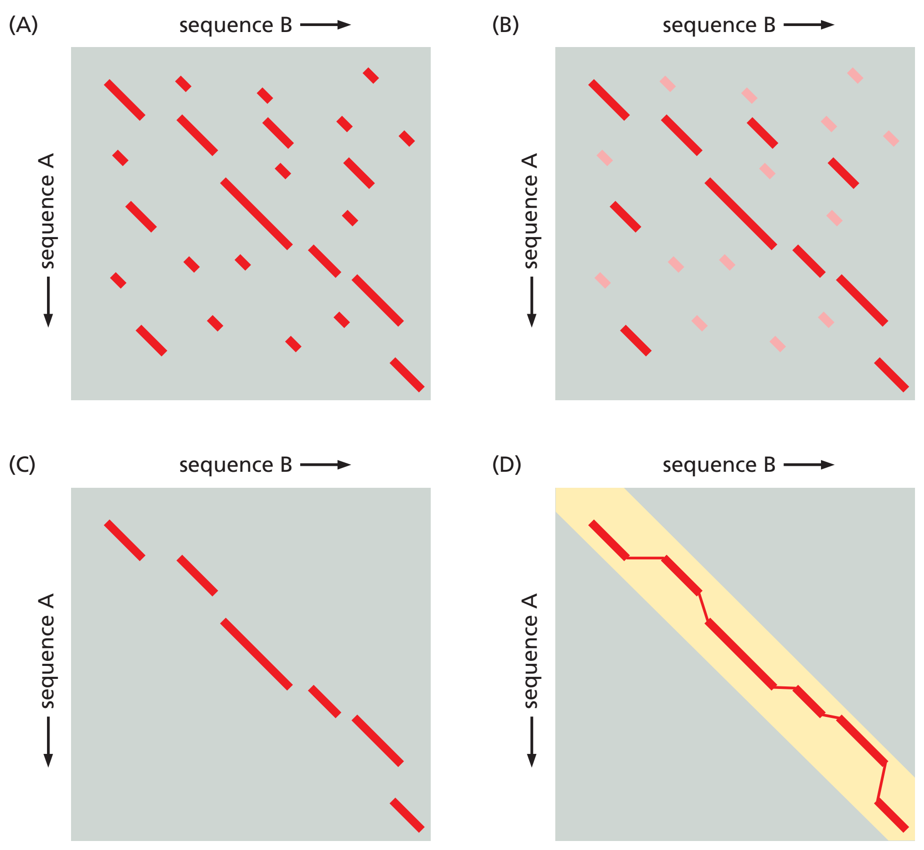


Figure 5.22
The four steps in the FASTA method for database searching. (A) For a given database sequence, all ungapped local alignments (against the query sequence) of suitably high score are shown. (B) The ten highest-scoring alignments are rescored with the PAM250 matrix, the highest having score init1. (C) Attempts are made to join together some of these ungapped alignments, allowing some gaps, to obtain score initn. (D) Dynamic programming is used to extend the alignment and give the final score opt.

sensitive. However, this is usually only beneficial for distantly related sequences, and searches based on family profiles (see Sections 4.6 and 6.1) would be expected to be even more sensitive.

In what follows, we will assume that the chaining arrays a and b have been calculated as described in the previous section. For each database entry, successive k-tuples of the sequence are searched against these arrays to identify where identical k-tuples align in the two sequences. As these alignments are ungapped, they lie on diagonals in an alignment matrix (such as the one in Figure 5.11). We want to find those diagonals that have many such k-tuples aligned along them.

Before giving the details of the search algorithm, a few comments about labeling diagonals in an alignment matrix will be helpful. As in the alignment matrices shown in Section 5.2, the alignment diagonals descend to the right. Suppose the two sequences have m and n residues, respectively. Remembering that there are $(m - k)$ k-tuples in an m residue sequence, there will be $(n + m - 2k + 1)$ diagonals, which can be labeled d_{k-m} to d_{n-k} (see Figure 5.21).

With this diagonal labeling, if the same k-tuple is located at position i of the query sequence and at position j of the database sequence, the alignment of these k-tuples will lie on the diagonal d_{j-i}. The chaining arrays make it simple to determine all the identical k-tuples, and on which diagonals they lie. All such aligned k-tuples are given the same (positive) score s. We just need to account for residues between these aligned k-tuples to complete the scoring of diagonals. These can be scored with a penalty $g(l)$ proportional to their length l. Using these scores, a straightforward algorithm finds the highest-scoring local regions of the diagonals. This is related to the Smith–Waterman algorithm, although as there are only matching residues it is much simpler (and quicker). A score is maintained for each diagonal of the alignment matrix. Let this score be s_{j-i} for diagonal d_{j-i}, and suppose the last k-tuple match on this diagonal was at location j_0. As the database sequence is scanned, another k-tuple match is found on d_{j-i}. The score of the diagonal is updated according to the formula

$$s_{j-i} = s + \max \begin{cases} s_{j-i} & + \quad g(l) \\ & 0 \end{cases} \tag{EQ5.29}$$

The value of l is calculated using j_0, and j_0 is updated to the current value of j. Note the zero alternative, as in Equation EQ5.27, which makes this a local alignment search method. The locations of the highest-scoring local alignments are recorded, so that more than one region of the same diagonal can be found. An example of the result of this step can be seen in Figure 5.22A.

Current implementations of this scheme identify the 10 highest-scoring ungapped alignments for each particular pair of sequences and use them in further analysis. In assessing these alignments, no account has yet been taken of conservative replacements (if comparing amino acid sequences) or even of identical matches in runs shorter than k residues. We now address this shortcoming by rescoring these 10 alignments using a substitution matrix such as PAM250, thus generating a more reasonable score. In the case of nucleotide sequences, matches and mismatches are by default scored +5 and –4, respectively. This stage is shown in Figure 5.22B.

Early versions of this program (called FASTP for protein sequences and FASTN for nucleotides) ranked all the database sequences according to the highest-scoring local alignment, the score being referred to as "init1." The later FASTA versions first combine some of the top-scoring alignments into a single longer alignment using a simple dynamic programming technique. The scoring scheme for this technique is based on the scores of the individual alignments and a joining penalty to score the regions between them. The resultant alignment score is called "initn," and is used to make a preliminary ranking of the database sequences. This stage is shown in Figure 5.22C.

In the last step, suitably high-scoring database sequences are further investigated with a Smith–Waterman local alignment procedure to produce the final alignment and score. This will introduce gaps to give the best alignment. In most cases the older versions of the program used a banded version of the algorithm, centered on the initial approximate alignment. In general, a band of 16-residue width was used, except in the case of protein sequence alignments with 1-tuple indexing, when a 32-residue band was employed. This stage is shown in Figure 5.22D. In later versions, such as FASTA3, all alignments of protein sequences use the full-matrix Smith–Waterman method by default. The default for nucleotide sequences is still to use the banded version, as the full-matrix method is very time consuming for the longer sequences.

The resultant alignment score, opt, is used for the final sequence ranking, but the ranking itself is according to the estimated significance of the score. This is based on theories such as the extreme-value distribution discussed in the final part of the chapter. The significance estimates involve the sequence lengths as well as the score, and thus the ranking reported can differ from that based directly on opt. By default, the alignments reported by FASTA are those given by the Smith–Waterman method, and therefore may contain gaps.

By restricting the initial search to ungapped perfect matches, joining these together in a simple way, and using the resulting score to filter out very dissimilar database sequences, FASTA can achieve a considerable speed-up in database searches over a straightforward application of the Smith–Waterman method. Furthermore this has been achieved for a very small loss of sensitivity.

The BLAST algorithm makes use of finite-state automata

To evaluate the alignment of a database sequence with the query sequence, one needs to know the significance of its score relative to that expected for a random sequence. It proved very hard to derive a theory from which the significance could be calculated. The inclusion of gaps in alignments proved to be one of the major complications. In 1990, Samuel Karlin and co-workers derived a theory for ungapped local alignment scores. The BLAST programs were written to take advantage of the rigorous scoring significance estimates that could now be derived for ungapped local alignments.

For this reason the original program did not consider gaps at all, and reported one or more local ungapped alignments per pair of sequences. Subsequent versions of BLAST allow gaps in alignments after an initial search without them, so that currently BLAST will produce a final local gapped alignment. In this respect the latest versions of BLAST are similar to FASTA.

The initial stage of BLAST uses short words to search for identities in the database sequence. It differs from FASTA in two respects. First, FASTA only looks for k-tuples that are identical to query-sequence k-tuples. BLAST, on the other hand, searches for k-mers that would score above a given threshold (*T*) when aligned with a query k-mer. These aligned k-mers need not be identical. Second, FASTA uses hashing and chaining to aid rapid identification of k-tuple matches. BLAST uses a scheme based on **finite-state automata** (**FSA**) to achieve the same goal. The input for such an automaton is a linear string of symbols taken one at a time. The automaton is designed to be able to identify particular patterns in the input string. These patterns may be more complex than a specific sequence and can cover a range of variation. Such automata usually report the existence of one or more of these patterns, possibly including location information, by emitting data under specific circumstances.

Each state of an automaton has well-defined responses to any possible input, responses that can include both the transition to a new state and the emission of symbols. A key response to a new symbol is not to accept it, meaning that the input string is rejected. Some automata always start reading a new string from a particular state, in which case rejection can also be written as transition back to this state. Figure 5.23 shows an example of such a finite-state automaton, in which rejection (denoted by $¶_1$ and $¶_2$) results in transition back to state 0.

As **hidden Markov models** (**HMMs**) are discussed in detail in further chapters (especially Sections 6.2 and 9.3) it is useful to note here that in contrast to HMMs, the transitions and emissions in FSA models are not probabilistic. Each input symbol leads to a deterministic outcome.

Unlike FASTA, at all stages BLAST calculates scores using a realistic substitution matrix such as BLOSUM-62 (that is, it does not use a simple sum of identities such

	INPUT	OUTPUT
$¶_1$:	~CHY	none
	H	CHH
	Y	CHY
$¶_2$:	~CH	none
	H	CYH

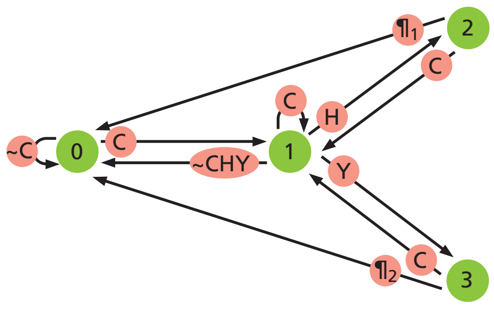


Figure 5.23
The deterministic finite-state automaton that can be used to find instances of the 3-mers CHH, CHY, and CYH in an input sequence. Input starts at state 0. The input that causes a particular transition is given near the start of the relevant arrow. Thus the transition from state 2 to state 1 is triggered by input C. The symbol ~ means "not," so ~CHY means any input except C, H or Y. The transitions 2 → 0 and 3 → 0 are triggered by several different inputs as listed, some of which also result in an output from the automaton. No other transitions produce an output. The output in this example is simply the 3-mer matched in the input, but it could be the residue number of the start of the matching 3-mers, or any other useful information.

as Equation EQ5.29); as in FASTA, nucleotide sequences are by default scored +5 and –4 for matches and mismatches, respectively. Typically, a k-mer of length 3 is used for protein sequences, and of length 11 for nucleotide sequences. In the following discussion, unless explicitly stated, we will assume we are dealing with protein sequences.

Electronic rights could not be obtained for this image.

Figure 5.24
Illustration of BLAST word hits for a comparison of horse β-globin and broad bean leghemoglobin I. The + symbols indicate the 15 hits with $T = 13$, as used in the original BLAST algorithm. All 15 would be extended to give ungapped HSPs. The • symbols show a further 22 hits with $T = 11$, the setting of the more recent gapped BLAST program. From this total of 37 hits, there are two pairs on the same diagonal within 40 residues. Only these two are extended, as shown by the lines. The left-hand one gives the higher ungapped HSP score, and is subsequently extended into a gapped alignment as shown in Figure 5.19. (Redrawn from S.F. Altschul et al., Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *N.A.R.* 25 (17):3389–3402, 1997, by permission of Oxford University Press.)

For each 3-mer in the query sequence all possible 3-mers that have an alignment score greater than T are generated. This list is then used to scan the database sequence to find all possible similar regions. A typical value for T was 14 in early versions of BLAST when used with the BLOSUM-62 matrix, meaning an average alignment score per residue of over 4. This meant that 3-mers composed only of alanine, isoleucine, leucine, serine, or valine could not reach the threshold (see Figure 5.5), as they all score a maximum of 4. (Some implementations of the algorithm allow the exact 3-mer, i.e., the identical tripeptide, in such a case.) In fact, the large number of negative scores in the BLOSUM-62 matrix, and the maximum possible score of 11 (for aligning two tryptophan residues), mean that all the possible 3-mers are highly similar, if not identical, to the query 3-mer. If T is raised, the number of possible 3-mers is reduced, and they will be even closer to the query 3-mer. This will be the case even for 3-mers including residues that have high scores, such as cysteine and histidine. For example, if $T = 19$, only CHH, CHY, and CYH will score sufficiently highly against the query sequence 3-mer CHH with BLOSUM-62.

For protein sequences there are a total of $20^3 = 8000$ possible different 3-mers, and an n-residue query protein will have $(n - 2)$ 3-mers. Each possible 3-mer must be associated with the position of the query sequence 3-mer. The default choice of parameters can result in about 50 words, making a long list of potentially tens of thousands of k-mers for even an average-length query sequence.

For nucleotide sequences, a word size of 11 is often used, and instead of allowing nonidentical matches, only exact matches to these query words are allowed. In this case, construction of the word list used for searching resembles that in FASTA.

Returning to the case where we are matching nonidentical k-mers, given the list of k-mers, a deterministic finite-state automaton can be constructed such that if the database sequences are input to the automaton, all possible matches will be output in an efficient way. Finite-state automata are best explained with an example, and Figure 5.23 shows one for the CHH 3-mer mentioned above. The figure assumes that only the three 3-mers CHH, CHY, and CYH are to be searched for.

The automaton is started in state 0. The database sequence is input to the automaton one residue at a time. From state 0, only state 1 can be reached, requiring a C, because all desired 3-mers start with a C. Note, however, that all inputs to state 0 that are not C (~C) are regarded as causing a transition from 0 to itself (shown by the curled arrow on the left in Figure 5,23). In fact the results of any input to any state are defined by an unambiguous transition, which is why this is called a **deterministic finite-state automaton**.

From state 1, an H input leads to state 2, which can only be reached this way, requiring an input sequence CH. Similarly, state 3 can only be reached with the input sequence CY. Thus we are gradually building up the desired 3-mers, in a way analogous to the suffix tree (see Figure 5.20). If the input at state 1 is C there is a transition back to state 1, because this second C can be regarded as a second attempt to start one of the 3-mers. All other inputs give a transition back to state 0, from where the search starts all over again with the next input.

From states 2 and 3, unless the input is C, the transition is to state 0. Certain inputs result in the output of a 3-mer name, and these transitions can only occur if the input sequence contains the relevant 3-mer at that point. Thus, if the input sequence is CHCYHC, the states visited will be 0121301 and the transition to state 0 will be accompanied by the output CYH. Thus the 3-mer has been identified. In the BLAST algorithm, instead of storing the 3-mer sequence, as in the first stage of FASTA the positions of the 3-mers in the two sequences are kept for further analysis in the next stage of the algorithm. An example of the results at this stage is shown in Figure 5.24.

In a real case, with several thousand k-mers to find, a diagram of the automaton is extremely large, with many crossing transitions. Whenever input is such as to require restarting the search with a new initial residue, transition is made to the state that can reuse as much as possible of the failed k-mer. For example, if a state was reached via input ABCD but no desired k-mer has sequence ABCDE, then input E will cause a transition to a state that can only be reached via input of BCDE, CDE, DE, or E, or else will return to state 0. These transitions will be considered in the order given, to try to retain as much information as possible. These transition choices depend only on the k-mer list, and so can be constructed with fixed transitions as soon as the list is available. Using this approach, all suitably high-scoring k-mer matches between the query and database sequences can be listed.

In the original version of BLAST, all k-mer matches scoring above T are extended in both directions without using gaps. Such extended ungapped local alignments are called high-scoring segment pairs (HSPs). The score is monitored and the extension is stopped when the score falls by some set amount X_u from the maximum found so far for this match. This procedure tries to allow an HSP to contain a region of lower similarity. For protein sequences, a typical value X_u for the permitted drop in score is 20. It is possible that the best-scoring HSP has a sufficiently poor-scoring internal region that the extension will stop prematurely, but the parameters are set to try to minimize the likelihood of this occurring. The highest-scoring region (MSP, maximal segment pair) for this pair of sequences is used with some statistical measures discussed in the final part of this chapter to estimate the significance of the alignment. In some cases, more than one HSP may score above some threshold value, in which case two or more HSP scores may be used in determining the significance. The alignment reported by BLAST can be a relatively short stretch of the whole sequence, and no attempt is made to extend it further using gaps.

Newer versions of BLAST take a different approach to generating alignments from the initial hits, called the two-hit method. This starts from the premise that any significant alignment is likely to have at least two high-scoring k-tuple matches on the same diagonal of the alignment matrix. Thus the initial matches are searched to find pairs on the same diagonal within a given distance (typically 40 residues) of each other. The scoring threshold T for k-mers is then lowered, for example to 11 for protein sequences scored with BLOSUM-62, producing more k-mers, and thus more initial hits (see Figure 5.24). However, only a few of these will pair up suitably close to each other on the same diagonal. Only the second hit of such pairs is extended, initially ungapped, as for the older BLAST version. The X_u parameter is used in obtaining the extension, as described above. This ungapped extension must produce an HSP with a score greater than a threshold value S_g if it is not to be discarded. S_g is set such that approximately 2% of database sequences will have an HSP of greater score.

Alignments with scores exceeding S_g are used to seed a dynamic programming calculation of a gapped alignment. This is started from the center of the highest-scoring 11-mer of the HSP. The matrix is filled both forward and backward, as described in Section 5.2, with elements calculated until the score falls a set amount X_g below the current highest-scoring alignment. In this way the amount of calculation is minimized without restricting the alignment to a predetermined band of the matrix. Only one gapped alignment is generated for a database sequence, and its score is used to determine the significance, as discussed in Section 5.4.

The parameters of the gapped BLAST program are set to make it approximately three times as fast as the ungapped version, yet more sensitive. This is achieved by severely reducing the number of extensions attempted. A gapped extension takes approximately 500 times as long as an ungapped one.

Comparing a nucleotide sequence directly with a protein sequence requires special modifications to the BLAST and FASTA algorithms

There are several situations where a comparison between a nucleotide sequence and a protein sequence is necessary. Most protein databases tend to be nonredundant and only contain highly reliable sequences (that is, minor variants and many hypothetical genes are excluded). When analyzing new sequences, a much stronger signal for significant similarity can be obtained by comparing against protein sequences. Thus, new nucleotide sequences are often compared with the protein sequence databases. Conversely, there are occasions when one wants to search for homologous proteins in large genomic sequences.

Two new problems arise in these circumstances. First, a nucleotide sequence can be translated into protein in six different reading frames (three on each strand), so that there are six different potential protein sequences to be examined for each nucleotide sequence. Second, insertion or deletion errors can be present in the nucleotide sequence. This can result in the actual protein sequence being in different reading frames for different parts of the sequence, a situation called **frameshift**. The algorithms described earlier for comparing protein with protein or nucleotide with nucleotide sequences require modifications to allow for these new factors.

In the BLAST program suite the two programs blastx and tblastx allow searches with different reading frames, but neither allows for frameshifts. In addition to the 20 possible amino acids, there may be some stop codons present, and by default, the score for aligning a stop codon with an amino acid is taken as the most negative score in the substitution matrix, although other scoring schemes are possible. Both these programs convert the nucleotide sequence into protein sequence and then work exactly as a standard protein BLAST search. The only difference is that the query-sequence reading frame used in each alignment is reported.

The FASTA program suite contains four programs relevant to this problem. Two of these (tfastx and tfasty) are for searching nucleotide databases with protein sequences, and the others (fastx and fasty) are for nucleotide query-sequence searches of protein databases. All these programs can account for frameshifts to some degree, as well as the alternative reading-frames, and thus are, in principle, more powerful than blastx at generating suitable alignments. However, in practice blastx is still very useful, and all these programs have their place in database searching.

The fastx and tfastx programs use an algorithm that only allows for nucleotide insertions and deletions. The possibility of a base being incorrectly given, for example an A where there should have been a T, is not considered. The nucleotide sequence is translated in all six frames, each set of three being considered in a single run of the alignment program.

Each of the three reading frames is analyzed in a separate matrix. However, at each step, the possibility of moving between matrices is considered. Thus, the alignment could arrive at matrix element $S_{i,j}$ from $S_{i-1,j-1}$ in the same frame, or from $T_{i-1,j-1}$ or $U_{i-1,j-1}$ in the alternative frames. The other ($i-1,j$ and i,$j-1$) elements are also possible sources of the new (i,j) element. Any move between matrices incurs an extra penalty — the frameshift penalty. This can be set higher when the sequences are expected to be more accurate, to prevent excessive numbers of frameshifts in the alignments, and is often set a little higher than the gap-opening penalty. The full details of the algorithm will not be given here, but a sequence example is given to illustrate the effects of frameshifts.

Box 5.2 Sometimes things just aren't complex enough

Many protein and nucleotide sequences contain regions that can be described as being of low compositional complexity: low-complexity regions or **simple sequences**. Examples include stretches of identical amino acids in proteins, repeated short sequences, and longer DNA repeats (see Box 1.1). It is estimated that roughly half of all database sequences contain at least one such region. If alignments are made with these regions present, many spurious similarities will be reported because many unrelated sequences contain similar low-complexity regions. These stretches also cause problems in database searches because they are nonrandom, violating the assumptions on which the calculations of statistical significance are based (Section 5.4). It is important to be able to define these stretches and mask them to prevent their biasing the database search results. Many databases have such sequences masked when they are made available for sequence searches, so that often there is only a need to find these regions in the query sequence.

Three different properties of these simple sequences can be distinguished. Precisely repeating sequences are referred to as patterns. These can be very short, for example ATT, and are not necessarily in a single contiguous block, for example ATTCATTGCATTATT. If there is a clear period of repeat—for example, ATTATTATTATT has a repeat of three—this is called a periodicity. The periodicity need not necessarily be an integer, as can occur, for example, in an amino acid repeat relating to an α-helix. Furthermore, patterns and periodicities can both involve some degree of error, in that the repeat need not be exact. The final property that we will discuss is the compositional complexity, which is a measure of bias in the sequence composition.

In general terms, a sequence of length L is made up from N_{type} possible different components (i.e., N_{type} is 20 for proteins, 4 for nucleic acids) in a composition defined by the ath component occurring n_a times. The general complexity-state vector is defined as a list of these integers n_a in numerical order, disregarding the specific components. Thus both nucleotide sequences ATA and CAC are represented by the same vector {2, 1, 0, 0}. The number of distinct sequences of length L with this composition is given by

$$\frac{L!}{\prod_{a=1}^{N_{\text{type}}} n_a!} \qquad \text{(BEQ5.2)}$$

(The term $L!$ is called "L factorial", and means the product $L \times (L-1) \times (L-2) \ldots 3 \times 2 \times 1$, so $4! = 4 \times 3 \times 2 \times 1 = 24$. By definition, $0! = 1$.) The n_a can vary in value from 0 to L, and will always sum to L. The number of these n_a that have the value c is written r_c, and the r_c will sum to N_{type}. The number of different compositions that will give rise to the same complexity-state vector is given by

$$\frac{N_{\text{type}}!}{\prod_{c=0}^{L} r_c!} \qquad \text{(BEQ5.3)}$$

Thus the total number of distinct sequences corresponding to a particular complexity-state vector is

$$\frac{L!}{\prod_{a=1}^{N_{\text{type}}} n_a!} \times \frac{N_{\text{type}}}{\prod_{c=0}^{L} r_c!} \qquad \text{(BEQ5.4)}$$

The more distinct sequences available to a complexity state, the more complex the sequence.

As an example consider the five-nucleotide sequence ATTAT. The composition of this sequence can be represented as {T_3, A_2, C_0, G_0} where the bases have been ordered according to their abundance. There are 10 possible sequences with the same composition: AATTT, ATATT, ATTAT, ATTTA, TAATT, TATAT, TATTA, TTAAT, TTATA, and TTTAA. Note that this is 5!/(3!2!). Now consider how many different compositions are possible by switching the bases around but maintaining the proportion of bases at 3:2:0:0. There are 12 of these: A_3C_2, A_3G_2, A_3T_2, C_3A_2, C_3G_2, C_3T_2, G_3A_2, G_3C_2, G_3T_2, T_3A_2, T_3C_2, and T_3G_2. Note that this is 4!/(2!1!1!). Therefore for the general complexity state represented by the vector {3, 2, 0, 0} there are a total of $10 \times 12 = 120$ distinct DNA sequences. For comparison, the complexity state represented by vector {5, 0, 0, 0} has only four unique DNA sequences [(5!/5!) × 4!/(3!1!)], compared to 360 for the state {2, 2, 1, 0} [5!/(2!2!1!) × 4!/(2!1!1!)].

Several programs are available that attempt to distinguish between simple and other regions of a sequence. We will only discuss one, the program SEG, which uses compositional complexity as a measure to determine regions of simple sequences. SEG works in two steps to determine regions that satisfy certain complexity constraints. The compositional complexity I_{SEG}, which is a measure of the information required per sequence position to specify a particular sequence, given the composition, is defined by

$$I_{\text{SEG}} = \frac{1}{L} \log_{N_{\text{type}}} \left(\frac{L!}{\prod_{a=1}^{N_{\text{type}}} n_a!} \right) \qquad \text{(BEQ5.5)}$$

Box 5.2 Sometimes things just aren't complex enough (continued)

For example, for a five-nucleotide sequence, I_{SEG} can vary from 0 for {5, 0, 0, 0} to ~1.92 for {2, 1, 1, 1}. In the first step of the SEG method, a more computationally efficient approximation is used to search the sequence. Windows of length L are identified for which the value of

$$I'_{\text{SEG}} = -\sum_{a=1}^{N_{\text{type}}} \frac{n_a}{L}\left(\log_2 \frac{n_a}{L}\right) \tag{BEQ5.6}$$

is less than a given threshold I_{SEG1}. Note that I'_{SEG} approximates I_{SEG} for large L. These initial low-complexity regions are augmented by any overlapping windows that have a value of I'_{SEG} that is exceeded by a less-strict threshold I_{SEG2} (that is, $I_{\text{SEG2}} > I_{\text{SEG1}}$).

In the second step, SEG determines the sub-sequence of each initial low-complexity region whose composition has the least probability of occurrence, based on a model with all residues equally likely. The probability is calculated using the formula

$$P_{\text{SEG}} = \frac{1}{N_{\text{type}}^L}\left(\frac{L!}{\prod_{a=1}^{N_{\text{type}}} n_a!}\right)\left(\frac{N_{\text{type}}!}{\prod_{c=0}^{L} r_c!}\right) \tag{BEQ5.7}$$

By inclusion of the first term, this value can be compared for different window sizes.

Versions of SEG are available that are designed to search for specific periodicities, which can be useful in some instances. For example, a number of proteins have regions of low complexity, often short repeats, which can indicate nonglobular structure. Since these sequences tend not to be exact repeats, SEG can be a powerful tool for identifying these regions. DUST is an equivalent program for DNA sequences. There are other programs designed to search for specific known repeat sequences such as the DNA repeats found in many genomes, usually defined in a small repeat database.

Consider the following short stretch of sequence and three forward translations:

```
                A C C A G A G C C A A C T

Frame 1          T     R     A     N
Frame 2            P     E     P     T
Frame 3              Q     S     Q
```

The translations have been placed under the central base of each codon, so that T is placed under the first C of ACC. Consider all the possible moves, including frameshifts, from a matrix element at position 2 in the translated sequence to one at position 3. This means that the alignment of TR, PE, and QS has already been considered, and we are now considering adding A, P, or Q, referring to frame 1, 2, or 3, respectively. When translated back into nucleotides, the possible interpretations of the sequence are:

$1 \rightarrow 2$ (AGA)*G*(CCA)

$1 \rightarrow 3$ (AGA)*GC*(CAA)

$2 \rightarrow 1$ (GA(**G**)CC) or (GAG)*CC*(AAC)

$2 \rightarrow 3$ (GAG)*C*(CAA)

$3 \rightarrow 1$ (A(**GC**)C) or (AGC)*C*(AAC)

$3 \rightarrow 2$ (AG(**C**)CA) or (AGC)*CA*(ACT) where codons are in parentheses. Bases that are ignored in translation (deletions) are in italic, while those used in two successive codons (insertions) are in bold. Thus the sequence written (A(**GC**)C) is interpreted as (AGC)(GCC). Similarly (AGC)*CA*(ACT) is interpreted as (AGC)(ACT). Note that deletions and insertions only occur at the boundaries of codons; that is, the sequence ACGT is only interpreted as ACG or CGT, never ACT or AGT.

A greater variety of errors in the nucleotide sequence can be allowed for by fasty and tfasty. Any two, three, or four consecutive nucleotides can be interpreted as a codon, of which only the middle alternative does not involve an indel. In each case, base errors are considered. A penalty scheme for modifying the codons is combined with a BLOSUM-50 scoring of the aligned amino acids to find the best codon for that alignment; if indels are involved they also incur a penalty. Note that in this case, as well as allowing all four interpretations of ACGT mentioned above, base-pair changes are also considered. If the base change and frameshift penalties are not sufficiently punitive, almost any pair of sequences could be aligned with a high score. Both of these are commonly set to up to twice the gap opening penalty.

5.4 Alignment Score Significance

In this section we will examine how to determine whether the score of an optimal alignment is significantly higher than would be expected for two unrelated sequences (see Flow Diagram 5.4). This is not the only way of trying to assess significance, but it is probably the most sensitive currently available. The simpler technique of observing the percentage of identical or similar residues in the alignment (see Section 4.2), although useful, is far less precise.

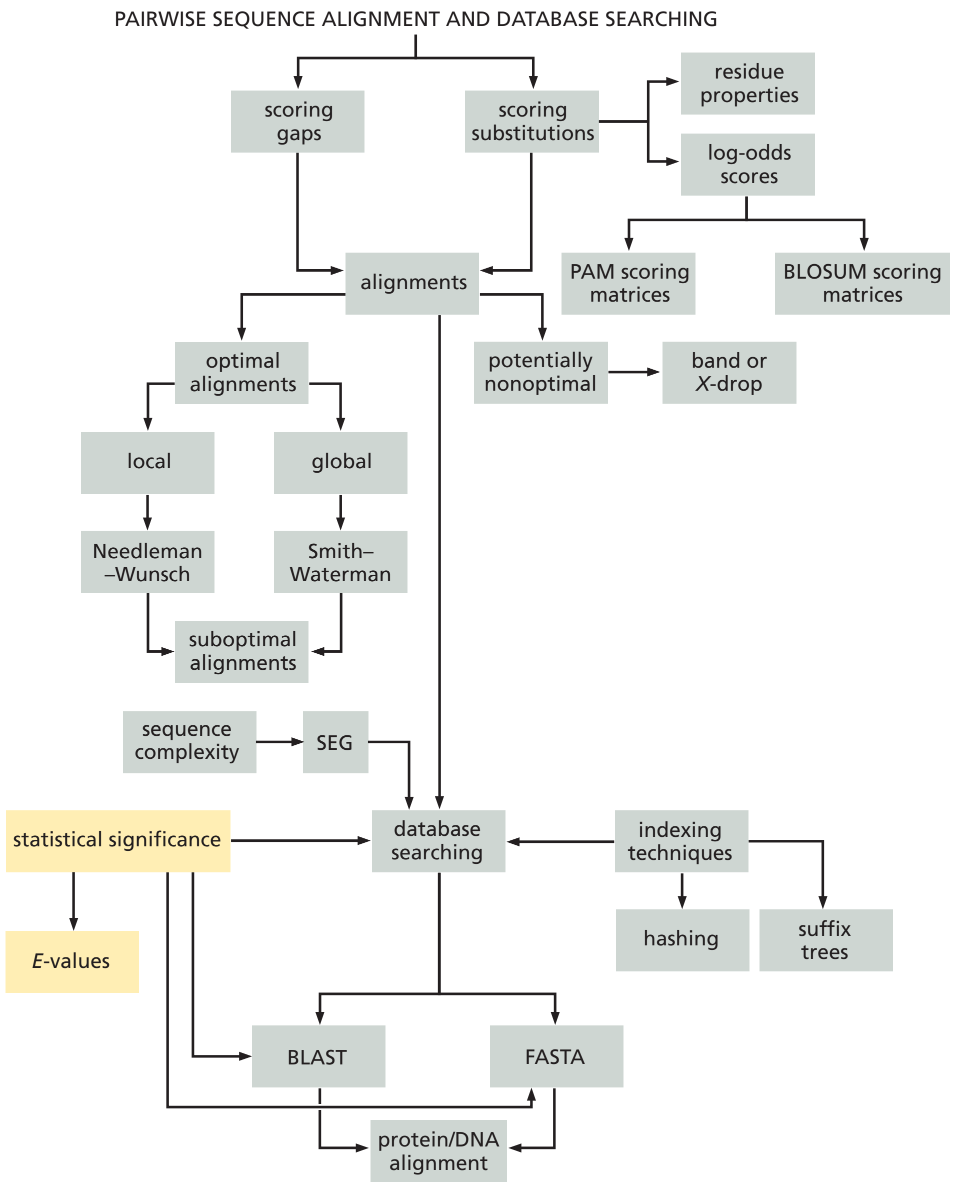
If the alignment scores were normally distributed—that is, Gaussian—then knowledge of their mean and standard deviation would allow us to calculate the probability of observing any given score using standard tables. The situation is not so simple, however, because the score that is reported in a database search is that of the optimal alignment; that is, it is the best possible score for that particular pair of sequences. This means that the scores are always from the extreme end of the distribution of all alignment scores.

The statistics of optimal alignment scores have only been rigorously derived in the case of ungapped local alignments, for which the scores follow an extreme-value distribution. Using this theory, precise evaluation of score significance for ungapped local alignments is readily calculable. For gapped local alignments, only approximations are available for the statistical score distribution. Both gapped and ungapped local alignments have been found to have very similar distributions, but differ in their parameterization.

Before discussing the details of the practical application of these score distributions, two general points about scores have to be made. First, one can ask how much information is required to define the position of an alignment in two sequences. Information is usually measured as bits, which can be regarded as yes or no answers. For example, distinguishing the numbers 0–255 requires eight digits in a binary (i.e., 0 or 1) number representation, which is $\log_2 256$. For a sequence of length m, we have to distinguish among the m possible alignment starting positions, which in general requires $\log_2 m$ bits of information. If this sequence is aligned to an n-residue sequence, positioning the start of the alignment on this second sequence will require a further $\log_2 n$ bits of information. Therefore the alignment requires a total of $\log_2 m + \log_2 n = \log_2(mn)$ bits of information to define its start position on both sequences.

Flow Diagram 5.4

The key concept introduced in this section is that when comparing optimal alignments between a query sequence and a database of sequences, care must be taken to evaluate the true significance of the alignment score, since even the best scoring database sequence cannot be assumed to show significant similarity.



In performing a database search, if m is the length of the query sequence, then n is the total length of the database entries. Considering a protein sequence database of 100 million residues and an average-length query sequence of 250 residues, a score of approximately 35 bits would be required to define the location of the best alignment between the sequences.

The second general aspect of optimal alignment scores relates to how they vary with the length of the alignment. Waterman has shown that for global alignments with gaps, the score grows linearly with sequence length. For local alignments the situation is more complex, and depends on a property of the substitution matrix used. If the expected score of a random sequence with a given substitution matrix, given by Equation EQ5.3 is positive, the local alignment score grows linearly with sequence length regardless of the gap penalties. However, if the expected score is negative, which is the case for most amino acid substitution matrices in common use, then unless gap penalties are very low, the optimal local alignment score will grow logarithmically; that is, as log n where n is the number of residues. In what follows, it is assumed we are using scoring parameters such that the scores grow logarithmically with sequence length. The alignment scores need to be corrected for length as part of the process of determining their significance.

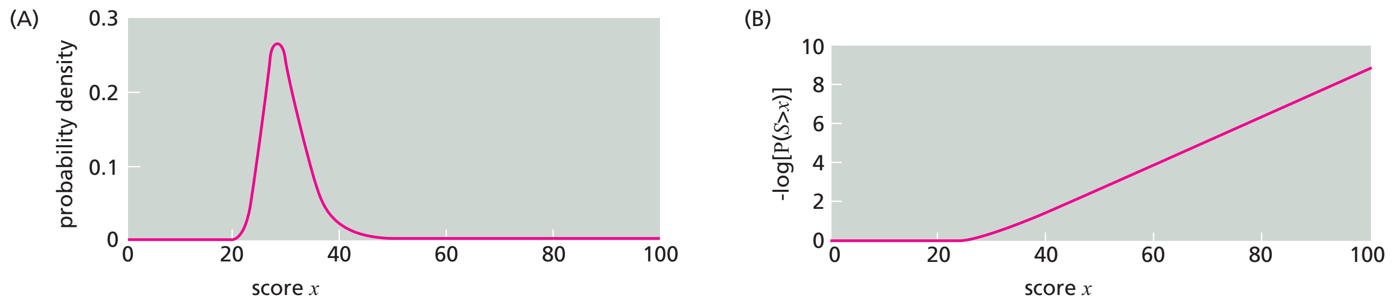


Figure 5.25
The extreme-value distribution of optimal alignment scores. (A) The distribution for λ and K of 0.286 and 0.055, respectively, for two sequences both of length 245 residues. (These values are from Altschul and Gish (1996) for a BLOSUM-62 substitution matrix, with affine gap opening and extension penalties –12 and –1, respectively.) (B) The probability, given the distribution in part A of an observed score S being at least x, plotted so that a value y on the vertical axis represents a probability 10^{-y}.

The distribution of alignment scores for optimal ungapped local alignments has been rigorously derived from first principles. A database search only carries out further analysis of the highest-scoring alignment of each database entry with the query sequence. Omitting the considerable amount of analysis required, it can be proved that the optimal ungapped local alignment score follows the Gumbel extreme-value distribution. With m and n defined as above, this distribution peaks at a value

$$U = \frac{\ln(Kmn)}{\lambda} \qquad \text{(EQ5.30)}$$

where K and λ are constants that depend on the scoring matrix used and the sequence composition. λ is a scaling parameter of the substitution matrix, and is the unique positive solution of the equation

$$\sum_{a,b} p_a p_b e^{s_{a,b}\lambda} = 1 \qquad \text{(EQ5.31)}$$

where the summation is over all residue types a and b, the p_a and p_b are frequencies of occurrence of the residues as defined in Section 5.1, and $s_{a,b}$ are the substitution scores.

The probability of the alignment score S being less than x is given by the cumulative distribution function

$$P(S < x) = \exp\left(-e^{-\lambda(x-U)}\right) \qquad \text{(EQ5.32)}$$

from which the complementary probability of the score being at least x is

$$P(S \geq x) = 1 - \exp\left(-e^{-\lambda(x-U)}\right) \qquad \text{(EQ5.33)}$$

Substituting for U we arrive at the formula for the probability of obtaining an alignment of score S greater than a value x:

$$P(S \geq x) = 1 - \exp\left(-Kmne^{-\lambda x}\right) \qquad \text{(EQ5.34)}$$

The extreme-value distribution is shown in Figure 5.25. The key feature of interest is the tail for high values. Notice that this decays much more slowly than the low-value tail; that is, the distribution is asymmetric with a bias to high values. It is important that we have the correct distribution if we are to estimate the significance of any given score accurately. In general, if $P(S \geq x)$ is less than 0.01, the alignment is significant at the 1% level. The level chosen as cut-off depends on the particular problem, and is discussed in more detail in Section 4.7.

Formulae exist for calculating K and λ for a given substitution matrix and sequence database composition. The original (ungapped) version of BLAST used this theory to estimate the significance of the alignments generated in a database search.

The statistics of gapped local alignments can be approximated by the same theory

For gapped local alignments, examination of actual database searches has shown that the scores of optimal alignments also fit an extreme-value distribution. However, in this case there is no rigorous theory to provide the parameters λ and K. These can be estimated by studying sample database searches for particular values of the scoring schemes (including gap penalties). Note, however, that they will depend on the composition of the database sequences, so that the parameterization should really be done according to the actual database to be searched. Both BLAST and FASTA use such methods to derive score significance.

The programs in the BLAST and FASTA suites report E-values, which are related to the probability $P(S \geq x)$. The value of P is calculated for the particular lengths of the query and database sequences, and varies from 0 to 1. The E-values are obtained from this by multiplying by the number of sequences in the database. Thus, if there are D database sequences, the E-values range from 0 to D.

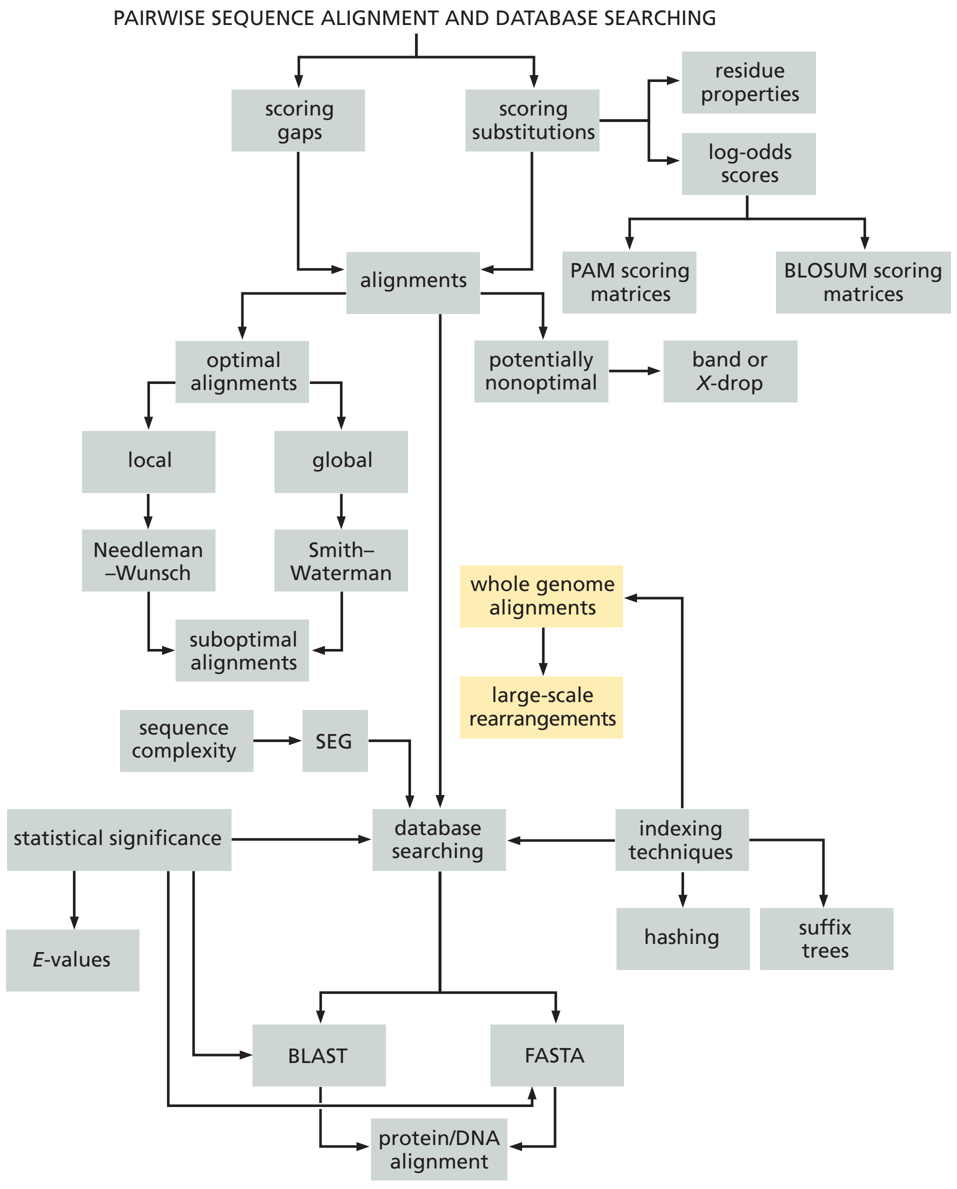
The E-value is the number of database sequences not related to the query sequence that are expected to have alignment scores greater than the observed score. Thus an E-value of 1 is not significant, as one unrelated database sequence would be expected to have such a score. The E-value deemed to indicate a significant similarity as shown by the alignment score is a practical problem discussed in Section 4.7.

5.5 Aligning Complete Genome Sequences

As more genome sequences have become available, of different species and also of different bacterial strains, there are several reasons for wanting to align entire genomes to each other. Such alignments can assist the prediction of genes because if they are conserved between species this will show up clearly in the alignment. In addition, other regions of high conservation between the genomes can indicate other functional sequences. The study of genome evolution needs global alignments to identify the large-scale rearrangements that may have occurred. Although in principle the dynamic programming methods presented in Section 5.2 could be used, the lengths of the sequences involved makes the demands on computer time and disk space prohibitive. As in the case of database searches, other techniques must be used that are not guaranteed to find the optimal scoring alignment.

When aligning two complete genome sequences, problems arise that are quite distinct from those discussed above in relation to aligning two related proteins or making simple database searches. In the case of complete genomes, the alignment problem is more complex because the intergene regions may be subject to higher mutation rates; in addition, large-scale rearrangements may have occurred. Many discrete, locally similar segments may exist, separated by dissimilar regions. Unlike in the BLAST and FASTA methods, therefore, it is insufficient to determine a single location from which to extend the alignment. The solution is to modify the indexing techniques for genome alignments to locate a series of anchors. The relationship of this subject to the other topics discussed in this chapter is shown in Flow Diagram 5.5.

A second problem is the linking together of the anchors to form a scaffold for the alignment, and doing this in such a way as to identify the large-scale rearrangements.



Flow Diagram 5.5
The key concept introduced in this section is that when aligning extremely long sequences, refinements are required in order to make the alignment search more efficient. The section also deals with the observation that long sequences often show large-scale rearrangements, a feature that needs to be included in the alignment methods.

Both of these problems require complex solutions that are only briefly outlined here. This area is still undergoing rapid development, and new techniques may yet emerge that are a significant advance in the field.

Another application involving a complete genome sequence is the alignment to it of many smaller nucleotide sequences. These could be from a variety of sources, including sequence data from related species at various stages of assembly. The latter task, in particular, can give rise to the problems noted above.

Indexing and scanning whole genome sequences efficiently is crucial for the sequence alignment of higher organisms

Both FASTA and BLAST use indexing techniques to speed up database searches, and both index the query sequence. For genome alignments it is now practical to index the complete genome sequence, as long as the computer used has sufficient memory to store the whole index. In contrast to usual database searches with single gene sequences, this requires careful planning of the data storage. If the index is larger than the available computer memory, the time required for the alignment

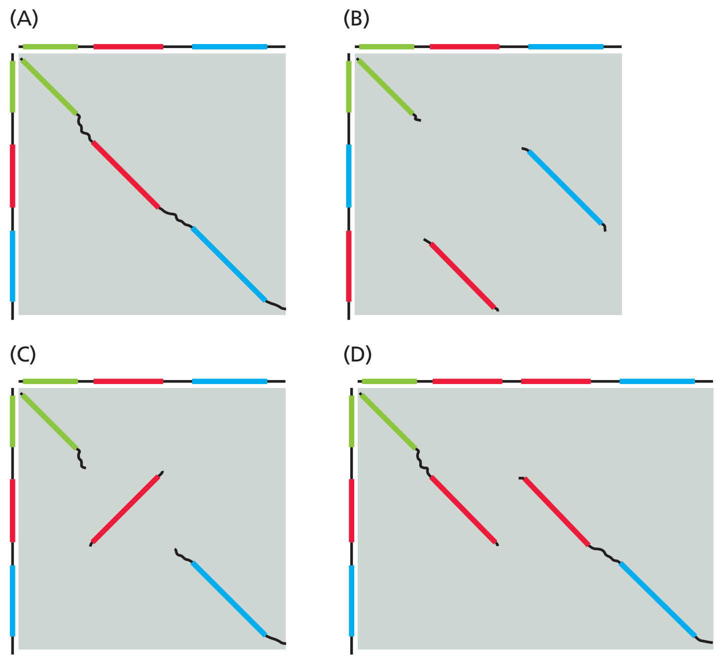


Figure 5.26
Four examples of possible relationships between two long nucleotide sequences. (A) Both sequences are similar along their whole length, with three particularly similar segments identified. No rearrangements during evolution from the most recent common ancestor need be proposed. This is the only case in which a global alignment can be given. (B) A translocation has changed the order of the red and blue segments for one of the sequences. (C) The red segment has been inverted in the horizontal sequence. (D) The red segment has been duplicated in the horizontal sequence. The black lines indicate regions of alignment outside the main segments. (Adapted from M. Brudno et al., Global alignment: finding rearrangement during alignment, *Bioinformatics* 19, Supplement 1:i54–i62, 2003.)

can increase by a factor of four or more. If suffix trees are used for the indexing, then with efficient techniques the space requirements are proportional to the sequence length. Hashing methods require storage according to the lengths of the sequence and the k-tuples. However, the space required can often be reduced significantly if the most common k-tuples, defined as those occurring more than a threshold number of times, are omitted on the grounds that they are likely to be uninformative for the alignment. The hashing methods often also reduce the index size by only considering nonoverlapping k-tuples; that is, bases $1 \rightarrow k$, $k + 1 \rightarrow 2k$, and so on.

The index is used to identify identical k-tuples in the two sequences. Because the sequences involved are very long, the k-tuples must be much longer than in the example of database searching discussed above, or many matches will be found with random sequence. The expected number of matches depends on the lengths of the k-tuples and the percentage identity of the aligned sequences, and was thoroughly analyzed during development of the BLAT (BLAST-like Alignment Tool) package. k-tuples of 10–15 bases are used by the Sequence Search and Alignment by Hashing Algorithms (SSAHA) program, and lengths of 20–50 bases have been used in suffix tree methods.

One technique that has proved to be particularly effective is the spaced seed method. When searching using the long k-tuples mentioned above, all k positions of the k-tuples are compared. It has been found that these are not the most efficient

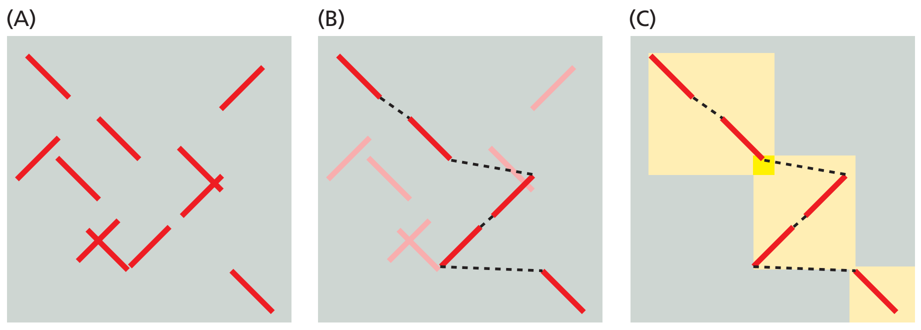


Figure 5.27
An example of the application of the SLAGAN method. (A) The set of identified potential anchors for the alignment of two long nucleotide sequences. (B) The path of anchors identified. The anchors are ordered progressively along the vertical sequence but not the horizontal one because of large-scale rearrangements. (C) The regions identified as locally consistent. Individual alignments are determined for each of the three regions. (Adapted from M. Brudno et al., Global alignment: finding rearrangement during alignment, *Bioinformatics* 19, Supplement 1:i54–i62, 2003.)

probes for identifying matches that are useful seeds for genome alignment. In the development of the PatternHunter program it was found that searching for a specific pattern of 12 bases in a stretch of 19 was more effective. The 19mer can be represented as 1110100110010101111, where 1 indicates a position whose match is sought, and 0 indicates a position whose base is disregarded. The later versions of the BLASTZ program made a further improvement by allowing any one of the 12 positions to align bases A and G or bases C and T, so long as the other 11 sites were perfect matches.

The complex evolutionary relationships between the genomes of even closely related organisms require novel alignment algorithms

Many anchor points must be identified between the two genome sequences, each initially as matched k-tuples which are usually extended in a gapless way as in FASTA and BLAST. The original MUMmer approach was to include only unique k-tuples common between the sequences; that is, k-tuples present just once on each sequence. Such matches can be expected with confidence to identify truly homologous segments. Most methods (including MUMmer 2) are not so restrictive, and include k-tuples that occur several times on a sequence. As a consequence, many matches are recorded, some of which are expected to be incorrect.

In contrast to the database search methods, which only need to select a single anchor for the alignment that is then extended using dynamic programming, for genome alignment a set of anchors is needed that spans the lengths of the genomes, identifying the true homologous segments. This step is usually similar to the third step of FASTA (see Figure 5.22C) and involves dynamic programming where each anchor and potential gap is given a score. Anchors are not allowed to overlap and must be arranged sequentially along the sequences.

In addition, a number of large-scale sequence rearrangements may have occurred since the last common ancestor of the species whose sequences are being aligned. As a consequence, the correct alignment will not have the straightforward character of the alignments discussed so far. Some examples of common rearrangements are shown in Figure 5.26, and a real example is shown for aligning equivalent chromosomes in mouse and rat in Figure 10.21. Most existing programs cannot automatically recognize large-scale rearrangements and thus cannot correctly assign segments to the overall alignment. An exception is the Shuffle-Limited Area Global Alignment of Nucleotides (SLAGAN) program, which attempts to identify rearrangements. The FASTA-type step referred to above requires each successive anchor to occur in succession along both sequences. However, any rearrangements that produce sequence reversal of a segment (such as in Figure 5.26C) will result in the successive anchors occurring progressively but in opposite directions along the two sequences. SLAGAN only requires the anchors to occur in succession along one of the sequences. A further step then identifies those regions that contain anchors that are locally consistent with a sequence reversal segment. An alignment is obtained for each of these regions. An example of a successful application of this method is shown in Figure 5.27.

In all the methods, an attempt is made to extend identified homologous segments, often using standard alignment techniques but trying to identify the limits beyond which the sequences are not recognizably similar. The alignments derived by these methods are often fragments separated by unaligned regions.

Summary

In the first part of this chapter, we looked at the derivation of the specialized algorithms that are used in automated methods to determine the optimal alignment between two sequences. Such algorithms are needed because if the insertion of gaps is allowed there are a very large number of possible different alignments of two sequences. Before an algorithm can be applied, a scoring scheme has to be devised so that the resulting alignments can be ranked in order of quality of matching. Scoring schemes involve two distinct components: a score for each pair of aligned residues, which assesses the likelihood of such a substitution having occurred during evolution, and a penalty score for adding gaps to account for the insertions or deletions that have also occurred during evolution. The first score has its foundations in the concept of log-odds, and is assigned on the basis of reference substitution matrices that have been derived from the analysis of real data in the form of multiple alignments. The penalty scores given to gaps have a more ad hoc basis, and have been assigned on the basis of the ability of combined (substitution and gap) scoring schemes to reproduce reference sequence alignments. (Confidence in these reference alignments is usually based on a combination of structural information and regions of sufficiently high similarity.)

The automated construction and assessment of gapped alignments only became possible with the development of dynamic programming techniques, which provide a rigorous means of exploring all the possible alignments of two sequences. The essence of dynamic programming is that optimal alignments are built up from optimal partial alignments, residue by residue, such that nonoptimal partial alignments are efficiently identified and discarded. The technique ensures that the optimal alignment(s) will be identified. A number of different schemes have been developed, which treat gaps in different ways. Only minor variations in the dynamic programming algorithms are required to alter the type of alignment produced from global to local, or to identify repeat or suboptimal alignments. Thus the same technique can be used to answer several different problems of sequence similarity and relatedness.

The problem with dynamic programming methods is that despite their efficiency they can place heavy demands on computer memory and take a long time to run. The speed of calculation is no longer as serious a barrier as it has been in the past, but the problem of insufficient computer memory persists, particularly as there are now many very long sequences, including those of whole genomes, available for comparison and analysis. Some modifications of the basic dynamic programming algorithm have been made that reduce the memory and time demands. One way of reducing memory requirements is by storing not the complete matrix but only the two rows required for calculations. However, to recover the alignment from such a calculation takes longer than if all the traceback information has been saved. By only calculating a limited region of the matrix, commonly a diagonal band, both time and space saving can be made, although at the risk of not identifying the correct optimal alignment.

Often the first step in a sequence analysis is to search databases to retrieve all related sequences. Such searches depend on making pairwise alignments of the query sequence against all the sequences in the databases, but because of the scale of this task, fast approximate methods are usually used to make such searches more practicable. The algorithms for two commonly used search programs—BLAST and FASTA—make use of indexing techniques such as suffix trees and hashing to locate short stretches of database sequences highly similar or identical to parts of the query sequence. Attempts are then made to extend these to longer, ungapped local alignments which are scored, the scores being used to identify database sequences that are likely to be significantly similar. This process is considerably faster than applying full-matrix dynamic programming to each database sequence. At this point, both techniques revert to the more accurate methods to examine the highest-scoring sequences, in order to determine the optimal local alignment and score, but this is only done for a tiny fraction of the database entries.

There are instances where it is necessary to align a protein sequence with a nucleotide sequence. In such cases one solution is simply to translate the nucleotide sequence in all possible reading frames and then use protein–protein alignments. However, this approach is rather bad at dealing with errors in the sequence that give rise to frameshifts. The dynamic programming method can be modified to overcome this problem by using three matrices, one for each possible reading frame in the given strand direction, with a set of rules for moving from one matrix to another. In this way, an optimal alignment can be generated that uses more than one reading frame, simultaneously proposing sequence errors.

Another problem that arises in database searches is the occurrence of regions of low sequence complexity, which can cause spuriously high alignment scores for unrelated sequences. Such regions can be defined either by identifying a repeating pattern or on the basis of their composition, and can then be omitted from the comparison.

To be sure that two sequences are indeed homologous, it is important to know when the alignment score reported is statistically significant. In the case of sequence alignments, the statistical analysis is very difficult; in fact the theory has not yet been fully developed for alignments including gaps. Part of the reason for this difficulty is that the alignments reported are always those with the best scores. These will not be expected to obey a normal distribution, but rather an extreme-value distribution. Furthermore, the scores are also dependent on the scoring scheme used, the sequence composition and length, and the size of the database searched. For alignments without gaps, formulae have been derived that allow the score to be converted to a probability from which the significance can be gauged.

The last section of the chapter deals with the alignment of very long sequences, for example those of whole genomes. The straightforward application of dynamic programming is often not feasible because of the lack of computer resources. This difficulty can be overcome by using similar indexing methods to those in the database search programs. However, there are significant differences, which lead to the indexing being applied to the genome sequence rather than the query sequence, and to searching for much longer identical segments than in database searches. Nevertheless, there is often an additional problem because frequently there are large-scale genome rearrangements even over relatively short periods of evolutionary time. To overcome this, local alignments must be identified that can then be joined together into a global alignment by a specific dynamic programming technique that allows for translations and inversions of segments of the sequence.

Further Reading

5.1 Substitution Matrices and Scoring

The PAM (MDM) substitution scoring matrices were designed to trace the evolutionary origins of proteins

Altschul SF (1991) Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.* 219, 555–565.

Dayhoff MO, Schwartz RM & Orcutt BC (1978) A model of evolutionary change in proteins. In Atlas of Protein Sequence and Structure, vol 5 suppl 3 (MO Dayhoff ed.), pp 345–352. Washington, DC: National Biomedical Research Foundation.

Jones DT, Taylor WR & Thornton JM (1992) The rapid generation of mutation data matrices from protein sequences. *Comput. Appl. Biosci.* 8, 275–282. *(The PET91 version of PAM matrices.)*

Yu Y-K, Wootton JC & Altshul SF (2003) The compositional adjustment of amino acid substitution matrices. *Proc. Natl Acad. Sci. USA* 100, 15688–15693.

See appendix for deriving target frequencies.

The BLOSUM matrices were designed to find conserved regions of proteins

Henikoff S & Henikoff JG (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA* 89, 10915–10919.

Scoring matrices for nucleotide sequence alignment can be derived in similar ways

Chiaromonte F, Yap VB & Miller W (2002) Scoring pairwise genomic sequence alignments. *Pac. Symp. Biocomput.* 7, 115–126.

The substitution scoring matrix used must be appropriate to the specific alignment problem

Altschul SF (1991) Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.* 219, 555–565.

May ACW (1999) Towards more meaningful hierarchical classification of amino acid scoring matrices. *Protein. Eng.* 12, 707–712. *(A comparative study of substitution matrices in terms of the way they group amino acids.)*

Yu Y-K & Altschul SF (2005) The construction of amino acid substitution matrices for the comparison of proteins with non-standard compositions. *Bioinformatics* 21, 902–911.

Yu Y-K, Wootton JC & Atschul SF (2003) The compositional adjustments of amnio acid substitution matrices. *Proc. Natl Acad. Sci USA* 100, 15688–15693.

Gaps are scored in a much more heuristic way than substitutions

Goonesekere NCW & Lee B (2004) Frequency of gaps observed in a structurally aligned protein pair database suggests a simple gap penalty function. *Nucleic Acids Res.* 32, 2838–2843. *(This recent paper on scoring gaps for protein sequences includes a good listing of older work.)*

5.2 Dynamic Programming Algorithms

Waterman MS (1995) Introduction to Computational Biology: Maps, Sequences and Genomes, chapter 9. London: Chapman & Hall. (*A more computational presentation of dynamic programming alignment algorithms.)*

Optimal global alignments are produced using efficient variations of the Needleman–Wunsch algorithm

Gotoh O (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162, 705–708.

Needleman SB & Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48, 443–453.

Local and suboptimal alignments can be produced by making small modifications to the dynamic programming algorithm

Smith TF & Waterman MS (1981) Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195–197.

Waterman MS & Eggert M (1987) A new algorithm for best subsequence alignments with application to tRNA–tRNA comparisons. *J. Mol. Biol.* 197, 723–728.

Time can be saved with a loss of rigor by not calculating the whole matrix

Zhang Z, Berman P & Miller W (1998) Alignments without low-scoring regions. *J. Comput. Biol.* 5, 197–210.

Zhang Z, Scwartz S, Wagner L & Miller W (2000) A greedy algorithm for aligning DNA sequences, *J. Comput. Biol.* 7 (1–2), 203–214.

5.3 Indexing Techniques and Algorithmic Approximations

Suffix trees locate the positions of repeats and unique sequences; Hashing is an indexing technique that lists the starting positions of all k-tuples

Gusfield D (1997) Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge: Cambridge University Press.

Waterman MS (1995) Introduction to Computational Biology: Maps, Sequences and Genomes, chapter 8. London: Chapman & Hall.

The FASTA algorithm uses hashing and chaining for fast database searching; The BLAST algorithm makes use of finite-state automata

Altschul SF, Gish W, Miller W et al. (1990) Basic Local Alignment Search Tool. *J. Mol. Biol.* 215, 403–410. (*The original BLAST paper.*)

Altschul SF, Madden TL, Schäffer AA et al. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25, 3389–3402.

Pearson WR & Lipman DJ (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA* 85, 2444–2448. *(The original FASTA paper.)*

Pearson WR, Wood T, Zhang Z & Miller W (1997) Comparison of DNA sequences with protein sequences. *Genomics* 46, 24–36. (*fastx, fasty.*)

Box 5.2: Sometimes things just aren't complex enough

Morgulis A, Gertz EM, Schäffer AA & Agarwala R (2006) A fast and symmetric DUST implementation to mask low-complexity DNA sequences. *J. Comput. Biol.* 13, 1028–1040.

Wootton JC & Federhen S (1996) Analysis of compositionally biased regions in sequence databases. *Methods Enzymol.* 266, 554–571 (*SEG.*)

5.4 Alignment Score Significance

Altshul SF & Gish W (1996) Local alignment statistics. *Methods Enzymol.* 266, 460–480.

Mott R (2000) Accurate formula for P-values of gapped local sequence and profile alignments. *J. Mol. Biol.* 300, 649–659.

Waterman MS (1995) Introduction to Computational Biology: Maps, Sequences and Genomes, chapter 11. London: Chapman & Hall.

5.5 Alignments Involving Complete Genome Sequences

The field of genome sequence alignment is moving very quickly. Some useful references in this area are:

Brudno M, Malde S, Poliakov A et al. (2003) Glocal alignment: finding rearrangements during alignment. *Bioinformatics* 19(suppl. 1), i54–i62. *SLAGAN.*

Delcher AL, Kasif S, Fleischmann RD et al. (1999) Alignment of whole genomes. *Nucleic Acids Res.* 27, 2369–2376. *MUMmer.*

Delcher AL, Phillippy A, Carlton J & Salzberg SL (2002) Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res.* 30, 2478–2483. *MUMmer 2.*

Kent WJ (2002) BLAT—the BLAST-like alignment tool. *Genome Res.* 12, 656–664.

Ma B, Tromp J & Li M (2002) PatternHunter: faster and more sensitive homology search. *Bioinformatics* 18, 440–445.

Ning Z, Cox AJ & Mullikin JC (2001) SSAHA: A fast search method for large DNA databases. *Genome Res.* 11, 1725–1729.

Schwartz S, Kent WJ, Smit A et al. (2003) Human–mouse alignments with BLASTZ. *Genome Res.* 13, 103–107.