Acta Cryst. (1998). D**54**, 1139–1146

Protein Sequence Alignment Techniques

Geoffrey J. Barton

European Molecular Biology Laboratory Outstation, The European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, England. E-mail: geoff@ebi.ac.uk

(*Received 21 May 1998; accepted 12 June 1998*)

Abstract

The basic algorithms for alignment of two or more protein sequences are explained. Alternative methods for scoring substitutions and gaps (insertions and deletions) are described, as are global and local alignment methods. Multiple alignment techniques are explained, including methods for profile comparison. A summary is given of programs for the alignment and analysis of protein sequences, either from sequence alone, or from three-dimensional structure.

1. Introduction

Sequence-alignment techniques are central to modern molecular biology. Alignment methods may be used to search sequence databases for potential homologues. Having identified a family of sequences, then alignment methods may be used to generate a multiple sequence alignment that allows the identification of structurally and functionally important residues. Many different programs for pair-wise and multiple sequence alignment exist. In this article the underlying algorithms used by these programs are summarized.

2. Alignment of two sequences

The most commonly used sequence-alignment techniques have three components. The scoring scheme, the gap model and the alignment-optimization algorithm.

2.1. *The scoring scheme, substitution or pair-score matrix*

The scoring scheme is a 20 × 20 matrix of numbers that defines the value for aligning each of the possible amino-acid pairs. The term substitution is often used for the alignment of two amino-acid residues, since scoring schemes are frequently derived from a model of evolution that considers two protein sequences to be related *via* a series of point mutations. The pair-score matrix is usually symmetrical, since Ala aligned with Gly has the same meaning as Gly aligned with Ala. The simplest scoring scheme is the identity matrix. This scores 1 for an exact match of two amino acids, and 0 for a mismatch. Although the identity matrix is appealing in its simplicity, it does not reflect adequately similarities observed between proteins that have similar three-dimensional structures. More sophisticated schemes take into account conservative substitutions. For example, Val aligned with Leu might score +4, but Glu with Leu, −3. Until recently, matrices referred to as PAM or Dayhoff were the most widely used. PAM matrices were derived by first aligning a small number of families of protein sequences by eye, then counting the observed amino-acid substitutions within the families and normalizing the counts before extrapolating the observed substitutions to those expected at different evolutionary distances (Dayhoff *et al.*, 1978). The measure of evolutionary distance used was the percentage of accepted mutations, or PAM, and the most commonly applied matrix was that at 250 PAMS, normally known as PAM250.

In spite of its small training-set size, the PAM250 matrix captures the principal physico-chemical properties of the amino acids (Taylor, 1986*a*). Furthermore, updates to the PAM matrix obtained from much larger data sets, for example, the PET92 matrix (Jones *et al.*, 1992), show few differences to PAM250, except for substitutions with the less common amino acids such as tryptophan.

The BLOSUM series of matrices (Henikoff & Henikoff, 1993) are also derived from an analysis of observed substitutions in protein families. Unlike PAM matrices, the starting point for BLOSUM is a set of alignments without gaps, obtained by the *BLAST* (Altschul *et al.*, 1990) algorithm (Henikoff & Henikoff, 1992). The alignments include sequences that share much lower sequence similarity than those used in the Dayhoff studies. In extensive tests of sequence database searching (Henikoff & Henikoff, 1992), pair-wise alignment (Vogt *et al.*, 1995) and multiple sequence alignment (Raghava & Barton, 1998) the BLOSUM series of matrices on average give results superior to the PAM matrices and most other matrices. For this reason, BLOSUM matrices are now the general matrix of choice for protein sequence alignment, and are the default matrices used by most popular sequence-alignment and database-searching software.

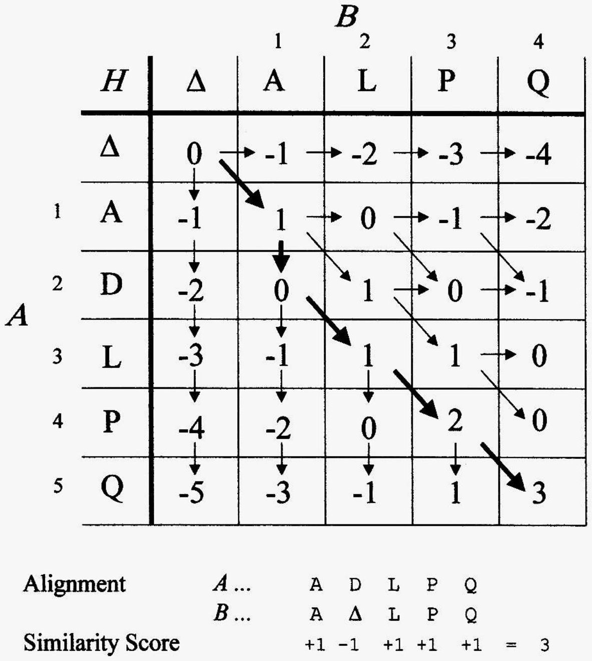
Rather than starting from alignments generated by sequence comparison, Overington *et al.* (1992) only consider proteins for which an experimentally determined three-dimensional structure is available. They then align similar proteins on the basis of their structure rather than sequence and use the resulting sequence alignments as their database from which to gather substitution statistics. In principle, the Overington matrices should give more reliable results than either PAM or BLOSUM. However, the comparatively small number of available protein structures currently limits the reliability of their statistics. Overington *et al.* (1992) develop further matrices that consider the local environment of the amino acids.

2.2. *Dealing with gaps*

Insertions and deletions are observed within protein families, and it is normally necessary to introduce such *indels* when producing an alignment. The simplest scheme for gaps introduces a new character that scores u when aligned with any amino acid. Since gaps are comparatively rare, u is usually made negative. As a consequence, u is often referred to as the gap-penalty. In this simple scheme, a gap of ten residues is penalized ten times more highly than a gap of one residue. Within protein families, this makes little sense since gaps of more than one residue are needed to obtain structurally reasonable alignments. The most commonly used scoring scheme for gaps is a function of the form: $ul + v$, where l is the length of the gap in residues. This form of penalty function is referred to as affine and has efficiency advantages over more elaborate penalty functions (Gotoh, 1982). The constants v and u are often referred to as the penalties for creation and extension of the gap, or length-independent and length-dependent penalties, respectively. Gap penalties need not be uniform across the sequence and such position specific gap penalties are discussed in §2.4.

2.3. *Finding the optimal alignment of two sequences*

The optimal alignment of two protein sequences is the alignment that maximizes the sum of pair-scores less any penalty for introduced gaps. The problem is to find an efficient way of locating an alignment that satisfies these conditions. If no gaps are allowed, then the task is simple, just slide one sequence over the other and for each position, sum the pair-scores from the chosen matrix (*e.g.* BLOSUM62). This simple alignment algorithm requires of the order of 200 alternative alignments to be evaluated for two sequences of length 100. In contrast, if gaps are allowed, then there are $>10^{75}$ alternative alignments that must be considered! An elegant solution to this intractable search problem is given by a class of computer algorithm known as *dynamic programming*. Dynamic programming allows the optimal alignment of two sequences to be found in of the order of mn steps, where m and n are the lengths of the sequences. Dynamic programming for sequence comparison was independently invented in several fields, many of which are discussed in Sankoff and Kruskal's book (Sankoff & Kruskal, 1983). An introduction to dynamic programming in the wider context of string comparison can be found in Gusfield (1997). Needleman & Wunsch (1970) are often attributed as the first application of dynamic programming in molecular biology, while slightly different formulations of the same algorithm were described by Sellers (1974) and Waterman *et al.* (1976). Here, the basic Sellers (Sellers, 1974) dynamic programming algorithm for two sequences is described, with a length-dependent gap-penalty. A simple example of this algorithm is illustrated



			B				
				1	2	3	4
		H	Δ	A	L	P	Q
		Δ	0	-1	-2	-3	-4
	1	A	-1	1	0	-1	-2
	2	D	-2	0	1	0	-1
A	3	L	-3	-1	1	1	0
	4	P	-4	-2	0	2	0
	5	Q	-5	-3	-1	1	3

Alignment *A*... A D L P Q
B... A Δ L P Q
Similarity Score +1 -1 +1 +1 +1 = 3

Fig. 1. Example of a completed dynamic programming H matrix for two short sequences ADLPQ and ALPQ aligning with the simple identity matrix, which scores +1 for a match of two amino acids and 0 for a mismatch. As explained in the text, for proteins, a more sophisticated scoring matrix is normally applied. The aim is to find the maximum score for the alignment of the two sequences allowing for deletions in either sequence. The H matrix is filled by starting at $H_{0,0}$ and proceeding one row at a time. At each cell, equation (1) is applied. In the 0th row and column, there is only one possible predecessor cell and that corresponds to aligning each residue with a gap. In all other cells, there are three possible predecessor cells. (i) A diagonal move corresponds to a substitution. (ii) A horizontal move corresponds to alignment of a residue in B with a gap. (iii) A vertical move corresponds to alignment of a residue in A with a gap. For example, at $H_{4,3}$, the comparison of P at A_4 with P at B_3, the score for the cell is the maximum of 1 + 1 (diagonal move), 0 − 1 (horizontal move), and 1 − 1 (vertical move). The arrows show the cells that contributed to each cell. Some cells have more than one arrow pointing into them since there can be ties for which cell gives the maximum value in equation (1). The cell in the bottom right-hand corner ($H_{5,4}$) contains the best score for the alignment of the two sequences. Following the bold arrows backwards to $H_{0,0}$ gives an alignment with the best score of 3. In this example, there is only one possible alignment with this score since there is no ambiguity on the path leading to $H_{5,4}$.

and explained in Fig. 1. At each aligned position between two sequences $A = (A_1, A_2, \ldots A_m)$, and $B = (B_1, B_2, \ldots B_n)$ of length m and n, there are three possible events,

substitution of A_i by B_j (scores w_{A_i,B_j}),

deletion of B_j (scores w_{Δ,B_j}),

and deletion of A_i (scores $w_{A_i,\Delta}$),

where Δ is the symbol for a single residue gap. The substitution score (w_{A_i,B_j}) is taken from the amino-acid pair-score matrix such as BLOSUM62 (Henikoff & Henikoff, 1992). The two deletion scores are the penalty for aligning a single residue with a single gap. The total score M for the alignment of A with B can be represented as $s(A_{1\ldots m}, B_{1\ldots n})$. This is found by working along each protein sequence from the N to the C terminus, successively finding the best score for aligning $A_{1\ldots i}$ with $B_{1\ldots j}$ for all i, j where $1 \leq i \leq m$ and $1 \leq j \leq n$. The values of $s(A_{1\ldots i}, B_{1\ldots j})$ are stored in a matrix H where each element of H is calculated as follows:

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i,j-1} + w_{\Delta,B_j} \\ H_{i-1,j} + w_{A_i,\Delta} \end{array} \right\}. \qquad (1)$$

Once the matrix is complete, the element $H_{m,n}$ contains the total score for the alignment of the complete sequences.

The processing steps described above determine the best score possible for the alignment of the two sequences given the gap-penalty and pair-score matrix. However, they do not give the alignment. In order to generate an alignment of A and B that gives the best score, it is necessary to trace back through H. An efficient way of completing the trace-back is to save an array of pointers that records which of the three possibilities was the maximum at each $H_{i,j}$. It is possible for there to be ties for the maximum chosen at any $H_{i,j}$. Ties represent equally valid alternative alignments and the manner in which ties are resolved is dependent on the program implementation. For this reason, two computer programs that correctly claim to implement the same dynamic programming algorithm and give the same total score for the alignment of two sequences, may produce subtly different alignments.

Calculation of the best score is more complex if gap-penalties of the form $ul + v$ are employed since it is necessary to determine for each cell whether a gap is starting at that cell, or continuing from an earlier cell. This algorithm typically runs a factor of three slower than the simple algorithm.

2.4. *Position-specific gap-penalties: domains and secondary structure*

In the simple example in the previous section, the penalty for a gap is equal at all locations in the alignment. However, it often makes sense to penalize gaps differently at the ends, or at different positions within each sequence. For example, if a protein domain is being aligned to a longer sequence that is known to contain the domain, the penalties at the end of the domain should be reduced to allow the domain to slide over the longer sequence. If the secondary structure of one protein in a pair to be aligned is known, then increasing the gap-penalty within core secondary-structure elements will reduce the likelihood of placing a gap in a secondary structure (Barton & Sternberg, 1987*a*; Lesk *et al.*, 1986). Both changes require simple modifications to the algorithm. End gaps are adjusted by changing the gap-penalty constants for the 0th and last row and column of the H matrix. Position-specific gaps are set by having a vector of penalties P of length m rather than a single constant Δ. This modifies the calculation of $H_{i,j}$ to

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i,j-1} + w_{\Delta,B_j} \\ H_{i-1,j} + P_i \end{array} \right\}. \qquad (2)$$

In this example, the gap-vector P refers to sequence A. Thus, the weight for aligning any residue in A with a gap will depend on where the residue is in A. In contrast, aligning a residue in B with a gap is penalized equally irrespective of position.

There are many ways of modifying position-specific gap-penalties. For example P can be applied to gaps in both sequences, but dependent only on the position in A, so eliminating the fixed constant Δ, or a second gap-penalty vector can be introduced for B.

2.5. *Position-specific weights: profile comparison*

In the examples in the previous sections, residue substitution weights have all come from a single pair-score matrix such as BLOSUM62. However, the substitution weights (w_{A_i,B_j}) can be made position-specific in the same way as gap-penalties. If the weights are position specific relative to A it means that the weight for matching a residue from A of a particular type to any other residue will depend upon the location of the residue in A as well as the type of the residue in B. For example, a Gly at A_{32} aligning with a Gly in B may have a weight of +7, while a Gly at A_{76} aligned with a Gly in B has a weight of −5. Position-specific weights are useful since they allow the importance of specific substitutions to be emphasized at particular positions along the sequence. This might mean increasing the weight for aligning a known active-site residue in A with a residue of the same type, or more general properties such as increasing the weight for known buried hydrophobic residues to align with similar residues in B.

In order to calculate an alignment score by dynamic programming that includes position-specific weights, a position-specific weight matrix or profile $Q_{m,20}$ for sequence A must be defined. Q contains m rows where each row has 20 weights for substitutions with each amino-acid type at that position. Thus, $Q_{3,S}$ is the substitution weight for a serine in B with position 3 of A. The equivalent of having no position-specific weights is to populate each $Q_{i,A\ldots Y}$ with the appropriate row from the BLOSUM62 or other pair-score matrix. However, if the power of position-specific weights is to be exploited, additional information about A must be included in the derivation of Q. This might be knowledge of the three-dimensional structure of the protein with sequence A, where position-specific weights reflect the local environment of the amino acids (*e.g.* Overington *et al.*, 1992). Or more commonly, the frequencies of observed amino acids at each position in a multiple sequence alignment of sequences similar to A (Gribskov *et al.*, 1987; Barton & Sternberg, 1987*b*, 1990). $H_{i,j}$ when calculated with position-specific weights and gaps for A is modified to

$$H_{i,j} = \max \left\{ \begin{matrix} H_{i-1,j-1} + Q_{i,B_j} \\ H_{i,j-1} + w_{\Delta,B_j} \\ H_{i-1,j} + P_i \end{matrix} \right\}. \qquad (3)$$

Since their parallel development by many authors in the mid 1980's (Gribskov *et al.*, 1987; Barton & Sternberg, 1987*b*; Taylor, 1986*b*), position-specific weighting schemes, or profiles, have formed the basis of many methods for sensitive sequence comparison. In addition, the principle of position-specific weights is at the heart of currently popular techniques such as Hidden Markov Models (HMM's) (Krogh *et al.*, 1994) and generalized profiles (Bucher *et al.*, 1996). A generalization of position-specific weight matrices is to create two Q matrices, one for each sequence. When calculating each element of H, the substitution weights may be combined by averaging. Comparison of two profiles is fundamental to hierarchical multiple alignment algorithms discussed in §3.3.

2.6. *Local alignment algorithm*

The general algorithm described in §2.3 is a global alignment algorithm. If it is not known in advance that the sequences are similar over most of their length, then a local alignment algorithm is preferred. To convert the basic algorithm to a local algorithm for a pair-score matrix that is centred on zero, the 0th row and column of H are initialized to zero, then the calculation of each element of H is modified to

$$H_{i,j} = \max \left\{ \begin{matrix} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i,j-1} + w_{\Delta,B_j} \\ H_{i-1,j} + w_{A_i,\Delta} \\ 0 \end{matrix} \right\}. \qquad (4)$$

The only difference is the addition of a zero (Smith & Waterman, 1981) which has the effect of terminating any path in the H matrix whose score drops below zero. The maximum local alignment score for comparison of A and B is given by the maximum value in H. The alignment is traced back from this cell until cells on the path drop to zero. There may be more than one local alignment between two sequences, perhaps representing repeats, or multiple shared domains. Algorithms exist to find most (Barton, 1993*a*) or all (Waterman & Eggert, 1987) of the alternative local alignments.

2.7. *Alternative alignments for the same two sequences*

The possibility of ties when constructing the H matrix, and hence alternative equally valid alignments has already been discussed in §2.3. A more general problem is that there may be many alternative alignments with scores close to the optimum. Any one of the alternative alignments could be the 'true' biologically meaningful alignment and so it is useful to be able to generate all alignments 'close' to the optimal alignment.

Saqi & Sternberg (1991) determine alternative sub-optimal alignments by first calculating the H matrix and best path. They then identify the cells that contributed to the best path and reduce these by a preset value (normally 10% of the typical scoring matrix value). A new H matrix is then calculated and a new best path and alignment. This process is repeated iteratively to generate a series of global sub-optimal alignments. Zuker (1991) and Vingron & Argos (1990) describe methods to visualize alternative sub-optimal alignments on a dot-plot. A simple way to probe for stable parts of an alignment is to generate an alignment with standard parameters, then modify the gap penalties slightly, re-align and observe which regions of the alignment change (if any).

2.8. *Aligning sequences in linear space*

The standard dynamic programming algorithm requires storage of at least one $m \times n$ matrix in order to calculate the alignment. On current computers, this is not a problem for protein sequences, but for large DNA sequences, or complete genomes, space requirements can be prohibitive. Linear-space algorithms for dynamic programming (Myers & Miller, 1988) overcome this problem by a recursive strategy, albeit at some sacrifice in execution time.

2.9. *Comparing sequences without alignment*

Few generally available programs allow the flexible investigation of alternative alignments. A way of avoiding the problem is to make use of one of the earliest methods of comparing two sequences, the dot-plot, or diagonal plot (Gibbs & McIntyre, 1970; Staden, 1982). Although the dot-plot does not give an alignment directly, it is an effective technique to view simultaneously all similarities between two sequences. Similarities may be obvious in a dot-plot, but missed entirely by a dynamic programming sequence alignment program that only displays the top-scoring alignment.

For two sequences A and B of length m and n, respectively, a matrix $D_{m,n}$ is generated where each element $D_{i,j}$ represents the similarity between sequence segments centred on A_i and B_j. In its simplest form, $D_{i,j} = 1$ if $A_i = B_j$ and $D_{i,j} = 0$ if $A_i \neq B_j$. A graph is plotted with A and B on each axis and a dot plotted whenever $D_{i,j} > 0$. Regions of sequence similarity appear as diagonal lines on the plot and repeats as parallel lines. Insertions and deletions show up as steps in the diagonals as the diagonal moves from one diagonal to another. The most sophisticated dot-plot techniques calculate $D_{i,j}$ from a sliding window, and score similarities by a pair-score matrix such as BLOSUM62 or PAM250. *Dotter* (Sonnhammer & Durbin, 1995) provides a particularly rich set of features including an interactive dot-plot 'contrast' adjustment that simplifies the interpretation of plots.

3. Multiple sequence alignment

3.1. *Extending dynamic programming to more than two sequences*

A multiple sequence alignment is an alignment that contains three or more sequences. In theory, dynamic programming methods for two sequences can be generalized to N sequences by adding dimensions to the H matrix. For three sequences H becomes a cube, and so on. In practice, although software has been developed that will allow three sequences to be aligned by full dynamic programming (*e.g.* Murata *et al.*, 1995), aligning more than three is impractical both in terms of CPU time and memory.

A common technique to save time and space when calculating the global alignment of two sequences is to 'cut corners'. Instead of calculating the entire H matrix, only a window either side of the main diagonal is calculated. This general principle has been extended to the global alignment of multiple sequences (Lipman *et al.*, 1989). The program *MSA* that implements these ideas reliably performs simultaneous alignment of small numbers of sequences (typically <10).

3.2. *Genetic algorithms*

As an alternative to dynamic programming, the genetic algorithm has been applied to multiple sequence alignment (Notredame & Higgins, 1996). It is shown that alignments as mathematically optimal as those generated by *MSA* (Lipman *et al.*, 1989) may be produced.

3.3. *Hierarchical methods of multiple sequence alignment*

Hierarchical methods for multiple sequence alignment are by far the most commonly applied technique since they are fast and accurate. Hierarchical methods proceed in three steps. A schematic example of the stages in hierarchical multiple alignment is illustrated for seven globin sequences in Fig. 2.

For N sequences, all $N(N-1)/2$ unique pair-wise comparisons are made and the similarity scores for the comparisons recorded in a table. The similarity scores, may be simple percentage identities, or more sophisticated measures. For example, the SD score is calculated by first aligning the pair of sequences by dynamic programming and saving the raw score V for the alignment. The two sequences are then shuffled and aligned, usually at least 100 times. The score for each shuffled pair comparison is saved and the mean $\bar{x}$ and standard deviation σ of the scores calculated. The SD score is then given by $(V - \bar{x})/\sigma$. SD scores correct for the length and composition of the sequences, and so are preferable to raw alignment scores, or percentage identity.

An example table of pair-wise SD scores is illustrated at the top left of Fig. 2. Hierarchical cluster analysis is then performed on the table of pair-wise scores. This process generates a dendrogram or tree that groups the most similar pair, the next most similar pair and so on. The tree is illustrated at the bottom left of Fig. 2. Finally, the multiple sequence alignment is generated by following the dendrogram from its leaves to the root. The detailed stages in this process are described in the legend to Fig. 2.

One often-stated drawback to hierarchical methods is that gaps once introduced are fixed. Thus, an error made at an early stage in the alignment process will propagate. Some methods allow a second pass through the alignment where each sequence is re-aligned to the complete alignment in order to correct the worst of these errors (Barton & Sternberg, 1987*b*; Gotoh, 1996).

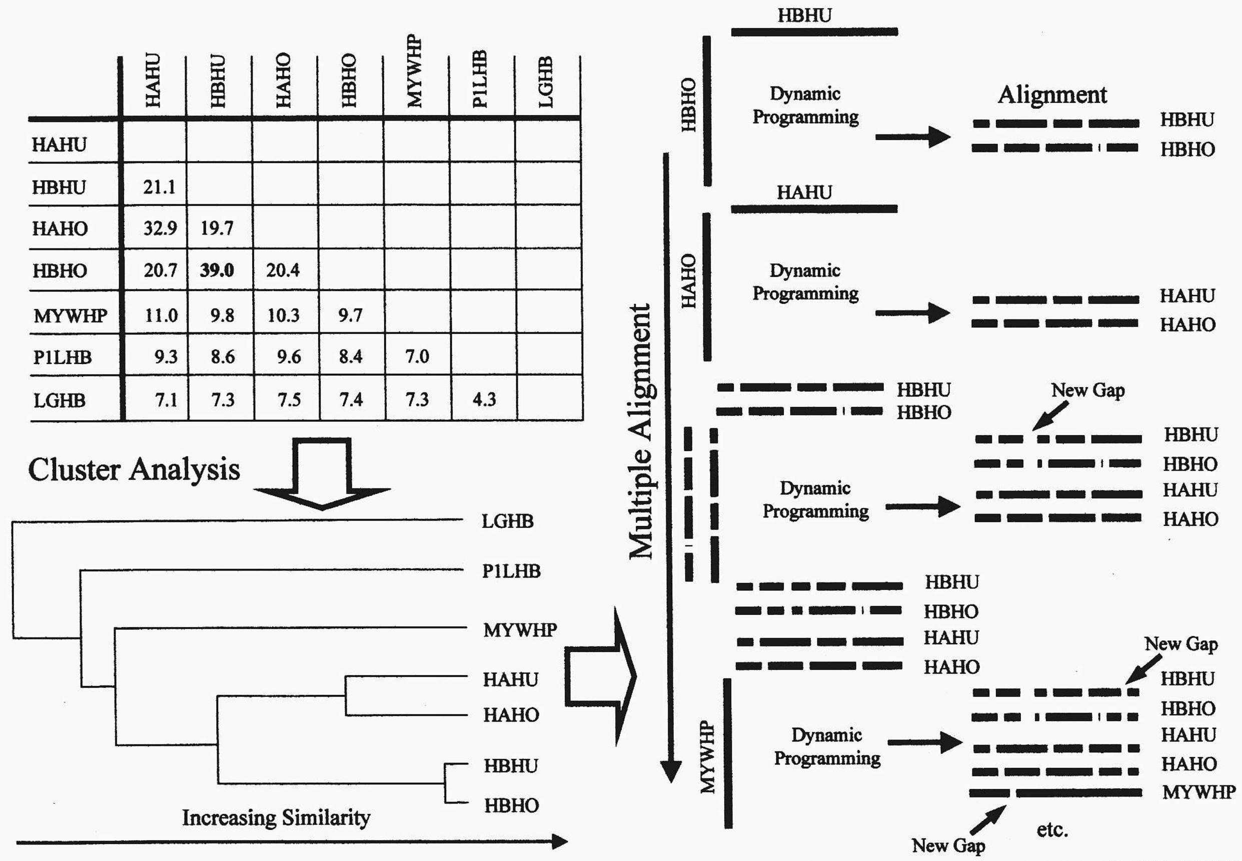
The main differences between different hierarchical methods rest with the techniques used to score the initial pair-wise alignments, the hierarchical clustering method and the treatment of gaps in the multiple alignment. For example, pair-wise alignments may be scored by simple percentage identity, normalized alignment score (where the raw score for alignment of two sequences is divided by the length of the alignment) or a statistical SD score from Monte Carlo shuffling of the sequences. Clustering may be a simple 'add one sequence at a time' method

(Barton & Sternberg, 1987*b*), or single linkage cluster analysis (Barton, 1990), or neighbour joining (Thompson *et al.*, 1994), though any clustering method may be applied. Gaps in the multiple-alignment process may be weighted simply, or position dependent (Thompson *et al.*, 1994). One of the most sophisticated hierarchical multiple-alignment programs is *CLUSTALW* (Thompson *et al.*, 1994). *CLUSTALW* applies different pair-score matrices when aligning sequences of differing similarity. The program also modifies gap-penalties in a position-specific fashion according to analyses of gap preferences in protein sequences (Pascarella & Argos, 1992).

4. How well do alignment methods work?

The pair-wise sequence-alignment algorithms outlined in the previous sections are guaranteed to produce a mathematically optimal alignment for two sequences given the chosen pair-score matrix (*e.g.* BLOSUM62) and gap-penalty function. However, the fact that an alignment is optimal for the matrix and penalty does not mean that the alignment is biologically or structurally meaningful. One can take any two sequences and optimally align them, but an optimal alignment of, say, a globin (an all-α protein) and an immunoglobulin (all-β) is meaningless.

The standard most commonly applied is to test how well the method reproduces the alignment obtained from comparison of the protein three-dimensional structures. In recent tests against structural alignments of 693 test families of domains (Raghava & Barton, 1998), most popular hierarchical methods for multiple sequence alignment aligned over 85% of core residues in agreement with the structural alignment. Alignment accuracy is correlated with sequence similarity (Barton & Sternberg, 1987*b*) and for families with average pair-wise sequence identities of over 30%, the sequence alignments were over 90% in agreement with the reference alignment in the core. In contrast, for those



	HAHU	HBHU	HAHO	HBHO	MYWHP	P1LHB	LGHB
HAHU							
HBHU	21.1						
HAHO	32.9	19.7					
HBHO	20.7	**39.0**	20.4				
MYWHP	11.0	9.8	10.3	9.7			
P1LHB	9.3	8.6	9.6	8.4	7.0		
LGHB	7.1	7.3	7.5	7.4	7.3	4.3	

Fig. 2. Illustration of the stages in hierarchical multiple alignment of seven sequences with the identifying codes: HAHU, HBHU, HAHO, HBHO, MYWHP, P1LHB, LGHB. The table at the top left of the figure shows the pair-wise SD scores for comparison of each sequence pair. Higher numbers mean greater similarity (see text). Hierarchical cluster analysis of the SD score table generates the dendrogram or tree shown at the bottom left of the figure. Items joined towards the right of the tree are more similar than those linked at the left. Thus, LGHB is the sequence that is least similar to the other sequences in the set, while HBHU and HBHO are the most similar pair. The first four steps in building the multiple alignment are shown on the right of the figure. The first two steps are pair-wise alignments by dynamic programming (see §2.3). The third step is a comparison of profiles from the two alignments generated in steps 1 and 2 (see §2.5). The fourth step adds a single sequence (MYWHP) to the alignment generated at step 3. Further sequences are added in a similar manner.

families with 0–10% sequence identity, the average agreement in the core was <25%.

5. Software for sequence and structure comparison and analysis

The basic algorithms for pair-wise and multiple sequence alignment have been described. Here, programs from the author's group that implement some of these algorithms are briefly summarized. All software is available *via* the URL http://barton.ebi.ac.uk.

5.1. *SCANPS – fast local alignment database searching*

SCANPS implements local alignment methods (Smith & Waterman, 1981; Barton, 1993*a*) for protein and nucleic acid sequence database searching. The program has been made to run efficiently in parallel on Silicon Graphics hardware. A WWW server that allows searching of the SWALL non-redundant database of protein sequences is available at the EBI on http://www2.ebi.ac.uk/scanps/. Alternative software is the *SSEARCH* program that is distributed in the *FASTA* package from W. Pearson (ftp://ftp.virginia.edu).

5.2. *AMPS – pair-wise and multiple alignment of sequences*

The *AMPS* package of programs implements pair-wise global alignment with assessment of statistical significance by Monte Carlo methods. It also implements position-specific weights and a variety of profile-based searching and alignment methods. However, the main function of *AMPS* is to produce multiple sequence alignments by a hierarchical method. Most features of the package are reviewed by Barton (1990). *CLUSTALW* (Thompson *et al.*, 1994) is an alternative to *AMPS* for multiple alignment. The program is available from many ftp sites.

5.3. *STAMP – multiple alignment from protein three-dimensional structure comparison*

STAMP (structural alignment of multiple proteins) (Russell & Barton, 1992) produces sequence alignments from comparison of protein three-dimensional structures. The algorithm is similar to that described for hierarchical multiple-sequence alignment, but *STAMP* works from the co-ordinates of Cα atoms rather than amino-acid characters. The program allows two or more structures to be compared, and a gapped pair-wise or multiple alignment and superposition are produced. The quality of each aligned position is assigned a value that allows the reliable regions of the structural alignment to be identified. *STAMP* also allows searching of a structure against a database of structures and has a number of utilities to simplify the manipulation of protein coordinate files and the preparation of publication-quality figures.

5.4. *AMAS – analysis of multiply aligned sequences*

AMAS (Livingstone & Barton, 1993, 1996) assists in the identification of functionally important residues from large protein-sequence alignments. A by-product of this process is a coloured and annotated representation of the alignment. An *AMAS* WWW server is available on http://barton.ebi.ac.uk/servers/amas_server.html.

5.5. *ALSCRIPT – annotation and colouring of a multiple sequence alignment*

ALSCRIPT (Barton, 1993*b*) allows flexible annotation of a sequence alignment. Annotations include the display of helix and strand symbols as well as alignment characters in any font and font size. Output is as a PostScript file that can be printed or viewed.

5.6. *OC – general hierarchical cluster analysis*

The *AMPS* package includes a simple clustering program, *ORDER*, to determine a tree for multiple alignment. *ORDER* is limited at compile time in the number of items it can cluster. In contrast, *OC* is limited only by the computer resources. *OC* also implements a wider range of clustering methods that may be applied to any data for hierarchical clustering.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990). *J. Mol. Biol.* **215**, 403–410.

Barton, G. J. (1990). *Methods Enzymol.* **183**, 403–428.

Barton, G. J. (1993*a*). *Comput. Appl. Biosci.* **9**, 729–734.

Barton, G. J. (1993*b*). *Protein Eng.* **6**, 37–40.

Barton, G. J. & Sternberg, M. J. E. (1987*a*). *Protein Eng.* **1**, 89–94.

Barton, G. J. & Sternberg, M. J. E. (1987*b*). *J. Mol. Biol.* **198**, 327–337.

Barton, G. J. & Sternberg, M. J. E. (1990). *J. Mol. Biol.* **212**, 389–402.

Bucher, P., Karplus, K., Moeri, N. & Hofmann, K. (1996). *Comput. Chem.* **20**(1), 3–23.

Dayhoff, M. O., Schwartz, R. M. & Orcutt, B. C. (1978). *A Model of Evolutionary Change in Proteins. Matrices for Detecting Distant Relationships.* In *Atlas of Protein Sequence and Structure*, Vol. 5, edited by M. O. Dayhoff, pp. 345–358. Washington DC: National Biomedical Research Foundation.

Gibbs, A. J. & McIntyre, G. A. (1970). *Eur. J. Biochem.* **16**, 1–11.

Gotoh, O. (1982). *J. Mol. Biol.* **162**, 705–708.

Gotoh, O. (1996). *J. Mol. Biol.* **264**(4), 823–838.

Gribskov, M., McLachlan, A. D. & Eisenberg, D. (1987). *Proc. Natl Acad. Sci. USA*, **84**, 4355–4358.

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

Henikoff, S. & Henikoff, J. G. (1992). *Proc. Natl Acad. Sci. USA*, **89**, 10915–10919.

Henikoff, S. & Henikoff, J. G. (1993). *Proteins Struct. Funct. Genet.* **17**, 49–61.

Jones, D. T., Taylor, W. R. & Thornton, J. M. (1992). *Comput. Appl. Biosci.* **8**, 275–82.

Krogh, A., Brown, M., Mian, I. S., Sjolander, K. & Haussler, D. (1994). *J. Mol. Biol.* **235**, 1501–1531.

Lesk, A. M., Levitt, M. & Chothia, C. (1986). *Protein Eng.* **1**, 77–78.

Lipman, D. J., Altschul, S. F. & Kececioglu, J. (1989). *Proc. Natl Acad. Sci. USA*, **86**, 4412–4415.

Livingstone, C. D. & Barton, G. J. (1993). *Comput. Appl. Biosci.* **9**, 745–756.

Livingstone, C. D. & Barton, G. J. (1996). *Methods Enzymol.* **266**, 497–512.

Murata, M., Richardson, J. S. & Sussman, J. L. (1985). *Proc. Natl Acad. Sci. USA*, **82**, 3073–3077.

Myers, E. W. & Miller, W. (1988). *Comput. Appl. Biosci.* **4**, 11–17.

Needleman, S. B. & Wunsch, C. D. (1970). *J. Mol. Biol.* **48**, 443–453.

Notredame, C. & Higgins, D. G. (1996). *Nucleic Acid. Res.* **24**(8), 1515–1524.

Overington, J., Donnelly, D., Johnson, M. S., Sali, A. & Blundell, T. L. (1992). *Protein Sci.* **1**, 216–226.

Pascarella, S. & Argos, P. (1992). *J. Mol. Biol.* **224**, 461–471.

Raghava, G. P. S. & Barton, G. J. (1998). *Protein Sci.* Submitted.

Russell, R. B. & Barton, G. J. (1992). *Proteins Struct. Funct. Genet.* **14**, 309–323.

Sankoff, D. & Kruskal, J. B. (1983). Editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison.* New York: Addison Wesley.

Saqi, M. A. S. & Sternberg, M. J. E. (1991). *J. Mol. Biol.* **219**, 727–732.

Sellers, P. H. (1974). *SIAM J. Appl. Math.* **26**, 787–793.

Smith, T. F. & Waterman, M. S. (1981). *J. Mol. Biol.* **147**, 195–197.

Sonnhammer, E. L. L. & Durbin, R. (1995). *Gene-Combis.* **167**, 1–10.

Staden, R. (1982). *Nucleic Acid. Res.* **10**(9), 2951–2961.

Taylor, W. R. (1986*a*). *J. Theor. Biol.* **119**, 205–218.

Taylor, W. R. (1986*b*). *J. Mol. Biol.* **188**, 233–258.

Thompson, J. D., Higgins, D. G. & Gibson, T. J. (1994). *Nucleic Acid. Res.* **22**, 4673–4680.

Vingron, M. & Argos, P. (1990). *Protein Eng.* **3**, 565–569.

Vogt, G., Etzold, T. & Argos, P. (1995). *J. Mol. Biol.* **249**, 816–831.

Waterman, M. S. & Eggert, M. (1987). *J. Mol. Biol.* **197**, 723–728.

Waterman, M. S., Smith, T. F. & Beyer, W. A. (1976). *Adv. Math.* **20**, 367–387.

Zuker, M. (1991). *J. Mol. Biol.* **221**, 403–420.