UWBC Biotechnical Resource Series

Richard R. Burgess, Series Editor
University of Wisconsin Biotechnology Center
Madison, Wisconsin

G. Grant, ed.	**Synthetic Peptides: A User's Guide**
M. Gribskov and J. Devereux, eds.	**Sequence Analysis Primer**
In Preparation:	
D. Nelson and B. Brownstein, eds.	**YAC Libraries: A User's Guide**

Sequence Analysis Primer

Edited by

Michael Gribskov
and
John Devereux

W.H. Freeman and Company
New York

the same type used to compute a dot matrix) and note the average displacements of the highest-scoring segments. This is conveniently done using NBRF's RELATE program. The average displacements of high-scoring segments tend to be multiples of the repeat unit length. For example, RELATE analysis of apo-AIV which contains a fundamental 11-residue repeat, showed additional displacements of 22, 44, 55, 66, etc. residues (Boguski et al., 1986b). Likewise, analysis of *cdc23* which contains a 34-residue repeat, resulted in additional displacements of 68 and 102 amino acids (Sikorski et al., 1990). Finally, both autocorrelation[26] (Kubota et al., 1981) and Fourier methods have been used to determine the period of sequence repeats.

Summary and Future Developments

Dot matrix analysis is a simple, yet powerful, technique for sequence comparison. To paraphrase Collins and Coulson (1987), any comparison of two sequences (or of a sequence with itself) should start with a dot plot. We have seen multiple instances in which failure to heed this advice has delayed the identification of important sequence features.

As useful as dot matrix methods are, there is still considerable room for improvement. No present implementation takes full advantage of modern computer hardware and graphical user interface technology. Dot matrix analysis would also benefit from integration with other types of data analysis and image display tools. The incorporation of multi-length probes (Argos, 1987; Argos and Vingron, 1990) and customized scoring matrices (Altschul, 1991) would improve sensitivity and specificity. Finally, although dot matrix analysis will fundamentally remain a heuristic method of exploratory data analysis, the ability to estimate the statistical significance of the patterns one observes is highly desirable and might be accomplished using a combination of new and traditional methods.

DYNAMIC PROGRAMMING METHODS

Dot matrix methods rely on the power of the human brain to recognize patterns indicative of similarity and to add gaps to the sequences to achieve an alignment. It is, however, quite difficult to be sure that one has obtained the highest scoring, or optimal, alignment when it is made by hand. Because assessments of homology are almost always made on the basis of alignments produced by dynamic programming approaches, we will discuss the method in detail.

[26] Autocorrelation analysis is available in Amos Bairoch's PC/GENE, marketed by IntelliGenetics, Inc. (Mountain View,CA)

A brute force approach of aligning sequences with the automatic insertion of gaps shows that the problem is very difficult. Simply comparing two sequences, without gaps, is equivalent to the computation that takes place in dot matrix analysis, and requires time proportional to the product of the lengths of the sequences (i.e., time proportional to NM, where N=length of sequence 1, and M=length of sequence 2). If the sequences are assumed to be approximately the same length (N), then time proportional to N^2 is required. To account for the presence of gaps, we would have to repeat this calculation 2N times to examine the possibility of gaps in each position of each sequence, for time proportional to N^{*N}. In actuality, the situation is not quite so bad since some of these alignments would be nonsensical, for instance aligning gaps with gaps. An explicit equation has been derived for the number of comparisons that would be required (Waterman, 1989). For two sequences 300 residues long about 10^{61} comparisons would be required, which compares favorably to the estimated 10^{80} elementary particles in the universe.

Fortunately, there is a more efficient way of aligning sequences based on an approach known as dynamic programming. Needleman and Wunsch (1970) introduced this approach to molecular biologists, which is, to this day, frequently referred to as the Needleman-Wunsch algorithm. The dynamic programming method requires only time proportional to N^2, and is based on a simple realization of what the term optimal alignment implies.

Derivation of Dynamic Programming Alignment

If we consider the optimal, or highest scoring, alignment shown in A (below), we can break the alignment into two parts as shown in B. The overall alignment score is the score for the left-hand alignment of four bases plus the score for aligning the two bases on the right. If we assume that the 5 base alignment in A is optimal, we must conclude that the four base alignment in B is also an optimal alignment. If it was not (for example, if we gave a positive score for aligning G with T), the alignment shown in C would give a higher score than the one shown in A. Then C rather than A would be the optimal alignment.

```
AATGC        AATG    C        AATGC
|  ||       |  |  +  |        | .||
AG-GC        AG-G    C        A-GGC
  A             B               C
```

In plain English then, the best alignment that ends at a given pair of bases or residues is the best alignment of the sequences up to that point, plus the score for aligning the two additional bases or residues. Mathematically, we

might say that, for sequence 1 and sequence 2 numbered 1 to i and 1 to j respectively

$$S_{ij} = s_{ij} + \max_{\substack{1\le k<i \\ 1\le l<j}} S_{kl} \qquad (3)$$

where S_{ij} is the score for the alignment ending at i in sequence 1 and j in sequence 2
s_{ij} is the score for aligning i with j.

Removing another pair of bases from B gives us the situation shown in D (below). The next step would require inserting a gap in sequence 2. Brief consideration shows that, at any step, there are only three possibilities: aligning the next base from sequence 1 with the next base from sequence 2; aligning the next base from sequence 1 with a gap; or aligning a gap in sequence 1 with the next base from sequence 2.

```
AAT     G       C
|   +   |   +   |
AG-     G       C
        D
```

This allows us to rewrite equation (3) in a more detailed form:

$$S_{ij} = s_{ij} + \max\begin{cases} S_{i-1\,j-1} & \text{or} \\ \max\limits_{2\le x\le i} S_{i-x\,j-1}+w_{x-1} & \text{or} \\ \max\limits_{2\le y\le i} S_{i-1\,j-y}+w_{y-1} & \end{cases} \qquad (4)$$

where S_{ij} is the score for the alignment ending at i in sequence 1 and j in sequence 2
s_{ij} is the score for aligning i with j
w_x is the score for making a x long gap in sequence 1
w_y is the score for making a y long gap in sequence 2
allowing gaps to be any length in either sequence.

The scores for gaps, w, are negative and are often referred to as gap penalties. This constitutes a virtually complete mathematical description of dynamic programming alignments since each of the terms on the right of equation 4 can, itself, be calculated from equation 4.

Simple Example of Dynamic Programming Alignment

Actual alignments are calculated in two stages. First, the two sequences are arranged on a lattice in much the same way as in dot matrix methods. For each point in the lattice, the alignment score, S_{ij}, is calculated. At the same time, the position of the best alignment in the previous row or column, i.e., the score of the best previous alignment which was used to calculate S_{ij}, is stored. This stored value is called a pointer and is represented by an arrow. In the second stage, the alignment is produced by starting at the highest alignment score in the lattice, and building up the alignment from right to left by following the pointers. This second stage is called the traceback. A graph of the pointers is sometimes referred to as a path graph because it defines the path through the lattice that corresponds to the optimal alignment.

Figure 20 shows a simple example of a dynamic programming alignment of the sequences AGGC (sequence 1) and AATGC (sequence 2). In this example no penalty is applied for introducing gaps so that the optimal alignment is simply the alignment with the most matches. The score matrix,

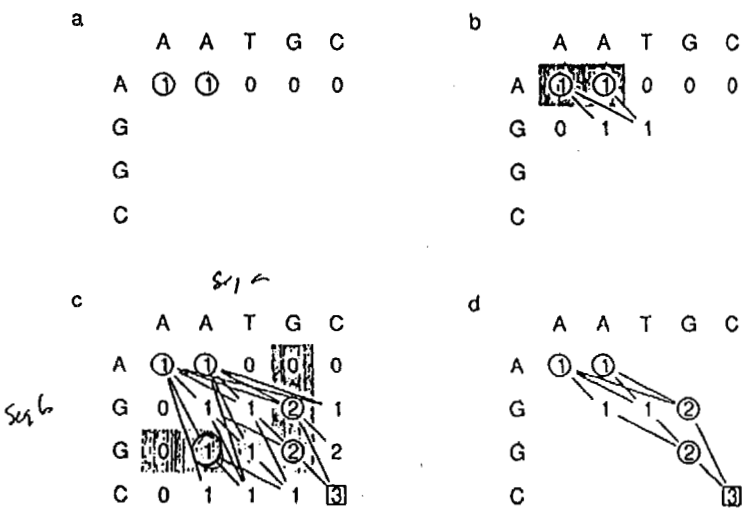

Figure 20: Calculation of dynamic programming alignments. The score matrix and path graph at several stages during the calculation of the alignment. In this calculation only identical matches receive positive scores, and no penalty is applied for gaps. The numbers represent the scores, S_{ij}, in the score matrix, and give the score for the best alignment ending at that pair of aligned residues. The alignment scores at positions where the comparison score for the corresponding two residues, s_{ij}, are positive are circled. The path graph is shown as arrows indicating the best previous alignment at each position in the path graph. a. after calculating the first row of the score matrix. b. after calculation of $S_{2,3}$, and the first several pointers in the path graph. c. after calculation of the last position, $S_{4,5}$, in the score matrix, and completed path graph. d. simplified score matrix and path graph showing only the best path or paths passing through each positive score, s_{ij} (circled).

containing the S_{ij} values, is filled in from left to right and top to bottom. Figure 20a shows the score matrix after filling the first row (i=1). Two positions, indicated by circled scores, are matches and receive a score of 1. All other positions receive scores of zero. The alignment score, S_{ij}, is the sum of the score for comparing the bases at i and j, plus the best previous alignment. Since all of these elements correspond to the first base in sequence 1, there are no best previous alignments and no pointers are saved at this point.

Figure 20b shows a later stage in the calculation of the score matrix. $S_{2,1}$ is an edge and therefore there is no best previous alignment to consider. $S_{2,2}$ has only one position that could contain a previous alignment, $S_{1,1}$, and this is therefore the position used for the pointer. To calculate $S_{2,3}$ we add the score for comparing the G in sequence 2 and the T in sequence 1 (mismatch so $s_{2,3}$=0), to the best previous alignment. The best previous alignment must end in either the previous row or the previous column, above and to the left of $S_{2,3}$. Therefore, we must look for the best previous alignment in both $S_{1,1}$ and $S_{1,2}$ (shown shaded). Since the scores for these positions are the same (in the absence of gap penalties), we store a pointer for each of them.

The final step in the generation of the score matrix is shown in Figure 20c. Setting the pointers for $S_{4,5}$ requires examination of the entire previous row and column (shown shaded) for the best previous alignments. Two equivalent positions are found at $S_{3,4}$ and $S_{3,5}$ and pointers set accordingly. The alignment is generated by following the pointers from the highest score in the score matrix, along a path leading up and to the left. Because an alignment must end in either the last base of sequence 1 or the last base of sequence 2 (the only other possibility being that both sequences end in a gap), the highest score in the score matrix is constrained to lie in either the last row (the last pair of bases in the alignment contain the last base of sequence 1) or the last column (the last pair of bases in the alignment contains the last base of sequence 2). In our example, the highest scoring position is found at $S_{4,5}$ and therefore aligns the last base from each sequence. This position is shown boxed.

Figure 20c is confusing because of the large number of unproductive paths, that is to say, paths that after an early match contain only mismatches. Because optimal alignments must contain matches, it is sufficient to show only the best path or paths passing through each matching position. This simplification applied to Figure 20c gives us Figure 20d. We now perform the traceback to generate the final alignment. Starting from the highest scoring position, $S_{4,5}$, we follow the pointers back, building up the alignment one pair of residues at a time, from right to left. If we follow the lower pointer at each position, we generate the following alignment in four steps:

```
C       GC      G.GC    AG.GC     Sequence 1
|       ||        ||    |  ||
C       GC      ATGC    AATGC     Sequence 2
step 1  step 2  step 3  step 4
```


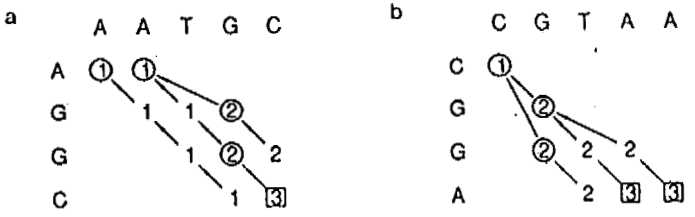

Figure 21: Effect of not saving all path pointers. The alignment in Figure 20 is repeated, but only one path pointer, the one which would introduce the shortest gap, is saved for each position, S_{ij}, in the score matrix. a. forward alignment b. reverse alignment - the same alignment with each sequence reversed. This will often, but not always, give a different alignment when only one pointer is saved for each position.

At step 3 we skip from the G in sequence 2 to the A, leaving the T unmatched. This is a gap; indicated by aligning the T with a null character, in this example a period. The final alignment, as indicated by the score for the alignment, $S_{4,5}$, contains 3 matches. There are actually five equivalent alignments beginninbg with $S_{4,5}$ in Figure 20d, all contatining three matches:

```
AG.GC   A.GGC   .AGGC   A..GGC   .A.GGC   Sequence 1
|  ||   |  ||   |  ||   |  | |   |  | |
AATGC   AATGC   AATGC   AATG.C   AATG.C   Sequence 2
  1       2       3       4        5
```

There are no other ways to align these sequences and get three matches unless gaps are aligned with gaps.

Many alignment programs allow only a single pointer to be set for each position in the score matrix. When two previous alignments have the same score, an arbitrary decision must be made about which pointer to store. Figure 21a shows an alignment example of the same two sequences used above, but storing only the pointer that corresponds to the shorter gap. The resulting optimal alignment is number 3 above and appears to be unique, judging from the path matrix.

This brings to light an important point about alignment programs. Although dynamic programming alignment techniques are guaranteed to find an optimal alignment, there may be other equally optimal alignments, and some of them may be more biologically relevant. The GCG program GAP[27] approaches this problem in a unique way, allowing the option to control which equivalent pointers are saved. The "highroad" option always saves the "upper" pointer, and would result in alignment 5 above, while the "lowroad" option, which always saves the "lower"pointer, would result in

[27] Genetics Computer Group, Inc., Madison, WI.

alignment 1 (Figure 20c). The highroad/lowroad options will always give different alignments if there are equivalent alignments. For long sequences there could be hundreds of equivalent alignments and it is prohibitive and confusing to list them all. The highroad/lowroad procedure can be thought of as establishing an upper and lower bound for the variation of the alignments.

Another way of detecting possible equivalent alignments is shown in Figure 21b. Simply perform the alignment a second time, keeping all parameters the same, but reverse both sequences. This method is not foolproof, as Figure 21b shows, since depending on which of the two optimal paths is reported, the result might be the same as that of 21a.

The problem of equivalent alignments is worse for nucleic acid sequences than for protein sequences because nucleic acids are usually aligned using a scoring system that gives all identical comparisons the same score (usually 1). In combination with the four character nucleic acid alphabet, this makes multiple equivalent paths common. Protein alignments are usually made based on a scoring system that gives a partial score for amino acid residues that are chemically or mutationally similar (see Scoring Systems). In combination with the larger 20 character protein sequence alphabet, this makes equivalent alignments less likely (although still possible).

More Complicated Alignments

The simple alignment shown in Figures 20 and 21 applied no penalties for the introduction of gaps. For longer sequences, gap penalties must be used to produce sensible alignments. For instance, when human pancreatic hormone precursor and chicken pancreatic hormone are aligned without gap penalties (Figure 22a), the high similarity of these homologous peptide hormones does not immediately strike the eye. However, when penalties are applied for the introduction of gaps, as shown in Figure 22b, the similarity is clear. It is worth noting that the last four residues of the chicken sequence are aligned in a non-homologous position in Figure 22a, but correctly in Figure 22b.

Gap penalties were originally applied either as a single penalty, regardless of the length of the gap (Needleman and Wunsch, 1970), or as penalty for each gap character inserted into the sequences (Sellers, 1974). More recently,

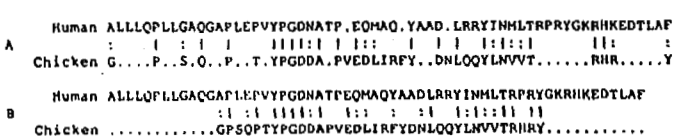

Figure 22: Gap penalties are required for sensible alignments. The alignment of the human pancreatic hormone precursor and chicken pancreatic hormone are shown. Identical matches are shown as '|', and conservative substitutions as ':'. a. alignment without gap penalties. b. alignment with gap penalty of 1.0 + 0.1 x gap length.

most programs have relied on gap penalties with both a length-dependent and a length-independent term (equation 5).

$$w_x = g + lx \qquad (5)$$

where w_x is the penalty for a gap of length x
g is the length-independent term (gap opening penalty)
l is the length-dependent term (gap extension penalty)

The length-independent term of the penalty (g) is applied to all gaps regardless of their length. It can be considered a penalty that is paid when the first base or residue is aligned with a gap character, and is therefore sometimes called a gap opening or gap creation penalty. The length-dependent term (l) of the gap penalty increases with the length of the gap and can be considered to be a penalty paid as each successive position is added to the gap. For this reason, it is sometimes called a gap extension penalty. Fitch and Smith (1983) showed that for globin mRNA sequences, correct alignments could only be made if both terms were non-zero. As can be seen from the Figure 22, gap penalties have a large effect on alignments and it is wise to sample a wide range of values in order to find the most interesting optimal alignments.

Typical values for the gap creation penalty are in the range of one half to five times the score for a match. The gap extension penalty is usually smaller than the gap creation penalty, often in the range of a tenth to one times the score of a match. When these gap penalty values are used, an alignment must gain a substantial number of matches to be worth adding a gap, but a long gap costs only slightly more than a short gap. This coincides with our knowledge of the mutational process which suggests that long insertions and deletions of various lengths can be produced by single mutational events.

Alignment programs vary in their treatment of gaps introduced at the ends of the sequences (end-gaps). If the gap penalties are applied for end-gaps, they are referred to as weighted end-gaps. If the penalties are not applied, we call them unweighted-end gaps. For sequences that are known to be homologous, it makes sense to weight end-gaps. However, if the sequences are different lengths, or of unknown homology, it is probably a good idea to not weight end-gaps. If you are unsure whether the program you are using weights end-gaps, you may be able to find out by adding small amounts of additional sequence to the ends of one sequence and observing the effect on the alignment.

The alignment procedures described above are known as global alignment algorithms, because the resulting alignments contain all characters in both sequences. Short but highly similar subsequences may not be aligned in a global alignment because they are outweighed by the rest of the sequence. One of the most important advances in dynamic programming sequence alignment techniques was the introduction of local alignment methods (Smith and Waterman, 1981). These methods find the two "most similar"

segments of the sequences, and generate an alignment that may contain only part of each sequence. Only small modifications to the algorithm described above are required:

- the scoring system must include negative scores for mismatches
- the minimum score recorded in the score matrix, S, is zero
- the end of the optimal path may be found anywhere in the score matrix, not only in the last row or column.

The key to the local alignment method is point 1. Use of negative scores for mismatches means that once the end of a region of similarity is reached, the scores along the path will begin to decrease. Because we want each short segment of similarity to start from zero, positions in the score matrix which would have a negative score are given a value of zero. A position in the score matrix will either be close enough to an existing path to be linked to it by a gap, or it will be the beginning of an independent region of local similarity.

Local alignment programs are probably the method of choice for unknown sequences, since they will find something quite close to a global alignment if the sequences are highly similar, or the region of highest similarity if they are not. Global alignment of distantly related sequences, on the other hand, may entirely miss their similarity. Figures 23a and 23b show global alignment and local alignments of segments of the phi-X174 *A* gene and *lac* promoters. The global alignment misses the similarity of the -35 regions of the promoters, which is found by the local alignment. Although the -35 regions of the promoters have 6 out of 7 identical bases (86% identical), the optimal global alignment matches only 7 out of 15 (47% identical). This clearly shows the differences between the approaches. The local algorithm finds the segment with the highest local similarity, while the global algorithm chooses a different alignment with higher overall similarity (7 matches) but lower local similarity (only 47% identical versus 86% identical).

Other Derivations

The presentation above follows the original Needleman and Wunsch paper, particularly in finding the alignment that optimizes the similarity between the sequences. Their paper, however, presented neither a mathematical outline of the algorithm nor a proof, and much of the early mathematical work centered on versions of the algorithm presented in terms of distance rather than similarity. In terms of distances, the alignment program finds the alignment that minimizes the distance between the sequences rather than maximizes the similarity, but the procedure is basically the same. The scores for matches are thus 0, and the scores for mismatches some larger value (dissimilar residues are more distant). Path matrices are often shown in a slightly different form, as well, with gaps shown as a series of horizontal and vertical steps along the lattice, rather than a single diagonal jump.

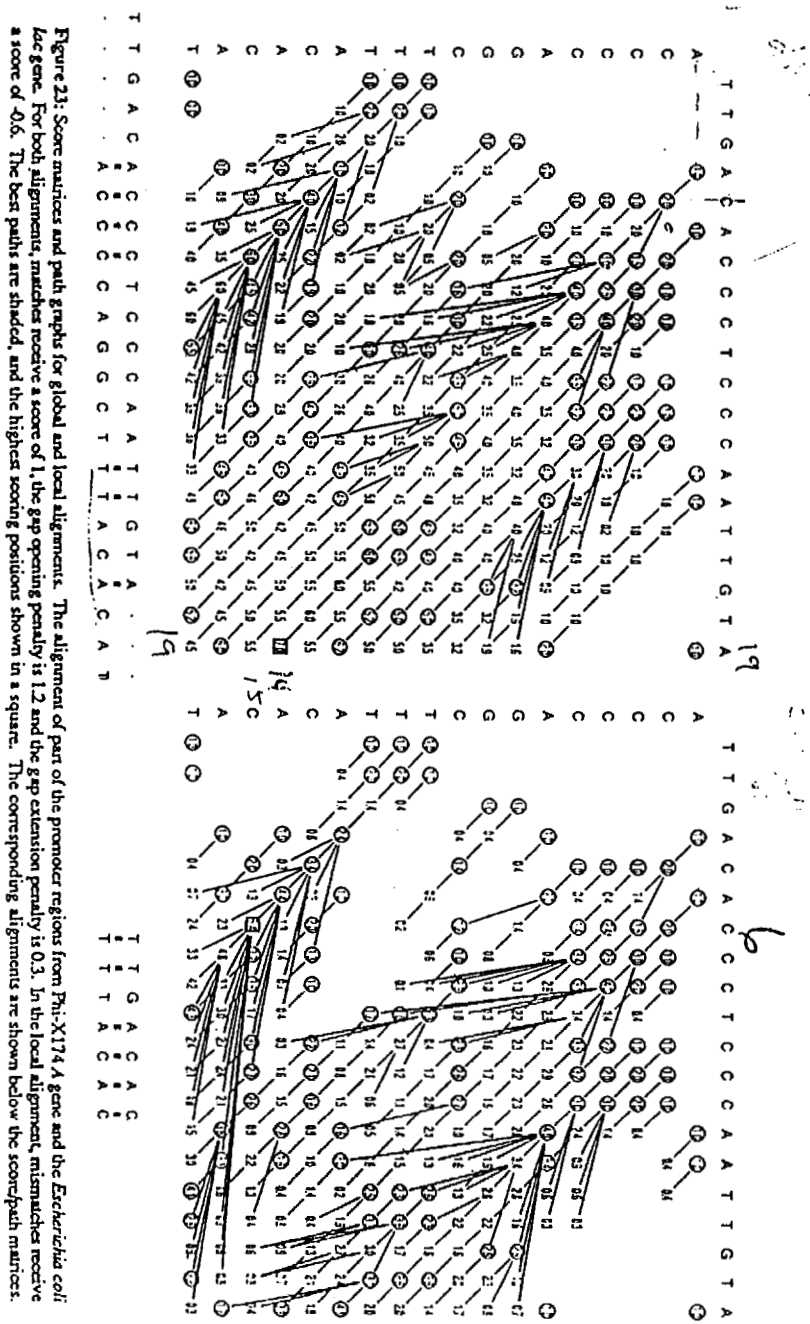

Figure 23: *Score matrices and path graphs for global and local alignments.* The alignment of part of the promoter regions from Phi-X174 *A* gene and the *Escherichia coli lac* gene. For both alignments, matches receive a score of 1, the gap opening penalty is 1.2 and the gap extension penalty is 0.3. In the local alignment, mismatches receive a score of -0.6. The best paths are shaded, and the highest scoring positions shown in a square. The corresponding alignments are shown below the scorepath matrices.

Extensions

The primary drawback to dynamic programming methods is that they require a considerable amount of computation. This limits their usefulness for tasks such as database searching. One simple way to speed up the alignment is to calculate only part of the score matrix, usually a diagonal band down the center (e.g., Sankoff and Kruskal, 1983). This can be safely done, for instance, if you know the sequences are homologous and do not require large gaps in their alignment, or if you have information from a faster method, such as hashing (see Hashing and Neighborhood Algorithms) that tells you where the most similar regions of the sequences are. Several methods that perform a banded alignment and iteratively increase the width of the band until the optimal alignment is found have been presented (Ukkonen, 1983).

A further great increase in alignment speed can be achieved through subdivision. If segments in each sequence can be identified, for instance by hashing methods (see Hashing and Neighborhood Algorithms), that are so similar they are unlikely to match with anything else, the alignment can be broken down into two smaller alignments, separated by the matching segment. Each equal subdivision increases the speed of the alignment by a factor of two.

Once a cDNA clone is sequenced, one usually wishes to identify the protein encoded by the message. One approach is to translate all three (or six) reading frames of the nucleic acid sequence and use the resulting protein sequences as probes in a fast database search (e.g., TFASTA - Lipman and Pearson, 1985; TBLASTN - Gish et al., in preparation). Unfortunately, this approach can be quite sensitive to frameshift errors in the cDNA sequence. An alternative to this approach (States and Botstein, 1990) uses dynamic programming methods to align the DNA and protein sequence.

SCORING SYSTEMS

The simplest scoring systems for molecular sequence analysis give positive scores only to comparisons of identical bases or residues. These scoring tables are referred to as an identity or unitary matrices and are still the primary scoring systems used for nucleic acid sequences.

The average rate of transition (purine to purine or pyrimidine to pyrimidine) mutations is about three times the average rate of transversion (purine to pyrimidine and vice versa) mutations. The rates of insertion/deletion mutations can also be determined from known homologous sequences. These values have been used to calculate scoring tables for nucleic acid sequences based on maximum likelihood methods (Bishop and Thompson, 1986). However, mutation rates and characteristics vary dramatically from species to species, from coding to non-coding regions, and from gene to gene, making it impossible to define a single best scoring system by this approach. In the absence of a single appropriate scoring table, most nucleic acid sequence alignments continue to be based on identity scoring systems.

Identity-based scoring systems often do not give the desired sensitivity when comparing distantly related sequences, especially for protein sequences. There is a strong consensus that, for proteins, scoring systems based on the chemical or mutational similarity of the amino acid residues are much better than identity scoring systems (Schwartz and Dayhoff, 1978; Feng and Doolittle, 1987). One early method of scoring the similarity of amino acid residues is known as the minimum base change or genetic code matrix. This scoring system calculates the similarity between residues as the minimum number of base changes required to change a codon for one residue to a codon for another. This system seemed especially plausible for evolutionary studies because it allowed the difference in amino acid residues to be stated in terms of the minimum number of mutational events needed to convert one residue to another.

The most commonly used scoring systems for protein sequences are based on the MDM_{78} table (mutation data matrix, 1978) of Dayhoff and coworkers (Schwartz and Dayhoff, 1979; George et al., 1990; see Appendix IV). Often called simply the "Dayhoff" table, this scoring table is derived using the "accepted point mutation" model of evolution (Dayhoff et al., 1978). A dataset was compiled from a group of closely related proteins (less than 15% amino acid differences), that could be unambiguously aligned. From these aligned sequences, Dayhoff and coworkers calculated a matrix describing the probability, for each residue, that a mutation would change the residue to each of the other possible residues. The matrix was calculated such that the probabilities represent the average mutational change that will take place when 1 residue out of 100 undergo mutation (1% accepted mutations or 1 PAM). This matrix is called the mutation probability matrix at 1 PAM, or simply the PAM-1 matrix. The specific model of evolution used by Dayhoff and coworkers assumes that more distantly related proteins arise by a series of uncorrelated mutations that can be described by the PAM-1 matrix. Mathematically, this is called a first order Markov chain transition model. To derive a mutational probability matrix for a protein sequence that has undergone N percent accepted mutations, a PAM-N matrix, the PAM-1 matrix is multiplied by itself N times. This results in a family of scoring matrices often referred to as PAM-120, PAM-250, etc.

Because one often desires to know if an alignment is more likely than one between unrelated sequences, scoring systems are often converted to a log-odds matrix. In a log-odds matrix, each element of the matrix, representing the probability that the two corresponding characters are evolutionarily related, is divided by the probability that the two characters could be aligned by chance. To aid in the calculation of probabilities, the values are converted to logarithms. The log-odds form of the PAM-250 matrix is called MDM_{78} and Dayhoff and coworkers recommend using it for all sequence comparisons

although it has been argued that it may be better to use an equivalent log-odds matrix calculated for a lower PAM for alignments of unknown sequences (Altschul, personal communication). The log of the probability of two sequences being evolutionarily related can, in principle, be calculated as the sum of the scores for each aligned pair of residues, i.e., the alignment score if a log-odds matrix is used as the scoring system for the alignment. However, this is overly simplistic since it ignores the effect of insertions and deletions on the probability.

The accepted point mutation model of protein sequence evolution is known to be imperfect in a number of ways. One common criticism is that the frequencies of mutations that require more than one base change in the DNA sequence is higher than would be expected for a simple Markovian model of DNA sequence evolution (Dayhoff and Eck, 1968; Wilbur,1986). This criticism, however, is based on a specific model of DNA sequence evolution which is, itself, open to criticism (George et al., 1990). More importantly, the accepted point mutation model assumes that all residues in a protein are equally mutable; an assumption that is clearly incorrect. This can be easily seen by examining alignments of families of homologous proteins. For a set of six proteins, each sharing a pairwise sequence identity of 35 % or less, one would expect to find not a single amino acid conserved in every sequence if all positions were equally mutable. In actual families of proteins, however, it is not unusual to find several residues that are absolutely conserved in dozens of distantly related sequences (the active site triad of the serine proteases, for example). Furthermore, by starting from aligned sequences with only one or two differences, Dayhoff and coworkers selected mutations occurring at the most mutable positions as the basis for their calculation. Since the most important features in alignments are those positions that are unusually conserved, it has been argued that scoring systems based on the chemical or structural properties of the amino acid residues may produce better alignments (for example, Kubota et al., 1981; Feng and Doolittle, 1985; Argos, 1987; Risler et al., 1988). Lastly, the matrix may be biased because it was derived using mainly small globular proteins as a basis. In spite of its flaws, the MDM_{78} table, or a scoring table derived from it, remains the only widely accepted means of scoring protein sequence alignments. The important position that the MDM_{78} table occupies in molecular sequence analysis is clearly indicated by the fact that every new scoring system compares itself to the MDM_{78} table as a standard.

A special class of scoring systems known as metric distances merits additional consideration. A metric is a distance measure that has the following properties:

- no distance is less than 0
- identical sequence characters have a distance of 0
- the distance is symmetric; that is, the distance from A to B is the same as the distance from B to A
- the scoring system as a whole obeys the triangle inequality; if you consider any three distances, the distance from A to C must be less than or equal to the distance from A to B plus the distance from B to C

Metric distances have received a great deal of attention by mathematicians and were extensively used in the early development of dynamic programming sequence alignment techniques (Sankoff, 1972; Sellers, 1974). The emphasis on metric distances is due mainly to the early use of sequence alignments to provide a measure of evolutionary distance between two sequences. An alignment score generated using a metric scoring system is itself a metric, and thus the most appropriate kind of measure to use in constructing a phylogenetic tree.

Many alignment programs permit one to use a user-specified scoring system. Usually, the alignment program reads the scoring information from a separate file. It is extremely important to use a scoring system appropriate to the algorithm you are using. For global alignments, all of the values in the scoring table must be greater than zero; if they are not, the highest scoring position is not guaranteed to occur at the last row or column of the score matrix, and the resulting alignment is likely to be less than optimal. Local alignment algorithms require that scores for mismatches be less than zero. If you inadvertently use a table with only positive scores with a local alignment program, the result is not so disastrous — the resulting alignment is simply a global alignment.

Fast Methods

To be useful, an algorithm must be able to sensitively and specifically identify sequence relationships while running on available computing systems with satisfactory execution times.[28] Efficiency is a complex issue influenced by many aspects of the computing resources available. High speed microprocessors and dedicated special purpose coprocessors may aide in some phases of a search, while overall performance remains limited by other aspects. I/O requirements, random access memory space and access patterns, and disk access all need to be considered. The dynamic programming algorithm described above is guaranteed to find the optimal alignment between two sequences, but because it requires order N^2 operations, the use of this algorithm for database searching on currently available PCs and workstations is prohibitive. The dynamic programming algorithm is amenable to parallel implementations processing an entire anti-diagonal stripe simultaneously.

[28] Although supercomputers, massively parallel architectures, and specialized chips all have their niches, the great majority of sequence analysis problems are adequately and efficiently handled using general purpose computers.