Sequencing and Sequence Alignment

Objectives:

1) To understand how DNA sequence data is collected and prepared;
2) To be aware of the importance of sequence similarity and sequence searching in biology and medicine;
3) To become familiar with different types of algorithms and scoring schemes used in sequence searching and alignment.

Introduction

Over the past 20 years, sequence comparison has evolved from an obscure pursuit of few evolutionary biologists to a routine event that is performed 100,000's times a day by more than 10,000 different scientists in 100 different countries. This is because sequence comparison is the simplest, quickest and most inexpensive way of determining whether a newly sequenced gene or protein is in fact "new" and whether this new gene might do something interesting. By comparing a sequence to others that have already been painstakingly characterized, it is possible to infer not only functional and structural similarity, but also detailed phylogenetic relationships -- simply on the basis of sequence similarity alone. In many respects, sequence searching and the assessment of sequence similarity lies at the **heart of bioinformatics**.

Sequencing Methodology

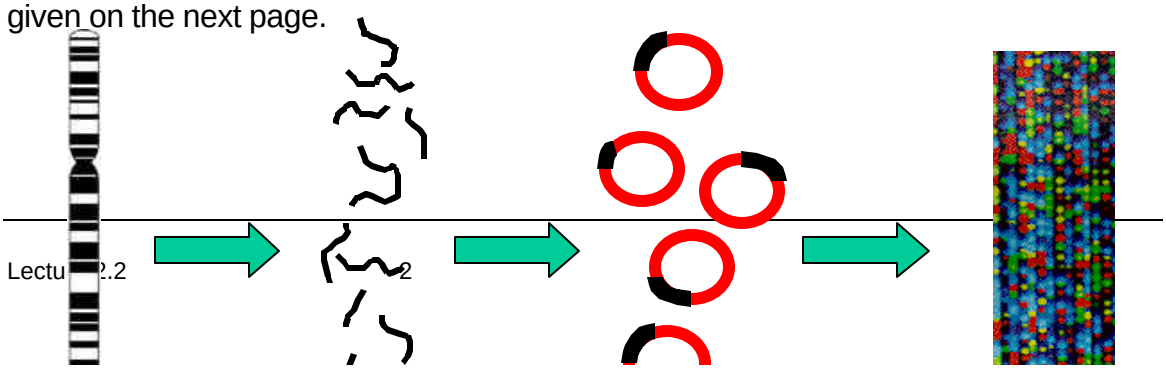
It is important to remember that without our capacity to Perform high throughput DNA sequencing, the field of bioinformatics would not exist. Because this technology is so key to bioinformatics, we will take a short diversion and briefly review the evolution of sequencing technology.

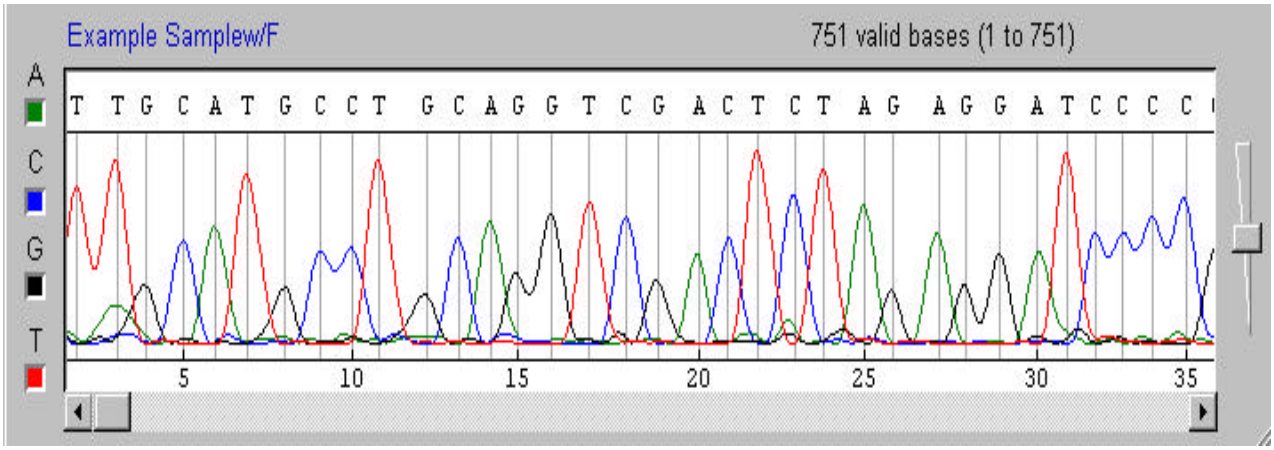
Prior to 1976, almost all sequencing was done directlyon proteins as opposed to DNA. The approach used back then is still the same as used today: Edman Degradation. With the development of Maxam & Gilbert and, later, Sanger sequencing methods in 1976, the field of DNA sequencing was born. The Maxam &

Gilbert is known as the chemical method while the Sanger method is known as the enzymatic method (because it uses a modified fragment of DNA polymerase called Klenow). In the early days of DNA sequencing, one was considered lucky (or very good) if one was able to generate or read more than a few hundred bases a day. Today, automated instruments like the ABI 3700 or the Pharmacia MegaBace 500 DNA Sequencer can routinely generate 500,000 base calls a day.

The development of high throughput sequencing owes much to the development of two key technologies: 1) multiplexed capillary electrophoresis (developed by Norm Dovichi at the U of A); and 2) the synthesis of fluorescently labelled dideoxy nucleotides (by Leroy Hood at Caltech). Multiplexed capillary electrophoresis allowed simultaneous (up to 96) sequencing separations to be performed more accurately, more quickly and with far fewer reagents than traditional slab gel methods. Fluorescently labelled dyes allowed robotic sequencing machines to be built because there was no threat of radioactive contamination.

With the capacity to do high throughput sequencing came the ability to sequence entire genomes. The approach that appears to work best is called Shotgun sequencing. In shotgun sequencing, a chromosome or chromosome fragment is isoloated, and then the DNA is physically sheared (which cuts the DNA at random places). This cut DNA is then ligated into sequencing vectors (pBR322 or M13). Short "universal" DNA primers are then mixed with the vectors along with appropriate Sanger sequencing reagents (ddN's, Klenow, dN's, etc.) and the reaction is allowed to go forward, leading to hundreds of DNA fragments of varying length. These are then run on a sequencing machine which reads off the different colored bands using a fluorescent scanner. From there the DNA is read off and converted to ASCI characters. It is these characters that are then manipulated and aligned in such a way as to reconstruct the original chromosomal sequence. This is called **Contig Assembly** and it is one of the most important applications for all of bioinformatics. A more visual description of what lies behind shotgun sequencing is given on the next page.





So What to Do with All this Sequence Data?

For the rest of this lecture we will focus some of the techniques and/or algorithms that are commonly used to determine if two (or more) sequences are similar and the mathematical techniques that have been developed to evaluate how similar they are. However, before going any further, it is important to develop two important definitions concerning sequence relatedness:

1) **Similarity** - in sequence analysis, this refers to the likeness or percent identity between any two sequences. Sequences are similar if they share a statistically significant number of amino acids in approximately the same positions. Similarity does not infer homology, it is only a descriptive term that carries no suggestion of shared ancestry or origin.

2) **Homology** - in sequence analysis, this refers to a shared ancestry. Two sequences are homologous if they are derived from a common ancestral sequence or if one of the two sequences has diverged (through evolution) to be different in its amino acid sequence from its parent. Homology usually implies similarity.

It is common for many biologists, biochemists and drug researchers to confuse similarity and homology when talking about protein or DNA sequences. Nevertheless, it is important to know that there is a difference and that difference can sometimes be quite profound.

How Can We Tell if Two Sequences are Similar?

Suppose someone gave you the following two "words"

a) `THESTORYOFGENESIS`
b) `THISBOOKONGENETICS`

How could you assess whether these two words were similar and how could you quantify their level of similarity? One way we might do it is to try and match these two words, letter for letter so that all the similar letters (vowels and consonants) could line up, as shown below:

```
THESTORYOFGENESI-S
```

```
THISBOOKONGENETICS
```

After introducing a gap between the last I and the last S in the upper "word" we could say that these two character strings share 10/17 characters or that they are 59% identical. Alternately we could try to parse the words so that similar words are grouped together such as:

```
THE  STORY  OF  GENESIS
THIS  BOOK  ON  GENETICS
```

If we evaluated the level of similarity on the basis of words alone, we would say these character strings have 0% identity, since none of the words match exactly. On the other hand if we evaluated the words on their semantic meaning, this pair of character strings might have 75% "similarity" since only the last two words (GENETICS and GENESIS) seem to have different meanings. Clearly the evaluation of character string and word similarity can get quite complicated depending on how you choose to evaluate the similarity score or how to parse the character strings. Furthermore, if the length of these character strings were to increase dramatically or the level of pairwise similarity were to drop, it might prove to be very difficult to mix and match the right letters or words. As a general rule if we are working at the **level of a DNA or protein sequence, we only have to worry about character-to-character similarity**. However if we were to look at "higher" levels of molecular structure, such as the location of intron/exon junctions in genes or secondary structures in proteins, the parsing and "context" of the character strings can play a more important role.

Now let's suppose that `"THESTORYOFGENESIS"` and `"THISBOOKONGENETICS"` are the (unusually spaced) first lines to two poems by different authors. How might we determine if the line `"THISBOOKONGENETICS"` had been plagiarized from the earlier poem: `"THESTORYOFGENESIS"`? Potentially if you had a large database of all known poems, you could use a computer to compare each line in question to such a database. While extensive

computer databases of written poetry do not yet exist, there are extensive, carefully maintained public databases containing essentially all known protein sequences (SWISS-PROT, TREMBL, GENPEPT, PIR) and all known gene sequences (GENBANK, EMBL, DDBJ). This fact makes it both practical and relevant to develop fast and accurate methods that can perform pairwise string comparisons for DNA and protein sequence data. Because of its potential importance mathematicians and computer scientists have been working, for more than 30 years, on a variety of ingenious methods to perform fuzzy matching and optimal string alignment. Results from their work suggest that sequence (or character string) similarity can be determined and quantified in any one of four different ways:

1) Dot Plots

2) Dynamic Programming

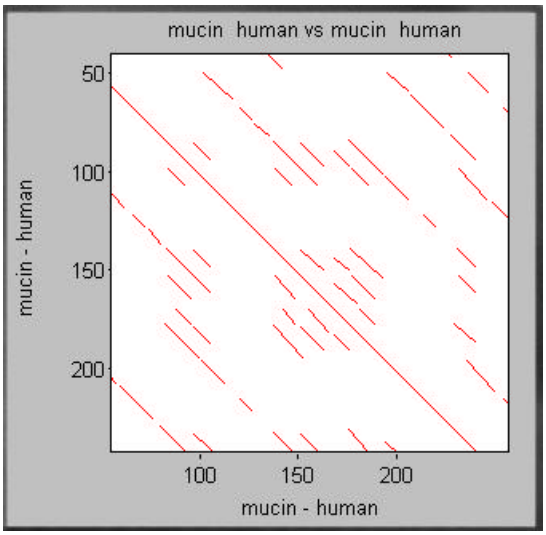
3) Heuristic (Fast) Local Alignment

4) Multiple Alignment

In addition to these methodological developments, mathematicians have also determined that sequence comparison is best done when the sequences (or character strings) exhibit "**complexity**", meaning that they are composed of a large number of different characters. In this regard, protein sequences (unless they are highly repetitive or only have a small number of amino acids) with their 20 letter alphabet are far more complex or informatically "richer" than DNA sequences which only have a 4 letter alphabet. Consequently, string comparisons between two DNA sequences are more likely to yield ambiguous results or potentially false alignments than string comparisons conducted against protein sequences. It is for this reason that most experienced bioinformaticians insist that DNA sequences be translated to protein sequences before they are submitted for pairwise alignment or database comparisons (Doolittle, 1986). Indeed, the only reason why one would not want to translate a DNA sequence into the corresponding protein sequence is if the DNA does not code for any protein product or if it corresponds to a tRNA or rRNA gene. So **ALWAYS TRANSLATE THOSE DNA SEQUENCES!**

Dot Plots

Dot plots were initially developed in 1970 by two biologists (Gibbs and McIntyre, 1970) and later refined by Andrew McLachlan (1971). Dot plots or **dot matrix** methods are the simplest methods for determining sequence similarity. They are primarily **graphical displays** that show similar stretches of amino acids or nucleotides as diagonal lines or diagonal strings of dots (pixels). An example of a dot plot is shown below:



If two sequences are exactly identical, their dot plot will show a single, unbroken line, going from right to left, down the full length of the diagonal. Small dashes or "islands" away from the diagonal (off-diagonal elements) will show up if there are small groups of amino acids that share some local similarity in the sequence. If two sequences are completely different, their dot plot will have almost no lengthy diagonal elements, but rather a series of randomly placed short diagonal dashes (spectral noise). Dot plots may be used to compare two different sequences (**inter-sequence** comparison) or they may be used to compare the same sequence to itself (**intra- sequence**) comparison. Intra-sequence comparisons are particularly good at identifying internal **repeats**, domains or "modules". Inter-sequence comparisons can offer a rich picture of "islands" of similarity between two sequences.

Dot plots are relatively memory intensive comparisons that provide a heuristic or qualitative assessment of similarity. They depend on the user's ability to recognize patterns or shapes. Dot plots are not particularly good for quantification and they are not a "fast" , automatic or necessarily rigorous method for determining alignments or optimal matches. However, because of their graphical nature and because of the many improvements in computer graphical user interface design, dot plots have enjoyed something of a resurgence in recent years.

The basic dot plot algorithm

1) Take two sequences you wish to compare and write out one sequence (sequence A, length=m) as a row and the other sequence (sequence B, length=n) as a column of letters

2) Create an "m x n" matrix with "m" columns and "n" rows.

3) Compare each letter of sequence A with every other letter in sequence B (scanning row by row or column by column) and mark a dot inside this "m x n" matrix whenever two letters match. If two letters do not match, leave the matrix position blank.

4) Scan the resulting dot matrix by looking for diagonals of a set length (say greater than 10 diagonal elements) having some threshold percentage of darkened dots (say 7/10 or 70%). Those diagonals falling below this "stringency" threshold are removed and only the strong or significant diagonals remain.

For protein sequences, in particular, the basic dot plot algorithm can be improved somewhat by making use of scoring functions that weight the similarity of amino acids on the basis of their physical characteristics or relatedness (via PAM matrices). This kind of enhancement allows one to generate a dot plot that displays sequence similarity on a more informative gray scale, instead of a simple binary

(black or white) scale. In addition, the use of a sliding diagonal window and a running calculation of the percent identity (which is compared to some %ID threshold) can allow significant improvements in the signal-to-noise ratio and the overall appearance of the dot plot. Furthermore, the intelligent use of memory and some smart programming (such as hash tables) can also allow dot plots to be generated so that the memory and time requirements scale as N instead of N^2.

Dynamic programming

Certainly dot plots offer a visually appealing and somewhat simple minded approach to identifying sequence similarity, however, they do not offer any hard suggestions as to what diagonals are significant and what diagonals should be connected to prepare a full-length pairwise sequence alignment. While someone with a trained eye might be able to select the three or four best diagonals and link them together to get such a pairwise alignment, it would be far better if we could come up with an algorithm which would automatically calculate the optimal or highest scoring set of diagonals. This is where dynamic programming comes in. Dynamic programming is an efficient mathematical technique that can be used to find optimal "paths" or routes to multiple destinations (such as the traveling salesman problem) or in locating paths that could be combined to score some maximum. The application of dynamic programming to sequence alignment was first shown by Needleman and Wunsch in 1970. Dynamic programming actually permits a **quantitative assessment** of similarity, while at the same time showing how two sequences can be **globally (completely) aligned**.

Dynamic programming is an N^2 algorithm, meaning that every time you double the length of the two sequences you are comparing, the time to analyze them will increase by a factor of four. This makes dynamic programming a rather slow algorithm, however, if you tried to do the same thing "intuitively" you would likely come up with an N^4 algorithm, which would be infinitely slower. While the dynamic programming algorithm is inherently slow, it is mathematically the most rigorous method for determining a pairwise sequence alignment. No other technique (not FASTA, not BLAST or BLAST2) can offer the guaranteed mathematically optimal alignment. While this is a very strong statement, one must always remember that what is mathematically correct, may not be biologically or phylogenetically correct.

In other words, one should always use common sense and a bit of biological intuition when evaluating any given sequence alignment.

In dynamic programming, alignments are calculated in **two stages**. The first stage involves putting the two sequences on a **lattice** or matrix (much like the dot plot algorithm) and then applying a special recursive scoring function to each position (in a defined order) in this matrix. Once the matrix is filled out, one scans through the matrix in a diagonal fashion (called a **traceback**) to look for the highest scores entered in the matrix. The path that is chosen is actually a series of "maximum" numbers. When all the scores in this path are added together, it gives a quantitative measure of the pairwise sequence similarity while at the same time defining which letters in one sequence should be matched with the letters in the other sequence.

$$S_{ij} = s_{ij} = \max \begin{cases} S_{i-1,} & \text{or} \\ \max\limits_{2<x<i} S_{i-xj-1} + w_{x-1} & \text{or} \\ \max\limits_{2<y<i} S_{i-1\,j-y} + w_{y-1} & \end{cases}$$

The recursive function that is used in scoring is written as follows:

S_{i-j} 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

The dynamic programming algorithm

1) Take two sequences you wish to compare and write out one sequence (sequence A, length=m) as a row and the other sequence (sequence B, length=n) as a column of letters

2) Create an "m x n" matrix with "m" columns and "n" rows.

3) Start with position (1,1) in this matrix and determine the score using the above recursive function. In particular, if there is a match put a score of 1, say, in this matrix position. If there is no match put a 0. (This is called the unity scoring matrix).

	A	A	T	V	D
A	1				
V					
V					
D					

4) Move to position (1,2) in the matrix and determine the score using the same procedure.

	A	A	T	V	D
A	1	1			
V					
V					
D					

5) Carry on with the same procedure through the entire first row. You should get the following:

	A	A	T	V	D
A	1	1	0	0	0
V					
V					
D					

6) Now move to row 2. This is where it gets more complicated. In position (2,1) you compare the V with the A and determine if there is a match (which there isn't) and assign a "temporary score" of 0. Now check to the upper left of position (2,1) and add the maximum value of any number listed to the left of (2,1) in row 1 to the temporary score given in (2,1). Since there isn't any number to the left of this position, we add 0 and so our "final score" is 0 + 0 = 0.

```
  A A T V D
A 1 1 0 0 0
V 0
V
D
```

7) Now moving to position (2,2) you compare V with A and determine if there is a match (which there isn't) and assign a "temporary score" of 0. Now check to the upper left of position (2,2) [which is position (1,1)] and add the maximum value of any number listed in (1,1) and to its left in row 1 to the temporary score given in (2,2). Since (1,1) has a value of 1, we add 1 and so our "final score" is 0 + 1 = 1.

```
  A A T V D
A 1 1 0 0 0
V 0 1
V
D
```

8) Now moving to position (2,3) you compare V with T and determine if there is a match (which there isn't) and assign a "temporary score" of 0. Now check to the upper left of position (2,3) [which is position (1,2)] and add the maximum value of any number listed in (1,2) and to its left in row 1 to the temporary score given in (2,3). Since (1,1) or (1,2) has a value of 1 we add 1 and our "final score" is 0+1=1.

```
  A A T V D
A 1 1 0 0 0
V 0 1 1
V
```

```
D
```

9) Now moving to position (2,4) you compare V with V and determine if there is a match (which there is) and assign a "temporary score" of 1. Now check to the upper left of position (2,4) [which is position (1,3)] and add the maximum value of any number listed in (1,3) and to its left in row 1 to the temporary score given in (2,4). Since both (1,1) and (1,2) have a value of 1 we add 1 to our temp score and our "final score" is 1 + 1 = 2.

```
  A A T V D
A 1 1 0 0 0
V 0 1 1 2
V
D
```

10) Now moving to position (2,5) you compare V with D and determine if there is a match (which there isn't) and assign a "temporary score" of 0. Now check to the upper left of position (2,5) [which is position (1,4)] and add the maximum value of any number listed in (1,4) and to its left in row 1 to the temporary score given in (2,4). Since both (1,1) and (1,2) have a value of 1 we add 1 to our temp score and our "final score" is 0 + 1 = 1.

```
  A A T V D
A 1 1 0 0 0
V 0 1 1 2 1
V
D
```

11) This process is repeated for row 3 and row 4, leading to the following "final" matrix

```
   A A T V D
A 1 1 0 0 0
V 0 1 1 2 1
V 0 1 1 2 2
D 0 1 1 1 3
```

12) At this stage you are ready to perform the traceback procedure which allows you to identify the optimal path and therefore determine the optimal (i.e. maximum) score and the true alignment. So starting at the bottom right corner (position [4,5]) we circle "3" and then move up and leftward looking for maximum values in any give row or column. Typically we would choose the number in [4,5]; [3,4]; [2,3] and [1,2].

```
   A A T V D
A 1 1 0 0 0
V 0 1 1 2 1
V 0 1 1 2 2
D 0 1 1 1 3
```

This gives a score of 3 + 2 + 1 + 1 = 8 and an alignment that could be written as:

```
A A T V D
  | | | |
  A V V D
```

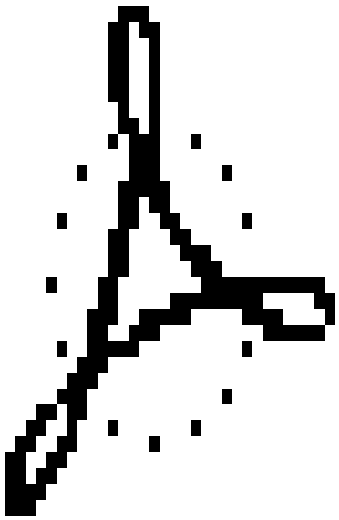
An equally plausible alignment with the same score is:

```
A A T V D
|   | | |
A - V V D
```

If we included gap penalties and incorporated a more realistic scoring matrix (say one that scored amino acids on their chemical similarity as well as their identity) we could have distinguished between these two possibilities more clearly.

Scoring Matrices

The scoring system we just described used a simple match/mismatch scoring function. Such a scoring system can be described by a simple table where we place all 20 amino acids on both the X and Y axes and indicate with a 1 or a 0 (inside the table) whether any two amino acids match or don't match. The scoring table would look like this:



Acrobat Document

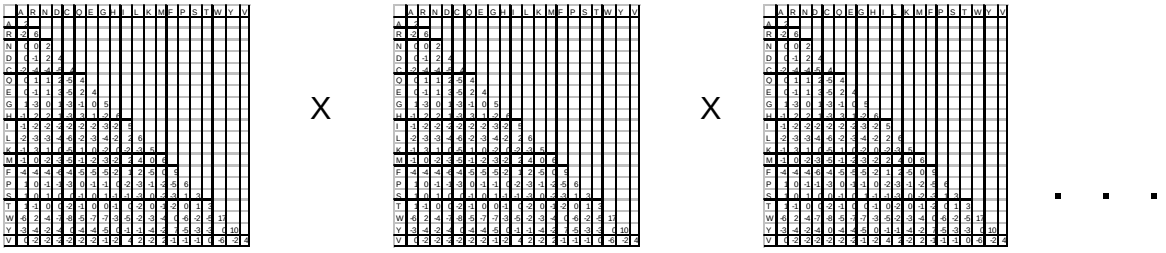
This table is referred to as an **identity** or **unitary** scoring matrix and it is still the most commonly used scoring system for aligning or matching nucleic acid sequences. Attempts have been made to improve on the unitary scoring matrix, especially for DNA, by taking into account the fact that certain types of mutations are more likely to occur than others (transitions are three times more common than transversions), however, these refined matrices appear to perform no better than the simple unitary methods. This may be due to the fact that the tremendous variability in DNA mutation rates and mutation locations overwhelms the small improvements expected from a more refined scoring system.

However, in the case of protein sequences it is quite clear that information on the chemical or mutational similarity of individual amino acids, if included in the scoring matrix, can make a significant difference (Feng et al., 1985). One of the earliest, and most evolutionarily logical methods of scoring amino acid similarity is based on the similarity of amino acid codons. It is called the **minimum based change** or **genetic code scoring matrix**. In this scoring system the similarity score is based on the number of base changes needed to change one amino acid codon to another. For instance, a common codon for Serine is AGC and a common codon for cysteine is UGC. Simply by changing an A to a U we can convert Ser to Cys. Thus the similarity score between Ser and Cys would be rated as high. On the other hand, Alanine is often coded for by a GCU codon. To go from Ala to Cys would require three base changes (both transitions and transversions) so we would score the similarity between Cys and Ala as low. By weighting the frequency of certain codons and the types of mutations needed to get from codon A to codon B, a very precise scoring matrix can be developed.

Dayhoff (PAM) Scoring Matrix

Up until recently the most common scoring system for proteins was based on the mutation data matrix (MDM_{78}) developed by Margaret Dayhoff and coworkers in 1978 (Schwartz and Dayhoff, 1979). It was derived using the **Point Accepted Mutation (PAM)** model of evolution developed by Dayhoff. Using a set of proteins that were >85% identical, Dayhoff and her colleagues manually aligned these proteins, taking great pains to ensure the alignments were completely unambiguous. From these groups of alignments, the probability for each residue to

change to any other residue was calculated. The initial scoring matrix was calculated such that the probabilities in the scoring table would represent the average mutational change when one residue in 100 mutates (1% point accepted mutations or 1 PAM). This particular matrix was called the PAM-1 matrix. Now by assuming that the mutations that occur in a protein sequence are essentially uncorrelated with previous mutations, it is possible to use the mathematics of Hidden Markov Models (or first order Markov chain transitions) to calculate a mutational probability matrix that has undergone N PAM's. To do so, all you have to do is multiple the PAM-1 matrix by itself N times.



Using this model Dayhoff created a family of scoring matrices, with the most useful being the PAM-120 and PAM-250 matrices. These matrices were created by multiplying the PAM-1 matrix by itself 120 and 250 times respectively. As a general rule, for the alignment of sequences that are highly divergent, the best results are obtained with matrices having higher PAM values. In other words, you can think of PAM-120 and PAM-250 as being the typical scoring matrices of proteins that have been separated by 120 million (PAM-120) and 250 million (PAM-250) years of evolution. Dayhoff also converted the PAM-120 and PAM-250 matrices into **log-odds matrices**. To do so, the original scoring matrix probabilities (called **target frequencies**) were divided by the frequencies of each amino acid found in the original alignment set (called **background frequencies**). The natural log of these ratios was then determined and the numbers were substituted into the scoring matrix. The log-odds form of the PAM-250 matrix is called $\mathbf{MDM_{78}}$ and this is the one that Dayhoff recommended for use in general sequence comparisons. The MDM_{78} or PAM-250 matrix is shown below:

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	4															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	3				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-2	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

As can be seen in this matrix, large positive values indicate a high level of mutational (and often chemical) similarity between a pair of amino acids. Large negative values indicate very little mutational or chemical similarity. In principle the sum of these log-odds scores over the length of any pair of aligned sequences is equal to the natural log of the probability that the two sequences are related.

The BLOSUM Scoring Matrix

A more recent addition and increasingly popular set of scoring matrices are the BLOSUM substitution matrices (Henikoff and Henikoff, 1992). These were prepared in a similar fashion to the PAM matrices described earlier, although the alignment data set and matrix construction techniques differ in detail. In particular, the substitution probabilities or target frequencies were estimated from the BLOCKS database (Henikoff and Henikoff, 1991) which is a database of local (contiguous) multiple alignments for various distantly related sequence blocks or domains found in proteins. As with the PAM matrices there is a numbered series of BLOSUM matrices, such as BLOSUM 30, BLOSUM 62 and BLOSUM 90. The numbers refer to the maximum level of pairwise sequence identity in the selected

sequence BLOCKS used to prepare the scoring matrix. So for the BLOSUM 62 matrix, substitution frequencies are influenced primarily by those sequences sharing greater than 62% sequence identity. In general one would use the BLOSUM 90 matrix to compare very similar sequences and the BLOSUM 30 matrix to compare very different sequences or to detect more remote homologies. The BLOSUM 62 scoring matrix, which is the default scoring matrix used in BLAST, is shown below:

Other Scoring Matrices

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

The assumptions built into the models behind the PAM and BLOSUM matrices are known to be slightly flawed. Both are constructed on the supposition that all positions in a sequence are equally mutateable. We know from practical experience that this is not the case, as active site sequences or critical regulatory regions will tend to be absolutely conserved even among the most remotely related sequences. For these and other reasons it has been argued that scoring systems based on the physico-chemical features of amino acids (hydrophobicity, secondary structure propensity, chemical similarity, etc.) could potentially produce better

alignments than those based on a mathematically convenient model of evolution. One example of a physico-chemical scoring matrix is the Boyko matrix (Wishart et al., 1994) which is based on amino acid secondary structure propensity and chemical similarity. The Boyko weight matrix was originally developed to serve as an alternative to the Levin Matrix (Levin et al. 1986; 1988) for homology based secondary structure prediction. The Levin matrix is a highly simplified, manually derived scoring matrix that uses only a small range of numbers (1, 0, -1, -2) inferred by studying amino acid substitutions in a small (<60) database of protein structures. In order to develop a more effective scoring scheme with a broader range of optimal values, the original Levin matrix was adjusted to allow its values to assume any positive integer between 0 and 21. Beginning with this scaled matrix a Monte Carlo procedure was used to iteratively modify each of the 200 values in the initial scoring matrix by adding or subtracting integer values (with the requirement that no scoring matrix entry could be less than zero). Each modified matrix was then tested for its secondary prediction accuracy (using the Levin homology algorithm) by comparing the predicted structures with a database of 267 non-homologous X-ray and NMR structures. This process was repeated until a stable high-scoring matrix was found. This new matrix has been found to produce consistently superior results in homology based structure prediction and to perform at least as well as the PAM250 matrix in identifying homologous sequences (Wishart et al., 1994). The Boyko matrix is shown on the next page:

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	10																			
C	2	17																		
D	2	2	10																	
E	2	2	6	10																
F	1	0	0	0	12															
G	2	2	2	2	0	10														
H	3	1	2	2	3	2	10													
I	2	2	0	0	6	0	1	10												
K	3	2	3	3	0	2	3	0	10											
L	2	2	0	0	6	0	0	4	0	10										
M	2	2	0	0	4	0	2	4	1	8	10									
N	2	0	6	4	0	3	3	0	7	0	1	10								
P	2	0	3	0	0	2	3	0	2	0	0	2	10							
Q	2	0	4	6	0	1	4	1	4	0	1	6	2	10						
R	2	2	2	2	0	2	4	0	7	0	1	2	2	4	10					
S	5	2	2	2	0	2	1	0	2	0	0	4	3	2	3	10				
T	3	2	2	2	0	2	2	2	2	2	2	2	2	2	2	4	10			
V	2	2	0	0	3	1	1	8	0	6	4	0	0	0	0	0	2	10		
W	0	0	0	0	5	0	1	0	0	1	1	0	0	1	3	1	0	0	16	
Y	0	1	0	1	9	0	2	3	0	2	3	0	1	1	0	1	0	3	5	12

Gap and Gap Extension Penalties

In addition to scoring matrices, the use of gap and gap extension penalties can also increase the sensitivity of a sequence search and the utility of the resulting alignment. Gaps and gap extensions must often be introduced in alignments to accommodate deletions or insertions that may have occurred during the evolutionary divergence of one sequence from another. An example of two sequences aligned with gaps (marked by dashes) is shown below:

```
ACDEFGH-K-MNPQRRQSTVW-ACDEFGH-K-MNPQRRQSTVW-
A-DEFGHIKLMNPQR--STVWYA-DEFGHIKLMNPQR--STVWY
```

Dynamic programming algorithms can be easily modified during the **traceback** step to include gap penalties and gap extension penalties. Most commonly, a fixed deduction is applied to the alignment score when a gap is first introduced then an extra deduction is added which is proportional to the length of the gap. Typically the gap opening penalty is called G and the gap extension penalty is called L.

Therefore the total deduction for a gap of length n would be G + Ln. The selection of gap parameters is very empirical and this represents one of the greatest weaknesses in dynamic programming and in sequence alignment in general. For the BLOSUM62 matrix, it is usually recommended that a high gap penalty (G) of 10 to 15 be used along with a low value for L (1 or 2). As a rule of thumb, the gap penalty G should be approximately equal to the difference between the highest and the lowest number in the chosen scoring matrix.

Other Dynamic Programming Techniques: The Smith-Waterman Algorithm

The Needleman-Wunsch algorithm was specifically developed to do global pairwise sequence alignments. In other words, the scoring function and traceback procedure implicitly assume that the optimal alignment will be found by going from the lower right corner to the upper left corner of the matrix. A simple modification to the NW algorithm introduced by Smith and Waterman (1981) showed that dynamic programming could actually be adapted to perform **local** alignments. In other words, the path for the traceback procedure did not have to start and end at the edges of the search matrix, but it could start and end internally (i.e. within the search matrix). Such an alignment, once found, would be locally optimal. In other words, the Smith-Waterman algorithm allows one to identify short subsequences between two long sequences that may have some shared similarity. The advantage of identifying a locally optimal alignment, particularly for proteins, is that it may be possible to pull out small phylogenetically or functionally related domains that would normally be missed by a global alignment algorithm. Because of their potential advantage in identifying remote or frequently missed sequence similarities, local alignment techniques have been the subject of considerable interest development over the past decade. Two local alignment methods in particular, namely FASTA (Lipman and Pearson, 1985; Pearson and Lipman, 1988) and BLAST (Altschul et al., 1990; Altschul et al., 1997) will be discussed in detail in the next section.

Fast Local Alignment Methods

Dynamic programming is an excellent and efficient method for comparing two sequences and obtaining a global (i.e. entire) sequence alignment. However, if one

wishes to perform 400,000+ comparisons using this approach -- as is typical of a database search these days -- it could easily take several hours on a fast (400 MHz) computer. Given that most people do not want to wait that long for an answer, there has been a great deal of effort (especially over the past 10 years) directed at developing methods to improve search speeds so that database comparisons could be done more quickly. However, improvements in speed usually come with a sacrifice in accuracy or precision. Nevertheless, the advent of such fast algorithms as FASTA (Lipman and Pearson, 1988) and BLAST (Altschul et al., 1990; Altschul et al., 1997) has revolutionized the process and frequency of sequence comparison and database searching. Their tremendous speed advantages seem to far outweigh their potential shortcomings in sensitivity.

Most fast local alignment algorithms are based on what are called **Hash tables**. Hash tables are lists of locations or addresses that allow computers to wade through and compare data more efficiently. Hash tables for sequence alignment are usually built by scanning the query sequence and assigning locations to short words or strings (called "k-tuples"). A **k-tuple** is a word or character string that is k letters long. Once this table is built, it is possible to compare other database sequences to the query sequence very quickly. In fact, it turns out that these hash-based methods scale linearly as the size of the sequence or the size of the database (i.e. they are **N-type algorithms as opposed to N^2** algorithms). This kind of algorithmic scaling makes them excellent choices to scan very large (and rapidly growing) databases with little cost in time.

A General Fast Local Alignment Algorithm (FASTALIGN)

1) Take the query sequence and generate a hash table indicating the location of each 3-tuple word in the sequence. For instance, if the query sequence was:

ACDEFGDEF

The hash table would be:

ACD 1

CDE 2

DEF 3,7

EFG 4

FGD 5

GDE 6

2) Using a previously defined set of homologous 3-tuples, expand this hash table so that related words could be included in the table. For instance the above hash table could be expanded to:

ACD 1

ACE 1

ACN 1

CDE 2

CDD 2

CNE 2

etc.

3) Once the hash table is built, begin to scan the database. 3-letter segments from each sequence in the database are sequentially tested against this query sequence hash table. If a match is found (say CDD in position 2 of the query matches CDD in position 187 of a database sequence) then this match information is saved. If no match is found, then no information about that 3-tuple is saved. Both the number of hits and location of these hits are calculated for each database sequence. During this scanning process, the whole hash table is scanned for each database sequence read in.

4) Those database sequences having an unusually high number of hits or matches (determined by statistics - see later) or a cluster of matches corresponding to the spatial location of the matches in the query protein are further analyzed. In particular, they may be forwarded to a dynamic programming algorithm for global alignment or, alternately, the clusters of matches previously

identified may be grouped and parsed to produce a short local alignment and the statistical significance of this subsequence alignment assessed.

FASTA

The above fast alignment protocol is very similar to the algorithm used in FASTA (Pearson and Lipman, 1988). Essentially FASTA looks for word (**k-tuple**) clusters or, using the terminology from dot plots, **clusters of nearby diagonals**. The ten regions with the highest density of matching words or k-tuples are then trimmed and scored (using a PAM250 or BLOSUM62 matrix). This best scoring region from this step is reported as the **init1 score**. Those regions around the best scoring diagonal are then joined to form a single region. The score for this set of linked diagonals is then calculated using appropriate gap penalties and reported as the **initn score**. The initn is used to sort the initial output of the search and select which sequences will be further evaluated. Finally a full blown dynamic programming (Smith-Waterman) step is performed over a narrow diagonal band (32 residues wide) centred about the best (init1) region. The score reported for this step is called the **opt or optimized score**. An example of a FASTA alignment is shown below:

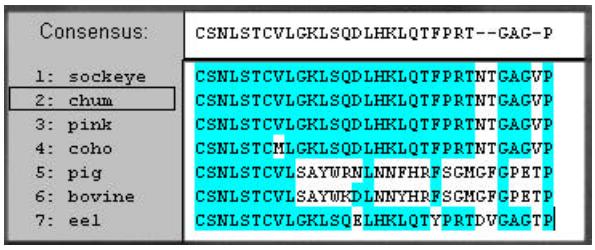
```
gi|135775|sp|P08628|THIO_RABIT THIOREDOXIN                                  (104
aa)
 initn: 641 init1: 641 opt: 642 Z-score: 806.4 expect() 3.2e-38
Smith-Waterman score: 642;  86.538% identity in 104 aa overlap (2-105:1-104)

gi|135    2- 105: ------------------------------------------------------------
----:

               10        20        30        40        50        60        70
80
thiore
MVKQIESKTAFQEALDAAGDKLVVVDFSATWCGPCKMINPFFHSLSEKYSNVIFLEVDVDDCQDVASECEVKCTPTFQFF
        ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::
gi|135
VKQIESKSAFQEVLDSAGDKLVVVDFSATWCGPCKMIKPFFHALSEKFNNVVFIEVDVDDCKDIAAECEVKCMPTFQFF
                10        20        30        40        50        60        70

               90       100
thiore KKGQKVGEFSGANKEKLEATINELV
       :::::::::::::::::::::::.
gi|135 KKGQKVGEFSGANKEKLEATINELL
      80        90       100
```

The sensitivity of a FASTA search can be controlled by several parameters, including the choice of the scoring matrix, the choice of the gap penalty and the size of the word or k-tup value. In FASTA the default ktup value for proteins is normally set to 2, but if you need to look for very distant relationships, the ktup value should be set to 1. The newest release of FASTA (FASTA 2.0) includes an estimate of the statistical significance of each initial alignment as determined by extreme value statistics (see later). It also calculates and expectation value E, which is similar to the value calculated by BLAST. In general FASTA is a somewhat slower program (about 5 X slower) than BLAST. However, one of the long-time advantages of FASTA over BLAST was that it not only identified high-scoring hits, it also produced a full-length or global alignment between two sequences. With the introduction of BLAST2 to the scene, this advantage has largely disappeared.

Multiple Sequence Alignment

Multiple sequence alignment is a technique for aligning more than two sequences at once. Given that there are often hundreds of highly similar proteins (like hemoglobins or serine proteases) or hundreds of highly similar genetic regulatory elements (like promoters or splice site signals) that are used by different animal and

plant species, it is important to be able to compare large groups of sequences in one large listing or table. In this way, it is often possible to detect certain signature sequences or common sequence motifs that define a common active site, a common signal or a common structural motif. Indeed, multiple sequence alignment is one of the most important techniques for extracting critical functional, structural and phylogenetic information from genes and proteins and it is being used more and more frequently as more and more scientists realize its potential. An example of a multiple alignment for several different calcitonins (small calcium sensitive peptides) is shown below:



Multiple Alignment Algorithm

1) Take the collection of all sequences to be multiply aligned and feed them into a pairwise (dynamic programming) alignment algorithm so that each sequence is compared with every other sequence in the list. This requires n(n-1)/2 comparison if "n" is the number of sequences to be aligned.*

2) Identify the pair of homologs that exhibits the highest sequence similarity (on the basis of % identity or some other score). Perform an alignment on this pair and generate a consensus sequence (this consensus sequence is an artificial sequence that shares the conserved residues and all of the gaps in the initial alignment).

3) From this initial "consensus" sequence, select the next most similar sequence (determined from step 1) and align it to the initial consensus sequence. From this

alignment (which now contains 3 sequences), generate a second consensus sequence.

4) Repeat step (3) until you have compared the last (or lowest scoring sequence) to the second last consensus sequence. At this point you should have a complete multiple alignment of all sequences you wished to align.

Note that this technique is heuristic. It is not guaranteed to find the most optimal multiple alignment. It turns out that if you wanted to do this correctly (in the mathematical sense) your algorithm would scale as N^x where "x" is the number of sequences to be multiply aligned. You could see that after "x" exceeds 3, the whole thing would get completely out of hand.

CLUSTAL W (http://www2.ebi.ac.uk/clustalw**)**

CLUSTAL W (Higgins et al., 1996) has essentially become the "industry standard" in multiple sequence alignment. This is in part because it is freely available, it works on almost all major platforms and it uses a very powerful "phylogenetic" algorithm. As with all multiple alignment algorithms, CLUSTAL W uses the progressive alignment algorithm described above (i.e. calculating a series of pairwise alignments). From this, a distance matrix (pairwise % ID) is calculated which provides information on the relatedness of each sequence. This distance matrix is then used to calculate a phylogenetic tree using a neighborhood joining method. This tree is then used as a basis for constructing the alignment, starting with the sequence pair that is most similar. CLUSTAL W can accomodate information about secondary structure by limiting the appearance of gaps among secondary structural elements.

MULTALIN (http://www.toulouse.inra.fr/multalin.html**)**

This multiple alignment program (Corpet, 1988) uses a slightly different approach than CLUSTAL W in that it iteratively refines its multiple alignment by evaluating pairwise alignment scores as the multiple alignment is built. This iterative analysis is repeated until the alignment score maximizes. MULTALIN has a Web interface

that accepts sequences in FASTA format. A series of pull down menus allows one to select different parameters (gap penalites, scoring functions, etc.). MULTALIN cannot accomodate known secondary structural information in its alignment process.

Contig Assembly

Recall how we began this discussion with the brief review of DNA sequencing and contig assembly. Now that we have got a better grasp of sequence alignment, we are in a position to return to that discussion and to explain the process of contig assembly in a bit more detail.

Contig assembly is an excellent example of the potential applications of multiple sequence alignment. In beginning the process of contig assembly one must be particularly careful to exclude any spurious or questionable experimental data. Remember the credo: "garbage in equals garbage out". Typically some effort has to be expended in looking and raw DNA chromatograms and editing or trimming (removing the 5' and 3' ends where overlaps occur) the computer's best guesses about what base is what. After this editing is completed, the files containing the ASCII characters can be read. For contig assembly to effective, there must be a fair bit of redundancy in the data. The term in the DNA world is "X-fold coverage". What this means is there must be enough sequence data to cover the entire sequence of interest, X-times over. Typically 5-fold coverage is the minimum, so for a 1,000,000 bp genome, one would need about 5,000,000 base calls. Given that the average edited read is about 600 bp, that suggests one may have to read in and align up to 10,000 sequence files. That's a lot!.

Because sequencing reactions don't distinguish between which strand they read, it is 50% likely that for every sequence corresponding to the forward reading frame, one will get a sequence in the reverse. To get around this mix of forward and reverse reads it is necessary to "reverse complement" every sequence in the initial set of DNA sequences. This always doubles the number of sequences to align. So for our 1,000,000 bp genome, the problem has grown to 20,000 files. But we're still not done! After the reverse complement step is complete, the entire data set has to

be screened for possible pBR322 contaminating sequences (vector sequences). Once these have been identified and removed, it is usually a good idea to remove or mask low complexity regions. These LCR's can seriously affect the overall quality of the assembly. Once this is done, one is able to launch the multiple aligment program – which performs the assembly.

Methods of Assessing Significance - Doolittle's Rules of Thumb

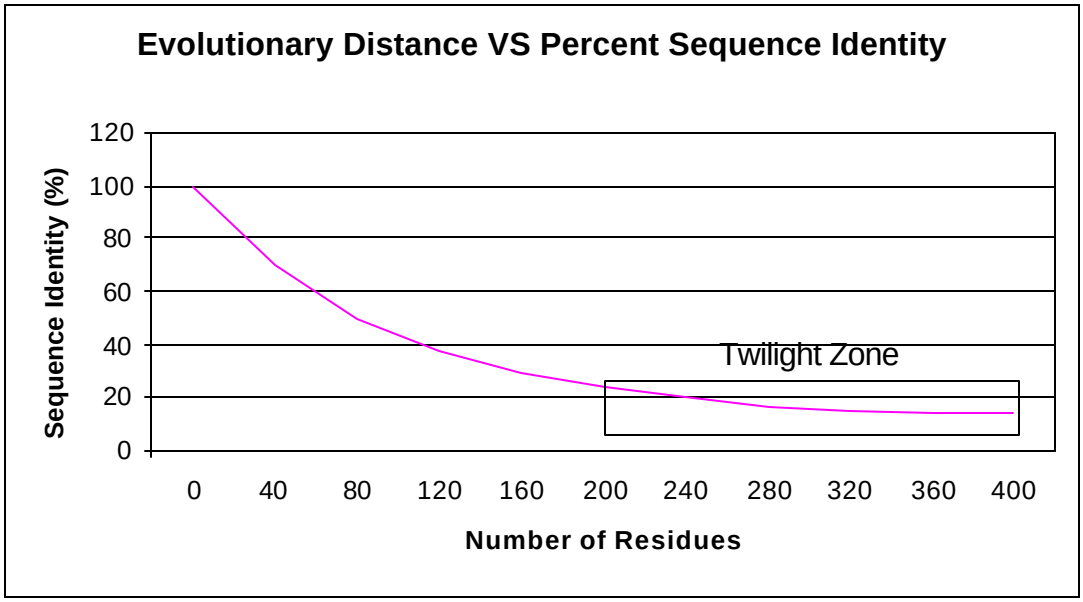
Perhaps the simplest and most lucid treatment of the assessment of protein sequence similarity has been given by R.F. Doolittle (1987). Doolittle was among the first to note that any two random or totally unrelated sequences will typically share a 10-20% identity when aligned by any standard computer alignment program. This is because most alignment algorithms allow gaps to be incorporated with almost unrestricted frequency. If a strict "no-gap" scoring system were used, sequence identities would more typically fall to between 5 and 6%. Given these empirical observations, Doolittle has proposed the following "rules of thumb".

1) If two sequences are longer than 100 residues and are more than 25% identical after suitable gapping, it is very likely that they are related.

2) If two sequences are 15-25% identical they may well be related but one should do further tests to confirm their putative relationship.

3) If two sequences are less than 15% identical, they probably are not related.

These rules may be summed up with the diagram on the following page This is an extremely useful chart and it would serve everyone well if they committed it to memory.



While these rules hold up for the vast majority of cases, it should be noted that there are two factors which can complicate the situation. These mainly have to do with sequence length and sequence composition. As we will show later on, it is remarkably simple to find pairs of short sequences which match identically to each other. Consequently, short peptides (such as certain endocrine hormones) often produce false homologies when subject to a pairwise comparison against large databases. In addition to these problems, it is also worthwhile noting that some proteins are unusually rich in certain amino acids (tracts of poly Glu or poly Asn, for instance). This enrichment of selected residues can often lead to the detection of (false) homologies between proteins which share nothing more than the same tract of repetitive residues. Because of these complications, one must be extra-careful when using Doolittle's rules of thumb around short peptides and proteins with repetitive sequences. As a result of these additional concerns, it may be worth amending Doolittle's rules to include the following suggestions:

4) If an alignment between two sequences requires more than about 1 gap for every 20 residues then the alignment should be viewed as suspicious.

5) If you find that an alignment changes drastically when you change the gap penalties by a small amount then the original alignment is probably not significant.

Semi-empirical Methods -- Database Sampling

Another route to get good statistical data is to use database sampling. This is based on the idea that the current sequence databases are now large enough and diverse enough that they could be considered to represent a truly random sample of protein sequences. If this were indeed the case, then the PIR or SWISS-PROT databases could be viewed essentially as an extremely large collection of jumbled query sequences. Consequently, when we compare any given query sequence against every sequence in the database, it is like comparing the query sequence to thousands of scrambled versions of itself. In effect, database sampling allows the scrambling, aligning and scoring steps of the "jumbling method", described above, to be done more than 400,000 times instead of just 40 times. Furthermore, this process can be done at no extra cost in computer time because the computer is actually performing the alignments at the same time that it is "jumbling".

While this kind of approximation is open to some criticism among purists, in practice it seems to work rather well. This is, in part, due to the fact that the sample space (400,000 points in the case of proteins) is so large that it overwhelms most statistical anomalies. Its success is also due to the very prudent practice of removing extremely high scoring outliers (i.e. homologous proteins) before calculating the mean and the standard deviation of the distribution. Typically the top 2-3% of scores are removed before and means, medians and standard deviations are calculated. This prevents the distribution from appearing bi-modal and allows a better estimate of the true character of the distribution. Most early programs (prior to BLAST) such as FASTA, FASTDB, FASTP used database sampling to calculate their statistics and determine the significance of their sequence alignments. A histogram derived from database sampling is shown below. Note the appearance of a second, high-scoring mode, at the extreme right edge of the graph. This region was excluded in the calculation of the histogram

Suggested Reading

Baxevanis, A.D. and Ouellette, B.F.F. (eds) Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins, Wiley Interscience, New York, 2001. (see Chapters 8-9)

Gribskov, M. and Devereux, J. (eds) Sequence Analysis Primer, W.H. Freeman, and Company, New York, 1992

Creghton, T.E. Proteins: Structures and Molecular Principles W.H. Freeman and Company, New York, 1984.

Websites

- http://www.ncbi.nlm.nih.gov/BLAST
- ftp://ftp.virginia.edu/pub/fasta
- http://www2.ebi.ac.uk
- http://www2.ebi.ac.uk/clustalw
- http://www.toulouse.inra.fr/multalin.html

References

Altschul, S. F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402 (1997)

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. "Basic Local Alignment Search Tool" J. Mol. Biol., 215:403-410 (1990).

Barton, G.J. and Sternberg, M.J.E. "A Strategy for the Rapid Multiple Alignment of Protein Sequences". J. Mol. Biol., 198:327-337 (1987).

Brendel, V., Bucher, P. Nourbachsh, Blaisdell, B.E. and Karlin, S. "Methods and Algorithms for Statistical Analysis of Protein Sequences" Proc. Natl. Acad. Sci. USA 89:2002-2006 (1992)

Dayhoff, M.O., Barker, W.C., Hunt, L.T. "Establishing homologies in protein sequences" Meth. Enzymol. 91:534-545 (1983).

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

Feng, D.F., Johnson, M.S. and Doolittle, R.F. "Aligning amino acid sequences: comparison of commonly used methods". J. Mol. Evol. 21:112-125 (1985).

Gibbs, A.J. and McIntyre, G.A. "The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences. Eur. J. Biochem. 18:1749-1756 (1970).

Henikoff, S. and Henikoff, J.G. "Amino acid substitution matrices from protein blocks" Proc. Natl. Acad. Sci. 89:10915-10919 (1992).

Henikoff, S. and Henikoff, J.G. "Automated assembly of protein blocks for database searching". Nucl. Acids Res. 19:6565-6572 (1991).

Higgins, D.G., Thompson, J.D. and Gibson, T.J. "Using CLUSTAL for multiple sequence alignments" Methods Enzymol. 266:383-402 (1996).

Karlin, S. and Altschul, S.F. "Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Systems". Proc. Natl. Acad. Sci. USA 87:2264-2268 (1990)

Levin, J.M. and Garnier, J. "Improvements in a Secondary Structure Method Based on a Search for Local Sequence Homologies and its use as a Model Building Tool". Biochim. Biophys. Acta, 955:283-295 (1988).

Levin, J.M., Robson, G. Garnier, J. "An algorithm for secondary structure determinatin based on sequence similarity" FEBS Lett. 205:303-308 (1986)

Lipman, D.J. and Pearson, W.R. "Rapid and sensitive protein similarity searches" Science 227:1435-1441 (1985).

McLachlan, A.D. "Tests for comparing related amino acid sequences: cytochrome c and cytochrome c551" J. Mol. Biol. 61:409-424 (1971).

Needleman, S.B. and Wunsch, C. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". J. Mol. Biol. 48:443-453 (1970).

Pearson, W.R. and Lipman, D.J. "Improved tools for biological sequence comparison". Proc. Natl. Acad. Sci. 85:2444-2448 (1988)

Smith, T.F. and Waterman, M.S. "Identification of common molecular subsequences" J. Mol. Biol. 47:195-197 (1981).

Wishart, D.S., Boyko, R.F., Willard, L., Richards, F.M. and Sykes, B.D. "SEQSEE: a comprehensive program suite for protein sequence analysis" CABIOS 10:121-132 (1994)

Wooton, J.C. and Federhas, S. "Analysis of compositionally biased regions in sequence databases". Methods. Enzymol. 266:554-571 (1996).

Wooton, J.C. and Ferehen, S. "Statistics of local complexity in amino acid sequences and sequence databases". Comput. Chem. 17:149-163 (1993).