Last modified August 13, 2018

Elementary Sequence Analysis

Brian Golding, Dick Morton and Wilfried Haerty

Department of Biology
McMaster University
Hamilton, Ontario
L8S 4K1

These notes are in Adobe Acrobat format (they are available upon request in other formats) and they can be obtained from the website http://helix.biology.mcmaster.ca/courses.html. Some of the programs that you will be using in this course and which will be run locally can be found at http://evol.mcmaster.ca/p3S03.html.

The "blue text" should designate links within this document while the "red text" designate links outside of this document. Clicking on the latter should activate your web browser and load the appropriate page into your browser. If these do not work please check your Acrobat reader setup. The web links are accurate to the best of our knowledge but the web changes quickly and we cannot guarantee that they are still accurate. The links designated next to the JAVA logo,  , require that JAVA be installed on your computer.

These notes are used in Biology 3S03. The purpose of this course is to introduce students to the basics of bioinformatics and to give them the opportunity to learn to manipulate and analyze DNA/protein sequences. Of necessity only some of the more simple algorithms will be examined.

The course will hopefully cover . . .

- databases of relevance to molecular biology.
- some common network servers/sites that provide access to these databases.
- methods to obtain sequence analysis software and data.
- methods of sequence alignment.
- methods of calculating genetic distance.
- methods of phylogenetic reconstruction.
- methods for detecting patterns and codon usage.
- methods for detecting gene coding regions.

The formal part of the course will consist of two approximately one hour lectures each week. Weekly assignments will be be provided to practice and explore the lecture material. In addition there will be an optional tutorial to help students with these assignments or other problems. These assignments will be 40% of your grade and three, in class quizzes will make up the remainder.

We would appreciate any comments, corrections or updates regarding these notes.

Golding@McMaster.CA Morton@McMaster.CA HaertyW@McMaster.CA

This document was entirely constructed with pdfLaTeX. Enormous kudos to all those that make this great software free to everyone.

Table of Contents in Brief

In order to speed download, I place here links to the individual chapters in pdf format. The contents of these are shown on the following 'Contents' pages but note that the links will function only for the individual chapter included here.

Contents

Chapter 6

Sequence Alignment

6.1 Dot Plots

The comparison of sequences can be done in many different ways. The most direct method is to make this comparison via a visual means and this is what "dot plots" attempt to do. Dot plots are a group of methods that visually compare two sequences and look for regions of close similarity between them.

6.1.1 The Exact Way

The sequences to be compared are arranged along the margins of a matrix. At every point in the matrix where the two sequences are identical a dot is placed (i.e. at the intersection of every row and column that have the same letter in both sequences). A diagonal stretch of dots will indicate regions where the two sequences are similar. Done in this fashion a dot plot as shown in Figure 6.1 will be obtained. This is a dot plot of the globin intergenic region in chimpanzees plotted against itself (bases 1 to 400 vs. 1 to 300) The solid line on the main diagonal is a reflection that every base of the sequence is trivially identical to itself. As can be seen this dot plot is not very useful unless applied to protein sequences (where the background is much less dense), however some statistical methods can still be applied to the results (Gibbs and McIntyre 1970, Eur. J. Biochem. 16:1).

Maizel and Lenk (1981, PNAS 78:7665) popularized the dot plot and suggested the use of a filter to reduce the noise demonstrated in Figure 6.1. This noise is caused by matches that have occurred by chance. Because only four different nucleotides are possible, nucleotides will match other nucleotides elsewhere in the sequence without any homology present and hence are not a true reflection of the similarities between the sequences but rather reflect the limited number of bases permitted in DNA sequences. There are a wide variety of filters that can be used, indeed they are only limited by your imagination. The one suggested by Maizel and Lenk was to place a dot only when a specified proportion of a small group of successive bases match. In Figure 6.2 the same dot plot is reproduced with a filter such that a window of 10 bases is highlighted only if 6 of these 10 bases match. In Figure 6.3 the same plot is again shown with a filter of 8 out of 10 matches. Note that these plots highlight the complete window while other programs might highlight a single point centered by the window. Another common way to filter the matches is to give them a weight according to their chemical similarity (Staden 1982, Nuc. Acids Res. 10:2951).

The computational work involved with the generation of these matrices can be quite time consuming. If you are comparing a sequence of length N with another sequence of length M, then the total number of windows for which matches must be calculated is $N \times M$. Hence the amount of work increases with the square of the sequence length. This rapidly becomes a large number. For example with $N = 700$ and $M = 400$, $N \times M = 280,000$.

Figure 6.1: Dot Plot - without filtering.

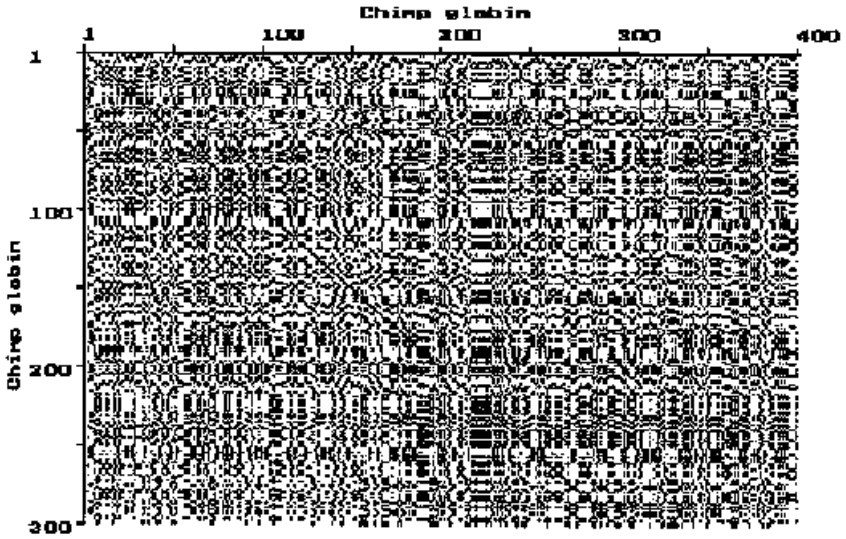


Figure 6.2: Dot Plot - filtered 6 of 10.

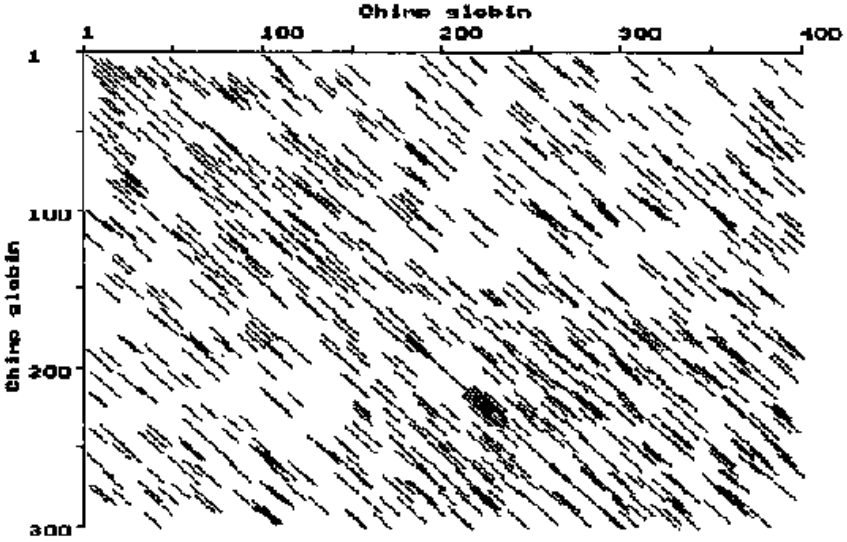
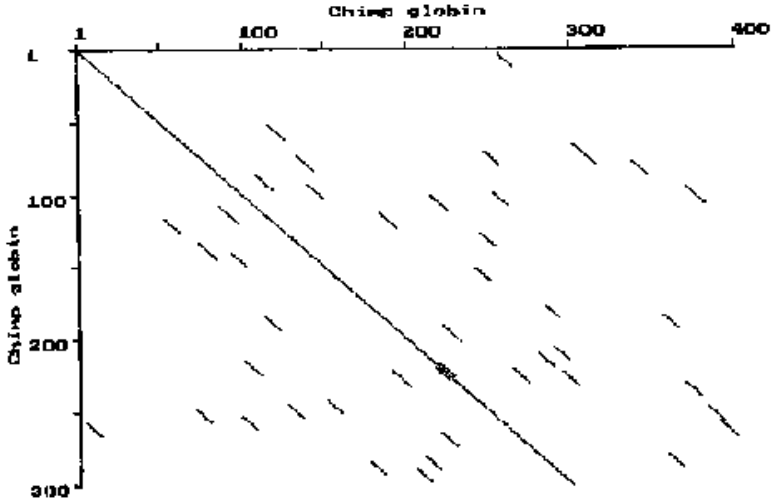


Figure 6.3: Dot Plot - filtered 8 of 10.



6.1.2 Identity Blocks

There is another way in which dot plots can be generated very quickly. This involves a computer method commonly known as "hashing" (list-sorting). As mentioned previously, these methods are incorporated into the FASTA algorithms. Basically, the idea is that instead of taking the complete matrix and calculating points for every entry in that matrix, a great saving can be made if the algorithm searches only for exact matches. Hence, this method looks only for blocks of perfect identity. The computational complexity of this algorithm grows linearly with increasing N.

The algorithm simply sub-divides the sequence into all "words" of a user specified block size. The same is done for the alternate sequence. In addition, for both sequences the location of each word is also recorded. These arrays of "words" are then sorted alphabetically and the arrays of locations are sorted in parallel with the "words". Then, by comparing the sorted array from one sequence with that from the other sequence immediately gives the location of all identical "words".

An algorithm which does be used to generate the dot plots shown in Figure 6.4 for identity blocks of length 5. The rapidity of this method compared to the exact method can be demonstrated by the dot plot shown in Figure 6.5 (with identity blocks of length 6). This figure extends the sequences compared in the chimpanzee globin intergenic region from (1-400 vs 1-300) up to (1-4000 vs 1-3000). The length of time required for a plot of the small region is not significantly shorter than the length of time it takes to calculate short identities on a 100 fold larger matrix.

The beauty of this method is demonstrated in Figure 6.6. This is a plot of all identities of length 6 between the chimpanzee and spider monkey sequences in the same region. The evolutionary homology between these sequences is easily discernible by the solid lines along the main diagonal despite the approx. 60 million years that separate these two groups. Further more, this is intergenic DNA with no known function to selectively maintain this homology (modulo an even more ancient eta-globin pseudogene). The insertion of some DNA is easily observed within chimpanzee sequence and then a corresponding deletion further down. These correspond to the insertion of an Alu element in the chimpanzee (and human and other ape) sequences (at approx. bp 1000) and then the presence of a truncated L1 element in the spider monkey (inserted at approx. bp 2600) that is not present in the great apes. These events are difficult to find by a simple inspection of the actual sequence code but are readily found by a visual inspection.

A more distant similarity can be seen in Figure 6.7. This is a plot of the identities of length 6 between the same region of the chimpanzee haemoglobin intergenic region and another intergenic region from the spider monkey. Note the similarity (the short diagonal line) in the circled region. This region of similarity corresponds to the location of another Alu element in the chimpanzee sequence.

There are many programs freely available to make dot plots. One which is particularly fast and interactive is the `dotter`

Figure 6.4: Identity Dot Plot.

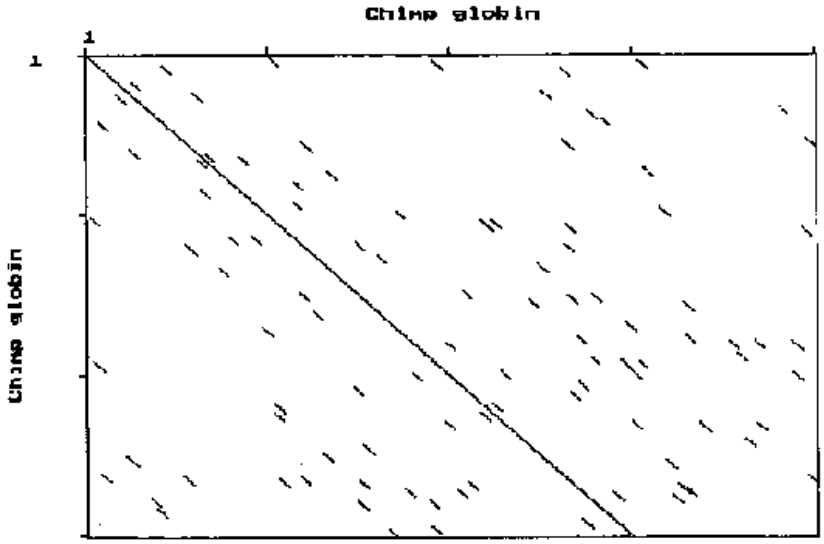


Figure 6.5: Identities of length 6bp. Chimpanzee hemoglobin intergenic DNA against itself.

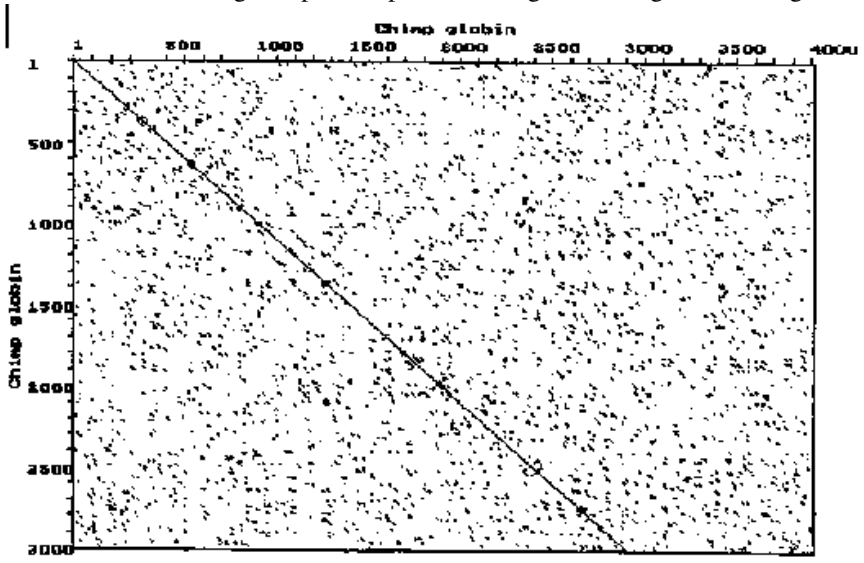


Figure 6.6: Identities of length 6bp. Chimpanzee hemoglobin intergenic DNA against spider monkey.

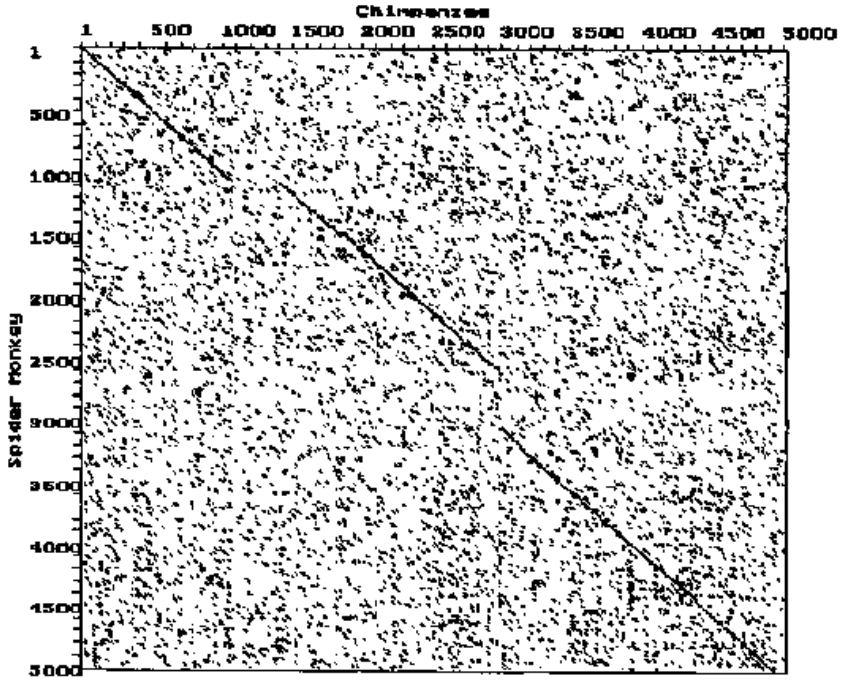


Figure 6.7: Identity dot plot. Chimpanzee hemoglobin intergenic region vs. Spider Monkey unrelated intergenic region.

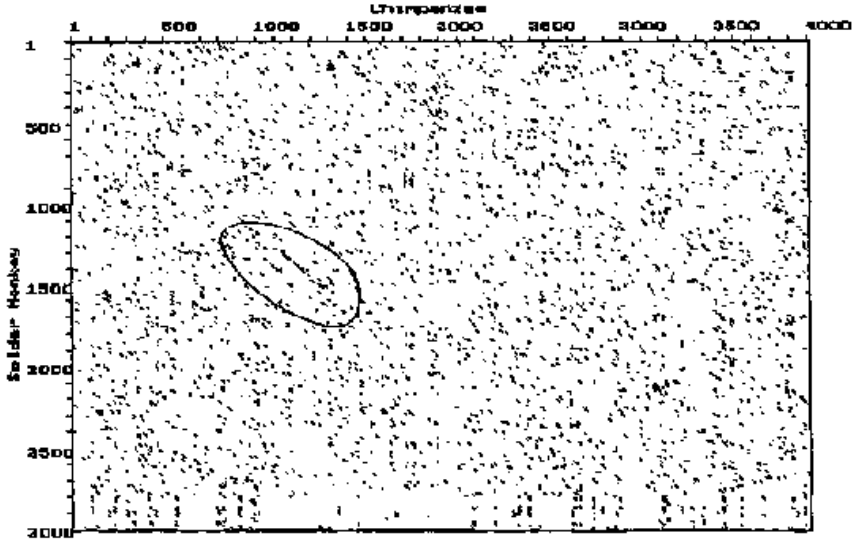


Figure 6.8: Human calmodulin protein sequence dot plotted against itself. Since the dotplot is symmetrical only the lower half is shown. Also note the margin around the edge where a complete window could not be calculated.

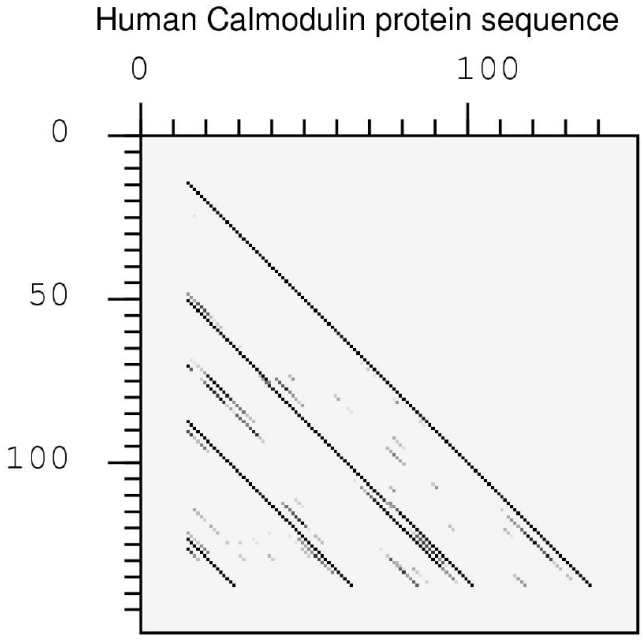


Figure 6.9: Human epidermal growth factor protein sequence dot plotted against itself. Since the dotplot is symmetrical only the lower half is shown.

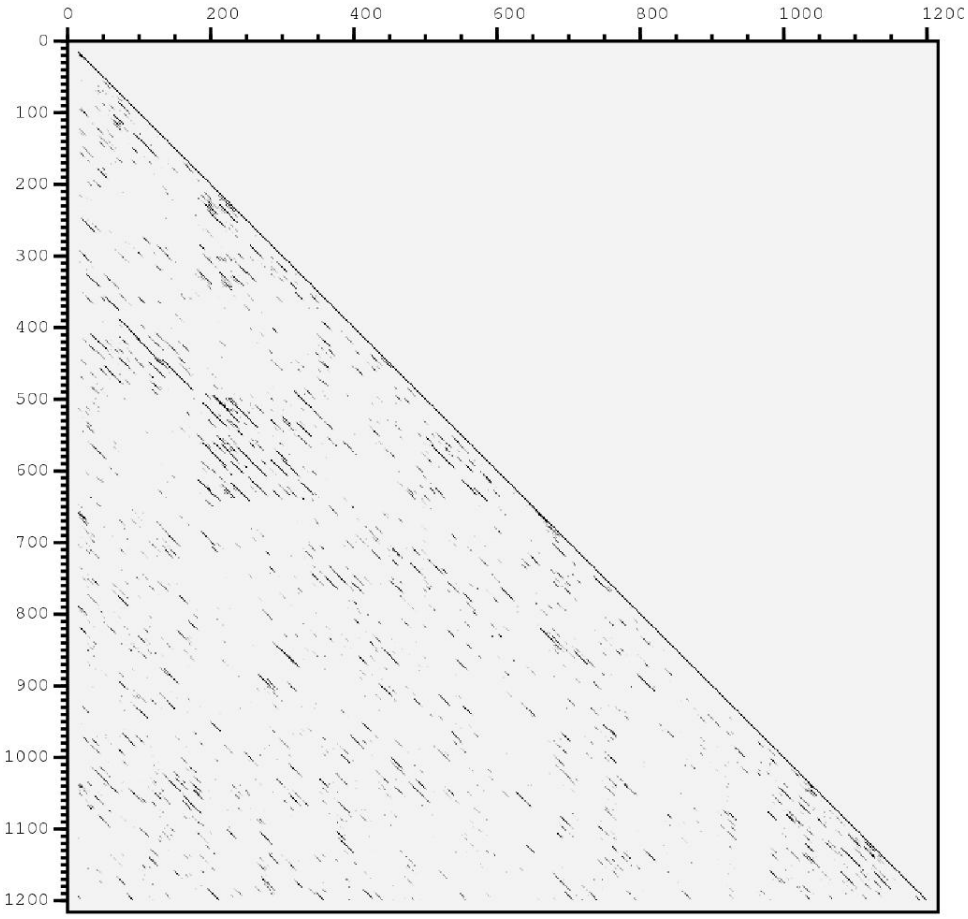


Figure 6.10: Human globin region (zeta, psizeta, psialpha1, alpha2, alpha1) dot plotted against itself. Since the dotplot is symmetrical only the lower half is shown.

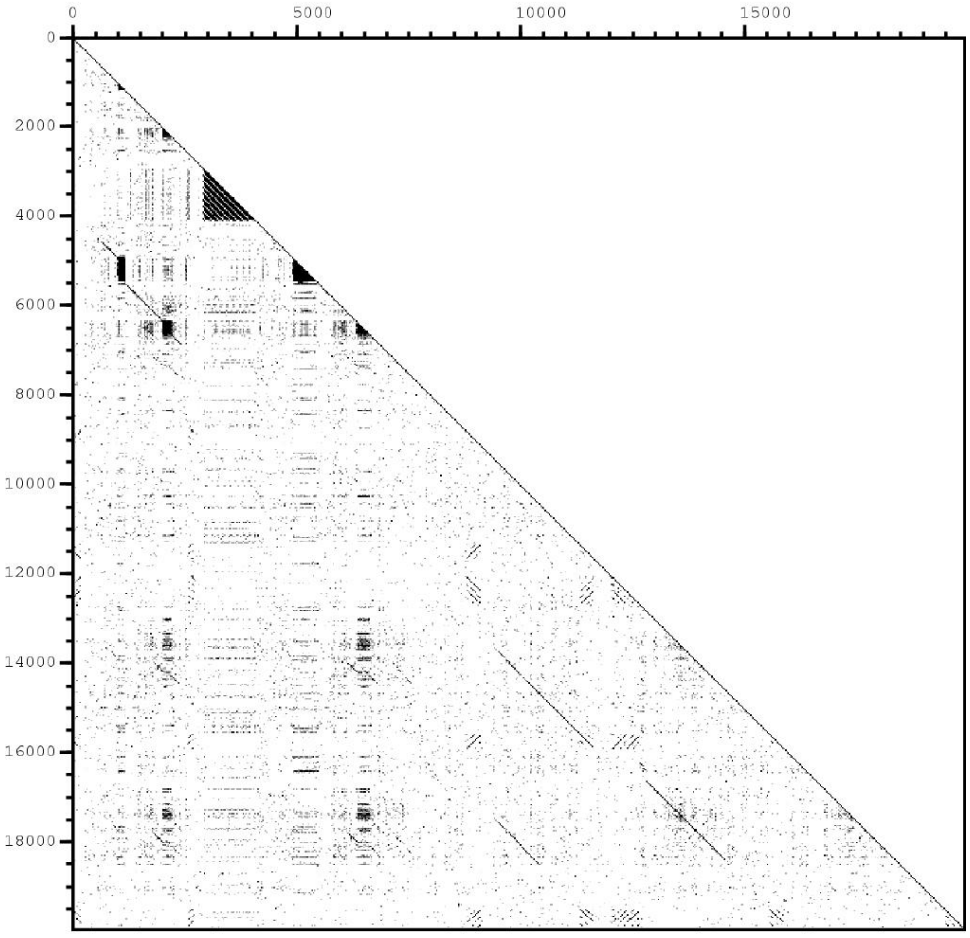
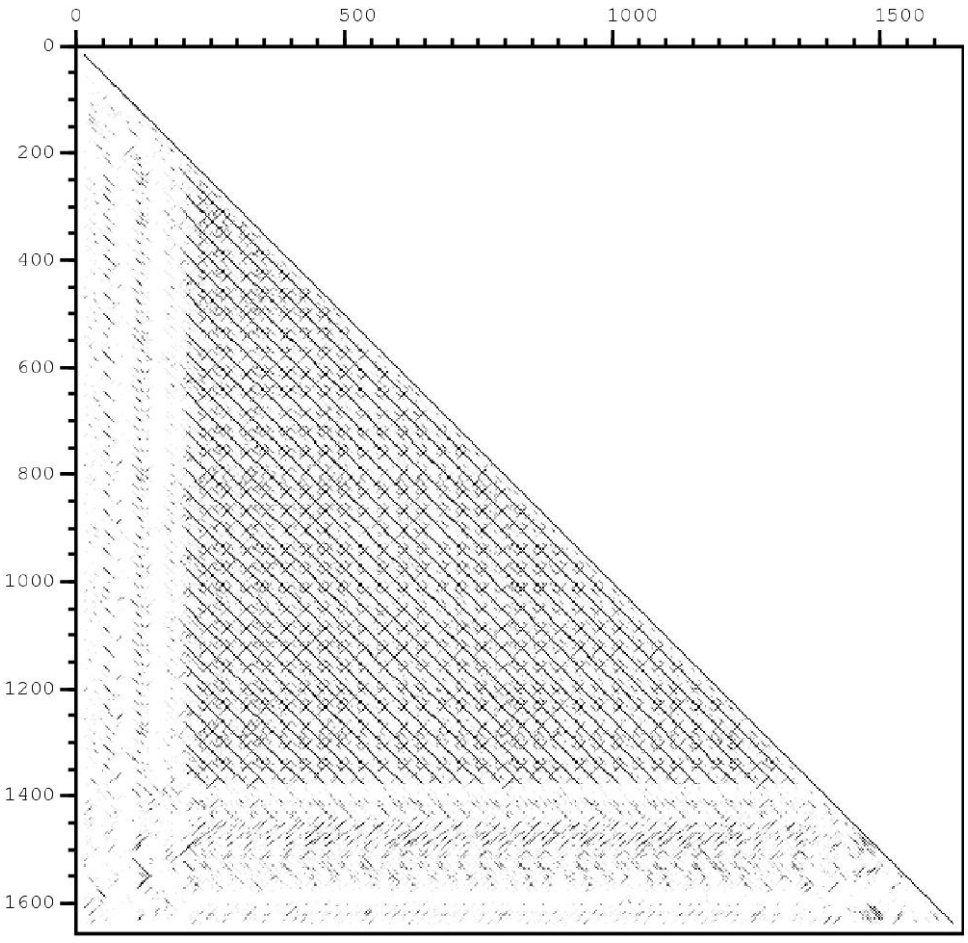


Figure 6.11: Human zeta globin intergenic region expanded from Figure 6.10. Since the dotplot is symmetrical only the lower half is shown.



program. Some other interesting dot plots are comparisons of the calmodulin (Figure 6.8) protein against itself and the human epidermal growth factor (Figure 6.9) against itself. Both show internal repetitive elements. The neatest dot plot that I have yet seen is the human zeta globin (Figure 6.10) region and if you zero in on the intergenic region (Figure 6.11) the plot becomes fantastic (try to interpret this dot plot).

6.2 Alignments

The dot plots provide a useful way to visualize the sequences being compared. They are not very useful however in providing an actual alignment between the two sequences. To do this, other algorithms are required.

First, a word on terminology. People in the field shudder when the terms similarity and homology are used indiscriminately. Similarity simply means that sequences are in some sense similar and has no evolutionary connotations. Homology refers to evolutionary related sequences stemming from a common ancestor.

The ability to calculate the correct alignment is crucial to many types of studies. It may, for example, alter from which part of a gene one segment was duplicated, it may alter the inferred number of point mutations, it may alter the inferred location of deletions / insertions, alter the inferred distance between species, and may alter the inferred phylogeny of the sequences along with whatever evolutionary hypotheses are dependent on these phylogenies.

An explicit and precise algorithm is also required. For example one paper in the prestigious journal NATURE stated that the alignment

```
------CCTTCAGAATACAGAATAGGGACATAGAGA
ATCCCACCCAGCCCCCTGGACCTGTAT---------
```

was optimal in the sense that gaps were inserted to maximize the number of base matches (the base matches are highlighted). They obviously did this alignment by eye and did not use an explicit algorithm. An alternate alignment (due to Fitch 1984, Nature 309:410) is

```
CCTTCAGAATACAGAATAGGGACATAGAGA
ATCCCA---CCCAGCCCCCTGGACCTGTAT
```

This alignment not only increases the number of base matches by 133 per cent, but also decreases the number of gaps by 50 per cent and reduces the number of gapped residues by 80 per cent. Hence, if the number of base matches can be increased by reducing the number of gaps, then clearly the original author's insertion of gaps did not maximize that number. Fitch recommends that the authors change their statement to the assertion that gaps were introduced to increase the number of base matches (rather than to maximize them). More generally this example shows the importance of i) using a well defined algorithm and ii) of using a computer based algorithm to perform these calculations. Even alignments that may appear simple and straightforward, if given to the computer, might yield alternatives that you did not consider.

6.2.1 The Needleman and Wunsch Algorithm

The most basic algorithm to align two sequences was developed by S.B. Needleman and C.D. Wunsch (1970, J. Mol. Biol. 48:443). The algorithm is a simple and beautiful way to find an alignment that maximizes a particular score[1]. (The score can be calculated in a variety of methods - as will be indicated below). The initial steps of the algorithm are reminiscent of the dot plot. The first step is to place the two sequences along the margins of a matrix as shown in Table 6.1.

In this first step, simply place a 1 anywhere the two sequences match and a 0 elsewhere. If done on a larger scale than is shown in Table 6.1, this would exactly recreate the dot plot shown in Figure 6.1. In this case however, we wish to find a path through this matrix which would define a more conventional alignment. For example, proceeding along the diagonal with no deviations would imply an alignment without any gaps. The introduction of a gap (either by an insertion or a deletion - an indel) in either sequence would correspond to moving either above or below the main diagonal.

To find the best route, Needleman and Wunsch suggested that you modify the matrix to represent this idea of tracing different pathways through the matrix. However, you want to include all possible pathways and from among these choose only that one which is best (in the sense of maximizing some score). Their method consists of two passes through the matrix. The first pass traces a score for all possible routes and moves right to left, bottom to top. Once the score for all possible routes are found, the maximum can be chosen (it will be somewhere on the topmost row or leftmost column) and a

[1] An alignment step by step example 

Table 6.1: Initial setup for Needleman-Wunsch

The first step is to place the two sequences, in this case two protein sequences, along the margins of a matrix. Then place a one in the matrix where ever the two sequences agree.

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	1	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
N	0	0	0	1	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	1	0	0	0	0	1	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
R	0	0	0	0	0	1	0	0	0	0	1	0	0
B	0	1	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

second pass can be carried out, this time running left to right, top to bottom to find that alignment that gives the maximum score.

The way to trace a score for all possible paths is shown in Table 6.2. For each element in the matrix you perform the following operation.

$$M_{i,j} = M_{i,j} + \max(M_{k,j+1}, M_{i+1,l})$$

where k is any integer larger than i and l is any integer larger than j. In words, alter the matrix by adding to each element the largest element from the row just below and to the right of that element and from the column just to the right and below the element of interest. This row and column for one element are shown in Table 6.2 by boxes. The number contained in each cell of the matrix, after this operation is completed, is the largest number of identical pairs that can be found if that element is the origin for a pathway which proceeds to the upper left.

We wish to have an alignment which covers the entire sequence. Hence, we can find on the upper row or on the left column the element of the matrix with maximum value. An alignment must begin at this point and can then proceed to the lower right. This is the second pass through the matrix. At each step of this pass, starting from the maximum, one moves one row *and* column to the lower right and finds the maximum in this row or column. The alignment must proceed through this point.

Continuing in this fashion one eventually hits either the bottom row or the rightmost column and the alignment is finished. This tracing pattern is shown in Table 6.3. Note that in this case the optimal alignment is not unique. There are two alignments and both give the optimal score of 8 matches.

These two alignments can be written in more familiar form as either

```
ABCNJ-RQCLCR-PM
* * * * * ** *
AJC-JNR-CKCRBP-
```

or as

Table 6.2: Half way through the second step

Add to each element of the matrix the largest number from the row or column to the lower right of each element proceeding right to left, bottom to top.

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	1	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	1	0	1	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0
N	0	0	0	1	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	1	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 6.3: Trace the alignment

The alignment is traced proceeding left to right, top to bottom choosing the largest numbers available. Alternate pathways both yielding the optimal alignment are shown .

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
J	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
J	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

```
ABC-NJRQCLCR-PM
* * * * * ** *
AJCJN-R-CKCRBP-
```

both with 8 asterisks to denote the 8 matches. Note that in this particular case, gaps are given the same penalty as a mismatch. They simply do not add to the score.

6.2.2 The Smith-Waterman Algorithm

The Needleman-Wunsch algorithm creates a global alignment. That is, it tries to take *all* of one sequence and align it with *all* of a second sequence. Short and highly similar subsequences may be missed in the alignment because they are outweighed by the rest of the sequence. Hence, one would like to create a locally optimal alignment. The Smith and Waterman (1981, J. Mol. Biol. 147:195-197) algorithm finds an alignment that determines the longest/best subsequence pair that give the maximum degree of similarity between the two original sequences. This means that not all of the sequences might be aligned together.

The Smith-Waterman algorithm is very similar to the Needleman-Wunsch algorithm and again finds the best subsequence matches and builds upon these. Three conceptual changes are required,

- A negative score/weight must be given to mismatches and/or gaps.
- Zero must be the minimum score recorded in the matrix.
- The beginning and end of an optimal path may be found anywhere in the matrix - not just the last row or column.

The first point is required to cause the score to drop as more and more mismatches are added. Hence, the score will rise in a region of high similarity and then fall outside of this region. If there are two segments of high similarity then these must be close enough to allow a path between them to be linked by a gap or they will be left as independent segments of local similarity. In general the Smith-Waterman algorithm includes gap penalties (to be discussed in section 6.4) and if this is the case, then the mismatch penalties are not required to be negative (to retain simplicity here, I have assumed a negative mismatch penalty). Either way, the essence of a local alignment is that the score must decline.

The second point is required so that each pathway begins fresh at its beginning. Thus each short segment of similarity should begin with a score of zero (and hence the matrix is intialized with the first row and column equal to zero). The third point indicates that the entire matrix must be searched for regions with high local similarity.

For each element in the matrix you perform the following operation.

$$M_{i,j} = \max(M_{i-1,j-1} + s_{i,j}, \quad \max_{k\geq 1} M_{i,j-k} - W_k, \quad \max_{k\geq 1} M_{i-k,j} - W_k, \quad 0)$$

By tradition, the matrix is filled out left to right, top to bottom (the opposite direction of Needleman and Wunsch). Here, $s_{i,j}$ is the score for characters i and j (they either match or mismatch). The terms W_k are the gap penalties for a gap of length k ending at either i or j. Similarly gap penalties can be added to the Needleman-Wunsch method.

As an example the previous alignment can be reproduced with score of 1.0 for a match, -0.5 for a mismatch and $W_k = -0.5k$ for a gap of length k. The matrix will then be as given in Table 6.4. In this case largely the same alignment is found. However, the Smith-Waterman algorithm ends at the maximum score and begins at the first no-zero score. In this case it includes the same ambiguity in the alignment but it ends with the alignment of the two P's. The M is not part of this alignment. More generally, large chucks of each sequence may be missing from a local alignment (as in the alignment presented by BLAST) as opposed to a global alignment .

Table 6.4: Smith-Waterman example

Here a score of 1.0 is given for a match, of -0.5 for a mismatch and a linear score of $-0.5k$ for a gap of length k.

		A	B	C	N	J	R	Q	C	L	C	R	P	M
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	1	0.5	0	0	0	0	0	0	0	0	0	0	0
J	0	0.5	0.5	0	0	1	0.5	0	0	0	0	0	0	0
C	0	0	0	1.5	1	0.5	0.5	0	1	0.5	1	0	0	0
J	0	0	0	1	1	2	1.5	1	0.5	0.5	0.5	0.5	0	0
N	0	0	0	0.5	2	1.5	1.5	1	0.5	0	0	0	0	0
R	0	0	0	0	1.5	1.5	2.5	2	1.5	1	0.5	1	0.5	0
C	0	0	0	1	1	1	2	2	3	2.5	2	1.5	1	0.5
K	0	0	0	0.5	0.5	0.5	1.5	1.5	2.5	2.5	2	1.5	1	0.5
C	0	0	0	1	0.5	0	1	1	2.5	2	3.5	3	2.5	2
R	0	0	0	0.5	0.5	0	1	0.5	2	2	3	4.5	4	3.5
B	0	0	1	0.5	0	0	0.5	0.5	1.5	1.5	2.5	4	4	3.5
P	0	0	0.5	0.5	0	0	0	0	1	1	1.5	3.5	4.5	4

It is seldom the case that the Smith-Waterman and the Needleman-Wunch algorithms give the same answer. For example, a global and a local alignment of TTGACACCCTCCCAATTGTA versus ACCCCAGGCTTTACACAT give

```
TTGACACCCTCC-CAATTGTA
    ::  ::   ::  :
ACCCCAGGCTTTACACAT---


---------TTGACACCCTCCCAATTGTA              TTGACAC
         :: ::::                   or      :: ::::
ACCCCAGGCTTTACACAT-----------              TTTACAC
```

respectively. The global alignment has considered a penalty for the end gaps but the local alignment has simply searched for the best substrings that can be put together.

If the sequences are not known to be homologous throughout their entire length, a local alignment should be the method of choice. Sometimes the two methods will give similar answers but if the homology is distant, a local alignment will be more likely to find the remaining patches of homology.

6.3 Testing Significance

The above algorithms are trying to find the best way to match up two sequences. This does not mean that they will find anything profound. For example, if I take these two sentences (and delete spaces and delete non-amino acids), they can be aligned.

```
THESEALGRITHMARETR--YINGTFINDTHEBESTWAYTMATCHPTWSEQENCES
:: :.. . ..  ...:   :    :::::.       ::  .  : ...
THISDESNTMEANTHATTHEYWILLFINDAN-------YTHIN-GPRFND------
```

Depending on your intuition you may or may not think this to be a pretty good alignment particularly at the amino-terminus. There are a total of 12 exact matches and 14 conservative substitutions. But there is obviously no homology between these two "sentence" sequences. How do we test whether or not an alignment is significant?

As a more biological example, consider the alignment of human alpha haemoglobin and human myoglobin. If you remember your basic biology, you should remember that these two proteins do similar functions of transporting oxygen in the blood and muscle respectively. But are they evolutionarily related? An alignment of the two looks like ...

```
Human alpha haemoglobin (141 aa) vs. Human myoglobin (153 aa)


VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQ
 :: ..   : ..::::.:.  ..:.:.: :.: . :.: . : .: .:.     ..:..
GLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASED


VKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLP
.: :: .: .::.. . . ..  .....:.. :: : ..    ....:.:.. .:... :
LKKHGATVLTALGGILKKKGHHEAEIKPLAQSHATKHKIPVKYLEFISECIIQVLQSKHP


AEFTPAVHASLDKFLASVSTVLTSKYR------
..:.........: :.  .. ..:.:.
GDFGADAQGAMNKALELFRKDMASNYKELGFQG
```

Again looks like a reasonably good alignment. Or how about chicken lysozyme and bovine ribonuclease. An alignment of these gives

```
Chicken lysozyme (129 aa) vs. Bovine ribonuclease (124 aa)


KVFGRCELAAAMKRHGLDNYRGYSLGNWVCAAKFESNFNTQATNRNTDGSTDYGILQINS
:  .    :: ..:. .:.  . . ..  :.....:.  :..  . ... .. .. ....
KETA----AAKFERQHMDSSTSAASSSNYCNQMMKSRNLTKDRCKPVNTFVHESLADVQA


RWWCNDGRTP--GSRNLCNIPCSALLSSDITASVNCAKKIVSDGDGMNAWVAWRNRCKGT
   :.. ...  .... :  ..:..  .:  .. ...:   .. .. .:       :.:.
V--CSQKNVACKNGQTNCYQSYSTMSITDCRET-GSSKYPNCAYKTTQANKHIIVACEGN


DVQAWIRGCRL
   .   . ..
PYVPVHFDASV
```

Again a reasonable alignment or so it seems. How do you know which sequences (if any) are homologous?

A common and simple test to determine if the alignment of two sequences is statistically significant is to carry out a simple permutation test. This consists of

1. Randomly rearrange the order of one or both sequences
2. Align the permuted sequences
3. Record the score for this alignment
4. Repeat steps 1-3 a large number of times.

Doing this say 10000 times, gives a distribution of alignment scores that could be expected for random sequences with a similar amino acid content. If the actual alignment has a score much higher than that of the permuted sequences, then you know that they must be homologous to some extent.

A plot of 10,000 alignment scores for the human myoglobin and human alpha haemoglobin sequences are shown in figure 6.12. The permuted scores range from 14 to 75 but most are less than 50. Also note the skewness of the distribution - statistics based on a normal distribution would be strongly biased. The skew is expected since in each case the alignment algorithm is trying to maximize the score. The score for the alignment of the two actual sequences is 179 (indicated by the

Figure 6.12: Histogram of alignment scores

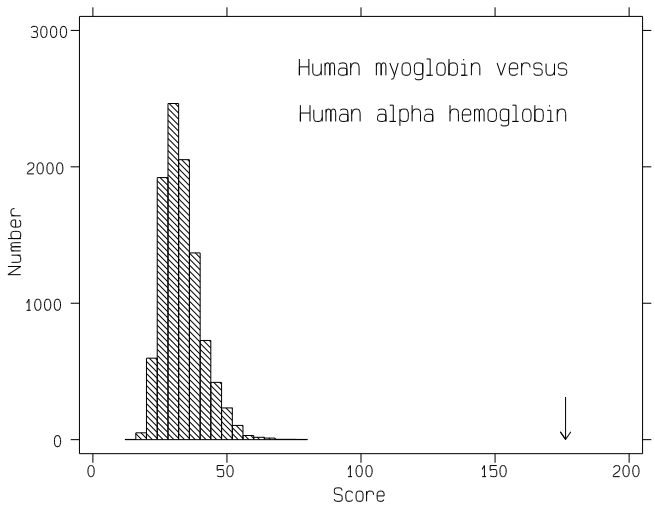
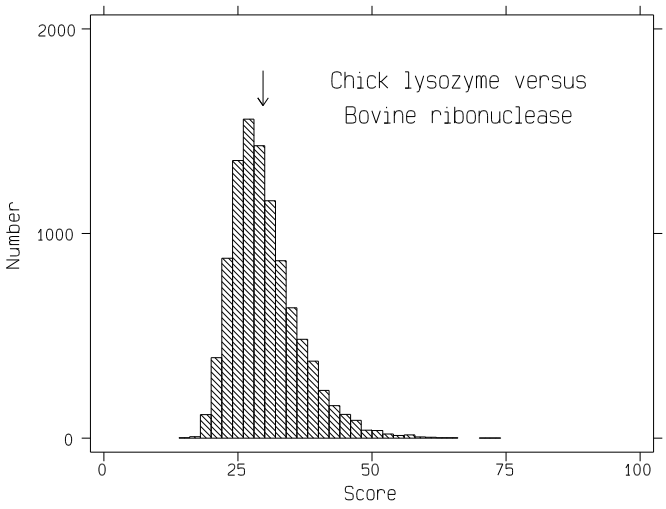


Figure 6.13: Histogram of alignment scores



arrow). Obviously, myoglobin and haemoglobin are evolutionarily related and still retain many features of their homology. This alignment has a probability of less than 0.0001 of occurring by chance alone.

A plot of 10,000 alignment scores for the chick lysozyme and bovine ribonuclease sequences are shown in figure 6.13. Again note, the skew and note that this "random" distribution is somewhat different from the haemoglobin "random" distribution. This is due to the differential effects of amino acid composition in these proteins. The permuted scores range from 14 to 72. The actual score for the proteins is 30 (indicated by the arrow). Obviously whatever homology that once existed between these proteins has been completely destroyed by time.

These two examples are clear cut. There is a large grey area where the tests may be uncertain of the degree of homology between sequences. For protein sequences Doolittle's rule of thumb is that greater than 25% identity will suggest homology, less than 15% is doubtful and for those cases between 15-25% identity a strong statistical argument is required. Personally, I would prefer the statistical test in all cases, since they are easy to do and things such as internal repeats and unusual amino acid compositions can sometimes confuse the picture.

6.4 Gaps and Indels

Gaps were rather freely permitted in the overly simple implementations of the Needleman-Wunsch and Smith-Waterman algorithms shown above. What would happen if you feel that gaps should be rarer events in your particular protein? It is possible to assign a different weight to gaps. This can be done by subtracting from the score, some predetermined value every time a gap is required. In this case you can define a weight as

$$W_k = a + bk$$

where k is the length of the gap. Hence you can control whether many short gaps occur or whether long gaps occur but more infrequently. Deletions do occur but when they occur it is seldom many small, short deletions but rather fewer and longer deletions. These types of penalties are termed affine gap penalties.

How do you choose a gap penalty? Unfortunately, there is little knowledge to help here. Most of the tests done so far depend on an empirical basis designed to achieve some end. For example, Smith and Fitch have derived (by exhaustive search) gap penalties that will best align distantly related haemoglobin genes. But there is no guarantee that these values would work well for the protein or (worse) the nucleotide sequence that you are interested in. Typical values are

$$0.5 < a < 5.0$$

$$0.05 < b < 1.0$$

but there is nothing special about these values other than the fact that they seem to work well for some of the common comparisons. Note that in general $a > b$. This corresponds with biological knowledge of how gaps are generated - it is easier to generate one gap of two residues rather than two gaps of one residue since the former can be created by a single mutational event.

More recently, Reese and Pearson (2002, Bioinformatics 18:1500-1507) examined how these parameters might change as a function of the distance between aligned sequences. Their criteria was the "correct" identification of distant homologues. They found that b did not change but that a did change with distance. Again, through empirical tests they showed that optimal penalties were $a = 25 - 0.1 \times (PAMdistance)$ (PAM is a method of measuring distance that will be explained in section 7.2.1) and $b = 5$ where these penalties are in $1/3$ bit units (see section 10.5.1).

6.4.1 "Natural" Gap Weights - Thorne, Kishino & Felsenstein

In a series of papers Thorne, Kishino and Felsenstein (JME 33:114, 1991) and (JME 34:3, 1992) have developed a method to find maximum likelihood estimates of the gap penalties (and transition/transversion rates) while doing an alignment. This eliminates the necessity for choosing an arbitrary gap penalty.

Table 6.5: Some rules of thumb for alignments

1. A gap and its length are distinct quantities. Different weights should be applied to each.
2. Weights for different mismatches should be permitted. A transition is more likely than a transversion; a Ile-Val more likely than Ile-Arg change.
3. If the two sequences have no obvious relationship at their right and left ends, then end gaps should not be penalized.
4. Unless two sequences are known to be homologous over their entire length, a local alignment is preferable to a global alignment.
5. An optimal alignment is by no means necessarily statistically significant. One must make some estimate of the probability that a given alignment is due to chance.
6. An alignment demonstrates similarity, not necessarily, homology. Homology is an evolutionary inference based on examination of the similarity and its biological meaning. Sequence similarity may result from homology but it may also result from chance, convergence or analogy.

They develop a maximum likelihood method to examine the possible paths of descent from a common ancestor for two sequences. The creation of gaps is modeled as a birth - death process with separate parameters for birth rate and death rate. The model then finds the likelihood of particular paths through the matrix given the transition parameters. It then examines alternative parameters and chooses that path and parameter set with the highest likelihood. The big difficulty with this method is the enormous computer time required to carry out the calculations.

A related question is the assignment of weights to individual differences in nucleotide or protein sequences (more on this later). There have also been advances in methods to try to find statistically bounded sets of alignments. That is, the set of alignments that are within 95% confidence limits of some best answer. Again another fertile area were many significant improvements are being made.

It is important to realize that an optimal alignment is optimal only for the particular values chosen for the mismatch and gap weights. When any of these are altered, the optimal alignment will also change. Also be aware of the fact that nature is seldom mathematically optimized. Fitch and Smith (1983, PNAS 80:1382) have derived a set of "rules of thumb" a subset of which are given in Table 6.5. Even with the very best programs it still requires some degree of experience to draw the right conclusions from the results produced and a good grasp of the biology of the problem is essential.

6.5 Multiple Sequence Alignments

Conceptually, there is no reason why a Needleman-Wunsch algorithm can not be performed with more than two sequences. The matrix simply becomes multi-dimensional and the algorithm would work successively through each dimension. There are however, significant practical problems with this approach. In this case instead of growing as an N^2 problem, the computational time will grow as N^m, where m is the number of sequences. Hence, even for just 100 nucleotides from 5 species, this is

$$100^5 = 10,000,000,000$$

operations or the equivalent of doing an alignment for two sequences each 100,000 nucleotides long. Obviously different methods need to be employed. In general these require more assumptions and are not as precise nor "all-encompassing" as the Needleman-Wunsch or Smith-Waterman algorithms.

If desired, simple scores of similarity can be readily found using rapid techniques on all pairs of sequences. But to take

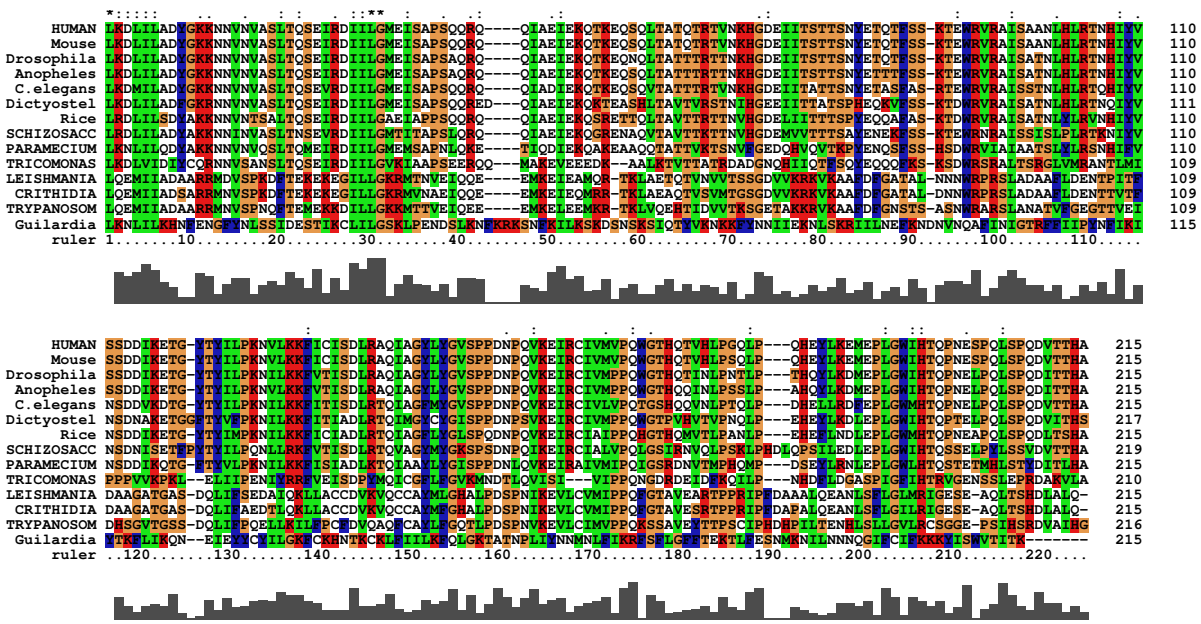


Figure 6.14: An example of the output from ClustalX, a popular multiple sequence alignment program. The shaded diagram at the base provides a measure of the similarity within each column.

these local regions of similarity and turn out an alignment is somewhat more complicated.

Bains (1986, Nuc. Acids Res. 14:159) suggested an iterative method which involves successive applications of the standard algorithms. It begins with a trial consensus alignment (say the alignment between sequences 1 and 2. Then the third sequence is aligned against the consensus sequence and a new consensus emerges. This continues until the consensus alignment converges to a global consensus. This type of method will be very dependent on the order that the sequences are introduced. Thus a different alignment could arise using the same technique and the same sequences but in a different order.

One of the most popular multiple alignment programs begins with all pairwise alignments and is called Clustal. It was written by Higgins and Sharp (1988, Gene 73:237; 1989, CABIOS 5:151). The alignments are done in four steps. In the first step, all pairwise similarity scores are calculated. This is done using rapid alignment methods. The second step is to create a similarity matrix and then to cluster the sequences based on this similarity using a cluster algorithm (see the section 9.2). The third step is to create an alignment of clusters via a consensus method. The final step is to create a progressive multiple alignment. This is performed by sequentially aligning groups of sequences, according to their branching order in the clustering. Three variants currently are being used, ClustalW, a companion program call ClustalX and the older ClustalV. An example of the output from ClustalX is shown in Figure 6.14.

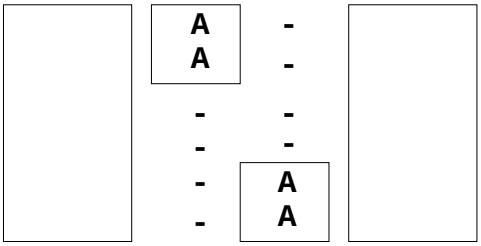
Other methods make use of a multiple dimensional dot plot and then look for dots that are common to each group (Vingron & Argos 1991 J.Mol.Biol. 218:33-43). Still others rely heavily on user input such as the popular windows program MACAW (Schuler, Altschul & Lipman, 1991 Proteins Struct. Func. Genet. 9: 180-190). Others such as MSA (Gupta, Kececioglu & Schaffer, 1995 J. Comput. Biol. 2:459-472) attempt to provide a near-optimal sum-of-pairs global solution to the multiple alignment. Most of these programs attempt to find a solution such that some measure of the multiple alignment is minimized (or maximized). Most however, can only provide a guess to the best solution. Kececioglu has developed a new branch and bound algorithm that is guaranteed to converge to the true optimal solution (just no guarantees on how long that will take). This whole area is ripe for major theoretical advances and for the creation of better interface programs.

One obvious extension of these algorithms is to construct an alignment and a phylogeny for sequences all at the same time. This is because the alignment will affect distances between sequences and this will affect the inferred phylogeny. Similarly a different phylogeny will imply a different alignment of the sequences. Now you are talking about a chicken and egg

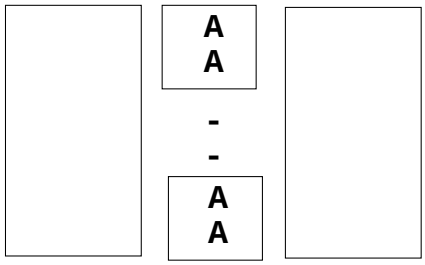
problem! Never-the-less, some progress has been made in this area. Jotun Hein has come up with a program TREEALIGN which will do exactly this. It is available in the list from EMBL software given at ftp.ebi.ac.uk/pub/software/unix but again it is a very slow program.

How well do they actually work? McClure, Vasi and Fitch (1994, Mol. Biol. Evol. 11:571-592) tested how well the different algorithms could detect and correctly align, ordered functional motifs in several proteins. They used haemoglobin (5 motifs), kinase (9 motifs), aspartic acid protease (3 motifs), and ribonuclease H (4 motifs) proteins. They calculated the number of times (out of 100) that different algorithms correctly aligned these motifs for each protien. The results obviously depend on the divergence of the proteins, the number of sequences, the length of the motifs and the indel penalties but were often disappointing. As an example, the results for just ClustalV with 6 sequences were (100, 92, 100, 100, 100), (100, 83, 67, 100, 100, 100,100, 100, 100), (100, 0, 67) and (100, 67, 50, 50), respectively. Note that these motifs should be highly conserved and retain the most information enabling a correct alignment. ClustalV was one of the better algorithms but would still often miss these motifs.

When all is said and done, people will still find that the alignments produced by the programs can be improved by a judicious and critical examination by eye. Spending time to slowly and carefully examine your alignments by hand is recommended. Occasionally you might see an alignment that contains

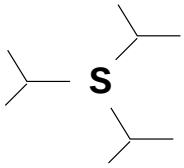


when an obviously better alignment would be



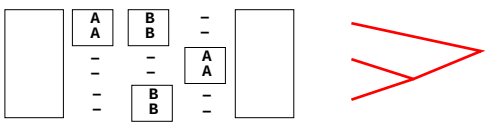
But why should this be necessary?

Many algorithms make some use of a tree or phylogeny in the construction of the alignment. It is how this information is used that can create some of these problems. If the nodes, S, containing the above deletion are central to the phylogeny, e.g.

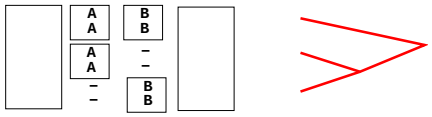


then insertions of the block 'A' must be made independently within each evolutionary branch. This will incur the same penalty in the local alignment whether it is placed to the left or to the right.

Similarly you might see



instead of



when the phylogeny shown on the right of the diagram (in red) is used. For many algorithms these are situations where the "apparent" score changes little or at all and hence the algorithm will not recognize it as a possibility for improvement (pers. comm. John Kececioglu).

In addition to these problems, all algorithms that I know of consider the penalty applied to gaps (and mismatches) as a constant throughout the length of the sequence. Yet all biologists recognize that this is not the case and understand that indels are more likely at the ends of a sequence and more likely in loop regions than in catalytic centers. We also have little idea of what are appropriate quantitative levels for the gap penalties. As a result, the alignments can always be improved by a careful examination. The algorithms can help with task. We routinely do several automated alignments for every comparison (minimally one with the default penalties, one with more severe and one with less severe penalties) and then compare these by hand.