

Pairwise sequence comparison

Frédérique Galisson

September 11, 2000

The topic that will be addressed here concerns the comparison between two biological sequences. The first part covers some algorithmic methods allowing to estimate (locate and quantify) the similarity between any two sequences, particularly through the obtention of an alignment of the sequences. The second part focuses on models used to measure the similarity between monomers of biological macromolecules.

1 Sequence comparison methods

1.1 Diagonals method

This method, often called “dot-plot matrix” was first introduced by Gibbs and McIntyre in 1970 [1], who used it for comparing cytochrome C sequences.

Its graphical version is most known: one sequence is represented on the horizontal axis, the other one on the vertical axis. If the x residue in the first sequence is identical to the y residue in the second sequence, the position of coordinates (x, y) in the matrix is labelled with a dot (thus, the “dot-plot” name). Regions of the two sequences sharing a high density of identical letters appear as diagonal fragments. This simple and fast method offers a visual representation of the similar regions between the two sequences. However, if one simply places a dot each time two positions of the sequences are identical, the signal to noise ratio is generally quite low because the background is very high: with proteins, the average probability that two random positions are identical is 0.05, and with nucleic acids, it is as high as 0.25.

One way to increase the stringency of the method is to use “word” (or “k-tuple”, or “window”) matches instead of residue matches. For each position x (or y) in a sequence, the word of size k for this position is defined as the consecutive group of letters starting at position x (or y) and continuing for a total of k letters. For each pair (x, y) of positions in the two sequences, a

dot is placed if and only if the words of size k starting at positions x and y in respectively the first and the second sequence are identical. The selectivity of the method increases with k . For example, with $k = 2$, in the case of protein sequences, the probability of a dot occurring randomly is yet no more than 0.0025 (1/400). On the other hand, this increase in selectivity comes with a decrease in sensitivity: for example, if $k = 2$, if two proteins share a similar region, with on average only one over two positions that are identical, this similarity may escape detection.

A more flexible way for modulating the sensitivity to selectivity ratio involves selecting both a size k for the sliding window and a threshold for the comparison of words: a dot is placed at position (x,y) if and only if the number of identical positions between the words starting at x in the first sequence and y in the second one is greater than or equal to the threshold. The size of the sliding window or word, and the chosen value for the threshold are two parameters allowing to modify the signal to noise ratio.

It is also possible to introduce here the notion of similarity between amino acids, by referring to a scoring matrix (see section 2.1), and to set a threshold such that one places a dot at the point (x,y) if and only if the similarity score (taken from the scoring matrix) between amino acids at respective positions x and y in the first and second sequences is greater than or equal to the threshold. Of course, taking into account the similarities between individual amino acids is also possible in combination with the use of a sliding window of size k .

The graphical version described above, provides an approximate but quick estimate of where and how much two sequences are similar. There are also non-graphical versions of the basic method, like those used in the first step of both the Fasta and Blast programs (see corresponding course).

The main limits of this method are:

- the difficulty in quantifying the similarity that is being observed;
- the difficulty in extending its use to the simultaneous comparison of more than two sequences.

Moreover, it is difficult to manipulate the resulting representation from such a comparison. So, for these reasons, the concept of alignment for comparing biological sequences has appeared, starting in the 70's.

1.2 Alignments of sequences

When two sequences are very similar, it is possible to align them just by eye, almost without any ambiguities. When the similarity decreases, and particularly when it becomes necessary to introduce deletions or insertions within one of the two sequences, the number of solutions one has to consider becomes too large to handle manually. Computational methods have been developed that allow the calculation of the best alignment between two sequences, given some criteria one wants to optimize.

1.2.1 Global optimal alignment between two sequences

As in the preceding section, we first ignore the question of how to score a substitution between two amino acids, as well as how to penalize gaps, and we consider the biological sequences simply as character strings, with no biological meaning. The problem of calculating the best alignment between two sequences reduces then to the optimization of a scoring function, whose variables are the studied sequences. The scores associated with matches, mismatches or gaps are the parameters of this function and constitute the “scoring system”.

Some definitions

- Σ denotes the alphabet of the sequences. E.g. for nucleic acids, $\Sigma = \{A, C, G, T\}$
- $\bar{\Sigma}$ denotes the set $\Sigma \cup \{-\}$, the symbol “-” representing a gap in one or the other of the sequences at one position in the alignment.
- The concatenation of n symbols taken from the alphabet, and forming the S string is denoted $S = s_1s_2 \cdots s_n$.
- The alignment between two symbols x and y is denoted $\begin{pmatrix} x \\ y \end{pmatrix}$
- $\begin{pmatrix} s_1s_2 \cdots s_n \\ r_1r_2 \cdots r_m \end{pmatrix}$ denotes the alignment between sequences $s_1s_2 \cdots s_n$ and $r_1r_2 \cdots r_m$.
- $opt_- \begin{pmatrix} s_1s_2 \cdots s_n \\ r_1r_2 \cdots r_m \end{pmatrix}$ denotes the optimal alignment of the two sequences $s_1s_2 \cdots s_n$ and $r_1r_2 \cdots r_m$.

Alignments without gaps

In the case of an alignment without gaps between two strings, these strings must be of the same length, and there is only one possible alignment, that is defined as the pairing of all the successive elements of the two strings $A = a_1a_2\dots a_q$ and $B = b_1b_2\dots b_q$. We represent the alignment as a sequence of pairs of letters:

$$\mathcal{A} = \begin{pmatrix} a_1a_2\dots a_q \\ b_1b_2\dots b_q \end{pmatrix} = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \dots \begin{pmatrix} a_q \\ b_q \end{pmatrix}$$

The score of the alignment is defined as:

$$\text{score}\mathcal{A} = \sum_{i=1}^q \text{score} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \quad (1)$$

from which one may derive the recursive formula:

$$\text{score}\mathcal{A} = \text{score}\mathcal{A}' + \text{score} \begin{pmatrix} a_q \\ b_q \end{pmatrix} \quad (2)$$

with

$$\mathcal{A}' = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \dots \begin{pmatrix} a_{q-1} \\ b_{q-1} \end{pmatrix} = \begin{pmatrix} a_1a_2\dots a_{q-1} \\ b_1b_2\dots b_{q-1} \end{pmatrix}$$

Having established these two equations will be useful below, in the case of alignments with gaps.

Scores: similarity or distance

The score of an alignment, with or without gaps, may represent a measure of the similarity between the sequences, or a measure of the distance between them. The distance will vary inversely with the similarity between the sequences.

Whatever x , y , and z , three elements taken from the alphabet, a distance metrics verifies:

- $d(x, y) \geq 0$,
- $d(x, y) = d(y, x)$,
- $d(x, x) = 0$,
- $d(x, y) \leq d(x, z) + d(y, z)$.

In the case of alignments without gaps, if one uses a score of 0 for the pairing of two identical letters, and a score of 1 for the pairing of non identical symbols, the global score is called the Hamming distance between the two strings, and it represents the number of differences between these two aligned strings.

Alignments with gaps

Let $A = a_1a_2 \cdots a_m$ and $B = b_1b_2 \cdots b_n$, two strings formed over the Σ alphabet, and of respective lengths m and n , with $m \leq n$. An alignment between strings A and B , is a sequence of paired symbols:

$$\mathcal{A} = \left(\begin{array}{c} a_1a_2 \cdots a_m \\ b_1b_2 \cdots b_n \end{array} \right) = \left(\begin{array}{c} \bar{a}_1 \\ \bar{b}_1 \end{array} \right) \left(\begin{array}{c} \bar{a}_2 \\ \bar{b}_2 \end{array} \right) \cdots \left(\begin{array}{c} \bar{a}_i \\ \bar{b}_i \end{array} \right) \cdots \left(\begin{array}{c} \bar{a}_p \\ \bar{b}_p \end{array} \right), \text{ with :}$$

- \bar{a}_i and \bar{b}_i being two elements from $\bar{\Sigma} = \Sigma \cup \{-\}$,
- $a_1a_2 \cdots a_m = \bar{a}_1\bar{a}_2 \cdots \bar{a}_p$ and $b_1b_2 \cdots b_n = \bar{b}_1\bar{b}_2 \cdots \bar{b}_p$ (if one ignores gaps, represented by “-”),
- then necessarily, $n \leq p \leq m + n$
- in one pair $\left(\begin{array}{c} \bar{a}_i \\ \bar{b}_i \end{array} \right)$, both elements cannot be the symbol “-” at the same time.

As in the case of ungapped alignments (equation 1), the score of the alignment is defined as the sum of the scores of every pair in the alignment:

$$score\mathcal{A} = \sum_{i=1}^p score \left(\begin{array}{c} \bar{a}_i \\ \bar{b}_i \end{array} \right) \quad (3)$$

If the set of scoring parameters is like this:

$$score \left(\begin{array}{c} \bar{a}_i \\ \bar{b}_i \end{array} \right) = \begin{cases} 0 & \text{if } \bar{a}_i = \bar{b}_i, \\ \text{else } 1, \end{cases} \quad (4)$$

then, the score of the optimal alignment between the sequences (the one whose score is the smallest) is called the minimum editing distance between A and B , or Levensthein distance. This is the scoring system that will be used for the rest of this section.

Searching the optimal alignment between A and B amounts then to looking for the minimum among the scores of all the possible alignments between

A and B . Fortunately, it will be possible to avoid the obvious but expensive method that would involve the computation of all these scores and comparing them. Let us consider this observation: any alignment of A and B may end in three different manners, indeed the last pair of the alignment, $\begin{pmatrix} \bar{a}_p \\ \bar{b}_p \end{pmatrix}$, may be either $\begin{pmatrix} a_m \\ b_n \end{pmatrix}$ or $\begin{pmatrix} a_m \\ - \end{pmatrix}$ or $\begin{pmatrix} - \\ b_n \end{pmatrix}$, and:

$$\mathcal{A} = \begin{pmatrix} a_1 a_2 \cdots a_m \\ b_1 b_2 \cdots b_n \end{pmatrix} = \begin{cases} \begin{pmatrix} a_1 a_2 \cdots a_{m-1} \\ b_1 b_2 \cdots b_{n-1} \end{pmatrix} \begin{pmatrix} a_m \\ b_n \end{pmatrix} \text{ or,} \\ \begin{pmatrix} a_1 a_2 \cdots a_{m-1} \\ b_1 b_2 \cdots b_n \end{pmatrix} \begin{pmatrix} a_m \\ - \end{pmatrix} \text{ or,} \\ \begin{pmatrix} a_1 a_2 \cdots a_m \\ b_1 b_2 \cdots b_{n-1} \end{pmatrix} \begin{pmatrix} - \\ b_n \end{pmatrix} \end{cases}$$

This observation leads to two consequences :

- This gives us a means for calculating the number of alignments of A and B , $f(m, n)$, which is equal to $f(m - 1, n - 1) + f(m - 1, n) + f(m, n - 1)$. It is possible to solve this recursion and express $f(m, n)$ as a function of m and n . For two sequences of length 1000, the number of possible alignments is on the order of 10^{600} (for comparison, the Avogadro number is on the order of 10^{23} , and the estimated number of particles in the universe is on the order of 10^{80}). Even using a computer, it is not conceivable to calculate all these alignments ([2]).
- The score of \mathcal{A} can be written, as in the case of ungapped alignments (equation 2) with a recursive formula:

$$\text{score}\mathcal{A} = \text{score}\mathcal{A}' + \text{score} \begin{pmatrix} \bar{a}_p \\ \bar{b}_p \end{pmatrix} \quad (5)$$

with $\mathcal{A}' = \begin{pmatrix} \bar{a}_1 \bar{a}_2 \cdots \bar{a}_{p-1} \\ \bar{b}_1 \bar{b}_2 \cdots \bar{b}_{p-1} \end{pmatrix} = \begin{pmatrix} \bar{a}_1 \\ \bar{b}_1 \end{pmatrix} \begin{pmatrix} \bar{a}_2 \\ \bar{b}_2 \end{pmatrix} \cdots \begin{pmatrix} \bar{a}_{p-1} \\ \bar{b}_{p-1} \end{pmatrix}$. If \mathcal{A} is the optimal alignment of A and B , then \mathcal{A}' is also an optimal alignment (rank $p - 1$) (from the ‘‘optimality principle’’, [2], [3]).

From the above observation, \mathcal{A}' may be either $\mathcal{A}'_1 = \text{opt}_- \begin{pmatrix} a_1 a_2 \cdots a_{m-1} \\ b_1 b_2 \cdots b_{n-1} \end{pmatrix}$,
or $\mathcal{A}'_2 = \text{opt}_- \begin{pmatrix} a_1 a_2 \cdots a_{m-1} \\ b_1 b_2 \cdots b_n \end{pmatrix}$, or $\mathcal{A}'_3 = \text{opt}_- \begin{pmatrix} a_1 a_2 \cdots a_m \\ b_1 b_2 \cdots b_{n-1} \end{pmatrix}$,

and it follows that :

$$score_{\mathcal{A}} = \min \begin{cases} score_{\mathcal{A}'_1} + score \begin{pmatrix} a_m \\ b_n \end{pmatrix} \\ score_{\mathcal{A}'_2} + score \begin{pmatrix} a_m \\ - \end{pmatrix} \\ score_{\mathcal{A}'_3} + score \begin{pmatrix} - \\ b_n \end{pmatrix} \end{cases}$$

Comment: of course, if one were reasoning in terms of similarity instead of distance, one would try to maximize the score, instead of minimize.

In an analogous way, each of the three \mathcal{A}' alignments may be obtained from one of its three preceding sub-alignments, and this is true at all ranks. For any i and j , both greater than 0, we have:

$$score_{opt-} \begin{pmatrix} a_1 \cdots a_i \\ b_1 \cdots b_j \end{pmatrix} = \min \begin{cases} score_{opt-} \begin{pmatrix} a_1 \cdots a_{i-1} \\ b_1 \cdots b_{j-1} \end{pmatrix} + score \begin{pmatrix} a_i \\ b_j \end{pmatrix} \\ score_{opt-} \begin{pmatrix} a_1 \cdots a_i \\ b_1 \cdots b_{j-1} \end{pmatrix} + score \begin{pmatrix} - \\ b_j \end{pmatrix} \\ score_{opt-} \begin{pmatrix} a_1 \cdots a_{i-1} \\ b_1 \cdots b_j \end{pmatrix} + score \begin{pmatrix} a_i \\ - \end{pmatrix} \end{cases} \quad (6)$$

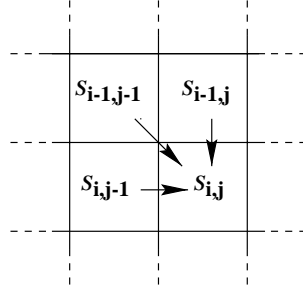


Figure 1: $S_{i,j}$ is the score of the optimal alignment between $a_1 \cdots a_i$ and $b_1 \cdots b_j$. The diagonal arrow represents the substitution of a_i with b_j , whereas horizontal and vertical arrows respectively represent the insertion of b_j and a_i in front of a gap in the other sequence. These arrows are “weighted” with the corresponding scores, given by the chosen scoring system.

The initial conditions (i and/or $j = 0$) of this recursion are easy to calculate:

- if i and j are both equal to 0, the score is 0 (nothing has been aligned);
- if $i = 0$ and $j \neq 0$,

$$\text{score opt}_- \left(\begin{array}{c} - \cdots - \\ b_1 \cdots b_j \end{array} \right) = \text{score opt}_- \left(\begin{array}{c} - \cdots - \\ b_1 \cdots b_{j-1} \end{array} \right) + \text{score} \left(\begin{array}{c} - \\ b_j \end{array} \right) \quad (7)$$

- if $i \neq 0$ et $j = 0$,

$$\text{score opt}_- \left(\begin{array}{c} - \cdots - \\ a_1 \cdots a_i \end{array} \right) = \text{score opt}_- \left(\begin{array}{c} - \cdots - \\ a_1 \cdots a_{i-1} \end{array} \right) + \text{score} \left(\begin{array}{c} - \\ a_i \end{array} \right) \quad (8)$$

Calculating the optimal alignment between two sequences may then be represented as the search for the optimal path in a graph: horizontal arrows correspond to the insertion of gaps in the sequence represented vertically, vertical arrows correspond to insertion of gaps in the horizontal sequence, and diagonals arrows correspond to the pairing of residues, one from each sequence. An example of such a graph is given on figure 2, which represents the alignments between sequences *ACACA* and *ACCACC* (reproduced with permission from [4]). The path that is emphasized models the alignment

$$\begin{array}{cccccc} A & - & C & A & - & C & A \\ A & C & C & A & C & C & - \end{array}$$

The goal is then to find the best path in such a weighted graph, each edge carrying the cost of its corresponding elementary editing operation. These costs, scores, or weights, are given by the chosen scoring system. The alignment is then built, step by step, representing it as a path in a table of dimensions $m + 1$ and $n + 1$, with the *A* sequence represented vertically from top to bottom, and the sequence *B*, horizontally from left to right.

- For all values of i and j , respectively between 0 and m and between 0 and n , the cell of indices i and j in the table contains the score of the optimal alignment of $a_1 \cdots a_i$ with $b_1 \cdots b_j$.
- The cell $(0, 0)$ contains the value 0 (nothing has been aligned). This is the start point of the alignment.
- The first row ($i = 0$) and the first column ($j = 0$) are easily filled in, since only one way is possible for reaching one of these cells (equations 7 and 8).

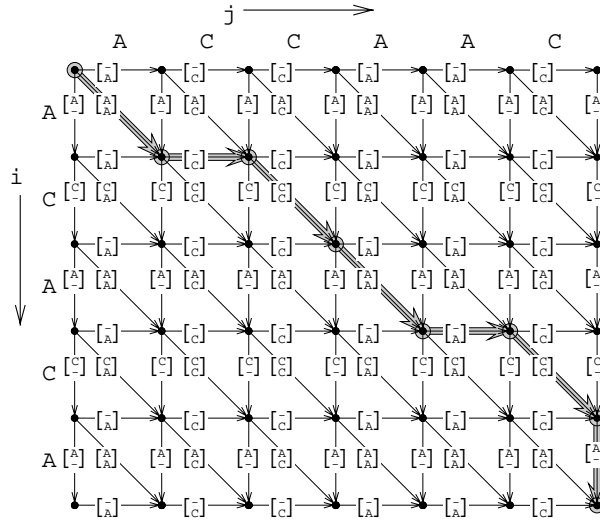


Figure 2: Graph representing all the possible alignments between sequences *ACACA* and *ACCACC*. Edges are labelled with the elementary operations, substitutions, deletions and insertions, that they represent.

- For i and j both greater than 0, the value in the cell of indices i and j is calculated from those of the three cells immediately on the left (optimal score of the alignment of $a_1 \cdots a_i$ with $b_1 \cdots b_{j-1}$), above (optimal score of the alignment of $a_1 \cdots a_{i-1}$ with $b_1 \cdots b_j$), and above on the left (optimal score of the alignment of $a_1 \cdots a_{i-1}$ with $b_1 \cdots b_{j-1}$), from the formula given in equation 6. This is illustrated on figure 1.
- The cell of indices m and n will contain the score of the optimal alignment between A and B .
- In order to build the corresponding alignment, one must have memorized, at each step, the path that has been chosen: a horizontal arrow means that one was coming from the cell on the left, and thus that one has introduced a gap in the vertical sequence; conversely if one comes from the cell just above; and if one comes from the cell that is above and on the left, a diagonal arrow will mean that one has paired the residues of respective indexes i and j in the sequences A and B .

An example of a scores and path table is shown in figure 3, of the same sequences as in figure 2.

	-	A	C	C	A	C	C
-	0	1	2	3	4	5	6
A	1	0	1	2	3	4	5
C	2	1	0	1	2	3	4
A	3	2	1	1	1	2	3
C	4	3	2	1	2	1	2
A	5	4	3	2	1	2	2

Figure 3: Scores table from the calculation of the best alignment between *ACACA* and *ACCACC*. The scoring system is the one of 4. Among the edges that have been kept during the computation, one can find two paths allowing to go backward from the rightmost cell on the last row (end of the alignment) to the leftmost cell on the first row (startpoint of the alignment). These two paths represent the two optimal alignments (they get the same score of 2) between the two sequences, under the scoring system that has been chosen.

Comments:

- When one calculates the optimal score of any sub-alignment, one has to calculate the minimum between three integers. The computer cannot do that directly, and the operation has to be done in several steps, e.g.:

- minimum of a , b and c : one first compares a and b then,
 - if $a \leq b$, then one compares a and c , and if $a \leq c$, the result is a , else it is c ;
 - else (if $a > b$), one compares b and c , and if $b \leq c$, the result b , else it is c .

In the case where a and b are equal, with the above algorithm, the result will always be a , never b , just because of the order with which the comparisons are made. This situation may arise when filling in the table, and corresponds to different alignments with identical scores.

- If A' and B' are sub-sequences from A and B such that A' corresponds to the i first residues of A , and B' corresponds to the j first residues of B , and if \mathcal{A} is an optimal alignment of A and B , and \mathcal{A}' is an optimal alignment of A' and B' , \mathcal{A}' is not necessarily included in \mathcal{A}

(the path representing \mathcal{A} in the table, does not necessarily include the cell of indexes (i, j)).

This method is known as the algorithm of Needleman and Wunsch, or sometimes Needleman-Wunsch-Sellers ([5], [6],[7]). The class of methods it belongs to, is called *Dynamic Programming*. *Dynamic Programming* may be applied to a wide range of problems, which fit the following criteria: the solution of a sub-problem may be derived from the solution of a sub-problem immediately preceding it. One may notice that this method does not require the examination of all the possible alignments between the two sequences. The alignments whose scores are indicated in the table are only a very small fraction of the whole. Indeed, because of the optimality criterion, the algorithm guarantees that at each step of the calculation, one can find the optimal alignment from one of its three immediate sub-alignments, and therefore at each step, two thirds of the possible alignments are eliminated. For these reasons, this method is said to be *exact* (it is guaranteed to find the *optimal* solution), even if it is not *exhaustive* (it does not explore the *whole* space of possible alignments).

1.2.2 Local alignments

With the method described in the preceding section, we align the sequences along their entire lengths, the alignment taking into account all the residues of each sequence. What we obtain is thus a *global alignment* of the sequences. With biological molecules, it may sometimes be more relevant to search the best *local alignment* of the two sequences.

There exist several slight modifications of the Needleman-Wunsch-Sellers algorithm. For example it is possible not to penalize the gaps when they are at one extremity of one of the sequences. These modifications make it possible to compute the best alignment corresponding to the inclusion of one sequence into the other, or the best overlap between the two sequences.

What is more difficult is if one wants to get the best alignment between a sub-sequence of A and a sub-sequence of B , given that the start and end positions of these sub-sequences are not *a priori* known: the start and end positions of the alignment in both sequences are yet part of the problem that we have to solve. An algorithm for solving this problem has been developed by Smith and Waterman (and is called the Smith-Waterman algorithm) in 1981 ([8]). This algorithm is derived from the Needleman-Wunsch-Sellers algorithm, and uses the same principle:

- The computation of the optimal alignment is done by *Dynamic Pro-*

gramming, by filling in a table of scores and another table for keeping in memory the chosen paths (or computing the paths afterwards, from the scores table, using a backtracking algorithm).

- In case of the local alignment problem, it becomes necessary to reason in terms of similarity instead of distance, and moreover there are some requirements about the scoring system that is to be chosen. Indeed, for determining the start and end positions of the alignment, we need the score to be able to vary in both ways: we want to begin the alignment where the similarity becomes to increase and we want to stop it when the similarity decreases. It means that we need the score to increase with the similarity and to decrease as soon as the sequences become to be no more similar (and we do not want the score to increase systematically with the length of the alignment). Thus, we choose a scoring system such that all that is considered as “good” (identities, and pairing between similar letters) gets a positive score, and all the other operations (gaps, and pairing between not similar amino acids) get negative scores.

More formally, we need a system in which an alignment taken randomly receives a negative score, in order to guarantee that alignments scores will not tend to increase just with length. For ungapped local alignments, it has been proven [9] that the scoring system must fulfill these two requirements:

$$\sum_{i,j} f_i f_j S_{i,j} \leq 0 \tag{9}$$

and, there exists at least one pair i, j , such that $S_{i,j} > 0$, i and j being taken from the sequences alphabet.

For alignments with gaps, there is no theory about the alignments scores and scoring systems have to be empirically validated.

- With the correct scoring system, we now just need to fill in the tables of scores, applying the same principles as for the global alignments, but with one difference: while the score is below 0, we just place 0 in the cell. Each cell containing a 0 is a potential alignment start point. Once the table is filled in, one has to locate in the table the cell containing the highest score. It corresponds to the end point of the best local alignment. Then, one has just to backtrack the path (as we do in the case of global alignment), until reaching a cell containing a 0, which will indicate the start point of the optimal local alignment.

There exist many variations from this algorithm, allowing for example to get also sub-optimal local alignments ([2] and [10]).

Conclusions

Dynamic programming methods for both global or local alignments guarantee at least one optimal solution to the problem that is formulated. This guarantee of optimality and correctness has a cost in terms of time and computation space that are required:

- The time required for performing the computation is a function of the number of elementary operations that have to be performed. Here, we need to fill in a table of dimensions $m + 1$ and $n + 1$. Each cell corresponds to one elementary computing operation. Thus, the required time will be roughly proportional to the product of the lengths of the two sequences.
- If one wants not only to calculate the score of the alignment, but also to be able to build the alignment, one has also to keep in the computer's memory the path that is being chosen, until the table is completely filled in. It means that the memory space that is required to make the program run will also grow with the product of the lengths of the two sequences.

There exist some improvements of the basic algorithms, allowing to save time and space. These two issues become really a problem when one wants to extend these solutions to bigger or more complex problems like comparing one sequence against a whole database of sequences, or simultaneously aligning multiple sequences. One should notice that in the case of the comparison of a sequence against a whole database, the size of the problem, even if “big”, grows only linearly with the size of the database. In the case of the extension to the multiple alignment problem, the size of the problem increases exponentially with the number of sequences, and becomes intractable with pure dynamic programming methods as soon as the number of sequences becomes greater than six or seven sequences. For these reasons, for both of these problems, *heuristic* (meaning they don't guarantee to find the “optimal” result) methods have been developed. These methods are explained in the corresponding chapters.

2 Scoring systems

In the preceding sections, we focused on computational methods that compare sequences, given a set of parameters, and particularly that find the best global or local alignment between these sequences. We temporarily left aside the fact that the sequences of interest are those of biological molecules and that the similarity that we want to estimate may be interpreted from a biological point of view. In order to give the alignments some biological meaning, we need scoring systems that attempt to be “biologically relevant”.

When the character strings being compared represent nucleic acids or proteins, an elementary operation like the pairing between two residues or the introduction of a gap at one position of a sequence, must be interpreted in the context in which the comparison is being performed: if one compares two sequences with the underlying idea that they may have evolved from a common ancestor, we would like the score of the pairing between two residues to be related to what we know about the evolution of biological sequences, and for instance reflecting the probability of the observed substitution. Similarly, if one wants to interpret the similarity between two protein sequences in terms of structural properties, we would like the scores to reflect structural similarities.

This section will address the question of how to estimate the similarity between two amino acids, and how to score gaps in biological alignments. One has to remember that there may be different answers to these questions depending on the biological context in which we perform the sequence comparison. The elementary scores are parameters of the sequence comparison programs, and even if there are default values that come with the programs, they may be changed by the users.

2.1 Scoring matrices for amino acids

The similarity scores between amino acids are usually symmetrical (i.e. $s_{i,j} = s_{j,i}$), and are proposed into symmetrical matrices of dimension 20.

The notion of similarity between amino acids refers implicitly to the biological - point of view from which one considers that two amino acids look more or less like each other. For example, the similarity scores will not be the same if one considers specifically some physical or chemical properties, or if one is interested in the propensity of two amino acids to be exchanged while keeping the structural or functional properties of the sequence, etc.

One may distinguish two classes of approaches that have been used to set these similarity scores:

- those considering precisely some particular features (for example chemical properties) of the amino acids, and setting similarity scores directly from these features.
- those that consist in observing in real biological molecules how the amino acids are indeed exchanged, and deriving from these observations a set of probabilities of substitutions, which are then converted into similarity scores.

2.1.1 Direct methods

This approach relies on the direct comparison of amino acids and consists in defining similarity values for all the possible pairs.

A first obvious example of this approach leads to the “identity matrix”: each diagonal cell contains the value 1 (or any other positive value), and non diagonal elements are all set to 0.

If one wants to choose a biochemical point of view, one has to try quantifying the similarities according to some physical and chemical properties. This has been done by Grantham in 1974 ([11]), who considered the atomic composition, polarity, and volume of the side chains of amino acids, and, differently by Miyata in 1979 ([12]). In 1987, Rao ([13]) considered properties such that hydrophobicity and propensity to form some secondary structures.

There have also been some attempts to adopt a strict genetic point of view: in 1966, Fitch has derived scores from the minimum number of mutations at the level of DNA that were required to observe a substitution at the level of amino acids ([14], [15]).

The biochemical and genetic points of view may also be considered together, which has been done by Feng and Doolittle in 1985 ([16]).

The first problem of the biochemical and/or genetic approaches is that they consider only a subset of the biological properties that may be involved when one compares amino acids, and relies only on some intuitive notions of biochemical or genetic similarities.

Finally, in 1971, MacLahan instead of focusing on some specific features, considered the propensity of amino acids to be exchangeable, as may be observed from an alignment of homologous proteins ([17]). From these observations, he derived similarity scores ranging from 1 to 9. So, his approach, conversely from the preceding ones, did not try to explain the similarity in terms of biochemical or genetic features, but just took into account the result of what natural evolution has done.

The second problem, common to all these approaches, is that whatever

the criteria used to judge the similarity between amino acids, the different scores are defined in an arbitrary manner.

2.1.2 Log-odds matrices

For the reasons explained above, the scores resulting from various direct estimations of the similarities between amino acids have not extensively been used in sequence comparison, and approaches deriving scores indirectly, from the estimation of the probabilities of substitutions between amino acid, have rapidly proven to be more accurate and efficient.

As above, different criteria may be taken into account, but yet the idea is to start from observed substitutions and to deduce from these observations the probabilities of specific substitutions. These probabilities, are then converted into scores. In order to observe these substitutions, one has to consider alignments. If these alignments correspond to structural alignments, the corresponding scores will reflect structural similarities. If the alignments one considers are those of homologous proteins, the probabilities will be those of the substitution from one amino acid to another one, with conservation of the function.

This later approach is the one that has been used by Margaret Dayhoff as early as 1968, and others followed in 1972 and 1978 ([18], [19]), and this is the one that will be explained in more detail in the next section, which explains the construction of "log-odds scoring matrices".

PAM matrices

The fundamental idea underlying the work of M. Dayhoff's group was to use a functional definition of the similarity between amino acids: the similarity between two amino acids should be reflected in the frequency of observed substitutions, with conservation of function. These conservative substitutions are identified after aligning protein sequences that are known (from external knowledge) to be homologous (which means that they have evolved from a common ancestor) and to perform the same function.

The methodology used for calculating the matrices was the same in 1968 and 1978, the only difference being the number and nature of the sequences that were considered. The "PAM" matrices that are proposed with many programs, and still used, are those that were developed in 1978.

«PAM» is an acronym for "Accepted Point Mutation"; it is the name given to the matrices described below, and also the name of the unit defined by the authors for expressing amounts of evolution: an evolutionary time

is characterized by the percentage of mutations that occurred during that time. A PAM1 similarity matrix contains similarity scores that one should apply to sequences that diverge from each other by an amount of evolution of one PAM, or 1% mutations. Similarly, a PAM250 represents the similarities between amino acids after a period of 250 PAMs. Since two successive mutations may arise at the same position in the sequences, a period of x PAMs does not correspond to an amount of $x\%$ of observed substitutions (see corresponding table).

The construction and the use of these matrices rely on two assumptions about the evolutionary process followed by the protein sequences:

- the mutational events at one position of a sequence are supposed not to depend on the context. It means they do not depend neither on the position itself, nor on the nature of the amino acids surrounding it, nor on the mutational events at other positions in the sequence.
- the mutational events at one position are supposed to be independent from the evolutionary history at that position

(One may notice that from our current understanding about the evolution and structure of proteins, both of these hypotheses are generally wrong. But, taking into account more realistic but complicated models is much more difficult.)

One may distinguish four steps in the process leading to the final matrices:

1. Collecting raw data:

- The sequences were chosen among different protein families, and were required (inside a family) to share at least 85% identity, for two reasons: first, this level of similarity allows to align the sequences without the help of a computer program, for there are few ambiguities and few gaps to introduce; second, if the sequences are very similar, the amount of evolution supposed to separate them is considered to be short, and one makes the hypothesis that multiple changes at the same positions have not occurred (a multiple change would mean that when we observe an exchange between i and j , the actual story is $i \rightarrow x \rightarrow j$), so that observing i in front of j in an alignment is interpreted as a substitution from i to j or from j to i .
- The observed substitutions are identified and counted: $A_{i,j} = A_{j,i}$ represents the number of observed substitutions from i to j or

from j to i . Indeed, when one observes i and j at the same position in an alignment, one doesn't know the direction of the exchange, and one considers that both probabilities are the same. The matrix A that is obtained is thus symmetrical.

- Frequencies of occurrences: n_i represents the number of times that i occurs in the sequences, from which are derived the relative frequencies: $f_i = \frac{n_i}{N}$, with $N = \sum_k n_k$

2. The transition matrix:

From the data contained in the A matrix are derived the probabilities of mutation that will form the M matrices, also called "transition matrices". An element $M_{i,j}$ of an M matrix represents the probability of mutation from j to i during a given amount of evolution that is the same for all the $M_{i,j}$ and is thus a feature of M . The M matrices are non-symmetrical: the intrinsic mutabilities of the amino acids are taken into account, which means that the probability that i mutates to j is different from the probability that j mutates to i , because the probabilities that they mutate at all are different.

- If $p_x(j \rightarrow i)$ is the probability that j mutates in i during an evolution period of x PAMs, it can be written like:

$$p_x(j \rightarrow i) = p(j \rightarrow i | j \rightarrow) p_x(j \rightarrow)$$

where $p(j \rightarrow i | j \rightarrow)$ is the probability that j mutates in i given that j mutates (does not depend on x) and $p_x(j \rightarrow)$ is the probability that j mutates (during the evolution period of x PAMs). $p(j \rightarrow i | j \rightarrow)$ is easily calculated from the data collected at first step. $p_x(j \rightarrow)$ may be written as $\lambda_x m_j$ where λ_x depends on x , and m_j , called the relative mutability of j is calculated from the data (equations and details of the calculation are not given here) and represents the probability for each occurrence of j that it mutates, during an amount of evolution of one PAM.

- Given that both $p(j \rightarrow i | j \rightarrow)$ and m_j can be calculated from the observed data, and that λ_x only depends on x , one may calculate the $M_{i,j}^x$ elements of an M^x matrix for any value of x . M. Dayhoff first calculated the M^1 matrix and then obtained the other ones by successive multiplications: it can easily be demonstrated that the M^{x+y} , whose elements are the probabilities of amino acids mutations during an evolution period of $x+y$ PAMs,

is the product of M^x by M^y . This is true because of the second hypothesis (see above), meaning that what happens during the x period doesn't depend on what happens during the y period: this allows to multiply the probabilities of events occurring respectively during the x and y periods. That is how it is possible to derive matrices representing long periods of evolution from sequence data representing only short evolutionary distances.

3. The relatedness odds matrix:

- Each $R_{i,j}$ element of this matrix is such that: $R_{i,j} = \frac{M_{i,j}}{f_i}$, which may also be written as $\frac{f_j \times M_{i,j}}{f_i \times f_j}$. $M_{i,j}$ is the probability that j mutates to i per occurrence of j , and thus $f_j \times M_{i,j}$ is the probability for observing a mutation of j to i in sequences where their respective relative frequencies are f_j and f_i . $f_i \times f_j$ is the probability of observing j exchanged with i in the same sequences, just by chance. This ratio is called "odds ratio".
- From the above definitions of M and R , one can show that $R_{i,j} \simeq R_{j,i}$, which means that the R matrix is symmetrical. So, $R_{i,j}$ is the probability of observing a substitution from j to i or from i to j , per occurrence of i and per occurrence of j , or said differently, it is the ratio of the chances that when we observe i in front of j in an alignment, it is because one has mutated into the other one, or just because of their respective probabilities (f_i , f_j) of being there.
- When one aligns two protein sequences, the product of the $R_{i,j}$ for all the aligned (i, j) pairs gives us the ratio of the probability that the alignment represents the evolution of sequence into the other one, to the probability of a chance alignment. Then, the logarithm of this ratio gives us a score, which is greater than 0 if the probability that this alignment represents the evolution of one sequence into the other one (of course, it means "during the amount of x PAMs that the matrix represents") is greater than the probability that this alignment is obtained "by chance", given the amino acids composition of the sequence (that is approximated to be the same for all the studied sequences...).

4. The log-odds matrix :

Because the different positions of the sequences are considered to be independent from each other (first hypothesis, see above), we could

multiply the $R_{i,j}$ values of all the aligned pairs in order to get an odds ratio for the whole alignment. This would lead to very small numbers, that are difficult to work with a computer. Also, in the first section, we have defined the score of an alignment as the sum of the scores for each pair of aligned residues. Since the logarithm of a product is equal to the sum of the logarithms, we just have to take the logarithms of the $R_{i,j}$ values in order to get scores that we can sum:

$$S_{i,j} = k \log R_{i,j}$$

k is just a scaling factor that allows us to work with integers. In the original Dayhoff's work, k was equal to 10. The Dayhoff matrices (the well known "PAM" series), as well as all those derived by similar ways are called "log-odds matrices".

Dayhoff matrices have been criticized (see below) and new "log-odds" matrices have been developed. They are slightly different, but the general method, leading to "log-odds scores" has been kept.

The main criticisms were:

- the data set that she used was small (it reflects the number of available sequences when she did the work), and some substitutions had not been observed; there was also a bias in the composition of the proteins from the dataset due to a bias in the available protein families). New PAM matrices have been developed, using the same methodology but taking advantage of the much bigger number of available sequences more than 10 years later ([20], [21]).
- as described above, the hypothesis saying that successive changes are independent from one another allows us to extrapolate probabilities of mutations for long periods of evolution from data corresponding to short periods. This hypothesis has been much criticized and is probably wrong ([22]). This is one aspect that is different between the PAM and Blossum matrices. The Blossum matrices are also log-odds matrices that are widely used (see below).
- the hypothesis saying that events at the different positions do not depend on one another has also been criticized, and there have been some attempts for developing matrices based on substitution frequencies of di-amino-acids ([23]).
- M. Dayhoff was considering the whole sequences from the data set, without distinguishing regions that are very strongly conserved from

regions that are subjected to little or no selection pressure (regions that are less important with respect to the structure and function of the protein). Taking these regions into account introduces “background” in the counts for observed changes, if we want these changes to be “accepted” changes that reflect functional similarities between amino acids. Some matrices based on alignments of conserved “blocks”, that do not contain gaps, have been developed. They are the series of the Blosom matrices ([24]). Their second difference when compared with Dayhoff matrices is that the scores for a given amount of evolution are derived directly from the data (no extrapolation by matrix products like was done by Dayhoff). For example “blosom62” means that the sequences used for deriving it shared at least 62% of identity.

Conclusion

- the log-odds matrices have proven to work better in scoring homologous proteins than those previously developed.
- the closer the amount of evolution represented by the matrix to the actual distance between the sequences being compared, the better the scores will discriminate “biologically meaningful” alignments from “chance” alignments ([25]). Of course, before aligning two sequences, one doesn’t know their evolutionary distance. That’s why it is often advised, particularly in the context of a database search, to use several different matrices corresponding to different amounts of evolution.
- some empirical evaluations of the different matrices have been done ([16], [26], [27], [28])). The results vary depending on the evaluation criteria, but it seems that on average the blosom50 and blosom62, as well as the “new PAMs” (like “pet91”, [20], and the “Gonnet series”, [21])) give the best results.

2.2 Gap penalties

The first algorithms and scoring systems that have been developed for biological sequence comparisons were scoring gaps with a fixed penalty for each residue aligned with a gap (as in section 1.2.1). The result is that the penalty for a gap is proportional to its length, which also means that we consider a gap of length k as being the same thing as k independent gaps of length l . This appears to lead to alignments that contain many short insertions and

deletions, which is not what is observed when considering proteins of known structure ([29]).

In a biological context, we may expect gap penalties to reflect our biological interpretation of their occurrence. If one interprets gaps into alignments as insertion and deletion events occurring at the level of DNA during evolution, we would like to distinguish two penalties: one for the existence of a gap, and another one depending on its length.

Thus, modifications of the algorithms, allowing to define a gap penalty of the form $a + b \times k$, k being the length of the gap, and where a is often called "the gap creation penalty", and b , "the gap extension penalty".

There is no theory allowing the calculation of the probability of a gap (as a function of the evolutionary distance and/or its length for example) and to derive a score for the alignment of a gap with k residues. Particularly, there is no theoretical background justifying to use an affine function for scoring gaps. Moreover, there exist some studies about the distribution of gaps in proteins as a function of their length that indicate that penalties of the form $a + b \times \log k$ would probably better fit biological observations and be more relevant ([30]). Algorithms using this kind of gap penalties have been developed ([31]), but probably because they are more difficult to implement, they are not currently used, and most programs propose affine gap penalties.

It has recently been proposed to model the gaps in a way that avoids aligning regions of low similarity, in order to prevent the introduction of artificial gaps inside them ([32]). Indeed, this means considering the alignment as a succession of ungapped blocks of similarity separated by unaligned regions of low similarity. These regions are considered to contain unpaired residues and pairs of residues that are left unaligned. One such region, involving k_1 residues in one sequence and k_2 residues in the other one ($k_1 \geq k_2$), is penalized with a negative weight equal to $a + b(k_1 - k_2) + ck_2$ (k_2 is the number of residues that are unpaired and $k_1 - k_2$ is the number of pairs of residues that are unaligned. This gap scoring scheme is called "generalized affine gap costs").

There exist also some variations on gaps weighting that have been implemented, like the one proposed in the Calign program ([33]): the gaps in the shorter sequence that are longer than l (a user-specified parameter) are penalized with a fixed cost. It enables one to get very long gaps in one of the sequences, separating similar regions, a situation that may arise when aligning genomic DNA with corresponding coding (spliced) sequences. It amounts to consider two different kinds of gaps in the alignment: those that are required for correctly align related segments in the two sequences, and those that correspond to introns in the genomic sequence.

2.3 Similarity scores between nucleotides

Most of time the similarity scores between nucleotides distinguish only between matches and mismatches. Their relative scores may reflect a PAM distance as described above ([34]). In some cases (mostly for phylogenetic analyses), there exist scoring systems for nucleotides that take into account the fact that transitions are supposed to occur more frequently than transversions, and even other more sophisticated evolutionary models of nucleic acids.

Acknowledgments

I wish to thank Emmanuel Chang for his careful reading and editing of the manuscript.

References

- [1] Adrian J. Gibbs and George A. McIntyre. The diagram, a method for comparing sequences. its use with amino acid and nucleotide sequences. *Eur. J. Biochem.*, 16:1–11, 1970.
- [2] Michael S. Waterman. *Introduction to computational biology - Maps, sequences and genomes*. Chapman & Hall, 1995.
- [3] Dan Gusfield. *Algorithms on strings, trees, and sequences - Computer science and computational biology*. Cambridge University Press, 1997.
- [4] manuscript in preparation, personal communication.
- [5] Saul B. Needleman and Christian D. Wusnch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.
- [6] Peter H. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26(4):787–793, 1974.
- [7] Temple F. Smith, Michael S. Waterman, and Walter M. Fitch. Comparative biosequence metrics. *J. Mol. Evol.*, 18:38–46, 1981.
- [8] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.
- [9] Stephen F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, 219:555–565, 1991.

- [10] <http://www-hto.usc.edu/software/seqaln/>.
- [11] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185:862–864, 1974.
- [12] T. Miyata, S. Miyasawa, and T. Yasunaga. Two types of amino acid substitutions in protein evolution. *J. Mol. Evol.*, 12:219–236, 1979.
- [13] J. K. M. Rao. New scoring matrix for amino acid residue exchanges based on residue characteristic physical parameters. *Int. J. Pept. Protein Res.*, 29, 1987.
- [14] W. M. Fitch. An improved method of testing for evolutionary homology. *J. Mol. Biol.*, 16:9–16, 1966.
- [15] W. M. Fitch. Construction of phylogenetic trees. *Science*, 15:299–304, 1967.
- [16] D. F. Feng, M. S. Johnson, and R. F. Doolittle. Aligning amino acids sequences: comparison of commonly used methods. *J. Mol. Evol.*, 21:112–125, 1985.
- [17] A.D. McLachlan. Tests for comparing related amino-acid sequences. cytochrome c and cytochrome c551. *J. Mol. Biol.*, 61:409–421, 1971.
- [18] M.O. Dayhoff and R.V. Eck. A model of evolutionary change in proteins. In Natl. Biomed. Res. Fnd., editor, *Atlas of Protein Sequence and Structure*, chapter 4, pages 33–41. Silver Spring MD, 1967-68.
- [19] M.O. Dayhoff and R.V. Eck. Matrices for detecting distant relationships. In Natl. Biomed. Res. Fnd., editor, *Atlas of Protein Sequence and Structure*, chapter 23, pages 353–358. Silver Spring MD, 1978.
- [20] David T. Jones, William R. Taylor, and Janet M. Thornton. The rapid generation of mutation data matrices from protein sequences. *CABIOS*, 8(3):275–282, 1992.
- [21] Steven A. Benner, Mark A. Cohen, and Gaston H. Gonnet. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.
- [22] Steven A. Benner, Mark A. Cohen, and Gaston H. Gonnet. Amino acid substitution during functionally constraint divergent evolution of proteins. *Protein Eng.*, 7(11):1323–1332, 1994.

- [23] Gaston H. Gonnet, Steven A. benner, and Mark A. Cohen. Analysis of amino acid substitution during divergent evolution: the 400 by 400 dipeptide substitution matrix. *Biochem. and Biophys. Res. Com.*, 199(2):489–496, 1994.
- [24] Steven Henikoff and Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.
- [25] Stephen F. Altschul. A protein alignment scoring system sensitive at all evolutionary distances. *J. Mol. Evol.*, 36:290–300, 1993.
- [26] Steven Henikoff and Jorja G. Henikoff. Performance evaluation of amino acid matrices. *Proteins: Structure, Function and Genetics*, 17:49–61, 1993.
- [27] Gerhard Vogt, Thure Etzold, and Patrick Argos. An assessment of amino acid exchange matrices in aligning protein sequences: the twilight zone revisited. *J. Mol. Biol.*, 249:816–831, 1995.
- [28] Mark S. Johnson and John P. Overington. A structural basis for sequence comparisons: an evaluation of scoring methodologies. *J. Mol. Biol.*, 233:716–738, 1993.
- [29] Stefano Pascarella and Patrick Argos. Analysis of insertions/deletions in protein structures. *J. Mol. Biol.*, 224:461–471, 1992.
- [30] Steven A. Benner, Mark A. Cohen, and Gaston H. Gonnet. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *J. Mol. Biol.*, 229:1065–1082, 1993.
- [31] Webb Miller and Eugene W. Myers. Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, 50(2):97–120, 1988.
- [32] Stephen F. Altschul. Generalized affine gap costs for protein sequence alignment. *Proteins: Structure, Function and Genetics*, 32:88–96, 1998.
- [33] Kun-Mao Chao. Calign: aligning sequences with restricted affine gap penalties. *Bioinformatics*, 15(4):298–304, 1999.
- [34] David J. States, Warren Gish, and Stephen Altschul. Improved sensitivity of nucleic acid database searches using application specific scoring matrices. *Methods: a companion to Methods in Enzymology*, 3(1), 1991.