

The analysis of multiple DNA or protein sequences (I)

Sequence similarity is an important issue in sequence alignments. In molecular biology, proteins and DNA can be similar with respect to their function, their structure, or their primary sequence of amino or nucleic acids. The general rule is that sequence determines shape, and shape determines function. So when we study sequence similarity, we eventually hope to discover or validate similarity in shape and function. This approach is often successful. However, there are many examples where two sequences have little or no similarity, but still the molecules fold into the same shape and share the same function. This lecture speaks of neither shape nor function. We are focusing on DNA sequences. Sequences are seen as strings of characters. In fact, the ideas and techniques we discuss in this lecture have important applications in text processing, too. Similarity has both a quantitative and a qualitative aspect: A *similarity measure* gives a quantitative answer, saying that two sequences show a certain degree of similarity. An *alignment* is a mutual arrangement of two sequences which is a sort of qualitative answer; it exhibits where the two sequences are similar, and where they differ. An optimal alignment, of course, is one that exhibits the most correspondences, and the least differences.

Edit Distance

Two ways are used to quantify similarity of two sequences: A *similarity measure* is a function that associates a numeric value with a pair of sequences, with the idea that a higher value indicates greater similarity. Beyond this, similarity measures vary widely, and care must be taken when interpreting similarity measures. The notion of distance is somewhat dual to similarity. It treats sequences as points in a metric space. A distance measure is a function that also associates a numeric value with a pair of sequences, but with the idea that the larger the distance, the smaller the similarity, and vice versa. Distance measures usually satisfy the mathematical axioms of a metric. In particular, distance values are never negative. In most cases, distance and similarity measures are interchangeable in the sense that a small distance means high similarity, and vice versa.

Hamming Distance

For two sequences of equal length, we just count the character positions in which they differ.

For example:

sequence <i>s</i>	AAT	AGCAT	GATCGTG
sequence <i>t</i>	TAA	ACAAT	GTCGTGG
Hamming distance(<i>s</i> , <i>t</i>)	2	2	5

This distance measure is very useful in some cases, but in general it is not flexible enough. First of all, the sequences may have different length. Second, there is generally no fixed correspondence between their character positions. In the mechanism of DNA replication, errors like deleting or inserting a nucleotide are not unusual. Although the rest of the sequences are identical, such a shift of position leads to exaggerated values in the Hamming distance. Look at the rightmost example above. The Hamming distance says that *s* and *t* are apart by 5 characters (out of 7). On the other hand, by deleting the second character **A** from *s* and the last character **G** from *t*, both become equal to *GTCGTG*. In this sense, they are only two characters apart! Let us model the distance of *s* and *t* by introducing a gap character "-" and edition operations including insertion, deletion, match, and

replacement (mismatch). We say that the pair

- (a, a) denotes a match (no change from s to t),
- $(a, -)$ denotes deletion of character a in s ,
- (a, b) denotes replacement of a (in s) by b (in t), where $a \neq b$,
- $(-, b)$ denotes insertion of character b (in s).

Since the problem is symmetric in s and t , a deletion in s can be seen as an insertion in t , and vice versa.

An alignment of two sequences s and t is an arrangement of s and t by position, where s and t can be padded with gap symbols to achieve the same length:

$s : \mathbf{G A T C G T G --}$

$t : \mathbf{G -- T C G T G G}$

or

$s : \mathbf{G A T C G T -- G}$

$t : \mathbf{G -- T C G T G G}$

Next we turn the edit protocol into a measure of distance by assigning a "cost" or "weight" w to each operation (deletion, insertion, match, replacement). We define

- $w(a, a) = 0$
- $w(a, b) = 1$ for $a \neq b$
- $w(a, -) = w(-, a) = 1$

This scheme is known as the Levenshtein Distance, also called unit cost model. Its predominant virtue is its simplicity. In general, more sophisticated cost models must be used. For example, replacing an amino acid by a biochemically similar one should weight less than a replacement by an amino acid with totally different properties. We will discuss those things later. Now we are ready to define the most important notion for sequence analysis:

- The *cost of an alignment* of two sequences s and t is the sum of the costs of all the edit operations that lead from s to t .
- An *optimal alignment* of s and t is an alignment which has minimal cost among all possible alignments.
- The *edit distance* of s and t is the cost of an optimal alignment of s and t under a cost function w . We denote it by $d_w(s, t)$.

Using the unit cost model for w in our previous example, we obtain the following cost:

$s : \mathbf{G A T C G T G --}$

$t : \mathbf{G -- T C G T G G}$

cost: 2

$d_w(s, t) = 2$

The analysis of multiple DNA or protein sequences (II)

Calculating Edit Distances and Optimal Alignments

The number of possible alignments between two sequences is gigantic, and unless the weight function is very simple, it may seem difficult to pick out an optimal alignment. But fortunately, there is an easy and systematic way to find it. The algorithm described now is very famous in biocomputing, it is usually called “the dynamic programming algorithm”.

For $s = s_1s_2\dots s_m$ and $t = t_1t_2\dots t_n$, let us consider two prefixes $\mathbf{0} : s : i = s_1s_2\dots s_i$ and $\mathbf{0} : t : j = t_1t_2\dots t_j$ with $i, j \geq 1$. We assume we already know optimal alignments between all shorter prefixes of \mathbf{s} and \mathbf{t} , in particular of

1. $\mathbf{0} : s : (i - 1)$ and $\mathbf{0} : t : (j - 1)$, of
2. $\mathbf{0} : s : (i - 1)$ and $\mathbf{0} : t : j$, and of
3. $\mathbf{0} : s : i$ and $\mathbf{0} : t : (j - 1)$.

An optimal alignment of $\mathbf{0} : s : i$ and $\mathbf{0} : t : j$ must be an extension of one of the above by

1. a Replacement (mismatch) (s_i, t_j) , or a Match (s_i, t_j) , depending on whether $s_i = t_j$.
2. a Deletion $(s_i, -)$, or
3. an Insertion $(-, t_j)$.

Edit Distance (Recursion)

We simply have to choose the minimum:

$$d_w(\mathbf{0} : s : i, \mathbf{0} : t : j) = \min\{d_w(\mathbf{0} : s : (i - 1), \mathbf{0} : t : (j - 1)) + w(s_i, t_j), \\ d_w(\mathbf{0} : s : (i - 1), \mathbf{0} : t : j) + w(s_i, -), \\ d_w(\mathbf{0} : s : i, \mathbf{0} : t : (j - 1)) + w(-, t_j)\}$$

There is no choice when one of the prefixes is empty, i.e. $i = 0$, or $j = 0$, or both:

$$d_w(\mathbf{0} : s : 0, \mathbf{0} : t : 0) = 0 \\ d_w(\mathbf{0} : s : i, \mathbf{0} : t : 0) = d_w(\mathbf{0} : s : (i - 1), \mathbf{0} : t : 0) + w(s_i, -) \text{ for } i = 1, \dots, m \\ d_w(\mathbf{0} : s : 0, \mathbf{0} : t : j) = d_w(\mathbf{0} : s : 0, \mathbf{0} : t : (j - 1)) + w(-, t_j) \text{ for } j = 1, \dots, n$$

According to this scheme, and for a given w , the edit distances of all prefixes of i and j define an $(m + 1) \times (n + 1)$ distance matrix $D = (d_{ij})$ with $d_{ij} = d_w(\mathbf{0} : s : i, \mathbf{0} : t : j)$.

The three-way choice in the minimization formula for d_{ij} leads to the following pattern of dependencies between matrix elements:

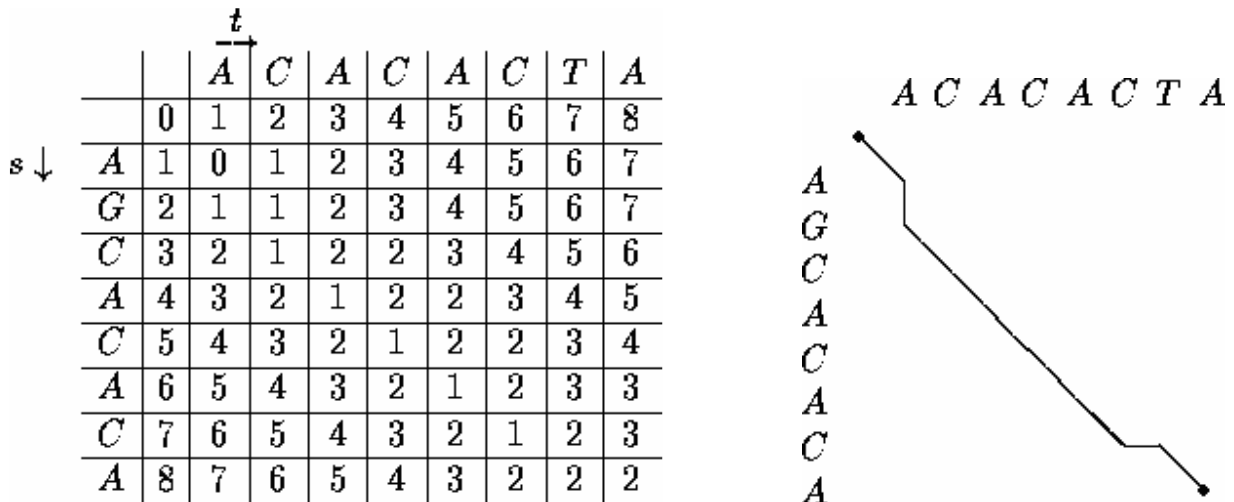
$$\begin{array}{ccc} d_{i-1, j-1} & & d_{i-1, j} \\ & \searrow & \downarrow \\ d_{i, j-1} & \rightarrow & d_{i, j} \end{array}$$

The bottom right corner of the distance matrix contains the desired result:

$$d_{mn} = d_w(\mathbf{0} : s : m, \mathbf{0} : t : n) = d_w(s, t).$$

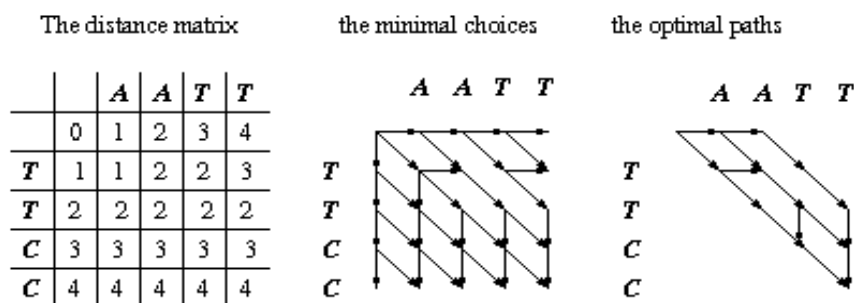
Edit Distance (Matrix)

Below is the distance matrix for the example with $s = AGCACACA$, $t = ACACACTA$ (the cost for a match is 0; the cost for a deletion or an insertion or a replacement is 1):



In the second diagram, we have drawn a path through the distance matrix indicating which case was chosen when taking the minimum. A diagonal line means Replacement (Mismatch) or Match, a vertical line means Deletion, and a horizontal line means Insertion. Thus, this path indicates the edit operation protocol of the optimal alignment with $d_w(s, t) = 2$. Please note that in some cases the minimal choice is not unique and different paths could have been drawn which indicate alternative optimal alignments.

Here is an example where the optimal alignment is not unique. Let $s = TTCC$, $t = AATT$.



In which order should we calculate the matrix entries? The only constraint is the above pattern of dependencies. The most common order of calculation is line by line (each line from left to right), or column by column (each column from top-to-bottom).

A Word on the Dynamic Programming Paradigm

“Dynamic Programming” is a very general programming technique. It is applicable when a large search space can be structured into a succession of stages, such that

- The initial stage contains trivial solutions to sub-problems,
- Each partial solution in a later stage can be calculated by recurring on only a fixed number of partial solutions in an earlier stage,

- The final stage contains the overall solution.

This applies to our distance matrix: The columns are the stages, the first column is trivial, the final one contains the overall result. A matrix entry $d_{i,j}$ is the partial solution $d_w(\theta:s:i, \theta:t:j)$ and can be determined from two solutions in the previous column $d_{i-1,j-1}$ and $d_{i,j-1}$ plus one in the same column, namely $d_{i-1,j}$. Since calculating edit distances is the predominant approach to sequence comparison, some people simply call this THE dynamic programming algorithm. Just note that the dynamic programming paradigm has many other applications as well, even within bioinformatics.

A Word on Scoring Functions and Related Notions

Many authors use different words for essentially the same idea: scores, weights, costs, distance and similarity functions all attribute a numeric value to a pair of sequences.

- “distance” should only be used when the metric axioms are satisfied. In particular, distance values are never negative. The optimal alignment minimizes distance.
- The term “costs” usually implies positive values, with the overall cost to be minimized. However, metric axioms are not assumed.
- “weights” and “scores” can be positive or negative. The most popular use is that a high score is good, i.e. it indicates a lot of similarity. Hence, the optimal alignments maximize scores.
- The term “similarity” immediately implies that large values are good, i.e. an optimal alignment maximizes similarity. Intuitively, one would expect that similarity values should not be negative (what is less than zero similarity?). But don't be surprised to see negative similarity scores shortly.

Mathematically, distances are a little more tractable than the others. In terms of programming, general scoring functions are a little more flexible.

Let us close with another caveat concerning the influence of sequence length on similarity. Let us just count exact matches and let us assume that two sequences of length n and m , respectively, have 99 exact matches. Let $c_w(s,t)$ be the similarity score calculated for s and t under this cost model. So, $c_w(s,t) = 99$. What this means depending on n and m : If $n = m = 100$, the sequences are very similar - almost identical. If $n = m = 1000$, we have only 10% identity! So if we relate sequences of varying length, it makes sense to use length-relative scores - rather than $c_w(s,t)$ we use $c_w(s,t)/(n+m)$ for sequence comparison.