4 Pairwise alignment

We will discuss:

1. Strings
2. Dot matrix method for comparing sequences
3. Edit distance
4. Scoring functions and matrices
5. Global-, local-, repeat- and overlap alignment of two sequences using dynamic programming

Sources and recommended reading:


4.1 Strings

Definition 4.1.1 An alphabet $\Sigma$ is a finite nonempty set. The elements of an alphabet are called symbols or letters. A string $S$ over an alphabet $\Sigma$ is a (finite) concatenation of symbols from $\Sigma$. The length of a string $S$ is the number of symbols in $S$, denoted by $|S|$. The number of strings of length $n$ over $\Sigma$ is denoted by $\Sigma^n$.

Example: For DNA-sequences $\Sigma = \{A, G, C, T\}$. Then $S = ATGGAAATGCTAATAG$ over $\Sigma$ is of length 15.

Note: Strings and sequences are usually synonyms in bioinformatics. Usually we will use the word sequence within the biological context (such as DNA and protein sequences) and strings within more abstract questions.

Definition 4.1.2 Let $S$ and $T$ be two strings over an alphabet $\Sigma$. Then by $ST$ we denote the concatenation of $S$ and $T$.

Note: Generally $ST \neq TS$ for strings.

Definition 4.1.3 Let $S$ and $T$ be two strings over an alphabet $\Sigma$.
- $S$ is a substring of $T$ if there are strings $U, V$ over $\Sigma$ such that $T = USV$.
- $S$ is a prefix of $T$ if there exists a string $V$ such that $T = SV$.
- $S$ is a suffix of $T$ if there exists a string $U$ such that $T = US$. 
Example:

\[
T = \text{ATGGAATGCTAATAG} \quad \text{then} \\
S = \text{AATGCT} \quad \text{is a substring of } T, \\
S = \text{ATGGAAT} \quad \text{is a prefix and} \\
S = \text{GCTAATAG} \quad \text{is a suffix of } T
\]

**Definition 4.1.4** Let \(\Sigma\) be an alphabet and let \(S = s_1 \ldots s_n\) with \(s_i \in \Sigma\). For all \(i, j \in \{1, \ldots, n\}, i < j\), we denote the substring \(s_i \ldots s_j\) by \(S[i, j]\).

### 4.2 Sequence formats

- **Single FASTA format:** In the first row of the file after the symbol “>” an identifier for a sequence follows. In the following lines the sequence itself follows. Empty lines are not read.

Example:

>gi|139617|sp|P09184|VSR_ECOLI PATCH REPAIR PROTEIN
MADVHDKATRSKNMRAIATDTAIEKRLASLLTGQLAFRVQDASLPGRPDFVDEVYRCVI
FTHGCFWHHHHHLFVKPATRTEFWELEKIGNVERDRRDISRLQELGWRVLIWECALR
GREKLTDEALTERLEEWICGEGASQIDTQGIHLLA

- **Multiple-FASTA-Format:** A file can consist of several sequences, which are written as a series of FASTA entries as described above.

Example:

>SEQUENCE 1
ACTCFTGGCGCGC........
>SEQUENCE 2
ACGGCGCTCFTGGCGCGCGC........
>SEQUENCE 3
GTTGGGACTCFTGGCGCGCGC........
>SEQUENCE 4
TGCGCACTCFTGGCGCGCGC........

### 4.3 Dot matrix sequence comparison

A dot matrix analysis is primarily a method for comparing two sequences. An \((n \times m)\) matrix relating two sequences of length \(n\) and \(m\) respectively is produced: by placing a dot at each cell for which the corresponding symbols match. Here is an example for the two sequences **IMISSMISSISSIPPI** and **MYMISISSAHIPIPE**:

```
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
```

```
Definition 4.3.1 Let $S = s_1 s_2 \ldots s_n$ and $T = t_1 \ldots t_m$ be two strings of length $n$ and $m$ respectively. Let $M$ be an $n \times m$ matrix. Then $M$ is a dot plot if for $i, j, 1 \leq i \leq n, 1 \leq j \leq m : M[i, j] = 1$ for $s_i = t_j$ and $M[i, j] = 0$ else.

Note: The longest common substring within the two strings $S$ and $T$ is then the longest matrix subdiagonal containing only 1s. However, rather than drawing the letter 1 we draw a dot, and instead of a 0 we leave the cell blank. Some of the properties of a dot plot are

- the visualization is easy to understand
- it is easy to find common substrings, they appear as contiguous dots along a diagonal
- it is easy to find reversed substrings (see assignment)
- it is easy to discover displacements
- it is easy to find repeats

Example: DNA sequences which encode the Bacteriophage lambda and Bacteriophage P22 repressor proteins:

Real dot plots of biological sequences will contain a lot of dots, many of which are considered as noise. To reduce the noise, a window size $w$ and a stringency $s$ are used and a dot is only drawn at point $(x, y)$ if in the next $w$ positions at least $s$ characters are equal. For the example above:

4.3.1 Dot matrix repeat detection

Dot matrix analysis of human LDL receptor against itself (protein sequence):
Exercise: Determine which \( w \) and \( s \) are best to use in this case, and interpret the result.

### 4.4 Sequence alignment

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple-alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in both sequences.

Two sequences are aligned by writing them across a page in two rows. Identical or similar characters are placed in the same column, whereas non-identical characters are either placed in the same column as a mismatch or are opposite a gap in the other sequence.

Two strings: \( \text{IMISSMISSISSIPPI} \) \( \text{I-MISSMISSISIPPI-} \) \( \text{MYMISSISAHIPPIE} \) \( \text{MYMISS-ISAH-IPPIE} \)

Align:

\[
\begin{align*}
\text{THIS IS A RATHER LONGER SENTENCE THAN THE NEXT.} \\
\text{THIS IS A SHORT SENTENCE.}
\end{align*}
\]

\[
\begin{align*}
\text{THIS IS A RATHER LONGER SENTENCE THAN THE NEXT.} \\
\text{THIS IS A ###### SHORT# SENTENCE##############.}
\end{align*}
\]

or

\[
\begin{align*}
\text{THIS IS A RATHER LONGER SENTENCE THAN THE NEXT.} \\
\text{THIS IS A SHORT# ###### SENTENCE##############.}
\end{align*}
\]

### 4.4.1 String alignment

**Definition 4.4.1** Given two strings \( X \) and \( Y \). An alignment \( A \) of \( X \) and \( Y \) is obtained by inserting dashes (\( '-' \)) so that both resulting strings \( X' \) and \( Y' \) of equal length can be written one above the other in such a way that each character in the one string is opposite a unique character in the other string. Usually, we require that no two dashes are aligned in this way.

Example:

\[
\begin{align*}
X &= \text{YE-STERDAY} \\
Y &= \text{EASTERS-}
\end{align*}
\]

In order to evaluate an alignment we need a scoring system. The score of two aligned characters in biology is mostly a similarity score, however we will first start with the mathematical concepts of distances of sequences and/or characters in sequences.
4.5 Distance

For this let us recall the formal definition of distance:

**Definition 4.5.1** A set $X$ of elements $x, y, \ldots \in X$ is called a metric space if for each pair $x, y \in X$ there exists a real number $d(x, y)$ with:

1. $d(x, y) \geq 0, d(x, y) = 0 \iff x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, y) \leq d(x, z) + d(y, z) \ \forall z \in X$

$d(x, y)$ is called the distance of $x$ and $y$.

**4.5.1 Minkowski metric**

**Definition 4.5.2** Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two elements of an $n$-dimensional space $X$. Then

$$d_M(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad \text{(4.1)}$$

is called the Minkowski distance with parameter $p$.

Note: For $p = 1$ the distance is also called the Manhattan (or city-block) distance, for $p = 2$ we have the well-known Euclidean distance.

**4.5.2 Hamming metric**

**Definition 4.5.3** Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be two strings of length $n$ over an alphabet $\Sigma$. Then

$$d_H(X, Y) = |\{i | i \in \{1, \ldots, n\}, x_i \neq y_i\}| \quad \text{(4.2)}$$

is called the Hamming distance.

Example:

The Hamming distance of the two sequences

$X = A \ T \ A \ T \ A \ T \ A \ T$
$Y = T \ A \ T \ A \ T \ A \ T \ A$

is equal to $d_H(X, Y) = \ldots$.

**4.5.3 Levenshtein or edit distance**

Another way of defining the distance between two strings follows from the possibilities and number of editing operations needed to transform one string into the other.

**Definition 4.5.4** The Levenshtein distance or edit distance $d_L$ between two strings $X$ and $Y$ is the minimum number of edit operations of type

$$\left\{ \frac{\text{Replacement},}{\text{Insertion, or} } \frac{\text{Deletion,}}{\text{}} \right\}$$

that one needs to transform string $X$ into string $Y$:

$$d_L(X, Y) = \min\{R(X, Y) + I(X, Y) + D(X, Y)\} \quad \text{(4.3)}$$
Note, that in comparison to the Hamming distance the two strings need not be of equal length. Using $M$ for match, an edit transcript is a string over the alphabet $I, D, R, M$ that describes a transformation of $X$ to $Y$.

Example: Given two strings

$X = \text{YESTERDAY}$

$Y = \text{EASTERS}$

The edit distance is equal to 5, which can be easily seen from the minimum edit transcript:

$\text{Edit transcript} = D \ M \ I \ M \ M \ M \ M \ R \ D \ D$

$X = \text{Y E S T E R D A Y}$

$Y = \text{E A S T E R S}$

As we see from this example, edit transcripts and alignments are mathematically equivalent ways of describing a relationship between two strings.

However, an edit transcript implies a set of putative mutational events, whereas an alignment presents a static picture of the relationship.

### 4.5.4 Dynamic programming calculation of edit distance

Though we learnt the definition and the concept for the edit distance, we have not learnt how to compute the minimal edit distance.

Given two strings $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_m$. We want to compute the edit distance $D_L(X,Y)$ between $X$ and $Y$.

Let $D(i,j)$ denote the edit distance of the two prefixes $x_1 \ldots x_i$ and $y_1 \ldots y_j$.

Clearly, it is $D_L(X,Y) = D(n,m)$, and we want to obtain $D(n,m)$ by computing $D(i,j)$ for all $i, j$ with $0 \leq i \leq n$ and $0 \leq j \leq m$.

This is the standard dynamic programming approach and has three essential components:

- the recurrence relation,
- the tabular computation, and
- the traceback.

So how do we compute $D(i,j)$ recursively?

We realize that when we want to compute $D(i,j)$, that this must be the result of three different operations: either we match the two characters at position $i$ and $j$, or character at position $i$ is matched with a gap or vice versa, character at position $j$ is matched with a gap.

In the first case we come from cell position $(i-1, j-1)$, in the second case we come from cell position $(i, j-1)$ and in the third case we come from cell position $(i-1, j)$.

We will actually see in Lemma 1 that this is correct.
The recursion is computed in tabular form:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>\ldots</th>
<th>x_{i-1}</th>
<th>x_i</th>
<th>\ldots</th>
<th>x_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D(0,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_{j-1}</td>
<td>D(i-1, j-1)</td>
<td>D(i, j-1)</td>
<td>\downarrow</td>
<td>\downarrow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_j</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>D(i-1, j)</td>
<td>\rightarrow</td>
<td>D(i, j)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The recurrence relation determines how we can obtain the value for $D(i, j)$ from values for smaller indices. When there are no smaller indices, then we must explicitly state base conditions.

Base conditions:

- We set $D(i, 0) = i$ for all $0 \leq i \leq n$. This corresponds to an alignment in which the first $i$ characters of $X$ are aligned to the left of the first character of $Y$.
- We set $D(0, j) = j$ for all $1 \leq j \leq m$. This corresponds to an alignment in which the first $j$ characters of $Y$ occur to the left of the first character of $X$.

### 4.5.5 The recurrence relation

The recurrence relation for $D(i, j)$ when both $i$ and $j$ are positive, is given by

$$D(i, j) = \min \begin{cases} D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + t(i, j) \end{cases}, \quad (4.4)$$

where $t(i, j) = \begin{cases} 0 & \text{if } x_i = y_j, \\ 1 & \text{else.} \end{cases}$

**Example:**

We can compute the edit score between sequences GATTAG and ATTAC by filling the following matrix:

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>0</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Distance:**

**Theorem** The above recurrence relation is correct.

We will show:

**Proof of Lemma 1** Given a minimum edit transcript $t_1 \ldots t_r$ of $x_1 \ldots x_i$ to $y_1 \ldots y_j$. There are four possibilities for the last symbol $t_r$: 

‘I’ The last operation was to insert \( y_j \) at the end of the first string. Hence, \( t_1 \ldots t_{r-1} \) is a minimum edit transcript of \( x_1 \ldots x_i \) to \( y_1 \ldots y_{j-1} \). By definition, the latter contains \( D(i, j - 1) \) edit operations, thus \( D(i, j) = D(i, j - 1) + 1 \).

‘D’ The last operation was to delete \( x_i \). Hence, \( t_1 \ldots t_{r-1} \) is a minimum edit transcript of \( x_1 \ldots x_{i-1} \) to \( y_1 \ldots y_j \). By definition, the latter contains \( D(i - 1, j) \) edit operations, thus \( D(i, j) = D(i - 1, j) + 1 \).

‘R’, ‘W’ The last operation was to replace (or match) \( x_i \) by \( y_j \). Thus \( t_1 \ldots t_{r-1} \) is a minimum edit transcript of \( x_1 \ldots x_{i-1} \) to \( y_1 \ldots y_{j-1} \) that contains \( D(i - 1, j - 1) + t(i, j) \) edit operations.

**Proof of Lemma 2** We show that all three scores can be achieved, thus so can their minimum:

- There exists a transcript with score \( D(i, j - 1) + 1 \):
  transform \( x_1 \ldots x_i \) to \( y_1 \ldots y_{j-1} \) using \( D(i, j - 1) \) edit operations, then use one more to insert \( y_j \).

- There exists a transcript with score \( D(i - 1, j) + 1 \):
  transform \( x_1 \ldots x_{i-1} \) to \( y_1 \ldots y_j \) using \( D(i - 1, j) \) edit operations, then use one more to delete \( x_i \).

- There exists a transcript with score \( D(i - 1, j - 1) + t(x_i, y_j) \):
  transform \( x_1 \ldots x_{i-1} \) to \( y_1 \ldots y_{j-1} \) using \( D(i - 1, j - 1) \) edit operations, then replace by, or match, \( x_i \) and \( y_j \), using one or zero more edit operations, respectively. \( \square \)

### 4.5.6 Traceback

Now we consider the question how to obtain the actual edit transcript corresponding to the minimal edit distance. While computing the values \( D(i, j) \) one also saves (in an independent matrix) which of the three terms in the recurrence relation was minimal and used for \( D(i, j) \). Then from the final \( D(n, m) \) the edit transcript (and therefore the alignment) can be achieved by backtracking or traceback of the entries in the second matrix.

<table>
<thead>
<tr>
<th>( D )</th>
<th>0</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edit Transcript: I M M M M R
Alignment: G A T T A G

### 4.5.7 Weighted edit distance

We can generalize the edit distance by weighting each of the edit operations I, D and R by a number. The *operation weighted edit distance* between two sequences \( X \) and \( Y \) is the minimum sum of weights of any edit transcript from \( X \) to \( Y \).

A second important generalization is obtained by making the score of an edit operation depend on the two characters involved. This gives rise to the *alphabet weighted edit distance*.

**Definition 4.5.5** Let \( \Sigma \) be a finite alphabet and \( d \) a metric on \( \Sigma \). Let \( \epsilon \in \Sigma \) denote the gap symbol. Then for two strings \( X \) and \( Y \) of length \( n \) and \( m \) respectively

\[
D_L(i, j) := \min \{ D_L(i, j - 1) + d(x_i, \epsilon), D_L(i - 1, j) + d(y_j, \epsilon), D_L(i - 1, j - 1) + d(x_i, y_j) \},
\]

and \( D_L(n, m) \) is the alphabet weighted Levenshtein distance of \( X \) and \( Y \).
4.6 Global distance alignment

When comparing two biological sequences, we want to determine whether and how they diverged from a common ancestor by a process of mutation and selection.

The basic mutational processes are substitutions, insertions and deletions. The latter two give rise to gaps.

The total score assigned to an alignment is the sum of terms for each aligned pair of residues, plus terms for each gap.

Using such an additive scoring scheme is based on the assumption that mutations at different sites occur independently of each other. This is often reasonable for DNA and proteins, but not for structural RNA, where base pairing introduces very important long-range dependences.

Let \( X = x_1 \ldots x_n \) and \( Y = y_1 \ldots y_m \) be two sequences over an alphabet \( \Sigma \). Let \( A \) be a global alignment of length \( l_A \) of \( X \) and \( Y \). Let \( \epsilon \) be the gap symbol. Let \( X' = x'_1 \ldots x'_{l_A} \) and \( Y' = y'_1 \ldots y'_{l_A} \) denote the two strings obtained after inserting dashes (for the gap symbol \( \epsilon \)). Let \( d(a, b), a, b \in \Sigma \cup \{ \epsilon \} \) be a distance on the alphabet. This represents the cost of a mutation of \( a \) into \( b \) or the cost of inserting or deleting a letter. Then define

\[
D(X, Y) = \min_A \left( \sum_{i=1}^{l_A} d(x'_i, y'_i) \right)
\]

(4.6)

The alignment for which the total score is minimal is called optimal.

4.7 String similarity

We have seen how to express string relatedness using the Levenshtein or edit distance. In biology, we are usually interested in similarity rather than distance, as we will see further below.

Given two sequences \( X = x_1 \ldots x_n \) and \( Y = y_1 \ldots y_m \) over an alphabet \( \Sigma \). A similarity score matrix \( S : \Sigma \cup \{ \epsilon \} \times \Sigma \cup \{ \epsilon \} \to \mathbb{R} \) assigns a similarity score to each pair of characters in \( \Sigma \cup \{ \epsilon \} \).

For a given alignment \( A \) of \( X \) and \( Y \) of length \( l_A \), the value of \( A \) is defined as

\[
\sum_{i=1}^{l_A} s(x'_i, y'_i).
\]

For example, for \( \Sigma = \{ A, B, L, - \} \) consider the following similarity score matrix \( S \):

\[
\begin{array}{cccc}
A & B & L & - \\
A & 3 & 1 & -1 & -2 \\
B & 2 & 0 & -3 \\
L & 1 & 2 \\
- & & & 0 \\
\end{array}
\]

so matches of symbols are rewarded, mismatches and gaps penalized.

Example:

\[
\begin{array}{ccccccccc}
X = & B & L & A & - & B & L & A \\
Y = & A & L & A & B & B & L & - \\
1 & +1 & +3 & -3 & +2 & +1 & -2 & = 3 \\
\end{array}
\]

Given a similarity score matrix. The similarity of two sequences \( X \) and \( Y \) is the value of any alignment \( A \) of \( X \) and \( Y \) that maximizes the alignment value. Such an alignment is called optimal.

1. Alignment between very similar human alpha- and beta globins:

\[
\begin{align*}
\text{HBA\_HUMAN} & \quad \text{GSAQVKGHGKKVADALTNAVAVHDDMPNALSALSDLHAKL} \\
\text{G+} & \quad +VK+HGKKV A+++++AH+D++ ++++LS+LH \quad \text{KL} \\
\text{HBB\_HUMAN} & \quad \text{GNPKVKAHGKKVLGA\_FSDLG\_AHLDNLKGT\_FATLSE\_LHCDKL}
\end{align*}
\]
2. Plausible alignment to leghaemoglobin from yellow lupin:

\[
\begin{align*}
\text{HBA\_HUMAN} & \quad \text{GSAQVKGHGKXVA\_D\_TNAVAH} & \quad \text{---D\_MP\_N\_ALS\_} & \quad \text{SDLHAHKL} \\
& \quad ++ \quad +++H+ \quad XV \quad +A \quad ++ \quad +L+ \quad L+++H+ \quad K \\
\text{LGB2\_LUPLU} & \quad \text{NNPE\_LQAH\_AKF\_K} & \quad \text{VY\_EAA\_I\_LQV\_T\_G\_V\_V\_T\_D\_A\_T\_L\_N\_L\_G\_S\_V\_H\_V\_S\_K\_G} \end{align*}
\]

3. A spurious high-scoring alignment of human alpha globin to a nematode glutathione S-transferase homologue:

\[
\begin{align*}
\text{HBA\_HUMAN} & \quad \text{GSAQVKGHGKXVA\_D\_TNAVAH} & \quad \text{DD\_M\_P\_N\_ALS\_A\_LSD} & \quad \text{\_\_\_\_LHAHKL} \\
& \quad GS+ \quad G + \quad +D \quad L \quad ++ \quad H+ \quad D+ \quad A \quad +A \quad L \quad D \quad ++ \quad A \quad H+ \\
\text{F11G11.2} & \quad \text{GSGYLVGD\_SLT\_F\_V\_D\_L\_V\_A\_Q\_H\_T\_A\_D\_L} & \quad \text{\_\_\_\_A\_A\_A\_L\_D\_E\_F\_P\_Q\_F\_K\_A\_H\_Q\_E} \end{align*}
\]

In (1), there are many positions at which the two corresponding residues are identical. Many others are functionally conserved, e.g. the D-E pairs, both negatively charged amino acids.

In (2), we also see a biologically meaningful alignment, as it is known that the two proteins are evolutionarily related, have the same 3D structure and both have the same function. However, there are many fewer identities and gaps have been introduced in the sequences.

In (3), we see an alignment with a similar number of identities or conservative changes as in (2). However, this is a spurious alignment between two proteins that have completely different structure and function.

The goal is to use similarity-based alignments to uncover **homology**, while avoiding **homoplasy**.

### 4.8 The scoring model

The algorithms that compute an alignment critically depend on the choice of the parameters for substitutions, deletions and insertions. Generally no existing scoring model can be applied to all situations. Here the underlying question and/or application always needs to be considered. Generally pairwise alignments are conducted when

- Evolutionary relationships between the sequences are reconstructed. Here scoring matrices based on mutation rates are usually applied.
- Protein domains are compared. Then the scoring matrices should be based on composition of domains and their substitution frequency.

There is a large number of publications on substitutions matrices esp. amino acid matrices. Most of them are based on a statistical theory, which we will look at in the following section.

#### 4.8.1 Substitution matrices

To be able to score an alignment, we need to determine score terms for each aligned residue pair.

**Definition 4.8.1** A substitution matrix \( S \) over an alphabet \( \Sigma = \{a_1, \ldots, a_\kappa\} \) has \( \kappa \times \kappa \) entries, where each entry \((i, j)\) assigns a score for a substitution of the letter \( a_i \) by the letter \( a_j \) in an alignment.

For the generation of scores one first computes relative frequencies of the letters \( f(a_i) \) as well as substitution frequencies \( f(a_i, a_j) \) in a representative data set. The generation of substitution matrices follows the scheme of statistical hypothesis testing as introduced in section 3.3.

Given two sequences

\[
X = (x_1, x_2, \ldots, x_n) \text{ and } Y = (y_1, y_2, \ldots, y_m).
\]

\footnote{Homoplasy\(=\)mutations that appear in parallel or convergently in two different lineages}
The symbols come from some alphabet $\Sigma$, e.g. the four bases \{A, G, C, T\} for DNA or, in the case of amino acids, the 20 symbols \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}.

For now we will only consider non-gapped alignments such as:

\[
\begin{align*}
\text{HBA\_HUMAN} & \quad \text{GSAQVKGHGKKVADLTNAVHDMPNASLDLHAKKL} \\
\text{G} & \quad +\text{VK}\text{HGK}K\text{A}\text{D}++\text{AL}\text{T}++\text{A++}\text{VS}\text{L}\text{D}++\text{LH} \text{KL} \\
\text{HBB\_HUMAN} & \quad \text{GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL}
\end{align*}
\]

Given a pair of aligned sequences (without gaps), the null hypothesis states that the two sequences are unrelated (not homologous). The alignment is then random with a probability described by the model $R$. The unrelated or random model $R$ assumes that in each aligned pairs of residues the two residues occur independently of each other. Then the probability of the two sequences is:

\[
P(X, Y \mid R) = \prod_i p_{x_i} \prod_i p_{y_i}.
\]

In the match model $M$, describing the alternative hypothesis, aligned pairs of residues occur with a joint probability $p_{ab}$, which is the probability that $a$ and $b$ have each evolved from some unknown original residue $c$ as their common ancestor. Thus, the probability for the whole alignment is:

\[
P(X, Y \mid M) = \prod_i p_{x_i y_i}.
\]

Note that the two probabilities $P(X, Y \mid R)$ and $P(X, Y \mid M)$ are likelihoods!

The ratio of the two gives a measure of the relative likelihood that the sequences are related (model $M$) as opposed to being unrelated (model $R$). This ratio is called odds ratio:

\[
\frac{P(X, Y \mid M)}{P(X, Y \mid R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i p_{x_i} \prod_i p_{y_i}} = \prod_i \frac{p_{x_i y_i}}{p_{x_i} p_{y_i}} \quad (4.7)
\]

To obtain an additive scoring scheme, we take the logarithm (base 2 is usually chosen) to get the log-odds ratio:

\[
S = \log \left( \frac{P(X, Y \mid M)}{P(X, Y \mid R)} \right) = \log \left( \prod_i \frac{p_{x_i y_i}}{p_{x_i} p_{y_i}} \right) = \sum_i s(x_i, y_i), \quad (4.8)
\]

with

\[
s(a, b) := \log \left( \frac{p_{ab}}{p_a p_b} \right). \quad (4.9)
\]

We thus obtain a matrix $S = s(a, b)$ that determines a score for each aligned residue pair, known as a score or substitution matrix.

For amino-acid alignments, commonly used matrices are the PAM and BLOSUM matrices.

### 4.8.2 BLOCKS and BLOSUM matrices

The BLOSUM matrices were derived from the database BLOCKS\textsuperscript{2}Blocks are multiply aligned ungapped segments corresponding to the most highly conserved regions of proteins. For the scoring matrices of the BLOSUM (=BLOcks SUBstitution Matrix) family all blocks of the database are evaluated columnwise. For each possible pair of amino acids the frequency $f(a_i, a_j)$ of common pairs $(a_i, a_j)$ in all columns is determined.

Here is an example for such a BLOCKS alignment:

Algorithms in Bioinformatics I, WS'06, ZBIT, C. Dieterich, October 19, 2006

For the following column of a fictitious alignment let us calculate the probabilities and odds ratio:

<table>
<thead>
<tr>
<th>Seq 1</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq 2</td>
<td>A</td>
</tr>
<tr>
<td>Seq 3</td>
<td>A</td>
</tr>
<tr>
<td>Seq 4</td>
<td>A</td>
</tr>
<tr>
<td>Seq 5</td>
<td>A</td>
</tr>
<tr>
<td>Seq 6</td>
<td>A</td>
</tr>
<tr>
<td>Seq 7</td>
<td>A</td>
</tr>
<tr>
<td>Seq 8</td>
<td>A</td>
</tr>
<tr>
<td>Seq 9</td>
<td>A</td>
</tr>
<tr>
<td>Seq 10</td>
<td>C</td>
</tr>
</tbody>
</table>

Altogether there are 45 possible pairs that we can draw from this alignment, of which 36 are AA and 9 are AC pairs. We now assume that the observed frequencies are equal to the frequencies in the population. Then

\[ p_{AA} = \frac{36}{45} \text{ and } p_{AC} = \frac{9}{45} \]

The observed frequency of a single amino acid is generally computed as

\[ p_a = p_{aa} + \sum_{b \neq a} p_{ab}/2 \]

For this example we then get

\[ p_A = 0.8 + 0.2/2 = 0.9 \text{ and } p_C = 0.1. \]

Different levels of the BLOSUM matrix can be created by differentially weighting the degree of similarity between sequences. For example, a BLOSUM62 matrix is calculated from protein blocks such that if two sequences are more than 62% identical, then the contribution of these sequences is weighted to sum to one. In this way the contributions of multiple entries of closely related sequences is reduced. Standard values are BLOSUM50 up to BLOSUM80, with the commonly used BLOSUM62 matrix.

Note that lower BLOSUMx values correspond to longer evolutionary time, and are applicable for more distantly related sequences.

### 4.8.3 BLOSUM62

BLOSUM62 is scaled so that its values are in half-bits, i.e. the log-odds were multiplied by \( \frac{2}{\log_2 2} \) and then rounded to the nearest integer value.
Let us look at two numbers from that matrix explicitly:

\[
\begin{array}{c|c|c|c|c|c|c}
 & A & A & R & -1 & \hline
 \text{p}_a & 0.074 & 0.074 & 0.052 & 0.0023 & \text{p}_b & 0.074 & 0.052 \\
 \text{p}_{ab} & 0.0215 & 0.0023 & 0.598 & -1.485 & \text{p}_{ab}/\text{p}_a \text{p}_b & 3.926 & 0.598 \\
 \text{A-A} & 3.946 & 3.946 & & & 2 \log_2(\text{p}_{ab}/\text{p}_a \text{p}_b) & & \\
 \text{A-R} & & & & & & & \\
\end{array}
\]

4.9 Gap penalties

Gaps are undesirable and thus penalized. The standard cost associated with a gap of length \(g\) is given either by a linear score

\[
\gamma(g) = -gd
\]

or an affine score

\[
\gamma(g) = -d - (g - 1)e,
\]

where \(d\) is the gap open penalty and \(e\) is the gap extension penalty.

Usually, \(e < d\), with the result that less isolated gaps are produced, as shown in the following comparison:

Linear gap penalty: GSAQVKHGKKVADALTNAVHVDMPNALSSALSDLHAKL
GSAQVKHGK-AAAA--D----A-SALSDLHAKL

Affine gap penalty: GSAQVKHGKKVADALTNAVHVDMPNALSSALSDLHAKL
GSAQVKHGK---------SALSDLHAKL

4.10 Alignment algorithms

Given a scoring scheme, we need to have an algorithm that computes the highest-scoring alignment of two sequences.

As for the edit distance-based alignments we will discuss alignment algorithms based on dynamic programming. They are guaranteed to find the optimal scoring alignment.

However, for large sequences they can be too slow and heuristics (such as BLAST, FASTA, MUMMER etc) are then used that usually perform very well, but will miss the best alignment for some sequence pairs.
Depending on the input data, there are a number of different variants of alignment that are considered, among them global alignment, local alignment and overlap alignment.

We will use the two short DNA sequences from section 4.4 for illustration: GATTAG and ATTAC. To score the alignment we will use \( s(a, a) = 1 \), \( s(a, b) = -1 \) and a linear gap cost of \( d = 2 \). (Later, we will also use affine gap costs.)

### 4.11 Global alignment: Needleman-Wunsch algorithm


Consider the problem of obtaining the best global alignment of two sequences. The Needleman-Wunsch algorithm is a dynamic program that solves this problem.

**Idea:** Build up an optimal alignment using previous solutions for optimal alignments of smaller substrings.

Given two sequences \( X = (x_1, x_2, \ldots, x_n) \) and \( Y = (y_1, y_2, \ldots, y_m) \). We will compute a matrix

\[
F : \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} \rightarrow \mathbb{R}
\]

in which \( F(i, j) \) equals the best score of the alignment of the two prefixes \( (x_1, x_2, \ldots, x_i) \) and \( (y_1, y_2, \ldots, y_j) \).

This will be done recursively by setting \( F(0, 0) = 0 \) and then computing \( F(i, j) \) from \( F(i-1, j-1) \), \( F(i-1, j) \) and \( F(i, j-1) \):

\[
\begin{array}{ccccccccc}
0 & x_1 & x_2 & x_3 & \ldots & x_{i-1} & x_i & \ldots & x_n \\
\hline
F(0, 0) & & & & & & & & \\
y_1 & & & & & & & & \\
y_2 & & & & & & & & \\
y_3 & & & & & & & & \\
\vdots & & & & & & & & \\
y_{j-1} & & & & & & & & F(i-1, j-1) \\
y_j & & & & & & F(i, j-1) & \downarrow & \Rightarrow F(i, j) \\
\vdots & & & & & & & & \\
y_m & & & & & & & & \\
\end{array}
\]

#### 4.11.1 The recursion

There are three ways in which an alignment can be extended up to \( (i, j) \):

- \( x_i \) aligns to \( y_j \):
  - \( x_i \) aligns to a gap:
  - \( y_j \) aligns to a gap:

\[
\begin{align*}
\text{A G A } x_i & \quad \text{A A G A } x_i & \quad \text{T G A } x_i \\
\text{A G G } y_j & \quad \text{A G G } y_j & \quad \text{T G G C } y_j \\
\end{align*}
\]

We obtain \( F(i, j) \) as the largest score arising from these three options:

\[
F(i, j) := \max \left\{ \begin{array}{l}
F(i-1, j-1) + s(x_i, y_j) \\
F(i-1, j) - d \\
F(i, j-1) - d
\end{array} \right. \quad (4.12)
\]

This is applied repeatedly until the whole matrix \( F(i, j) \) is filled with values.
To complete the description of the recursion, we need to set the values of $F(i,0)$ and $F(0,j)$ for $i \neq 0$ and $j \neq 0$:

We set $F(i,0) = \underline{\text{________}}$ for $i = 0,1,\ldots,n$ and
we set $F(0,j) = \underline{\text{________}}$ for $j = 0,1,\ldots,m$.

The final value $F(n,m)$ contains the score of the best global alignment between $X$ and $Y$.
To obtain an alignment corresponding to this score, we must find the path of choices that the recursion made to obtain the score. This is called a traceback.

4.11.2 Example of a global alignment matrix

Needleman-Wunsch matrix of the sequences GATTAG and ATTAC:

```
        D 0 G A T T A G
0  0  0 -2 -4 -6 -8 -10 -12
A -2 -1 -1 -3 -5 -7 -9
T -4 -3 -2  0 -2 -4 -6
T -6 -5 -4 -1  1 -1 -3
A -8 -7 -4 -3  0  2  0
C -10 -9 -6 -5 -2  0  1
```

Score= \underline{\text{________}} ;

Alignment

4.11.3 Needleman-Wunsch algorithm

**Input:** two sequences $X$ and $Y$

**Output:** optimal alignment and score $\alpha$

**Initialization:**
Set $F(i,0) := -i \cdot d$ for all $i = 0,1,2,\ldots,n$
Set $F(0,j) := -j \cdot d$ for all $j = 0,1,2,\ldots,m$

For $i = 1,2,\ldots,n$ do:
  For $j = 1,2,\ldots,m$ do:
    Set $F(i,j) := \max \left\{ \begin{array}{l} F(i-1,j-1) + s(x_i,y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{array} \right.$
    Set backtrace $T(i,j)$ to the maximizing pair $(i',j')$

The score is $\alpha := F(n,m)$
Set $(i,j) := (n,m)$

repeat
  if $T(i,j) = (i-1,j-1)$ print $(x_i \ y_j)$
  else if $T(i,j) = (i-1,j)$ print $(x_i \ -y_j)$
  else print $(\ -y_j)$
  Set $(i,j) := T(i,j)$
until $(i,j) = (0,0)$.

4.11.4 Complexity

Complexity of the Needleman-Wunsch algorithm:
We need to store $(n+1) \times (m+1)$ numbers. Each number takes a constant number of calculations to compute: three sums and a max.
Hence, for filling the matrix, the algorithm requires $O(nm)$ time and memory. Given the filled matrix, the construction of the alignment is done in time $O(n + m)$.

For biological sequence analysis, we prefer algorithms that have time and space requirements that are linear in the length of the sequences. Quadratic time algorithms are a little slow, but feasible. $O(n^3)$ algorithms are only feasible for very short sequences.

Something to think about: if we are only interested in the best score, but not the actual alignment, then it is easy to reduce the space requirement to linear.

### 4.12 Local alignment: Smith-Waterman algorithm

Global alignment is applicable when we have two similar sequences that we want to align from end-to-end, e.g. two homologous genes from related species.

Often, however, we have two sequences $X$ and $Y$ and we would like to find the best match between substrings of both. For example, two proteins may share a common domain, but they are no homologs.

Here the score of an alignment between two substrings would be larger than the score of an alignment between the full length strings.

**Definition 4.12.1** Let $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_m$ be two sequences over an alphabet $\Sigma$. Let $\delta$ be a score function for an alignment. A local alignment of $X$ and $Y$ is a global alignment of substrings $X' = x_{i_1} \ldots x_{i_2}$ and $Y' = y_{j_1} \ldots y_{j_2}$. An alignment $A = (X', Y')$ of substrings $X'$ and $Y'$ is an optimal local alignment of $X$ and $Y$ with respect to $\delta$ if

$$
\delta(A) = \max_{A'}\{\delta(X', Y') | X' \text{ is a substring of } X, Y' \text{ is a substring of } Y\}
$$

Example: Let $X = \text{AAAAACTCTCTCT}$ and $Y = \text{GCGCGCGCAAAAA}$. Let $s(a, a) = +1$, $s(a, b) = -1$ and $s(a, -) = s(-, a) = -2$ be a scoring function. Then an optimal local alignment

```
AAAA (CTCTCTCT)
    |||||
(GCGCGCGC)AAAAA
```

in this case has a score 5 whereas the optimal global alignment

```
AAAAACTCTCTCT
    ||
GCGCGCGCAAAAA
```

has score -11.

The Smith-Waterman algorithm is obtained by making two simple modifications to the global alignment algorithm.

1. In the main recursion, we set the value of $F(i, j)$ to zero, if all attainable values at position $(i, j)$ are negative:

$$
F(i, j) = \max \begin{cases} 
0, \\
F(i - 1, j - 1) + s(x_i, y_j), \\
F(i - 1, j) - d, \\
F(i, j - 1) - d.
\end{cases}
$$

\[ (4.13) \]
The value \( F(i, j) = 0 \) indicates that we should start a new alignment at \((i, j)\). This is because, if the best alignment up to \((i, j)\) has a negative score, then it is better to start a new one, rather than to extend the old one.

But what about the base conditions? For local alignments we need to set \( F(i, 0) = \)____ and \( F(0, j) = \)____ for all \( i = 0, 1, 2, \ldots, n \) and \( j = 0, 1, 2, \ldots, m \).

(2) Instead of starting the traceback at \((n, m)\), we start it at the cell with the highest score: \( \arg \max F(i, j) \). The traceback ends upon arrival at a cell with score 0, with corresponds to the start of the alignment.

For this algorithm to work, we require that the expected score for a random match is negative, i.e. that

\[
\sum_{a, b \in \Sigma} p_a \cdot p_b \cdot s(a, b) < 0,
\]

where \( p_a \) and \( p_b \) are the probabilities for seeing the symbol \( a \) or \( b \) respectively, at any given position. Otherwise, matrix entries will tend to be positive, producing long matches between random sequences.

Local vs. Global Alignment

The Global Alignment Problem tries to find the optimal path between vertices \((0, 0)\) and \((n, m)\) in the matrix graph.

The Local Alignment Problem tries to find the optimal path among paths between arbitrary vertices \((i, j)\) and \((i', j')\) in the matrix graph.

### 4.12.1 Example of a local alignment matrix

Smith-Waterman matrix of the sequences \text{GATTAG} and \text{ATTAC} with \( s(a, a) = 1 \), \( s(a, b) = -1 \) and \( s(a, -) = s(-, a) = -2 \):

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Score: ____;
Alignment =

### 4.12.2 Smith-Waterman algorithm

**Input:** two sequences \( X \) and \( Y \)

**Output:** optimal local alignment and score \( \alpha \)

**Initialization:** Set \( F(i, 0) := 0 \) for all \( i = 0, 1, 2, \ldots, n \)

Set \( F(0, j) := 0 \) for all \( j = 1, 2, \ldots, m \)

**For** \( i = 1, 2, \ldots, n \) **do:**

**For** \( j = 1, 2, \ldots, m \) **do:**

Set \( F(i, j) := \max \left\{ \begin{array}{l}
0 \\
F(i - 1, j - 1) + s(x_i, y_j) \\
F(i - 1, j) - d \\
F(i, j - 1) - d \\
\end{array} \right. \)

Set backtrace \( T(i, j) \) to the maximizing pair \((i', j')\)
Set \((i, j) := \text{arg max}\{ F(i, j) \mid i = 1, 2, \ldots, n, j = 1, 2, \ldots, m\}\)

The best score is \(\alpha := F(i, j)\)

repeat

\[
\begin{align*}
\text{if } T(i, j) &= (i - 1, j - 1) \text{ print } (x_{i-1}, y_{j-1}) \\
\text{else if } T(i, j) &= (i - 1, j) \text{ print } (x_{i-1}) \text{ else print } (y_{j-1}) \\
\text{Set } (i', j') &= T(i, j) \\
\end{align*}
\]

until \(F(i, j) = 0\).

### 4.13 Algorithm for finding repeated matches

The local alignment algorithm can also be used to find repeats (including overlapping repeats) within a sequence. How does one have to modify the original local alignment algorithm?

Since we want to compute a sequence \(X\) with itself, but we do not want to find \(X = X\) (which is the trivial but largest repeat) we set \(F_{0,0} = 0\) and compute only \(F_{i,j}\) with \(i < j\).

Thus the pseudo code is:

| Input: the sequence \(X\) (or input \(X\) and \(Y = X\)) of length \(n\) |
| Output: best repeat within \(X\) and score \(\alpha\) |
| Initialization: Set \(F(i, i) := 0\) for all \(i = 0, 1, 2, \ldots, n\) |
| Set \(F(0, j) := 0\) for all \(j = 1, 2, \ldots, n\) |
| For \(i = 1, 2, \ldots, n\) do: |
| For \(j = i + 1, \ldots, n\) do: |
| \(F(i, j) := \text{max} \begin{cases} 0 \\ F(i - 1, j - 1) + s(x_i, x_j) \\ F(i - 1, j) - d \\ F(i, j - 1) - d \end{cases} \) |
| Set backtrace \(T(i, j)\) to the maximizing pair \((i', j')\) |
| Set \((i, j) := \text{arg max}\{ F(i, j) \mid i = 1, 2, \ldots, n, j = i + 1, \ldots, n\}\) |
| The best score is \(\alpha := F(i, j)\) |

### 4.13.1 Example of a repeated alignment matrix

We want to find the best and second best repeat in \(ATATATAT\) using \(s(a, a) = +1\), \(s(a, b) = -1\) and \(s(a, -) = s(-, a) = -2\) for scoring parameters.

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

Best repeat is: \(ATATAT\), because optimal local alignment has highest score of +6.

Second best repeat is: \(TATAT\), because local alignment has score of +5.
4.14 Algorithm for finding overlap alignments

If we are given different fragments of genomic DNA that we would like to put together, then we need an alignment method that does not penalize overhanging ends:

For an overlap alignment, matches should be allowed to start anywhere on the top or left boundary of the matrix, and should be allowed to end anywhere on the bottom or right boundary.

To allow the former, simply set \( F(i, 0) = 0 \) and \( F(0, j) = 0 \) for \( i = 0, 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \). The recurrence relations are those used for global alignment.

To allow the latter, start the traceback at the best scoring cell contained in the bottommost row or rightmost column, i.e. start at \( \arg \max \{ F(i, j) \mid i = n \text{ or } j = m \} \).

4.14.1 Example of an overlap alignment

Given two sequences \( X = \text{T TT T A C} \) and \( Y = \text{A C A T A T T} \). Let \( s(a, a) = 1 \), \( s(a, b) = -1 \) and \( s(a, -) = s(-, a) = -2 \) for the match, mismatch and gap scores, respectively:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.15 Dynamic programming with more complex models

So far, we have considered the case of a linear gap score. This type of scoring scheme is not ideal for biological sequences, as it penalizes additional gap steps as much as the initial one. However, gaps are often longer than one residue and one would like a scheme that makes it expensive to open a gap, but once open, makes it less expensive to extend a gap.
Let $\gamma(k)$ be the score for gaps of length $k$. Then if $F(0,0) = 0$, $F(i,0) = \gamma(i)$ and $F(0,j) = \gamma(j)$

$$F(i,j) := \max \left\{ \begin{array}{l} F(i-1,j-1) + s(x_i, y_j), \\
\max_{1 \leq k \leq j} \{F(i,j-k) + \gamma(k)\}, \\
\max_{1 \leq l \leq i} \{F(i-l,j) + \gamma(l)\} \end{array} \right\}.$$  

Problem: requires $O(n^3)$ time, because for each cell we need to inspect $i + j + 1$ predecessors.

### 4.15.1 Affine gap scores

The standard alternative to using the above recursion is to use an affine gap score

$$\gamma(g) = -d - (g - 1)e,$$

with $d$ the gap-open score and $e$ the gap-extension score.

We will discuss how to modify the Needleman-Wunsch algorithm for global alignment so as to incorporate affine gap costs. Approach is due to Osamu Gotoh (1982)\(^4\)

Instead of using one matrix $F(i,j)$ to represent the best score attainable up to $x_i$ and $y_j$, we will now use three matrices $M$, $I_x$, and $I_y$:

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$ aligns to $y_j$:</td>
<td>$x_i$ aligns to a gap:</td>
<td>$y_j$ aligns to a gap:</td>
</tr>
<tr>
<td>I G A $x_i$</td>
<td>A I G A $x_i$</td>
<td>S G A $x_i$ -</td>
</tr>
<tr>
<td>L G V $y_j$</td>
<td>A G V $y_j$ -</td>
<td>S L G V $y_j$</td>
</tr>
</tbody>
</table>

1. $M(i,j)$ is the best score up to $(i,j)$, given that $x_i$ is aligned to $y_j$,
2. $I_x(i,j)$ is the best score up to $(i,j)$, given that $x_i$ is aligned to a gap, and
3. $I_y(i,j)$ is the best score up to $(i,j)$, given that $y_j$ is aligned to a gap.

### 4.15.2 Recursion for global alignment with affine gap costs

Initialization:

- $M(0,0) = 0$, $I_x(0,0) = I_y(0,0) = -\infty$
- $I_x(i,0) = -d - (i - 1)e$, $M(i,0) = I_y(i,0) = -\infty$, for $i = 1, \ldots, n$, and
- $I_y(0,j) = -d - (j - 1)e$, $M(0,j) = I_x(0,j) = -\infty$, for $j = 1, \ldots, m$.

Recursion:

\[
I_x(i,j) = \max \left\{ \begin{array}{l} M(i-1,j) - d, \\
I_x(i-1,j) - e; \end{array} \right\} \begin{cases} \text{begin gap in X} \\
\text{continue gap in X} \end{cases}
\]

\[
I_y(i,j) = \max \left\{ \begin{array}{l} M(i,j-1) - d, \\
I_y(i,j-1) - e; \end{array} \right\} \begin{cases} \text{begin gap in Y} \\
\text{continue gap in Y} \end{cases}
\]

\[
M(i,j) = \max \left\{ \begin{array}{l} M(i-1,j-1) + s(x_i, y_j), \\
I_x(i-1,j-1) + s(x_i, y_j), \\
I_y(i-1,j-1) + s(x_i, y_j), \end{array} \right\} \begin{cases} \text{match or mismatch} \\
\text{end gaps in X} \\
\text{end gaps in Y} \end{cases}
\]

We make the assumption that a gap in one sequence is not immediately followed by a gap in the other. This is true for the optimal path, if $-d - e$ is less than the lowest mismatch score.

4.15.3 Example of a global alignment with affine gap costs

Given two sequences \( X = \text{TTG} \) and \( Y = \text{TTAGAT} \). We use \( s(a,a) = 1, s(a,b) = -1, \gamma(g) = -d - (g-1)e \) with \( d = 4 \) and \( e = 1 \) for the match, mismatch, gap-open and gap-extension scores, respectively:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( M )</td>
<td>( I_x )</td>
<td>( I_y )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>( M )</td>
<td>( I_x )</td>
<td>( I_y )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>( M )</td>
<td>( I_x )</td>
<td>( I_y )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>( M )</td>
<td>( I_x )</td>
<td>( I_y )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are two global optimal alignments. They are?

4.15.4 Simplifying the affine-gap algorithm

In practice, the affine-gap formulation of the dynamic programs for sequence alignment usually uses only two matrices, \( M \) and \( I \), corresponding to an alignment of two symbols and an insertion in one of the two sequences, respectively. For global alignment, the recursions are:

\[
M(i, j) = \max \left\{ M(i-1, j-1) + s(x_i, y_j), I(i-1, j-1) + s(x_i, y_j) \right\},
\]

\[
I(i, j) = \begin{cases} 
M(i-1, j) - d, \\
I(i-1, j) - e, \\
M(i, j-1) - d, \\
I(i, j-1) - e. 
\end{cases}
\]

Exercise: how does one have to initialize the matrices?

This produces the same result as the original algorithm, if the lowest mismatch score is \( > -2e \). But, even if this does not hold, then the difference in score and alignment will be insignificant (in a poorly matched gapped region).

Assume that the original algorithm (using \( M, I_x \) and \( I_y \)) produces the following optimal alignment:

\[
\begin{array}{ccccccccccccccc}
\times & \times & \times & x & - & - & a & x & x & x & x & x & x & x \\
y & y & y & y & y & y & y & b & - & - & - & y & y & y & y
\end{array}
\]

If \( s(a, b) < -2e \), then the modified algorithm (using \( M \) and \( I \)) will produce the following higher scoring alignment, adding a gap before \( a \) and one after \( b \):

\[
\begin{array}{ccccccccccccccc}
\times & \times & \times & x & - & - & - & a & x & x & x & x & x & x & x \\
y & y & y & y & y & y & y & b & - & - & - & y & y & y & y
\end{array}
\]

However, situations in which \( (\frac{-b}{a}) \) is directly followed by \( (\frac{a}{b}) \) (or vice-versa) are uninteresting and so the original algorithm rules them out.
4.16 Alignment in linear space

Can we compute a best alignment between two sequences
$X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ using only linear space?

The best score of an alignment is easy to compute in linear space, as $F(i, j)$ is computed locally from
the values in the previous and current column only.

However, to obtain an actual alignment in linear space, we need to replace the traceback matrix.

We will discuss this for the case of global alignments.

**Idea:** Divide-and-conquer! Consider the middle column $u = \lfloor \frac{n}{2} \rfloor$ of the $F$ matrix.

If we knew at which cell $(v, u)$ the best scoring alignment passes through the $u^{th}$ column, then we
could split the dynamic-programming problem into two parts:

- align from $(0,0)$ to $(v,u)$, and then
- align from $(v,u)$ to $(m,n)$.

If we knew at which cell $(v,u)$ the best scoring alignment passes through the $u^{th}$ column, then we
could split the dynamic-programming problem into two parts:

<table>
<thead>
<tr>
<th>$Y \setminus X$</th>
<th>0</th>
<th>1</th>
<th>$\cdots$</th>
<th>$u$</th>
<th>$\cdots$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$v$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$(v,u)$</td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Concatenate the two solutions to obtain the final result.

How to determine $v$, the row in which a best path crosses the $u^{th}$ column? The key point of this
algorithm is, that we can find this row without actually knowing the optimal full path through the
whole matrix. Define $l(i)$ to be the score of the path from $(0,0)$ to $(m,n)$ that passes through the
vertex $(i, \lfloor \frac{n}{2} \rfloor)$. The cell $(i, \lfloor \frac{n}{2} \rfloor)$ splits the $l(i)$-path into two subpaths. The first runs from $(0,0)$ to
$(i, \lfloor \frac{n}{2} \rfloor)$, the second runs from $(i, \lfloor \frac{n}{2} \rfloor)$ to $(m,n)$. Then $l(i)$ is equal to the sum of the scores of both
subpaths. Clearly the score of the first subpath is equal to $F(i, \lfloor \frac{n}{2} \rfloor)$. The score of the second path is
equal to $F^R(i, \lfloor \frac{n}{2} \rfloor)$ of the path from $(m,n)$ to $(i, \lfloor \frac{n}{2} \rfloor)$ in the reverse matrix.

Then clearly $v = \arg \max_{0 \leq i \leq m} l(i)$.

Note that since the $l(i)$ values are determined through the $F(i,j)$ values, computing all $l(i)$ values
takes $O(m)$ space.

Once we have determined $(v,u)$, we recurse, as indicated here:
We obtain the actual alignment as a sequence of pairs \((v_1, 1), (v_2, 2), \ldots, (v_n, n)\).

What is the time complexity? We first look at \(1 \times nm\) cells, then at \(\frac{1}{2}nm\) cells, then at \(\frac{1}{4}nm\) cells etc. As \(\sum_{i=0}^{n} \frac{1}{2^i} < 2\), this algorithm is only twice as slow as the quadratic-space one!

### 4.17 Where do we stand?

So far, we have discussed:

- the dot matrix for visual comparison of two sequences,
- global alignments and the Needleman-Wunsch algorithm,
- local alignments and the Smith-Waterman algorithm, and
- repeat and overlap alignments and their corresponding dynamic programs.

An implementation of any of the dynamic programming algorithms uses \(O(nm)\) time and \(O(nm)\) space. We saw that:

- the space complexity can be reduced to \(O(n)\).

Now we ask:

- can we reduce the time complexity, too?

### 4.18 Banded global alignment

For simplicity, we consider DNA sequences, assume \(n = m\) and use a linear gap score \(d\).

**Idea:** Instead of computing the whole matrix \(F\), use only a *band* of cells along the main diagonal:

\[
\begin{array}{cccccccc}
\hline
i \backslash j & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
5 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
6 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\end{array}
\]

Let \(2k\) denote the height of the band. Obviously, the time complexity of the banded algorithm will be \(O(kn)\).

**Questions:** Will this algorithm produce an optimal global alignment? What should \(k\) be set to?
4.18.1 The KBand algorithm

**Input:** two sequences $X$ and $Y$ of equal length $n$, integer $k$

**Output:** best score $\alpha$ of global alignment at most $k$ diagonals away from main diagonal

**Initialization:** Set $F(i, 0) := -i \cdot d$ for all $i = 0, 1, 2, \ldots, k$.  
Set $F(0, j) := -j \cdot d$ for all $j = 1, 2, \ldots, k$.

for $i = 1$ to $n$ do
  for $h = -k$ to $k$ do
    $j := i + h$
    if $1 \leq j \leq n$ then
      $F(i, j) := F(i - 1, j - 1) + s(x_i, y_j)$
      if $\text{insideBand}(i - 1, j, k)$ then
        $F(i, j) := \max\{F(i, j), F(i - 1, j) - d\}$
      if $\text{insideBand}(i, j - 1, k)$ then
        $F(i, j) := \max\{F(i, j), F(i, j - 1) - d\}$
  return $F(n, n)$

To test whether $(i, j)$ is inside the band, we use:

$$\text{insideBand}(i, j, k) := (-k \leq i - j \leq k).$$

4.18.2 Optimal alignments using KBand

Given two sequences $X$ and $Y$ of the same length $n$. Let $M$ be the match score and $d$ the gap penalty.

**Question:** Let $\alpha_k$ be the best score obtained using the KBand algorithm for a given $k$. When is $\alpha_k$ equal to the optimal global alignment score $\alpha$?

**Lemma 4.18.1** If $\alpha_k \geq M(n - k - 1) - 2(k + 1)d$, then $\alpha_k = \alpha$.

**Proof** If there exists an optimal alignment with score $\alpha$ that does not leave the band, then clearly $\alpha_k = \alpha$. Else, all optimal alignments leave the band somewhere. This requires insertion of at least $k + 1$ gaps in each sequence, and allows only at most $n - k - 1$ matches, giving the desired bound. \(\square\)

The following algorithm computes an optimal alignment by repeated application of the KBand algorithm, with larger and larger $k$:

**Input:** two sequences $X$ and $Y$ of the same length $n$

**Output:** an optimal global alignment of $x$ and $y$

Initialize $k := 1$
repeat
  compute $\alpha_k$ using KBand
  if $\alpha_k \geq M(n - k - 1) - 2(k + 1)d$ then
    return $\alpha_k$
  $k := 2k$
end

As usual, we omit details of the traceback.
4.18.3 Analysis of time complexity

The algorithm terminates when:
\[
\alpha_k \geq M(n - k - 1) - 2(k + 1)d \\
\alpha_k - Mn + M + 2d \geq -(M + 2d)k \\
-\alpha_k + Mn - (M + 2d) \leq (M + 2d)k \\
\frac{Mn - \alpha_k}{M + 2d} - 1 \leq k
\]

At this point, the total complexity is:
\[
n + 2n + 4n + \ldots + kn \leq 2kn.
\]

So far, this doesn’t look better than \(nn\). To bound the total complexity, we need a bound on \(k\).

When the algorithm stops for \(k\), we must have:
\[
\frac{k}{2} < \frac{Mn - \alpha_{k/2}}{M + 2d} - 1.
\]

There are two cases: If \(\alpha_{k/2} = \alpha_k = \alpha\), then
\[
k < 2\left(\frac{Mn - \alpha}{M + 2d} - 1\right).
\]

Otherwise, \(\alpha_{k/2} < \alpha_k = \alpha\). Then any optimal alignment must have more than \(\frac{k}{2}\) spaces, and thus
\[
\alpha \leq M(n - \frac{k}{2} - 1) + 2\left(\frac{k}{2} + 1\right)d \quad \Rightarrow \quad k \leq 2\left(\frac{Mn - \alpha}{M + 2d} - 1\right).
\]

As \(M + 2d\) is a constant, it follows that \(k\) is bounded by \(O(\Delta)\), with \(\Delta = Mn - \alpha\), and thus the total bound is \(O(\Delta n)\).

In consequence, the more similar the sequences, the faster the KBand algorithm will run!

4.18.4 Searching for high-identity alignments

We can use the KBand algorithm as a fast method for finding high-identity alignments:

If we know that the two input sequences are highly similar and we have a bound \(b\) on the number of gaps that will occur in the best alignment, then the KBand algorithm with \(k = b\) will compute an optimal alignment.

For example, in forensics, one must sometimes determine whether a sample of human mtDNA obtained from a victim matches a sample obtained from a relative (or from a hair brush etc). If two such sequences differ by more than a couple of base-pairs or gaps, then they are not considered a match.

4.19 Historical Note

Saul Needleman and Christian Wunsch were the first to publish a dynamic programming algorithm for DNA sequence comparison. That was as early as 1970. However, a very similar algorithm had been published in 1968 by the Russian Vintsyuk on speech recognition. Yet again, two years earlier, 1966, Vladimir Levenshtein introduced the notion of edit distance, which we today also coin the Levenshtein distance. The original paper by NW presented a cubic algorithm, and it was David Sankoff who reduced the complexity to a squared algorithm. Nowadays we always learn the Sankoff-variant of the original NW algorithm.