Pairwise Global Alignment of Sequences

Comparing sequences, structures (and sequences with structures) is the most fundamental operation in protein sequence and structure analysis. When a comparison indicates a similarity between two proteins, it can immediately suggest relationships involving structure, function and the evolution of the two proteins from a common ancestor protein. When one of the proteins is well characterized (in terms of structure and function), the connection with a novel sequence allows all the hard-earned biological data to be transferred to the new protein. The degree of certainty with which this transfer can be made depends on how similar the two sequences are, but even for distant relationships it is likely that the overall structure of the two proteins (their fold) will have remained the same and even tentative suggestions of function can be used as a basis to suggest further experiments on the novel protein.

The comparison of two proteins is mostly made by trying to align the sequences (structures or sequences/structures). In making an alignment, a 1:1 correspondence is set up between the residues of the two proteins. This has the evolutionary implication that at one time the paired residues were the same in an ancestral protein and have diverged through the accumulation of point mutations (in their DNA). Point mutation is not the only process at work and extra residues may have been inserted or deleted giving rise to breaks or gaps in the alignment. These are referred to as insertions and deletions or, jointly, as indels. The simplest operation to explain is the global alignment of two sequences, in which the two proteins have maintained a correspondence over their entire length. An alternative is to align only the most similar part of the proteins, which is called local alignment and which will be considered in the next chapter.

In this chapter, we describe the basic algorithm for making an alignment (called dynamic programming) before considering more specialized comparison methods in later chapters. The basic dynamic programming algorithm will recur throughout these and other chapters and is perhaps the most widely used and important algorithm in bioinformatics. Variations of it are used for local alignment and it can be extended to align more that two sequences (multiple alignment). In later chapters we will also
describe how it has been adapted to compare two structures and for the hybrid task of comparing a sequence with a structure.

1.1 Alignment and Evolution

An evolutionary perspective is important for getting an understanding of the function of proteins. That means, given two proteins, one often wants to find the evolutionary relationship between them. When only the sequences of the proteins are known, one attempts to reveal the relationship by aligning the sequences. The alignment should therefore show the mutations that have happened in the evolution of the two sequences.

Example

Let \( h = \text{GLVST} \) be the ancestor of two sequences \( q = \text{GLISVT} \) and \( d = \text{GIVT} \). Assume that the evolution is as shown in Figure 1.1, where \( a \rightarrow b \) means substitution from \( a \) to \( b \), \( a \rightarrow \) means deletion of \( a \), and \( \rightarrow a \) means insertion of \( a \).

An alignment should show the corresponding positions of \( q \) and \( d \), and where insertions and deletions have occurred. Thus, the ‘true’ alignment can be found by using \( h \) as a template,

\[
\begin{align*}
\text{h: } & \quad \text{GLVS T} \\
\text{q': } & \quad \text{GLISVT} \\
\text{d': } & \quad \text{GIV--T}
\end{align*}
\]

where ‘-’ (denoted by blank) means deletion or insertion (indels). One or several contiguous blanks are called a gap. \( d' \) is \( d \) with possible insertions of gaps. \( \triangle \)

When the evolutionary history is not known (and \( h \) is not known), a given alignment can be interpreted in different ways. If we assume that only single mutations have happened (only one residue change in each mutation), the alignment between \( q \) and \( d \) in the example can be interpreted as two substitutions, and either two insertions,
two deletions, or one deletion and one insertion. That means, even by knowing the
correct alignment, we are not able to reconstruct for certain the evolutionary history,
the true one is only one of several possibilities.

When trying to reconstruct the evolution, one needs to have a model, telling how to
construct the tree from an alignment. One such model can be to not introduce more
mutations than necessary, resulting in the following relation between \( q \) and \( d \):

\[
q = \text{GLISVT}: \ I \leftrightarrow L; V \leftrightarrow I; \leftrightarrow S \rightarrow; \leftarrow V \rightarrow : \ d = \text{GIVT}.
\]

The new symbols are introduced to show that we do not know the direction of the
substitutions, and for each blank we do not know whether an insertion or a deletion
has happened. Several histories can be constructed from this relation (where the true
history is one of those), for example,

\[
q = \text{GLISVT}: \ L \rightarrow I; I \rightarrow V; S \rightarrow; V \rightarrow : \ d = \text{GIVT}.
\]

meaning that \( q \) is an ancestor of \( d \).

When only the sequences are known, it is even more difficult to reconstruct the true
evolutionary history. First, one can try to align them, and then construct the history
from the constructed alignment. For constructing an alignment we again need a model,
and the same simple model can be used: try to minimize the number of mutations.
An alignment of \( q \) and \( d \) in accordance with this would be

\[
q' = \text{GLISVT} \quad d' = \text{G-I-VT}
\]

showing two indels. One history could be

\[
\text{h*}: \text{GLIVT} \\
/ \backslash \\
-\rightarrow S/ \backslash L-\rightarrow \\
/ \backslash
\]

\[
q: \text{GLISVT} \quad d: \text{GIVT}
\]

Since \( h^* \) is not the same as the true \( h \), using our model with this example does not
give us the true evolutionary history from the alignment only. Despite this drawback,
and in the absence of a better alternative that is not too complicated, we will often use
this model, since it is so simple. It should be mentioned that constructing alignments
for predicting evolution is only meaningful for sequences of homologous proteins,
i.e. proteins with a common ancestor. But whether the sequences are homologous or
not is not often known, and in this context we see an important aspect of making
alignments: to assess if homology exists. Being able to construct a ‘good’ alignment
can indicate homology. Homology can then be used to predict the structure and/or the
function of proteins for which those are not known, since two homologous proteins
often have similar structures. This is one motivation for database searching: given a
query sequence \( q \), find the sequences in a database \( D \) which make ‘good’ alignments
to \( q \). This is treated in Chapter 2.
1.2 What is an Alignment?

An alignment of two sequences \( q \) and \( d \) must satisfy the following constraints

- All symbols (residues) in \( q \) and \( d \) have to be in the alignment, and in the same order as they appear in \( q \) and \( d \).
- We can align one symbol from \( q \) with one from \( d \).
- A symbol can be aligned with a blank, written as ‘-’.
- Two blanks cannot be aligned.

Example

A possible alignment of the insulin proteins from sheep and zebrafish is

Fish: \text{MAVWLQAGALLLLV-LV-SSVSTNPGTPQLCGSLVLDAIYLVC Pert GFFYNPQ--R}
Sheep: \text{MALWTRLVPLLALLALLWAPAPAHAFVNQHLC GSHLVEALYLVCSERGFFYTPKARR}

Fish: \text{DVE-PLLGLPKSAQETEVEADF VAKDHAELIRKGRIVEQCHKPCSIFELQNYCN}
Sheep: \text{EVEGPQVGAL--ELAGGFG-AG-GL-EGPP-Q-KR GI EQCCAGCVS L YQLEN YCN}

1.3 A Scoring Scheme for the Model

From our simple model for constructing alignments, we can define a scoring scheme for scoring the alignments.

- Each column can be given a score, independent of the other columns, meaning that we think of all mutations as single mutations:
  - the score of a column with two amino acids \( a, b \) is denoted by \( R_{ab} \);
  - the score of a column with blank can be \(-g\), where \( g \) is called the penalty of a blank.
- The score of the alignment can be found as the sum of the score of all columns (additive scoring scheme).

Note the correspondence between score and penalty of a column with blank, the score is the negative of the penalty.

Example

Let a scoring scheme be

- \( R_{ab} = 1 \) for \( a = b \), 0 for \( a \neq b \);
- \( g = 1 \).

Then the score of some different alignments of the same sequences are
PAIRWISE GLOBAL ALIGNMENT OF SEQUENCES

ALIGN1:
q': V E I T G E I S T
d': P R E - T E R I - T
  0 -1 1 -1 1 0 0 1 -1 1 Score 1

ALIGN2:
q': V E I T G E I S T
d': P R E T - E R I T
  0 0 0 1 -1 1 0 0 1 Score 2

ALIGN3:
q': - V E I T G E - I S T
d': P R E - T - E R I - T
  -1 0 1 -1 1 -1 1 -1 1 -1 1 Score 0

\[ \triangle \]

Note that which alignment will score highest depends on the scoring scheme used. Hence, finding the highest-scoring alignment does not necessarily mean the ‘best alignment’, if the scoring scheme is bad. Therefore, choosing which scoring scheme to use is an important and difficult task. Also note that there may be more than one alignment with the maximum score.

1.4 Finding Highest-Scoring Alignments with Dynamic Programming

We now realize that, even for small sequences, there exists a large number of possible alignments, and it is impractical to generate all of them and calculate their scores in order to find the best. Fortunately, there exists a method which in an efficient way can be used to find the best alignment, for a given scoring scheme.

This method is based on a general programming paradigm, called dynamic programming. The main idea is that results found early in the solution procedure are used in later calculations. This paradigm was first used for biosequences by Needleman and Wunsch (1970). The task of finding the highest-scoring alignment(s) is done in two steps.

1. Using dynamic programming, find the highest possible score.

2. Find (one, several or all) alignments achieving the highest score by using the intermediate results from the first step.

To explain the method we introduce some notation.

- We have sequence \( q \) of length \( m \), and sequence \( d \) of length \( n \). For example, \( q = \text{VEITGEIST} \ (m = 9) \), \( d = \text{PRETERIT} \ (n = 8) \).
- \( q_i \) is the \( i \)th symbol of \( q \), \( d_j \) is \( j \)th symbol of \( d \).
• $q_{1...i}$ is the sequence of the first $i$ symbols of $q$. For example, $q_{1...0} = \varepsilon$ (the empty sequence), $q_{1...1} = \forall$, $q_{1...4} = \forall \varepsilon I T$, $q_{1...m} = q$.

• $d_{1...j}$ is the sequence of the first $j$ symbols of $d$.

• $R_{ab}$ is the scoring between $a, b$.

• $H_{i,j}$ is the highest score which can be achieved by aligning $q_{1...i}, d_{1...j}$.

• $g$ is the penalty for a blank.

Note that $H_{m,n}$ will be the highest score which can be achieved by aligning $q$ and $d$.

Use of the dynamic programming paradigm here implies that we can determine $H_{i,j}$ by using one or more of $H_{k,l}$, $0 \leq k \leq i$, $0 \leq l \leq j$. This means that $H_{m,n}$ can be found by first finding some $H_{i,j}$ for $i \leq m$, $j \leq n$. This calculation can be done in a systematic way, as will be described in the following subsections.

### 1.4.1 Determine $H_{i,j}$

The alignment for $(q_{1...i}, d_{1...j})$ can only end with one of three different columns:

\[
\begin{array}{ccc}
q_i & \rightarrow & q_i \\
\rightarrow & d_j & \rightarrow \\
\end{array}
\]

We will find an expression for $H_{i,j}$ by regarding each of these cases, and from that determine the correct value for $H_{i,j}$.

We use $i = 3$ and $j = 4$ as an example in the explanation, $(q_{1...i} = \forall \varepsilon I T, d_{1...j} = \text{PRE} T)$. Assume we know $H_{i-1,j}, H_{i,j-1}, H_{i-1,j-1}$.

1. The alignment ends with $(q_i, \rightarrow)$. For the example it is $(\forall, \rightarrow)$. The alignment is then

\[
\begin{array}{ccc}
q'_1...i-1 & \rightarrow & q_i \\
d'_1...j & \rightarrow & d_j \\
\end{array}
\]

for the example it may be

\[
\begin{array}{ccc}
\forall & \rightarrow & \varepsilon - I \\
\rightarrow & \text{PRE} & - \\
\end{array}
\]

Since we have additive scoring, we see that we must add the penalty for blank to the score of aligning $q_{1...i-1}, d_{1...j}$, which is $H_{i-1,j}$, hence, $H_{i,j}^{(1)} = H_{i-1,j} - g$ ($g = 1$ in the example).

2. The alignment ends with $(\rightarrow, d_j)$ $(\rightarrow, \top)$. By using the same explanation as above we find the alignment to be

\[
\begin{array}{ccc}
q'_1...i & \rightarrow & \top \\
d'_1...j-1 & \rightarrow & d_j \\
\end{array}
\]

for the example it may be

\[
\begin{array}{ccc}
\forall \varepsilon I T & \rightarrow & \top \\
\rightarrow & \text{PRE} & \top \\
\end{array}
\]

$H_{i,j}^{(2)} = H_{i,j-1} - g$. 

3. The alignment ends with \((q_i, d_j)\). The alignment is then

\[
\begin{array}{c|c}
q_i'_{1..i-1} & q_i \\
\hline
d_j'_{1..j-1} & d_j \\
\end{array}
\]

for the example it may be

\[
\begin{array}{c|c}
\text{V-E} & \text{I} \\
\hline
\text{PRE} & \text{T} \\
\end{array}
\]

\[
H^{(3)}_{i,j} = H_{i-1,j-1} + R_{q_id_j} (R_{\text{IT}} = 0 \text{ in the example}).
\]

We then have three alternatives for aligning \(q_{1..i}, d_{1..j}\), depending on the last column. We choose one with highest score, such that the value for \(H_{i,j}\) becomes

\[
H_{i,j} = \max\{H_{i-1,j} - g, H_{i,j-1} - g, H_{i-1,j-1} + R_{q_id_j}\}.
\]

Note that for this to be correct the scoring scheme must be additive, and that each blank must have the same score (linear scoring). Then \(H_{m,n}\) will get the score of the best alignment of \((q, d)\).

### 1.4.2 Use of matrices

To help in the aligning process, it is appropriate to arrange the scores \(H_{i,j}\) in a two-dimensional matrix of size \((m + 1) \cdot (n + 1)\) as shown in Figure 1.2. The arrows show which earlier filled cells are used for calculating the value of a cell, \(H_{i,j}\) (i = 3, j = 4 in the example).

We see that the matrix can be filled in row by row from the upper left corner down to the bottom right corner. However, we must have start values, otherwise, for example, \(H_{1,1}\) cannot be calculated. Therefore, we have to initialize the values in row and column 0. \(H_{0,j}\) is highest score for aligning \(q_{1..0}, d_{1..j}\), which means aligning the empty sequence \((\varepsilon)\) to \(d_{1..j}\). This is done by expanding \(\varepsilon\) with \(j\) blanks, such that, for example, the alignment of \(q_{1..0}, d_{1..3}\) becomes

\[
---
\]

\[
\text{PRE}
\]

Each blank has a score \(-g\), meaning that \(H_{0,j} = -jg\). Figure 1.2 shows the initialized matrix when \(g = 1\).

Now the rest of the values can be filled in, to \(H_{m,n}\), as shown in Figure 1.3(a), where our simple scoring scheme is used. The arrows show which neighbour cells are used for getting the maximum score of a cell. Note that in some cases, two of the neighbour cells will give the maximum value. Generally, it could happen that all three neighbour cells would lead to the maximum value. Note that in the figure not all arrows are drawn.

**Example**

Let us find the value for

\[
H_{8,7} = \max\{H_{8,6} - g, H_{7,7} - g, H_{7,6} + R_{q_8d_7}\} = \max\{0 - 1, 2 - 1, 1 + 0\} = 1.
\]

We see that the maximum value (1) is found from both \(H_{7,7}\) and \(H_{7,6}\). Therefore, there are two arrows to \(H_{8,7}\). \(\triangle\)
Algorithm 1.1 shows the dynamic programming procedure for global alignment.

**Algorithm 1.1. Dynamic programming for global alignment.**

Aligning sequences \( q \) and \( d \) of length \( m \) and \( n \), respectively, with linear gap penalty

```plaintext
const.
g, penalty for one blank
R_{ab} the score of aligning a and b
var
H the dynamic programming matrix
begin
  for \( i : = 0 \) to \( m \) do \( H_{0,i} := -ig \) end /* initialize */
  for \( j : = 1 \) to \( n \) do \( H_{j,0} := -jg \) end
  for \( i : = 1 \) to \( m \) do
    for \( j : = 1 \) to \( n \) do
      \( H_{i,j} := \max\{H_{i-1,j} - g, H_{i,j-1} - g, H_{i-1,j-1} + R_{q_i,d_j}\} \)
    end
  end
end
```

It follows directly that the time complexity of the algorithm is \( O(mn) \), the number of cells. The space complexity is the same, but by means of a more complex storage administration one can achieve linear space (see Bibliographic notes).

One can give a formal inductive proof that the algorithm above does find the maximum alignment score for a pair of sequences. The proof can be based on the fact that the scoring of a column is independent on how the other parts of the sequences are aligned.

### 1.4.3 Finding the alignments that give the highest score

The arrows constitute paths in the matrix, and for finding the highest-scoring alignments, we can follow the paths from \( H_{m,n} \) backwards to \( H_{0,0} \). The arrows to follow for the example are shown in Figure 1.3(b).

From the arrows we can find the corresponding columns of the alignment. We remember that if the arrow comes from either the same row or the same column, a blank is introduced when extending the alignment to include \((q_i, d_j)\), which means that the corresponding columns should contain a blank. Using those rules (Section 1.4.1), we find the column corresponding to cell \( H_{i,j} \) as follows:

- if the arrow comes from \( H_{i-1,j} \), the column is \((q_i, -)\);
- if the arrow comes from \( H_{i,j-1} \), the column is \((- d_j)\);
- if the arrow comes from \( H_{i-1,j-1} \), the column is \((q_i, d_j)\).
PAIRWISE GLOBAL ALIGNMENT OF SEQUENCES

<table>
<thead>
<tr>
<th>q^d</th>
<th>P</th>
<th>R</th>
<th>E</th>
<th>T</th>
<th>E</th>
<th>R</th>
<th>I</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>i,j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
</tr>
<tr>
<td>V</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>-2</td>
<td>H_{i-1,j-1}</td>
<td>H_{i-1,j}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td>-3</td>
<td>H_{i,j-1}</td>
<td>H_{i,j}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>8</td>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.2 The dynamic programming matrix for the example sequences, and how the values of the cells are calculated. Row and column 0 are initialized for the score of a blank equal to −1. To calculate the value of $H_{i,j}$ one needs the values of $H_{i,j-1}$, $H_{i-1,j}$ and $H_{i-1,j-1}$.

Example

Let us find the column for $i = 8$, $j = 7$. Two arrows are coming in, from $i = 7$, $j = 6$ and from $i = 7$, $j = 7$. Backtracking to $H_{7,6}$ means that we go one position back in both sequences, hence the column becomes

```
S
I
```

Backtracking to $H_{7,7}$ means that we go one position back in $q$ and none in $d$, hence the column becomes

```
S
-  
```

$\Delta$
Several alignments can give the highest score; in our example it is two (note that
the alignments are found inverted of how it is presented):

\[
\begin{align*}
q' : & \text{V E I T G E I S T} \\
d' : & \text{P R E T E R I T} \\
0 & 0 0 1 -1 1 0 0 1 \quad \text{Score 2}
\end{align*}
\]

\[
\begin{align*}
q' : & \text{V E I T G E I S T} \\
d' : & \text{P R E T E R I - T} \\
0 & 0 0 1 0 0 1 -1 1 \quad \text{Score 2}
\end{align*}
\]

In programs, the arrows can be represented by variables. An alternative is to not
store this direction information in the forward process, but calculate the direction in
the backward process. This is done in Algorithm 1.2.

**Algorithm 1.2. Backtracking for the best global alignments.**
The best alignments are stored in B, one at the time

**proc** backtrack\((i, j, B, k)\) recursive procedure
called the first time as backtrack\((m, n, B, 1)\)

**const**
\(R_{ab}\) the scoring matrix
\(g\) penalty of one blank, linear gap penalty

**var**
\(B\) the alignment is filled in table B in reversed order,
q’ in row 1, d’ in row 2.
\(k\) column in B
\(i, j\) indices for q and d

**begin**
if \(i = 0\) then \(gaps\) at the beginning of q’
while \(j > 0\) do\(B_{1,k} = ' -' ; B_{2,k} = d_j ; k := k + 1; j := j - 1\) end
write(B) \(one\) best alignment found
elseif \(j = 0\) then \(gaps\) at the beginning of d’
while \(i > 0\) do\(B_{1,k} = q_i ; B_{2,k} = ' -' ; k := k + 1; i := i - 1\) end
write(B)
else
if \(H_{i,j} = H_{i-1,j} - g\) then
\(B_{1,k} = q_i ; B_{2,k} = ' -' ; backtrack(i - 1, j, B, k + 1)\) end
if \(H_{i,j} = H_{i,j-1} - g\) then
\(B_{1,k} = ' -' ; B_{2,k} = d_j ; backtrack(i, j - 1, B, k + 1)\) end
if \(H_{i,j} = H_{i-1,j-1} + R_{q_idi}\) then
\(B_{1,k} = q_i ; B_{2,k} = d_j ; backtrack(i - 1, j - 1, B, k + 1)\) end
end
end
### 1.4.4 Gaps

In the example there are only single blanks. Generally, more blanks might follow each other for getting the best alignment. One or more following blanks is called a *gap*. Also, there might be several gaps in an alignment.

**Example**

An example of an alignment with more gaps is

$$AC--GRTV$$  
$$ACMTG--TV$$

### 1.5 Scoring Matrices

The scoring used in Section 1.4 is too simple to be used when aligning real protein or DNA sequences. The main issue of a database search is to find sequences homologous to a query sequence, and this is done by scoring a similarity between the query and the database sequences. Hence, a scoring scheme should be based on the similarity of the residues occurring in the sequences. For two residues \((q_i, d_j)\), we need a measure of the probability (or likelihood) that they have a common ancestor, or that one is a result of one or several mutations of the other. The position of the residues is ignored, and a general measure for the similarity of the occurring amino acids is used. This

![Figure 1.3](image-url)
measure can then be given as a \( l \times l \) scoring matrix, where \( l \) is the number of amino acids. If we claim that the scoring for \( a \rightarrow b \) should be equal to the scoring for \( b \rightarrow a \) (a reasonable claim), the scoring matrices must be symmetrical, hence a triangular \( l \times l \) matrix is sufficient.

The most common scoring matrices are the PAM and BLOSUM series. Those are developed based on observed mutations in the nature, and are explained in Chapter 5. Figure 1.1 shows one of the PAM matrices. Note the great variation in the scoring values, and that both positive and negative values occur. Note also that the score of aligning equal amino acids vary, aligning A with A scores 2, but aligning W with W scores 17.

**Example**
The score of the columns for one of the alignments found in Section 1.4 is by use of the 250 PAM matrix:

\[
\begin{array}{cccccccc}
q' & V & E & I & T & G & E & I & S & T \\
d' & P & R & E & T & - & E & R & I & T \\
\end{array}
\]

\[
\begin{array}{cccccccc}
-1 & -1 & -2 & 3 & 4 & -2 & -1 & 3 \\
\end{array}
\]

The score for gap is not specified here, but is discussed in the following section. △

## 1.6 Scoring Gaps: Gap Penalties

Deciding how to score gaps is perhaps the most difficult task in performing sequence alignments. Usually, a local form of gap penalty is used, which means that the penalty of a gap is found independently of other gaps in the alignment. Here we only treat local gap penalties.

Scoring gaps should mirror the model we use for constructing alignments. A gap might be the result of one or several mutations (insertions or deletions). Below we will assume that a gap has occurred by a single mutation.

**Example**
Let an alignment be

\[
\begin{array}{cccccccc}
A & S & D & E & D & F & G & H \\
A & S & - & - & - & - & G & H \\
\end{array}
\]

We assume that the deletion (or insertion, if the evolution has gone the other way) of the four amino acids has happened in one mutation. Another way of modelling, for example, could allow two mutations, first deletion of DE and then of DF. △

Following our model, the penalty for a gap of length four should be less than or equal to the penalty of, for example, two gaps each of length two. This can be formulated generally as a constraint on the penalty \( g_l \) for a gap of length \( l \):

\[
\forall r : 0 < r < l : g_l \leq g_{l-r} + g_r. \quad (1.1)
\]
### Table 1.1
Scoring matrix for the evolutionary distance of 250 PAM, rounded to one digit.
The amino acids occur in alphabetic order of their full names.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>R</th>
<th>N</th>
<th>D</th>
<th>C</th>
<th>Q</th>
<th>E</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>L</th>
<th>K</th>
<th>M</th>
<th>F</th>
<th>P</th>
<th>S</th>
<th>T</th>
<th>W</th>
<th>Y</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>-2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td>-4</td>
<td>-5</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>-5</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>-3</td>
<td>0</td>
<td>1</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>-1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>-6</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-2</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>-1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-5</td>
<td>1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>-5</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>-4</td>
<td>-5</td>
<td>-4</td>
<td>-6</td>
<td>-4</td>
<td>-5</td>
<td>-5</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-5</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>1</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>-6</td>
<td>2</td>
<td>-4</td>
<td>-7</td>
<td>-8</td>
<td>-5</td>
<td>-7</td>
<td>-3</td>
<td>-5</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
<td>0</td>
<td>-6</td>
<td>-3</td>
<td>-5</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>-4</td>
<td>-4</td>
<td>-2</td>
<td>-4</td>
<td>0</td>
<td>-4</td>
<td>-4</td>
<td>-5</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>-2</td>
<td>7</td>
<td>-5</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>-2</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td>-3</td>
<td>4</td>
</tr>
</tbody>
</table>

| A  | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V |
| A |   | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V |
| R |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| N |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| D |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| C |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Q |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| E |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| G |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| H |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| I |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| L |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| K |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| M |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| F |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| P |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| S |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| T |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Y |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| V |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
This constraint also tends to prefer one longer gap over several neighbouring short ones.

Formulae for gap penalties satisfying Equation (1.1) are said to be concave gap penalties.\footnote{Gap penalties satisfying Equation (1.1) satisfy the definition of the concave gap function as defined in Waterman (1995).}

The linear gap penalty function ($g_l = gl$), which we have used previously, is concave. Biologically (and following our model), extending a gap should be penalized less than opening one, hence a better formula for the gap penalty should be an affine gap penalty function (which is also concave). The function for the affine gap penalty is either $g_l = g_{open} + lg_{extend}$ or $g_l = g_{open} + (l - 1)g_{extend}$, meaning that in some programs the penalty for opening a gap is $g_{open} + g_{extend}$, in other programs it is $g_{open}$. Some also argue that the penalty for extending should decrease with the length; $g_l = g_{open} + \log l$ is an example of such a function.

For completeness we also mention the constant gap penalty function ($g_l = g$), where the penalty is independent of the gap length.

**Example**

The alignment of the insulin proteins in Section 1.2 was found by using the PAM 250 matrix, and a linear gap penalty of $g = 5$. It has nine gaps. If we change to an affine gap penalty, $5 + (l - 1)0.5$ we get the alignment:

```
MAVLQQLAVLTVLVRASAVLQALLVLLVLVSLVNLGVLLVVLALNLVLAAVASHGLTLSALVLALALYTTVLYLVAV
```

which contains fewer (four) gaps.

Changing the gap penalty to $1 + (l - 1)0.1$ results in the alignment:

```
MANW--V--PLALLL----ALNA----P--APAHAVFNQHLCGSHLVEALYLYLCPTGFYFYPK--RDVE--PLL
MALNTRLVPLALLIAWAAPAHAFVNHLCGSHLVEALYLYLCPTGFFYYPKARREVEEYPQV
```

```
GFLPPKSAQETEVAIFAFKDHAEIRKRGIVEQCCCHKPSCIFELQNYCN
GAELEGGGPGAG---GLGEGPQQ---KRIGIVEQCAGVCSSLQLENYC
```

\[ \triangle \]

The example illustrates that the problem of determining the gap penalty is difficult. Affine gap penalties are the most used, and typically $g_{open} \approx 10g_{extend}$. We will discuss this more in the context of local alignments (see Chapter 2.3).

Another aspect to discuss is if gaps at the end shall have the same penalties as gaps not at ends.
Example

Assume two sequences AGVARTLR and AGTLR, and make two alignments:

\[
\begin{array}{c|c}
\text{AL1} & \text{AL2} \\
AGVARTLR & AGVARTLR \\
AG---TLR & ---AGTLR \\
\end{array}
\]

AL1 will get the highest score if the end gaps have the same penalty as other gaps. However, often one of the sequences is a subsequence of the other, and AL2 would here be the correct one in that case (and would be found if end gaps were not penalized).

\[\triangle\]

1.7 Dynamic Programming for General Gap Penalty

The recurrence formula presented for dynamic programming is only valid for linear gap penalties. The reason is that we have assumed that each blank in a gap has the same penalty, independent of how long the gap is.

For finding the value (score) in \(H_{i,j}\) when general gap penalties are used, we must compare

- the score if the subalignment ends with the pair \(q_i, d_j\),
- the score if the subalignment ends with a gap in \(q\) of length \(l\), \(1 \leq l \leq j\),
- the score if the subalignment ends with a gap in \(d\) of length \(l\), \(1 \leq l \leq i\).

Figure 1.4(a) shows which elements must be used to calculate \(H_{i,j}\).

The recurrence formula for this is

\[
H_{i,j} = \max \left[ H_{i-1,j-1} + R_{qi,dj}, \max_{1 \leq l \leq j} (H_{i,j-l} - g_l), \max_{1 \leq l \leq i} (H_{i-l,j} - g_l) \right].
\] (1.2)

The time complexity of this recursion can be found by noting that the number of cells examined for finding \(H_{i,j}\) is \(1 + i + j\), hence the total number of cells examined is

\[
\sum_{i=1}^{m} \sum_{j=1}^{n} (1 + i + j) = mn + \sum_{i=1}^{m} ni + \sum_{j=1}^{n} j
\]

\[
= mn + O(nm^2 + mn^2) = O(nm^2 + mn^2).
\]

Figure 1.4(b) shows some of the values in the dynamic programming (DP) table for an example of using affine gap penalty. The scoring scheme is defined in the figure’s text.

We see that the work for finding the best alignment when a general gap penalty is used is an order larger than when using a linear gap penalty.

Note that the general recurrence formula supports gap penalties that are not concave (see Section 1.6).
1.8 Dynamic Programming for Affine Gap Penalty

Let the affine gap penalty be \( g_I = g_{\text{open}} + I g_{\text{extend}} \). We can look at the algorithm for the linear gap penalty (Section 1.4.1), and see how it must be changed in order to
use affine gap penalties. When a blank is to be inserted, we must find if it is the start of a gap \((g_{\text{open}} + g_{\text{extend}})\), or an extension \((g_{\text{extend}})\). For determining \(H_{i,j}\) we looked at the three neighbouring cells \(H_{i-1,j-1}\), \(H_{i,j-1}\) and \(H_{i-1,j}\). The formula for using \(H_{i-1,j-1}\) is

\[
H_{1,j}^{(3)} = H_{i-1,j-1} + R_{q_id_j},
\]

and this can still be used, since it involves no gap (see Section 1.4.1 for \(H_{1,j}^{(3)}\)).

For calculating \(H_{1,j}^{(1)}\) (the alternative via \(H_{i-1,j}\)), we must take into account how the alignment for \((q_1...i-1, d_1...j)\) can end. Three cases have to be considered (see Figure 1.5).

(a) Let \(E_{i-1,j}\) be the score at \(i-1, j\) when coming from \(i-2, j\). Then

\[
H_{1,j}^{(1),a} = E_{i-1,j} - g_{\text{extend}}.
\]

(b) Let \(F_{i-1,j}\) be the score at \(i-1, j\) when coming from \(i-1, j-1\). This is unlikely, for it would produce an alignment ending in

\[
\ldots q_i \ldots d_j 
\]

It is more likely that the two last columns would be one, without a blank. But it must be considered, hence

\[
H_{1,j}^{(1),b} = F_{i-1,j} - g_{\text{open}} - g_{\text{extend}}.
\]

(c) Let \(G_{i-1,j}\) be the score at \(i-1, j\) when coming from \(i-2, j-1\). Then

\[
H_{1,j}^{(1),c} = G_{i-1,j} - g_{\text{open}} - g_{\text{extend}}.
\]

So the maximum score when coming to cell \((i, j)\) from \((i-1, j)\) is

\[
H_{1,j}^{(1)} = \max\{E_{i-1,j} - g_{\text{extend}}, F_{i-1,j} - g_{\text{open}} - g_{\text{extend}}, G_{i-1,j} - g_{\text{open}} - g_{\text{extend}}\}.
\]
Therefore, three variables have to be saved at $H_{i-1,j}$ to be able to calculate the correct value of $H_{i,j}^{(1)}$. For finding the correct value of $H_{i,j}^{(2)}$ by use of symmetry we can conclude that three variables have to be saved at $H_{i,j-1}$. As a conclusion, for changing the procedure for a linear gap so that it can handle an affine gap, it is only necessary to introduce three variables in each cell, and change the assignment equations. Hence, the algorithm is still of order $O(mn)$.

### 1.9 Alignment Score and Sequence Distance

In the preceding subsections we have shown how to score the similarity of sequences. We can also measure the distance between two sequences. The edit distance is a common measure for strings: the edit distance between two strings is the minimum number of operations for transforming one of the strings to the other, where the operations are substitution, deletion and insertion of single symbols.

#### Example

Using our sequences $q = \text{VEITGIEIST}$, $d = \text{PRETERIT}$, we can transform $q$ to $d$ by the following six operations:

$\rightarrow P; V \rightarrow R; I \rightarrow ; G \rightarrow ; \rightarrow R; S \rightarrow$

This is the minimum number of operations, hence the edit distance between them is six.

We can now define a scoring scheme that allows us to find the edit distance between two strings from an alignment with maximum score. Let the scoring scheme be as follows:

- $R_{ab} = 0$ for $a = b$; $-1$ for $a \neq b$, and $g = 1$;
- let $T$ be the score of the best alignment.

Then there are $-T$ columns containing either a mismatch or a blank, and this is the minimum number of operations for transformation; hence the edit distance is $-T$.

#### Example

The best alignment of $q, d$, using the scoring scheme defined above, becomes

$q' : \quad \text{-VEITGIE-IST}$

$d' : \quad \text{PRE-T-ERI-T}$

with score $T = -6$, so the edit distance is six, as found in the example above.

Often, the distances between objects constitute a metric space. A set $X$ of elements is said to be a metric space if for any two elements $x$ and $y$ there is a real number $d_{xy}$ called the distance from $x$ to $y$, such that
1. $d_{xy} = 0$ for $x = y$.
2. $d_{xy} > 0$ for $x \neq y$.
3. $d_{xy} = d_{yx}$.
4. $d_{xy} \leq d_{xz} + d_{zy}$ for any $z \in X$ (the triangle inequality).

The edit distance constitutes a metric space. Note, however, that the minimum transformation (the transformation with the minimum number of operations) between a pair of strings is not necessarily unique.

For comparison of biological sequences, the edit distance can be used under the assumption that each observed difference in the sequences represents one mutation, which can be reasonable for very similar sequences. However, when the number of observed changes is large, there might be several mutations to each observed change.

**Example**

Assume an evolutionary history:

$$\text{AKLDC} : K \rightarrow ; L \rightarrow V; \rightarrow R; V \rightarrow M : \text{AMRDC}$$

The edit distance between the two sequences is two, corresponding to the first alignment below. The correct alignment (corresponding to the history) is, however, the second alignment below, showing three mutations, but the correct number of mutations is four.

<table>
<thead>
<tr>
<th>AKLDC</th>
<th>AKL-DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMRDC</td>
<td>A-MRDC</td>
</tr>
</tbody>
</table>

$\triangle$

The example shows that there might be several substitutions in one site: here $L \rightarrow V; V \rightarrow M$.

For comparing distances between different pair of sequences, it is common to divide the observed distance by the length of the longest sequence, resulting in (relative) distances not greater than 1. Also, it is not unusual to only count the columns in the alignment which do not contain a blank, and divide by the number of those columns.

Several models for correcting for multiple mutations are presented. Of course, the growth of the function for the corrected distance must increase with the observed distance, and most models result in a formula with a logarithmic function. Let $D$ be the observed (relative) distance; then a common model for finding the corrected (relative) number of mutations is

$$K = -a \ln(1 - f(D)),$$

where $a$ is a constant, and $f(D)$ is a positive function less than 1. One simple formula used for proteins (when columns with blank are ignored) is $f(D) = D + \frac{1}{2}D^2$ (Kimura 1983). (Note, however, that this cannot be used for large $D$ ($D$ greater than
0.85, since then \( f(D) \) becomes greater than 1.) Using these values for \( a \) and \( f(D) \) gives us an expression for \( K \), the number of estimated substitutions per column as
\[
K = -\ln(1 - D - \frac{1}{5}D^2).
\]
(1.3)

This can be greater than 1. For example, if the observed value is 0.8 (eight of ten columns have different amino acids), then the number of estimated substitutions becomes 2.6 substitutions per column during the evolutionary time since the two sequences diverged.

An analogue to a metric space for similarity would inverse the triangle inequality:
\[
R_{ab} \geq R_{ac} + R_{cb}.
\]
This is generally not satisfied when using scoring matrices such as the PAM series. For example, in the PAM 250 matrix, \( R_{GR} = -3 \), \( R_{GA} = 1 \), \( R_{A,R} = -2 \), hence \( R_{GR} < R_{GA} + R_{AR} \).

### 1.10 Exercises

1. Let two sequences be \( q = CDAA \) and \( d = AEECA \), and a scoring matrix:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td></td>
<td></td>
<td>-2</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

(a) Find the highest score by aligning \( q \) and \( d \) when the gap penalty is \( g_l = 2l \). Then find the best alignments.

(b) Now use gap penalty \( g_l = 1.8 + 0.4l \). The dynamic programming table will be partly filled as below, using the general DP procedure:

<table>
<thead>
<tr>
<th>( q ) | ( d )</th>
<th>A</th>
<th>E</th>
<th>E</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>-2.2</td>
<td>-2.0</td>
<td>-2.2</td>
<td>-2.6</td>
<td>-3.0</td>
</tr>
<tr>
<td>D</td>
<td>-2.6</td>
<td>-4.2</td>
<td>-4.0</td>
<td>-4.2</td>
<td>-2.6</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>-0.6</td>
<td>-2.8</td>
<td>-3.2</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>-3.4</td>
<td>-1.0</td>
<td>-1.6</td>
<td></td>
</tr>
</tbody>
</table>

Note especially how the value -3.2 in \( H_{3,3} \) is found, the value is \( H_{3,1} = g_2 = -0.6 - (1.8 + 0.8) \). Fill in the rest of the table, and find the best alignment(s).

(c) Compare the alignments found under (a) and (b), and find for each of them the minimum number of mutations which might have occurred, when we suppose that only one residue is included in a substitution.

2. In some cases one wants to score gaps at the ends of an alignment as 0.
PAIRWISE GLOBAL ALIGNMENT OF SEQUENCES

(a) In what cases is this reasonable (what is the relation between the two sequences)?

(b) The general procedure for dynamic programming can be changed in the following way to take care of this:

- initialize all cells in row and column 0 to 0;
- blanks in last row and column shall score 0.

Explain why these changes will produce the best alignment.

(c) Change Algorithm 1.1 to take into account end gaps with zero score.

(d) We have the sequences \( q = \text{ART} \) and \( d = \text{AARRTRT} \). Use score 1 for equal symbols, –1 for unequal, and a (linear) gap penalty of 1. Find the best alignments when a score of 0 is used for the end gaps.

3. How would you find the alignments if a constant gap penalty is used?

4. Change Algorithm 1.1 so that it can be used for an affine gap penalty. Use the method explained in Section 1.8.

5. Suppose \( q = \text{LARKTLVAKVLSV} \), \( d = \text{KLVASTVLRKRSA} \). By using a score of –1 for mismatches and blanks, and 0 for matches, one best alignment is

\( q' : -\text{L-ARKT-LVAKVLSV} \)
\( d' : \text{KLVAS-TVL-RKR-SA} \)

(a) What is the edit distance between the sequences?

(b) Estimate the relative evolutionary distance between them using Equation (1.3). What does the distance found mean?

(c) Could other best alignments result in other relative evolutionary distances?

1.11 Bibliographic notes

The first to use dynamic programming for comparing biological sequences was Needleman and Wunsch (1970). A linear space algorithm is presented in Hirschberg (1975).

A discussion of gap penalties can be found in Pascarella and Argos (1992) and Benner et al. (1993).

Formulae for correcting the sequence distance for multiple mutations (for DNA or proteins) are presented in Kimura (1980, 1983), Li (1993, 1997), Li and Gu (1996) and Swofford et al. (1996).