CHAPTER 7

7.1 INTRODUCTION

This chapter discusses the problem of aligning biological sequences to probe for homologous or functional similarities. Given two sequences, for instance AGHGHVWR and AHGIIRN, an alignment is simply a superposition

\[
\begin{align*}
\text{AGHGHVWR} & - \\
\text{A - H G - I I R N}
\end{align*}
\]

in which each letter of each sequence is aligned with a letter or gap in the other sequence. When biological sequences are homologous, there will be a natural alignment between them that expresses their biological similarity. Recall from Section 6.7 that two protein sequences are homologous if they have descended from a common, ancestral sequence. We gave a made-up example in Section 6.7.1 of a genealogical tree showing the evolution of the ancestral sequence ARNGVW into the homologous family \{GRQGFW, GRGFW, ARWNGFW, ARNGFW\}. To capture the homological relationship between, say, the second and third sequences of this family, we wrote the alignment

\[
\begin{align*}
\text{G R - - G V W} \\
\text{A R W N G V W}
\end{align*}
\]

(7.1)

The aligned pairs of letters in (7.1) represent amino acids descending from amino acids in the ancestral sequence. Thus, in the first pair G - G, G came from a substitution mutation replacing the A in ARNGVW by G, while the A in G is just the
original A of ARNGVW. The pairs \( R, G, V, W \)
in the alignment all represent descent without modification
of the original \( R, G, V, \) and \( W \) in the ancestral sequence.
On the other hand, at some point in the evolution of ARWNGVW,
a \( W \) was inserted between \( R \) and \( N \) of the ancestral ARNGVW.
There is no amino acid in the sequence GRGVW related to the
inserted \( W \) and so \( \overline{W} \) is used in the alignment of (7.1).
Likewise, the pair \( \overline{N} \) represents the deletion of \( N \) in the evolution
of GRGVW. In general, in a correct alignment of two
homologous sequences, an aligned pair \( \overline{X} \) represents amino acids in
each sequence that have evolved from a common amino acid in an
ancestral protein sequence; an amino acid \( X \) aligned with a gap,
as in \( X \) or \( \overline{X} \), represents either an insertion or deletion event.

The alignment in (7.1) was derived from full knowledge of
the evolutionary history of its two sequences. But of course,
biologists are not privy to such full information. Rather, the problem
they face is, given several protein sequences, to decide whether the
sequences, or even just parts of the sequences, are homologous, and,
if so, to infer how they might be aligned. In chapter 6 we
discussed how to score aligned sequences to test for homology. But
before applying a test it is necessary to align the sequences in the first
place. Two sequences may be aligned in a myriad of ways when gaps
are allowed; even more possibilities are generated if the aim is not to
align the entire sequences but to look for parts of each that may
be similar. Even when it is known that two sequences are homologous
it is not clear how they should be aligned. For example, the alignment
in (7.1) of GRGKW and ARWNGKW looks good by inspection because it aligns many matching pairs, but without knowledge of the genealogy, other alignments are certainly within the realm of possibility.

The strategy for choosing an alignment from among many possibilities for two given strings of letters is first, to choose a scoring method for scoring alignments, and then to find that alignment maximizing the score. Given the scoring method, the challenge is to find a computationally efficient method for finding maximally scoring alignments. That is the question to be addressed in this chapter. We will use a technique called dynamic programming which has importance in applied mathematics going far beyond the alignment problem.

We shall treat two scoring methods both based on using a substitution matrix \( S(a,b) \) where \( A \) is the alphabet. In the first method we fix a penalty \(-d\) for each gap -- this is equivalent to adding element \( S(-,\cdot) = S(\cdot,-) = -d \) for every \( a \in A \) to the substitution matrix. (gaps are never aligned with gaps). This is called the linear gap penalty method because a gap \( \overline{x_1 y_1 y_2 y_3 \ldots y_g y_{g+1}} \)

of length \( g \) adds a penalty \(-dg\) to the score (the penalty is linear in \( g \)). For example the score of

\[
\begin{align*}
\text{A} & \quad \text{GG} & \quad \text{K} & \quad \text{H} \\
\text{A} & \quad \text{--} & \quad \text{K} & \quad \text{P}
\end{align*}
\]

would be \( S(A,A) - 2d + S(K,K) + S(H,P) \).
The second scoring method is called the affine gap penalty method. In this method there is a penalty $-d$ for opening a gap and an additional penalty $-e$ for each new aligned pair $y$ or $x$ in the same gap. To explain, suppose we see

$$\ldots\ x_1\ -\ -\ -\ -\ x_2\ \ldots\ y_1\ y_2\ y_3\ \ldots\ y_g\ y_{g+1}\$$

in an alignment. Here we see a gap of length $g$.

The first alignment $y_2$ costs $-d$; each following alignment $y_3$, $y_4$, etc. costs $-e$. Thus this gap contributes

$$-d - (g-1)e$$

to the total score. Since the expression ($\ast$) is an affine function of $g$, the gap penalty is called affine. Always, $d$ is chosen to be larger than $e$. 
7.2 LINEAR GAP PENALTY ALGORITHMS

7.2.1 The edit graph and representation of alignments.

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be two sequences.

We shall represent gapped alignments of these sequences using paths in an edit graph. The edit graph for the two sequences is the directed graph

```
x_1, x_2, x_3, \ldots, x_n

y_1
y_2
y_3
\vdots
y_m
```

Figure 7.1

This is a graph with vertices in a rectangular array with a column for each $x_i$ and an unlabelled column at the left hand side; similarly, the first row is not labelled, but the remaining are labelled by the letters $y_1, \ldots, y_m$. The only directed edges allowed are of the type shown, either pointing right horizontally one column, down one row, or diagonally down and to the right one column and one row.

Usually we shall suppress writing in all the directed edges --- they are there implicitly.
We think of the first row as row 0 and of the first column as column 0. We label the vertex in column $i$ and row $j$ by $(i, j)$.

| $\phi$ | $x_1$ | \cdots | $x_i$ | \cdots | $x_n$ |
|--------|-------|--------|-------|--------|
| $\phi$ | $(0,0)$ | $(1,0)$ | \cdots | $(i,0)$ | \cdots | $(n,0)$ |
| $y_1$ | $(0,1)$ | $(1,1)$ | \cdots | \cdots | \cdots | \cdots |
| \vdots | \vdots | \vdots | \cdots | \vdots | \cdots | \cdots |
| $y_j$ | $(0,j)$ | $(1,j)$ | \cdots | $(i,j)$ | \cdots | $(n,j)$ |
| \vdots | \vdots | \vdots | \cdots | \vdots | \cdots | \cdots |
| $y_m$ | $(0,m)$ | \cdots | \cdots | $(i,m)$ | \cdots | $(n,m)$ |

**Figure 7.2**

The viewpoint from vertex $(i,j)$ with $i \geq 1$, $j \geq 1$ look like (with directed edges into $(i,j)$ shown only):

```
  \bullet (i,j-1)
  \downarrow
  \bullet \cdots \bullet (i,j)
  \bullet (i-1,j)
```

**Figure 7.3**
A path in the edit graph is a continuous sequence of directed edges, that is each successive edge departs from the vertex attained by the previous edge.

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \]
\[ y_1 \rightarrow \downarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \]
\[ y_2 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \]
\[ y_3 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \]
\[ y_4 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \right}
Example 7.1

This path represents the alignment:

HAG - K
P - HKKK

of the subsegment HAGK in LAHAGKP with the subsegment PHKKK in QPHKKK.

7.2.2 Types of alignments

Let \( x_1, \ldots, x_n, y_1, \ldots, y_m \) be two sequences.

A global alignment of these sequences is an alignment of the entire sequence \( x_1, \ldots, x_n \) with the entire sequence \( y_1, \ldots, y_m \).

A global alignment corresponds to a path in the edit graph from vertex \((0,0)\) (upper left) to vertex \((n,m)\) (lower right):
A local alignment is an alignment of a subsegment of \( x_1 \ldots x_n \) with a subsegment of \( y_1 \ldots y_m \), where at least one subsegment is proper, that is, is not the entire segment. The alignment of Exercise 7.1 is local. We look for local alignments when we have long strings and want to determine if they contain homologous domains. Any path in the edit graph, except those giving global alignments, represents a local alignment.

An overlap alignment is a special type of local alignment in which either:

a) an initial segment \( x_1 \ldots x_i \) of \( x_1 \ldots x_n \) is aligned with a terminal segment \( y_j \ldots y_m \) of \( y_1 \ldots y_m \); or

b) a terminal segment \( x_{n-k} \ldots x_n \) is aligned with an initial segment \( y_1 \ldots y_k \); or

c) the entire segment \( x_1 \ldots x_n \) aligns with a subsegment of \( y_1 \ldots y_m \) or vice-versa.

Example 7.2

Each path in this graph represents an overlap alignment.

For example, path I, with the unaligned parts appended in parentheses, is:

\[
LAHA(GKP)
\]

\[
(QP)HKK
\]
Similarly, path II is
\[(L) - A \rightarrow H \rightarrow A \rightarrow G \rightarrow (K \rightarrow P)\]
\[Q \rightarrow P \rightarrow H \rightarrow K \rightarrow J\]
and path III is
\[(LA) A \rightarrow G \rightarrow K \rightarrow P\]
\[Q \rightarrow P \rightarrow H \rightarrow K \rightarrow (K)\]

Notice that the path of an overlap alignment must start at a vertex in the topmost row or leftmost column and must end in the bottom row or rightmost column.

Suppose we are interested in isolating non-overlapping subsegments of \(x_1, \ldots, x_n\) that align to subsegments of \(y_1, \ldots, y_n\). Such an alignment is called a repeat match alignment.

**Example 7.3** Let \(x = \text{HAPWHAKPILA}G\) and \(y = \text{PKAHAPG}\). Here is the representation of a repeat match alignment:

\[
\begin{align*}
\text{HAPWHAKPILA} & \rightarrow G \\
\text{HAP} & \rightarrow \text{HA-P} \quad \text{APG} \\
\text{y} & = \text{PKA} \cancel{\text{HAP}} \text{G}
\end{align*}
\]

In the top row the sequence \(x_1, \ldots, x_n\) appears in its entirety in proper order, with possible gaps. In the bottom row are the various subsegments of \(y\) that are being aligned with the non-overlapping subsegments of \(x\). We have indicated
where the subsegments of \( y \) come from by drawing \( y \) below with arrows and bars to indicate what is aligning with what. Dots (not gaps) are used to separate the subalignments. We represent the alignment again, this time with brackets to enclosed the non-overlapping, aligned subsegments of the \( x \) sequence.

\[
[HAP][W][HAKP][IL][A-G]
\]

In this example the non-overlapping segments are strictly separated (we do not see \([HAP][HAKP]\) for example). Ideally one should allow adjacent aligned subsegments without separation, but for technical simplicity we shall impose the condition that the aligned segments in the \( x \) sequence be strictly separated.

To graph repeat match alignments we add two new types of directed edges to the edit graph:

**Type (i)**

In a path this edge ends a subsegment alignment (a match) by adding \([x_{i+1}]\) to the alignment.
In a path, this edge marks the start of a new subsegment alignment starting at \((i,j)\).

Also, we add vertex to the edit graph in position \((n+1,0)\) and end every path representing a repeat match alignment with an edge to \((n+1,0)\).

Here is the path representing the repeat match alignment of Example 7.3.
7.2.3 Scoring paths

We will show how to score paths so that the score of a path is the score of the alignment it represents.

Assume we are given a substitution matrix $S$ and a gap penalty $-d$. The idea is simple: each directed edge is assigned a weight which equals the contribution to the alignment score of the aligned pair or gap represented by the directed edge.

(i) The edge $\text{(i-1,j-1)} \rightarrow \text{(i,j)}$ adds $x_i$ to the alignment, so the weight of this edge is $S(x_i, y_j)$.

(ii) The edge $\text{(i-1,j)} \rightarrow \text{(i,j)}$ adds $y_j$ to the alignment, so it has weight $-d$.

(iii) The edge $\text{(i-1,j)} \rightarrow \text{(i,j)}$ adds $\overline{z_i}$ to the alignment, so it has weight $-d$.

(So far we have not dealt with the extra, repeat match alignment edges. We do this later.)
To summarize:

\[ (i-1,j-1) \rightarrow S(x_i,y_j) \rightarrow (i,j-1) \rightarrow (i,j) \]

\[ (i-1,j) \rightarrow (i,j) \rightarrow -d \]

Figure 7.5  Edge weights for scoring.

The score of a path is then defined to be the sum of the weights of its edges.

The path score will then give the score of the alignment it represents using linear gap penalties. This is easy to see by example.

**Example 7.4** Here we show a path representing a global alignment

\[
\begin{array}{ccccccc}
\text{L} & \text{A} & \text{H} & \text{A} & \text{G} & \text{K} & \text{P} \\
\text{Q} & & & & & & \\
\text{P} & & & & & & \\
\text{H} & & & & & & \\
\text{K} & & & & & & \\
\text{K} & & & & & & \\
\end{array}
\]

The weight of the 1st edge is \( s(Q,L) \), of the 2nd \( s(P,A) \), of the third \(-d\), etc. The path score is thus

\[ s(Q,L) + s(P,A) - d - d + s(H,G) + s(K,K) + s(K,P) \]
and this is precisely the score of the alignment

\[ \text{L A H A G K P} \]
\[ \text{Q P H K K} \]

7.2.4 Finding optimal (highest scoring) global alignments

One way to find the highest scoring global alignment is simply to list each global alignment, score it, and go down the list to find the highest scoring ones. This is not practical the number of paths in the edit graph representing alignments grows exponentially with \( n \) and \( m \). For even moderately sized graphs there are just too many possibilities.

Rather, we take an approach called dynamic programming, working in the edit graph. We define the following function, called the value function, on the points \( (i, j) \) with \( 0 \leq i \leq n, 0 \leq j \leq m \), of the edit graph

\[ B(0,0) = 0 \]

\[ B(i, j) \triangleq \text{maximum score of all paths from} \]
\[ (0,0) \text{ to } (i, j) \]
\[ = \text{maximum score of all alignments of} \]
\[ x_1, x_2, \ldots, x_i \text{ with } y_1, \ldots, y_j \]
We propose to find \( B(i,j) : 0 \leq i \leq n, 0 \leq j \leq m \).

At first this might seem crazy. Haven't we just said there are too many paths to count to do this practically? True, but it turns out that \( B \) satisfies a simple, recursive equation.

Recall that \( B(i,j) \) is the maximal score of all paths from \((0,0)\) to \((i,j)\). Now all paths from \((0,0)\) to \((i,j)\) end with one of the edges in the diagram

\[
\begin{array}{c}
\bullet \quad \bullet \\
(0,0) \quad (i,j) \\
\downarrow \quad \downarrow \\
(-d) \quad (-d) \\
(0,0) \quad (i,j) \\
\end{array}
\]

What is the maximum score from among all paths from \((0,0)\) to \((i,j)\) the end with the diagonal edge from \((i-1,j-1)\) to \((i,j)\)? Clearly it is

\[ B(i-1,j-1) + s(x_i, y_j) \]

because \( B(i-1,j-1) \) is the maximum score of all paths from \((0,0)\) to \((i-1,j-1)\) and \( s(x_i, y_j) \) is the weight added to this score by the final alignment of \( x_i \) and \( y_j \).

Similarly

\[ B(i,j-1) - d \] is the maximum score of all paths from \((0,0)\) to \((i,j)\) that end in the edge \((i,j-1) \leftarrow (i,j)\).

\[ B(i-1,j) - d \] is the maximum score of all paths from \((0,0)\) to \((i,j)\) that end in the edge \((i-1,j) \rightarrow (i,j)\).
These three cases exhaust all the possibilities from paths from \((0,0)\) to \((i,j)\), so we can draw two conclusions:

(i) \[ B(i,j) = \max \left\{ B(i-1,j-1) + s(x_i, y_j), B(i,j-1) - d_j, B(i-1,j) - d_i \right\} \tag{DPE} \]

(ii) The last edge on a maximally scoring path from \((0,0)\) to \((i,j)\) points from the vertex that gives the maximum in (i). (For example if \( B(i-1,j-1) + s(x_i, y_j) \) is greater than the other two terms, the edge \((i-1,j-1) \rightarrow (i,j)\) will be the last edge of a maximally scoring path from \((0,0)\) to \((i,j)\) ) \tag{DPE}

Equation (i) is called the dynamic programming equation for \(B\). The beauty of it is that it can be solved simply and recursively starting from vertex \((0,0)\) and moving down and rightward in the edit graph. We get a simple algorithm for finding maximal score global alignments:

A) Solve for \(B\) at all vertices in the edit graph by using the dynamic programming equation.

B) At each step draw a 'traceback' arrow from \((i,j)\) to the vertex (or vertices) giving the maximum in the dynamic programming equation.

C) When A) and B) are done, follow the traceback arrows from the final vertex \((n,m)\) to \((0,0)\). When read in a forward direction this path will have maximal score.
Before we give an example we have to add one more part of the dynamic programming equation, because we derived this only for \((i,j)\) with \(i \geq 1, j \geq 0\).

What if \(j = 0\)? Then there is only one edge into \((i,0)\), namely

\[
\begin{array}{c}
\cdot \\
(i-1,0) \\
\cdot \\
(i,0)
\end{array}
\]

Thus

\[
B(i,0) = B(i-1,0) - d, \quad i \geq 1, \quad (DPE_1)
\]

Similarly

\[
B(0,0) = B(0,0) - d, \quad j \geq 1, \quad (DPE_2)
\]

In both cases the traceback is the reverse of the single edge into \((i,0)\) or \((0,j)\).

We work a full example below.
**Example 7.5.** Find the maximal score alignment of
\( x = \text{AAQCCDN} \) and \( y = \text{ACC} \) using \( d = 6 \)
and the substitution matrix

\[
S = \begin{bmatrix}
A & R & N & D & C & Q \\
A & 5 & -2 & -1 & -2 & -1 & -1 \\
R & 7 & -1 & -2 & -4 & 1 \\
N & 7 & 2 & -2 & 0 \\
D & 8 & -4 & 0 \\
C & 13 & -3 \\
Q & 7
\end{bmatrix}
\]

We shall do this problem by entering the values \( d \) \( B(i,j) \) directly in the edit graph.

Let's start by finding \( B(1,0) \) for \( 1 \leq i \leq 7 \) and \( B(0,j) \) for \( 1 \leq j \leq 5 \).
This is easy using \((DPE_2)\) and \((DPE_3)\).

First, using \((DPE_2)\): \( B(0,0) = 0 \), \( B(1,0) = B(0,0) - d = -6 \), \( B(2,0) = B(1,0) - 6 = -12 \), and continuing the obvious trend: \( B(3,0) = -3(6) = -18 \), \( B(4,0) = -4(6) = -24 \), \( B(5,0) = -5(6) = -30 \), \( B(6,0) = -6(6) = -36 \), \( B(7,0) = -7(6) = -42 \).

Next, using \((DPE_3)\) and similarly reasoning: \( B(3,0) = -3(6) \) for each \( j \).

We fill those values in the edit graph, adding in the traceback in the next figure.
First step: Filling in the border values.

Now we are set up to find \( B(1,1), B(2,1) \), etc. recursively

\[
B(1,1) = \max \left\{ B(0,0) + s(A,A), B(1,0) - 6, B(0,1) - 6 \right\}
\]
\[
= \max \left\{ 0 + 5, -6 - 6, -6 - 6 \right\} = 5
\]

The trace back is from \((1,1)\) to \((0,0)\). Thus we can add

\[
\begin{array}{c}
0 \\
-6 \\
-12
\end{array}
\]
\[
\begin{array}{c}
A \\
A \\
C
\end{array}
\]
\[
\begin{array}{c}
A \\
C
\end{array}
\]

Next

\[
B(2,1) = \max \left\{ B(1,0) + s(A,A), B(2,0) - 6, B(1,1) - 6 \right\}
\]
\[
= \max \left\{ -6 + 5, -12 - 6, 5 - 6 \right\} = -1
\]

and the trace back is from \((2,1)\) to \((1,0)\), or from \((2,1)\) to \((1,1)\) (both lead to the max of -1)
Adding this to the table:

\[
\begin{array}{cccccc}
\text{A} & \text{A} & \text{Q} & \text{C} & \text{C} & \text{D} \\
0 & -6 & -12 & -18 & -24 & -30 & -36 & -42 \\
\text{A} & \text{-6} & \text{5} & \text{-1} & \text{-7} & \text{-13} & \text{-19} & \text{-25} & \text{-31} \\
\text{C} & \text{-12} & \text{-1} & \text{4} & \text{-2} & \text{6} & \text{0} & \text{-6} & \text{-12} \\
\text{C} & \text{-18} & \text{-7} & \text{-2} & \text{4} & \text{11} & \text{19} & \text{13} & \text{7} \\
\text{Q} & \text{-24} & \text{-13} & \text{-8} & \text{5} & \text{5} & \text{13} & \text{19} & \text{13} \\
\end{array}
\]

We continue in this manner, moving from row to row; we could also go column by column. The algorithm could be easily implemented on a spreadsheet. We give here the final result. The reader should check some of the new entries using the (DPE) (we hope to have made no mistakes!)

This is cluttered so we draw the graph only with those tracebacks along paths connecting (0,0) to (7,4)
By taking different branches of the paths going forward we can read off 4 alignments each achieving the maximum score $B(7,4)=13$. Those are

\[
\begin{align*}
A & \quad A & \quad A & \quad Q & \quad C & \quad C & \quad D & \quad N \\
A & - & - & C & C & Q & -
\end{align*}
\]

\[
\begin{align*}
A & \quad A & \quad A & \quad Q & \quad C & \quad C & \quad D & \quad N \\
A & - & - & C & C & - & Q
\end{align*}
\]

\[
\begin{align*}
A & \quad A & \quad A & \quad Q & \quad C & \quad C & \quad D & \quad N \\
- & A & - & C & C & Q & -
\end{align*}
\]

\[
\begin{align*}
A & \quad A & \quad A & \quad Q & \quad C & \quad C & \quad D & \quad N \\
- & A & - & C & C & - & Q
\end{align*}
\]
Errata for Chapter 7, pages 1-22

1. page 12, first full line of text: "Also, we add a vertex...

2. page 6, last line before Figure 7.3: "looks like"

3. page 19, line 2: "y = ACC Q"

4. page 17, Dynamic programming equation 3rd line:

   \[ B(i,j) = \max \left\{ B(i-1,j) + S(x, y), B(i,j-1) - d, B(i-1,j) - d \right\} \]

5. page 10, line 9: "to \((i,j)\) that end with...

6. page 19, line 10 from bottom: "and \(B(0,j)\), \(1 \leq j \leq y\)."
7.2.5 The augmented edit graph and dynamic programming.

We want to adapt the dynamic programming method to find optimal alignments of other types -- local, overlap, etc. These other types of alignments are often represented by paths that can start at other vertices in the edit graph; for example, a local alignment path can start at any vertex, except the bottom-right vertex. However, it is convenient for dynamic programming to have all paths start at a common vertex. We will accomplish this by adding a special 'start' vertex I to the edit graph. By adding directed edges from I to selected vertices in the edit graph, according to the type of alignment, we will be able easily to derive optimal alignment algorithms.

The dynamic programming principle is always the same. In each case, we define a function \( F(v) \) on the vertices \( v \) of the graph. Always \( F(I)=0 \).

For each vertex \( v \), let \( \mathcal{P}(v) \) be the set of all vertices \( e \) such that the graph admits a directed edge \( e \rightarrow v \) from \( e \) to \( v \); one can think of \( \mathcal{P}(v) \) as the set of 'predecessors' of \( v \). Let the weight of \( e \rightarrow v \) be denoted \( W(ev) \). The dynamic programming equation will have the form:

\[
F(v) = \max \left\{ F(e) + W(ev) : e \in \mathcal{P}(v) \right\}.
\]
7.2.6. Finding optimal local alignments

We add the 'start' vertex $I$ to the edit graph and allow directed vertices of weight $0$ from $I$ to every vertex in the edit graph. An edge from $I$ to vertex $(i,j)$, when it occurs in a path, has the meaning: "start a local alignment path with a directed edge from $(i,j)". With this device, all paths representing local alignments start at $I$.

Example 7.6 Representing local alignments with paths starting from $I$.

Path I represents the local alignment $AGKP$ as $P = H K$.

It has score

$$0 + s(PA) - d + s(KP) + s(P)$$

where the first $0$ is the weight of the edge from $(0,0)$ to $(3,1)$.

Path II represents the local alignment $AHAG$ as $H K$.

The path score is $0 - d + s(H,H) + s(K) + s(A) - d$, which is the score of the local alignment.
We now define the value function \( L \).

(i) \( L(I) = 0 \)

(ii) For each vertex \((i,j)\) in the edit graph

\[ L(i,j) = \text{maximum score of all paths from } I \text{ to } (i,j) \]

We will derive the dynamic programming equation for \( L \), given the edit graph for \( x = x_1 \ldots x_n \) and \( y = y_1 \ldots y_m \).

Case 1: \( i > 1, j > 1 \). The picture is

\[ \begin{array}{c}
\text{(i,j-1)} \\
\downarrow \\
\text{(i,j)} \\
\text{(i-1,j)} \\
\text{(i-1,j-1)}
\end{array} \]

\( s(x_i, y_j) \)

Using the dynamic programming reasoning

\[ L(i,j) = \max \left\{ \begin{array}{l}
L(I) + 0 \\
L(i-1,j-1) + s(x_i, y_j) \\
L(i-1,j) - d \\
L(i,j-1) - d
\end{array} \right\} \]

\[ = \max \left\{ 0, L(i-1,j-1) + s(x_i, y_j), L(i-1,j) - d, L(i,j-1) - d \right\} \]

\((\text{DPEL}_L)\)

since \( F(I) = 0 \).
Case (ii) \( i \geq 0, \; j = 0 \)

The picture of the edges leading into \((i,0)\) is

\[
\begin{array}{c}
0 \quad \rightarrow \\
\left|_{(i-1,0)-d}^{(i,0)} \right| \\
\end{array}
\]

\(( \text{Note: this edge does not exist if } i = 0 \))

Thus

\[
\begin{align*}
\left( \text{DPEL}_2 \right) \quad & \quad L(i,0) = \max \{ L(i)+0, \; L(i-1,0)-d \} \quad \text{if } i \geq 1 \\
& = \max \{ 0, \; L(i-1,0)-d \} \\
L(0,0) &= 0
\end{align*}
\]

Case (iii) \( i = 0, \; j \geq 1 \). This is similar to case (ii)

\[
\left( \text{DPEL}_3 \right) \quad L(0,j) = \max \{ 0, \; L(j-1,0)-d \} , \; j \geq 1
\]

Algorithm to find maximum scoring paths:

Follow closely the prescription for the global alignment algorithm:

1) Solve \((\text{DPEL}_1), (\text{DPEL}_2), (\text{DPEL}_3)\) recursively for
   \(L\), recording traceback.
2) Search the table \(L(i,j)\) and find
   the vertex (or vertices) \((i^*, j^*)\) at which \(L(i,j)\) achieves
   its maximum value. Follow tracebacks from \((i^*, j^*)\) to
   find maximum scoring paths. These paths represent the
   maximum scoring local alignments.
Remarks

1. It will be assumed that $S(x,y) > 0$ for any letter $x$ and $y$, where $S$ is the substitution matrix.

Suppose that no letter in the sequence $x = x_1 \ldots x_n$ matches a letter in $y = y_1 \ldots y_m$. Then it could be possible that every local alignment has negative score. However, the algorithm proposed above based on $(DPEL_i) - (DPEL_j)$ will not pick out any negative scoring local alignments. Instead we will get $L(i,j) = 0$ for each $(i,j)$ and each path of the form $(i,j) \rightarrow (i-1,j) \rightarrow (i,j)$ consisting of one directed edge from $I$ to $(i,j)$ will be optimal. But this path does not give a true local alignment. So the algorithm will not work if all real local alignments have negative score. However we need not worry about this case in practice. Two long protein sequences are likely to have amino acids in common and if they do not we are probably not interested in trying to align them.

2. It may be that the alignment identified by this algorithm is a global alignment; that is there may be a global alignment whose score is bigger than that of any proper local alignment.

3. In practice, we do not explicitly include the vertex $I$ in performing the algorithm, nor do we draw in the traceback when it points to $I$. This will be illustrated in the examples below.
4. The dynamic programming equations \((DPEL_2)\) and \((DPEL_3)\) lead to the following result for any two sequences \(x = x_1 \ldots x_n\) and \(y = y_1 \ldots y_m\):

\[
L(0, j) = 0 \quad 0 \leq j \leq m \\
L(i, 0) = 0 \quad 0 \leq i \leq n
\]

In other words, when filling in the table for \(L\) values we find always:

\[
\begin{array}{cccccc}
  & x_1 & x_2 & x_3 & \ldots & x_n \\
\hline
0 & 0 & 0 & 0 & \ldots & 0 \\
y_1 & 0 & \quad & \quad & \quad & \quad \\
y_2 & 0 & \quad & \quad & \quad & \quad \\
\vdots & \quad & \quad & \quad & \quad & \quad \\
y_m & 0 & \quad & \quad & \quad & \quad
\end{array}
\]

This is easy to see: We know \(L(0, 0) = 0\). So, according to \((DPEL_2)\)

\[
L(1, 0) = \max\{0, L(0, 0) - d_1\} = \max\{0, -d_1\} = 0
\]

\[
L(2, 0) = \max\{0, L(1, 0) - d_2\} = \max\{0, -d_2\} = 0
\]

etc.

\[
L(0, 1) = \max\{0, L(0, 0) - d_2\} = \max\{0, -d_2\} = 0
\]

\[
L(0, 2) = \max\{0, L(0, 1) - d_3\} = 0
\]

etc.
Therefore, in doing a local alignment, we can always begin by entering 0's in the 0\textsuperscript{th} row and 0\textsuperscript{th} column. The tracebacks would be to \( I \) and are omitted, as will be seen in the example.

**Example 7.7** Using the substitution matrix \( S \) defined on page 19 in Example 7.5, find the maximum scoring local alignment between \( x = \text{AAQCCCDN} \) and \( y = \text{ACCAQ} \).

Use \( d = 6 \).

From remark 4 we know \( L(i, 0) = 0 \) and \( L(0, j) = 0 \) for all \( i \) and \( j \). We can then find \( L(i, i), i \geq 1 \) (the next row in the table): using (DPEL_1)

\[
L(1, 1) = \max\{0, L(0, 0) + 5(A, A), L(0, 1) - 6, L(1, 0) - 6\}
\]

\[
= \max\{0, 0 + 5, 0 - 6, 0 - 6\} = 5
\]

\[
L(2, 1) = \max\{0, L(1, 0) + 5(C, A), L(1, 1) - 6, L(2, 0) - 6\}
\]

\[
= \max\{0, 0 + 5, 5 - 6, -6\} = 5
\]

We enter these values in the table, with the tracebacks, and the remaining values of \( L \) in row 2, which the reader should verify.

\[
\begin{array}{ccccccc}
A & A & Q & C & C & D & N \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 5 & 5 & 0 & 0 & 0 & 0 \\
C & 0 & & & & & \\
C & 0 & & & & & \\
Q & 0 & & & & & \\
\end{array}
\]

Notice that we do not draw tracebacks from the 0 entry vertices to \( I \). They are implicitly understood.
The values of $L$ for the second row are calculated recursively by
$$L(i,j) = \max\{0, L(i-1,j) + S(x_i, y_j), L(i, j-1) - 6, L(i-1,j-1) - 6\}$$
For example,
$$L(1,2) = \max\{0, L(0,2) + S(A,C), L(1,1) - 6, L(0,2) - 6\}$$
$$= \max\{0, 5 - 6, 0 - 6\}$$
$$= 0$$
(In this case the traceback is to I and is omitted.)
We continue in this manner across rows and down columns.
The final result is

$$\begin{array}{cccccccc}
& A & A & Q & C & C & D & N \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 5 & 5 & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 4 & 2 & 13 & 13 & 7 & 1 \\
C & 0 & 0 & 0 & 1 & 15 & 20 & 20 & 14 \\
Q & 0 & 0 & 0 & 7 & 9 & 20 & 20 & 20 \\
\end{array}$$

The largest value of $L$ in this table is 26 and is achieved at $(5, 3)$ and at $(6, 4)$. Tracebacks from either of these two vertices give optimal local alignments.

In the next two tables we circle each of these alignments.

The local alignment circled here is

$$\begin{array}{cccccccc}
& A & A & Q & C & C & D & N \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 5 & 5 & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 4 & 2 & 13 & 13 & 7 & 1 \\
C & 0 & 0 & 0 & 1 & 15 & 20 & 20 & 14 \\
Q & 0 & 0 & 0 & 7 & 9 & 20 & 20 & 20 \\
\end{array}$$

The local alignment circled here is

$$\begin{array}{cccccccc}
& C & C & C & C \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 5 & 5 & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 4 & 2 & 13 & 13 & 7 & 1 \\
C & 0 & 0 & 0 & 1 & 15 & 20 & 20 & 14 \\
Q & 0 & 0 & 0 & 7 & 9 & 20 & 20 & 20 \\
\end{array}$$
Conclusion: We find two optimal local alignments, $\mathcal{EE}$, and $\mathcal{CCO}$, each with a score of 26.

7.2.7 Finding optimal overlap alignments

Recall that an overlap alignment path must start on the top row or the left-most column. To handle this we allow directed edges from the start vertex I only to vertices of the top row or left-most column:

But no edges from vertex I to other vertices are allowed. (Remember that an edge from vertex I to $(i,j)$ means starting a path at $(i,j)$, so this is the correct way to proceed so that alignments determined by paths starting at vertex I will start from the top row or left-most column.)
We define

\[ M(I) = 0 \]

\[ M(i,j) = \text{maximum score of all paths starting from vertex } I \text{ and ending at } (i,j) \]

The dynamic programming equations are now easy to derive. For the vertices \((i,0), i \geq 0,\) of the topmost row and the vertices \((0,j), j \geq 0,\) of the leftmost column, the situation is exactly the same as that of cases (ii) and (iii) on page 26 for local alignments. Therefore (DPEL\(_2\)) and (DPEL\(_3\)), with \(L\) being replaced by \(M\), are valid for \(M\) also. We can therefore conclude from the analysis of local alignments that \(M(i,0) = 0, i \geq 0,\) and \(M(0,j) = 0, j \geq 0.\) Thus for the overlap alignments algorithm one can start by placing 0's in the topmost row and leftmost column.

However, for a vertex \((i,j)\) with \(i > 1\) and \(j > 0,\) the picture of edges leading into \((i,j)\) is

\[ (i-1,j-1) \rightarrow (i,j) \]

\[ s(x_j, y_j) \rightarrow (i,j) \]

\[ (i,j) \rightarrow (i,j+1) \]

without the directed edge from vertex \(I\) to \((i,j).\) This is exactly the same picture that occurred for doing global alignments. Hence the dynamic programming equation is

\[ M(i,j) = \max\{ M(i-1,j-1) + s(x_j, y_j), M(i,j-1) - d, M(i,j+1) - d \} \]

(DPEO)
Algorithm for maximum scoring overlap alignments

1. Solve for $M(i,j)$, 0 $\leq i, j$ $\leq m$ using (DPED)
   and $M(i,j) = M(i-1,j-1)$ for all $i$ and $j$. Record traceback.
2. An overlap alignment must end on a vertex of the bottom row or rightmost column. Examine the values
   
   \[
   \begin{align*}
   &M(n,0) \\
   &M(n,1) \\
   &\vdots \\
   &M(n,j) \\
   &M(m) \quad \cdots \quad M(i,m) \quad M(n,m) \\
   \end{align*}
   \]

   In the bottom row and rightmost column and find the vertex which maximizes $M$ among all these values.

   The traceback from these vertices defines the best overlap alignment.

Example 7.8. We perform the overlap algorithm for the sequences $x = AAQC C D N$ and $y = ACCQ$ used in example 7.7 and for the same $S$.

The reader should derive the result as an exercise.

We have underlined the maximum value 25 achieved in the bottom row and rightmost column and circled the traceback.
paths from this vertex. We find two overlap alignments each with score 25:

\[(AA)Q\times C\times C\times D\times (N)\]
\[(A)A\times C\times C\times C\times D\times (N)\]
\[A\times C\times C\times Q\]
\[A\times C\times C\times Q\]

7.2.8 Repeat match alignments.

First we need to complete the description of a scoring method for repeat match alignments. We wish to avoid many small segment matches. For this reason, we impose a penalty \( T \) on ending a match \((T70)\). This means an edge of the type

\[
\begin{array}{c}
(i,0) \\
\downarrow
\end{array}
\]

\[(i,j)\]

has weight \(-T\) if \(i > 1\).

The edges along the top row \((i,0)\) however, fill in between unmatched subsegments of \(x_1x_2\ldots x_n\) and receive \(0\) weight.

Finally, the edge

\[
\begin{array}{c}
(i,j) \\
\downarrow
\end{array}
\]

which starts a new match also receives weight \(0\).
Example 7.9. Return to the repeat match alignment of Example 7.3. It is graphed on the bottom of page 12. From this graph we can read off the score (again using -d as the gap penalty)

\[
0 + S(H,H) + S(A,A) + S(P,P) - T + \alpha + S(H,H) + S(A,A) - d + S(P,P) \\
- T + 0 + 0 + S(A,A) - d + S(G,G) - T.
\]

By now the dynamic programming method should be clear. We write the dynamic programming equations for \( R(i,j) \), where

\[
R(0,0) = 0
\]

\[
R(i,j) = \text{maximum score of all repeat match paths starting at } (0,0) \text{ and ending at } (i,j).
\]

We get:

(i) \( R(0,j) = 0 \) for all \( j \geq 0 \).

(ii) For \( i > 1 \),

\[
R(i,0) = \max \left\{ R(i-1,0), R(i,1) - T, R(i,2) - T, \ldots, R(i,m) - T \right\}
\]

(iii) For \( (i,j) \), \( i \geq 1, j \geq 1 \)

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

(DPRM)
The algorithm is:

1. Compute $R(i,j)$ for all entries, including $(n+1,0)$, and record tracebacks.
2. Follow tracebacks from $(n+1,0)$ to $(0,0)$ to find a maximum scoring repeat match alignment.

A worked example will be given in the exercises.

7.3 An (abbreviated and incomplete) treatment of affine gap penalties

We promised in the introduction to discuss the case of affine gap penalties. However, in the interest of time, we will go back on our word. We will just show how the dynamic programming method can be applied to derive an algorithm for global alignments. But our algorithm will not be the most efficient one — clever adaptions of the method lead to better algorithms. Our purpose here are mostly pedagogical. (The restriction to global alignments is not important. The method can be extended to other alignment types following two ideas of section 7.2.)

Remember that for the affine gap penalty method we set parameters $d > 70$ and give a sequence of $g$ consecutive alignments of a letter with a gap weight $-d - (g-1)e$.

Something of the form
\[ a \ x_1 \ x_2 \ldots \ x_g \ b \ \ldots \ldots \ (x) \]
\[ y_1 \quad - \quad - \quad y_2 \]
therefore contributes \( s(a, y_1) - [d + (g-1)c] + s(b, y_2) \) to the overall score.

To handle this we will add directed edges to the edit graph to represent a sequence of \( g \) consecutive gaps and give the edges appropriate weights. For example, to handle the segment shown in (x) we would use edges as follows:

\[ a \ x_1 \ x_2 \ldots \ x_g \ b \]
\[ s(a, y_1) \quad \rightarrow \quad -d-(g-1)c \]
\[ y_1 \quad \bullet \quad \bullet \quad \bullet \quad \ldots \]
\[ y_2 \]

Thus we allow horizontal edges that point to a vertex \( g \) columns over to represent a gap of length \( g \) letters.

Following this idea we draw a picture showing the 'predecessors' of vertex \( (i,j) \) that we now include in the revised edit graph.

**Figure 7.6.**
Any horizontal or vertical edge of length gets weight 

To return to our favorite example consider the paths

```
A   A   Q   C   C   D   N
```

The path shown represents the alignment

```
-     A   Q   C   C   D
A   C   C   -   -   Q
```

and it has score

```
- [d+e] + s(A,c) - [d+2e] + s(B,Q)
```

**Dynamic programming for global alignment:**

Let \( G(0,0) = 0 \)

\( G(i,j) = \text{maximum score of all paths from (0,0) to (i,j)} \)

where the paths can now include horizontal or vertical edges connecting any vertices in the same row or same column, respectively.
First we derive the values of $G(i,0)$, $i \geq 1$, directly from the definition.

$$G(i,0) = -d - (i-1)e$$

This value is achieved by the edge

$$
\begin{align*}
\bullet & \quad \rightarrow \\
(0,0) & \quad (i,0)
\end{align*}
$$

According to our definitions we could also imagine a path

$$
\begin{align*}
\rightarrow & \quad \rightarrow \\
(0,0) & \quad (k,0) \quad (i,0)
\end{align*}
$$

But this has weight

$$-\left[d + (k-1)e\right] - \left[d + (i-k-1)e\right]$$

This equals

$$-2d = (i-2)e = -d - (i-1)e + (e-d)$$

Since we are assuming $d > e$ we see that this weight is yet more negative than $-d - (i-1)e$. Similar reasoning applies to any other sequence of horizontal edges leading from $(0,0)$ to $(i,0)$, and thus $G(i,0) = -d - (i-1)e$ as claimed.

By the same argument $G(j,0) = -d - (j-1)e$ for $j \geq 1$.

Next we look at $G(i,j)$ for $i \geq 1$, $j \geq 1$. By referring to
Figure 7.6 on the bottom of page 36 we obtain

\[
G(i,j) = \max \left\{ \begin{array}{l}
G(i-1,j-1) + S(x_i, y_j) \\
G(i-1,j) - d \\
G(i-2,j) - [d+e] \\
G(i-3,j) - [d+2e] \\
\vdots \\
G(i,j) - [d+(i-1)e] \\
G(i,j-1) - d \\
G(i,j-2) - [d+e] \\
\vdots \\
G(i,j) - [d+(j-1)e] 
\end{array} \right. \]

The justification of this equation should be clear by now. Clearly it will not be efficient to implement on large edit graphs. But its derivation illustrates again the dynamic programming method, which was our only aim here.