Errata for Chapter 7, pages 1-22

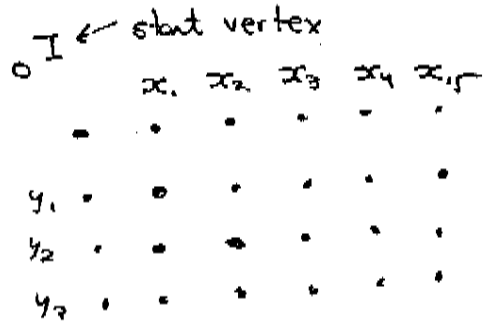
1. page 12; first full line of text: "Also, we add a vertex ..."
2. page 6; last line before Figure 7.3: "looks like"
3. page 19; line 2: "$y = ACCQ$"
4. page 17: Dynamic programming equation 3rd line:

$$B(i,j) = \max\{B(i-1,j-1) + s(x_i, y_j),\ B(i,j-1) - d,\ B(i-1,j) - d\}$$

5. page 16, line 9: "to (i,j) that end with ..."
6. page 19, line 10 from bottom: "and $B(0,j)$, $1 \le j \le 4$."

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.



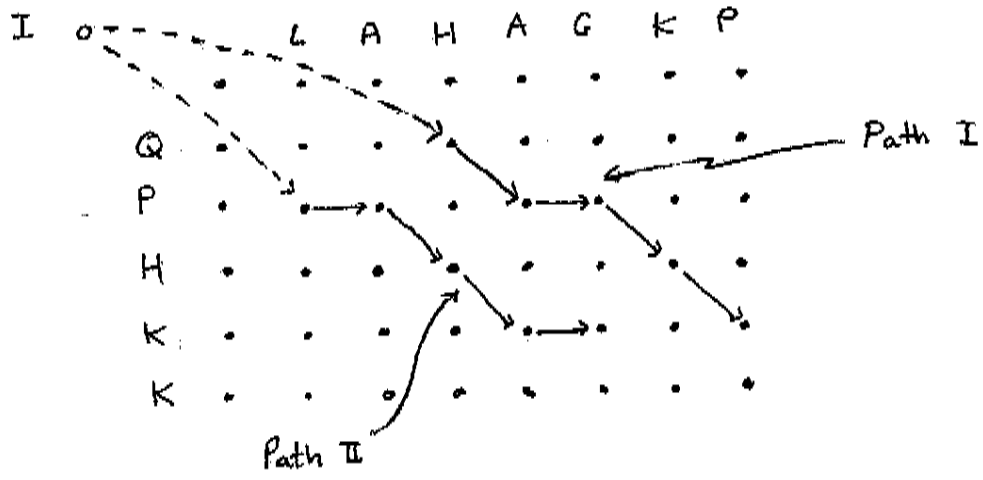
The dynamic programming principle is always the same. In each case, we define a function $F(v)$ on the vertices v of the graph: Always $F(I) = 0$. For each vertex v, let $\mathcal{P}(v)$ be the set of all vertices e such that the graph admits a directed edge $\vec{ev}$ from e to v; one can think of $\mathcal{P}(v)$ as the set of 'predecessors' of v. Let the weight of $\vec{ev}$ be denoted $W(\vec{ev})$. The dynamic programming equation will have the form:

$$F(v) = \max\{ F(e) + W(\vec{ev}) ;\ e \in \mathcal{P}(v) \}.$$

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 path representing local alignments start at I.

Example 7.6 Representing local alignments with paths starting from I.



Path I represents the local alignment
A G K P
P – H K .
It has score

$$0 + s(P,A) - d + s(H,K) + s(K,P)$$

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

Path II represents the local alignment
A H A G
– H K –
The path score is $0 - d + s(H,H) + s(K,A) - d$, which is the score of the local alignment.

We now define the value function L.

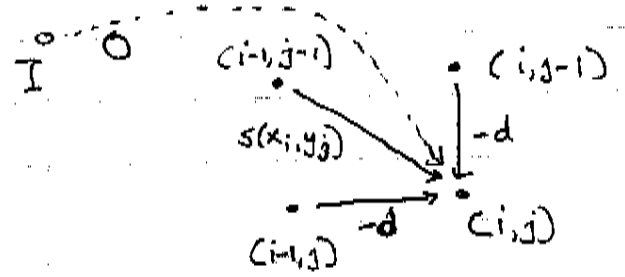
(i) $L(I) \triangleq 0$

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

$L(i,j) \triangleq$ maximum score of all paths from I to (i,j)

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

Case 1 $i \geq 1$, $j \geq 1$. The picture is



Using the dynamic programming reasoning

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

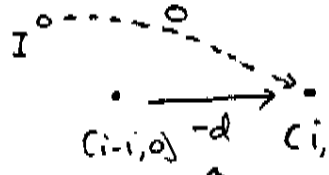
$$= \max \{0,\ L(i-1,j-1) + s(x_i,y_j),\ L(i-1,j) - d,\ L(i,j-1) - d\}$$

(DPEL$_1$)

since $F(I) = 0$.

Case (ii) $i \geq 0,\ j = 0$

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



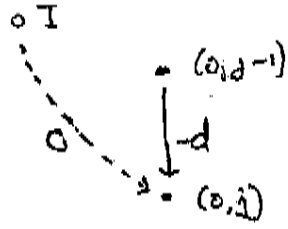
(Note: this edge does not exist if $i=0$

Then

$$(DPEL_2)\quad \begin{cases} L(i,0) = \max\{L(I)+0,\ L(i-1,0)-d\} \\ \qquad\quad = \max\{0,\ L(i-1,0)-d\} & \text{if } i \geq 1 \\ L(0,0) = 0 \end{cases}$$

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

$$(DPEL_3)\quad L(0,j) = \max\{0,\ L(j-1,0)-d\},\ j \geq 1$$



Algorithm to find maximum scoring paths.

Follow closely the prescription for the global alignment algorithm:

1) Solve $(DPEL_1)$, $(DPEL_2)$, $(DPEL_3)$ recursively for L, recording tracebacks
2) Search the table of values $L(i,j)$ 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 $S(a,a) > 0$ for any letter where $\{S(a,b)\}$ is the substitution matrix.

Suppose that no letter in the sequence $x = x_1 \cdots x_n$ matches a letter in $y = y_1 \cdots y_m$. Then it could be possible that every local alignment has negative score. However, the algorithm proposed above based on $(DPEL_1) - (DPEL_3)$ 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

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 example below.

4. The dynamic programming equations $(DPEL_2)$ and $(DPEL)_3$ lead to the following result for any two sequences $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_m$:

$$L(0,j) = 0 \qquad 0 \le j \le m$$
$$L(i,0) = 0 \qquad 0 \le i \le n$$

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

	.	x_1	x_2	x_3	. . .	x_n
.	0	0	0	0	. . . .	0
y_1	0					
y_2	0					
.	.					
.	.					
.	.					
y_m	0					

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

$$L(1,0) = \max\{0, L(0,0) - d\} = \max\{0, -d\} = 0$$
$$L(2,0) = \max\{0, L(1,0) - d\} = \max\{0, -d\} = 0$$

etc.

and from $(DPEL_3)$

$$L(0,1) = \max\{0, L(0,0) - d\} = \max\{0, -d\} = 0$$
$$L(0,2) = \max\{0, L(0,1) - d\} = 0$$

etc.

Therefore, in doing a local alignment, we can always begin by entering 0's in the 0^{th} row and 0^{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 = AAQCCDN and y = ACCQ. 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 $L(i,1)$, $i \geq 1$ (the next row in the table): using $(DPEL_1)$

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

$$L(2,1) = \max\{0, L(1,0) + S(A,A), L(1,1) - 6, L(2,0) - 6\}$$
$$= \max\{0, 0+5, 5-6, -6\} = 5$$

We enter these values in the table, with the tracebacks, and the remaining values of L in row 1, which the reader should verify.

		A	A	Q	C	C	D	N
	0	0	0	0	0	0	0	0
A	0	5	5	0	0	0	0	0
C	0							
C	0							
Q	0							

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 $$L(i,2) = \max\{0, L(i-1,1) + s(x_i, y_j), L(i,1)-6, L(i-1,2)-6\}$$

For example

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

[in this case the trace back is to I and is omitted]

We continue in this manner across rows and down columns.

The final result is

		A	A	Q	C	C	D	N
	0	0	0	0	0	0	0	0
A	0	5	5	0	0	0	0	0
C	0	0	4	2	13	13 ←	7 ←	1
C	0	0	0	1	15	26 ←	20 ←	14
Q	0	0	0	0	7	9	20	26 ← 20

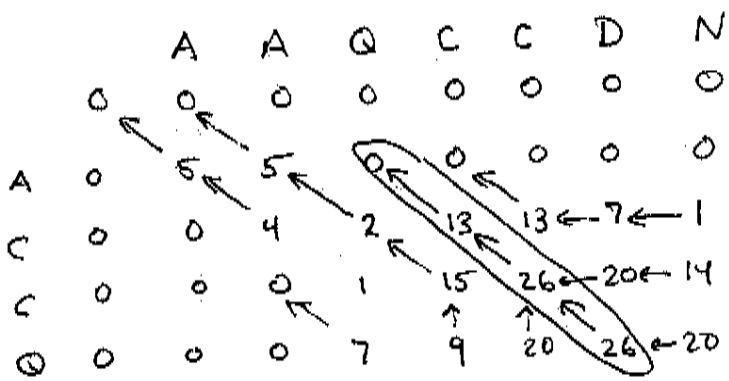
The largest value of L in this table is 26 and is achieved at (5, 3) and at (6, 4). Trace backs from either of those two vertices give optimal local alignments

In the next two tables we circle each of those alignments

		A	A	Q	C	C	D	N
	0	0	0	0	0	0	0	0
A	0	5	5	0	0	0	0	0
C	0	0	4	2	13	13 ←	7 ←	1
C	0	0	0	1	15	26 ←	20 ←	14
Q	0	0	0	0	7	9	20	26 ← 20

The local alignment circled here is

C C
C C



The local alignment circled here is C C N / C C Q

Conclusion: We find two optimal local alignments, $\begin{matrix} C & C \\ C & C \end{matrix}$, and $\begin{matrix} C & C & N \\ C & C & Q \end{matrix}$, 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 leftmost column.)

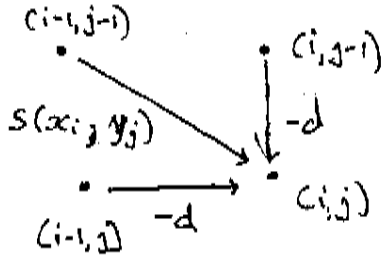
We define

$$M(I) \triangleq 0$$

$M(i,j) \triangleq$ maximum score of all paths starting from vertex I 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 $M(j,0) = 0$, $j \geq 0$. Thus for the overlap alignments algorithm one can start by placing 0's in the topmost row and left most column.

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



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

$$M(i,j) = \max\{M(i-1,j-1) + s(x_i, y_j),\ M(i-1,j) - d,\ M(i,j-1) - d\} \qquad (DPEO)$$

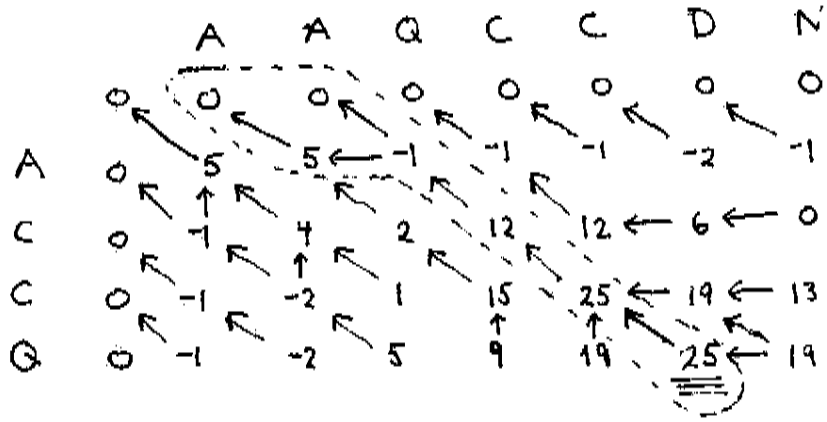
Algorithm for maximum scoring overlap alignments

1. Solve for $M(i,j)$, $0 \le i \le n$, $0 \le j \le m$ using (DPEO) and $M(i,0) = M(0,j) = 0$ for all i and j. Record tracebacks
2. An overlap alignment must end on a vertex of the bottom row or rightmost column. Examine the values

$$M(n,0)$$
$$M(n,1)$$
$$\vdots$$
$$M(n,j)$$
$$\vdots$$
$$M(0,m) \quad \cdots \quad M(i,m) \quad \cdots \quad M(n-1,m) \quad M(n,m)$$

in the bottom row and rightmost column and find the vertex which maximizes M among all those values. The traceback from these vertices defines the best overlap alignment.

Example 7.8. We perform the overlap algorithm for the sequences $x = AAQCCDN$ 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 right-most column and circled the traceback

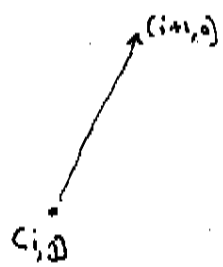
paths from this vertex. We find two overlap alignments each with score 25 :

```
(A A) Q C C D (N)        (A) A Q C C D (N)
      A C C Q      ,           A - C C Q      .
```

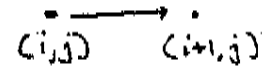
■

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 $\mathcal{T} - T$ on ending a match ($T > 0$). This means an edge of the type



has weight $-T$, if $i \geq 1$

The edges along the top row 

, however, fill in between matched subsegments of $x_1 x_2 \cdots x_n$ and receive 0 weight.

Finally, the edge 

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)

$$0 + S(H,H) + S(A,A) + S(P,P) - T + 0 + S(H,H) + S(A,A) - d + S(P,P)$$
$$- T + 0 + 0 + S(A,A) - d + S(G,G) - T .$$

□

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

$$R(0,0) = 0$$

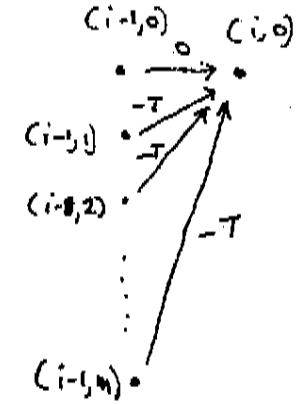
$R(i,j)$ = maximum score of all repeat match paths starting at $(0,0)$ and ending at (i,j).

We get

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

(ii) For $i \geq 1$,

$$R(i,0) = \max \begin{cases} R(i-1,0) \\ R(i,1) - T \\ R(i,2) - T \\ \vdots \\ R(i,m) - T \end{cases}$$



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

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

(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 the ideas of section 7.2.)

Remember that for the affine gap penalty method we set parameters $d > e > 0$ and give a sequence of g consecutive alignments of a letter with a gap weight

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

---

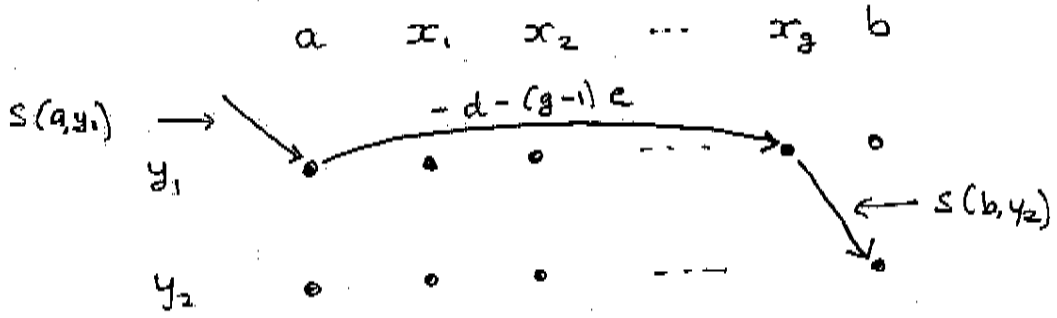
* See Durbin, Eddy, Krogh, Mitchison, Biological Sequence Analysis Cambridge Univ. Press 1998, chapter 2

Something of the form

$$\begin{array}{lll} & a\ x_1 x_2 \cdots x_g\ b & \\ \cdots & y_1 - - \quad - y_2 & \cdots \end{array} \qquad (*)$$

therefore contributes $s(a,y_1) - [d+(g-1)e] + s(b,y_2)$ to the overall score.

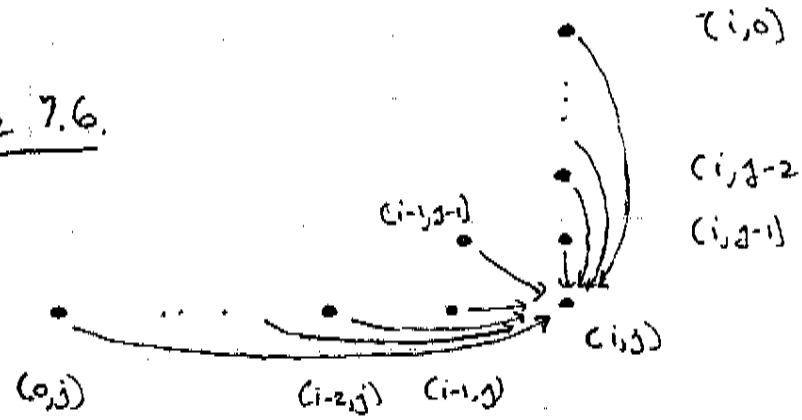
To handle this we will add directed edges to the edit graph to represent a sequence of consecutive gaps and give the edges appropriate weights. For example, to handle the segment shown in (*) we would use edges as follows



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

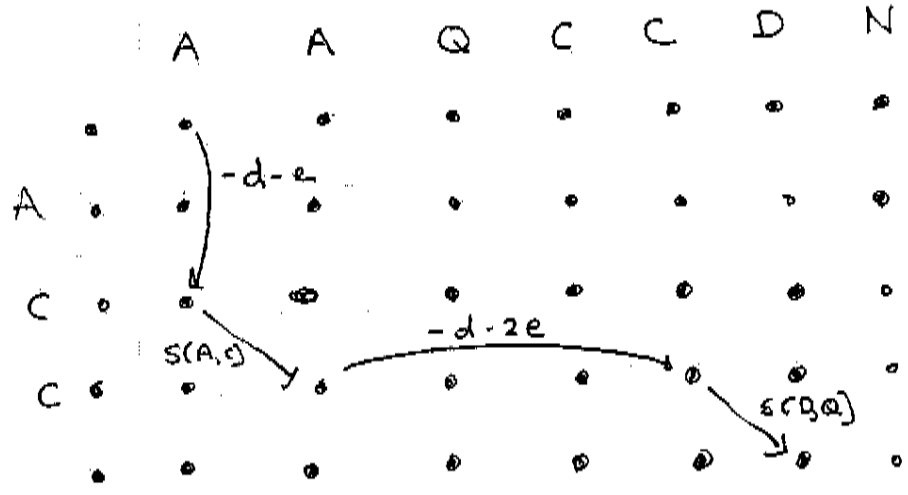
Following this idea we draw a picture showing the 'predecessors' of vertex (i,j) that we now include in the revised edit graph.

Figure 7.6.



Any horizontal or vertical edge of length g gets weight $-[d+(g-1)e]$

To return to our favorite example consider the paths in:



The path shown represents the alignment

$$\begin{array}{cccccccc} - & - & A & Q & C & C & D \\ A & C & C & - & - & - & Q \end{array}$$

and it has score

$$-[d+e] + s(A,C) - [d+2e] + s(D,Q)$$

Dynamic programming for global alignment:

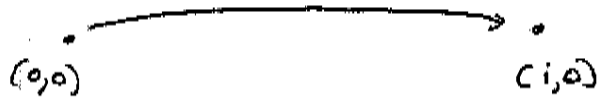
Let $G(0,0) \triangleq 0$

$G(i,j)$ = 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.

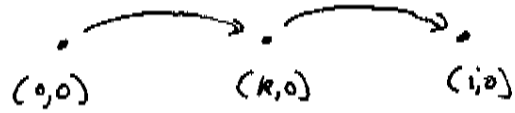
First we derive the values of $G(i,0)$, $i \geq 1$, directly from the definition.

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

This value is achieved by the edge



According to our definitions we could also imagine a path



But this has weight $-[d + (k-1)e] - [d + (i-k-1)e]$

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 refering to

Figure 7.6 on the bottom of page 36 we obtain

$$G(i,j) = \max \begin{cases} G(i-1, j-1) + s(x_i, y_j) \\ G(i-1,j) - d \\ G(i-2,j) - [d+e] \\ G(i-3,j) - [d+2e] \\ \vdots \\ G(0,j) - [d+(i-1)e] \\ G(i,j-1) - d \\ G(i,j-2) - [d+e] \\ \vdots \\ G(i,0) - [d+(j-1)e] \end{cases}$$

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.