Notes on Dynamic-Programming Sequence Alignment

Introduction. Following its introduction by Needleman and Wunsch (1970), dynamic programming has become the method of choice for ''rigorous'' alignment of DNA and protein sequences. For a number of useful alignment-scoring schemes, this method is guaranteed to produce an alignment of two given sequences having the highest possible score.

For alignment scores that are popular with molecular biologists, dynamic-programming alignment of two sequences requires quadratic time, i.e., time proportional to the product of the two sequence lengths. In particular, this holds for *affine gap costs*, that is, scoring schemes under which a gap of length k is penalized $g + ek$, where g is a fixed ''gap-opening penalty'' and e is a ''gap-extension penalty'' (Gotoh, 1982). (More general alignment scores, which are more expensive to optimize, were considered by Waterman *et al*., 1976, but have not found wide-spread use.) Quadratic time is necessitated by the inspection of every pair (i, j), where i is a position in the first sequence and j is a position in the second sequence. For many applications, e.g., database searches, such an exhaustive examination of position pairs may not be worth the effort, and a number of faster methods have been proposed.

For long sequences, computer memory is another limiting factor, but very space-efficient versions of dynamic programming are possible. The original formulation (Hirschberg, 1975) was for an alignment-scoring scheme that is too restrictive to be of general utility in molecular biology, but the basic idea is quite robust and works readily for affine gap penalties (Myers and Miller, 1988).

The Dynamic-Programming Alignment Algorithm. It is quite helpful to recast the problem of aligning two sequences as an equivalent problem of finding a maximum-score path in a certain graph, as has been observed by a number of authors, including Myers and Miller (1989). This alternative formulation allows the problem to be visualized in a way that permits the use of geometric intuition. We find this visual imagery critical for keeping track of the low-level details that arise in development and implementation of dynamic-programming alignment algorithms.

An *alignment* of two sequences, say S and T, is a rectangular array of symbols having two rows, such that removing all dash characters from the first row (if any are there) gives S, and removing dashes from the second row gives T. Also, we do not allow columns containing two dash symbols. For instance,

```
AAGCAA-A
A-GCTACA
```

is an alignment of `AAGCAAA` and `AGCTACA`.

For the current discussion, we assume the following simple alignment-scoring scheme. For each possible aligned pair $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, where each of x and y is either a normal sequence entry or the symbol ''–'', there is an assigned score $\sigma(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])$. The score of a pairwise alignment is defined to be the sum of the σ-values of its aligned pairs (i.e., columns). For instance if we score each match (i.e., column of identical symbols) 1, and each other column −1, then the above alignment scores $1 - 1 + 1 + 1 - 1 + 1 - 1 + 1 = 2$.

Recall that a directed graph $G = (V, E)$ consists of a set V of *nodes* (also called *vertices*) and a set E of *edges*. The edge from node u to node v, if it exists, is denoted $u \to v$. A sequence of consecutive edges $u_1 \to u_2$, $u_2 \to u_3$, $\cdots$, $u_{k-1} \to u_k$ is a *path* from u_1 to u_k. If each edge $u \to v$ is assigned a score $\sigma(u \to v)$, then the *score* of such a path is $\sum_{i=1}^{k-1} \sigma(u_i \to u_{i+1})$.

We now describe the relationship between maximum-score paths and optimal alignments. Consider two sequences, $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$. That is, A contains M symbols and B contains N symbols, where the symbols are from an arbitrary "alphabet" that does not contain the dash symbol, "−". The *alignment graph* for A and B, denoted $G_{A,B}$, is an edge-labeled directed graph. The nodes of $G_{A,B}$ are the pairs (i, j) where $i \in [0, M]$ and $j \in [0, N]$. (We use the notation $[p, q]$ for the set $\{p, p+1, \cdots, q-1, q\}$.) When graphed, these nodes are arrayed in $M+1$ rows (row i corresponds to a_i for $i \in [1, M]$, with an additional row 0) and $N+1$ columns (column j corresponds to b_j for $j \in [1, N]$). The edge set for $G_{A,B}$ consists of the following edges, labeled as indicated.

1. $(i-1, j) \rightarrow (i, j)$ for $i \in [1, M]$ and $j \in [0, N]$, labeled $\begin{bmatrix} a_i \\ - \end{bmatrix}$
2. $(i, j-1) \rightarrow (i, j)$ for $i \in [0, M]$ and $j \in [1, N]$, labeled $\begin{bmatrix} - \\ b_j \end{bmatrix}$
3. $(i-1, j-1) \rightarrow (i, j)$ for $i \in [1, M]$ and $j \in [1, N]$, labeled $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$

Fig. 1 provides an example of the construction.

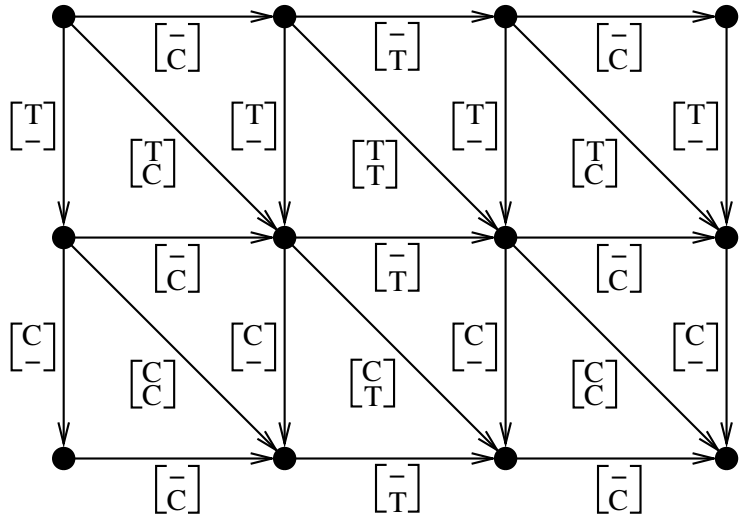

FIG. 1. Alignment graph $G_{A,B}$ for the sequences $A = TC$ and $B = CTC$.

It is instructive to look for a path from $(0, 0)$ (the upper left corner of the graph of Fig. 1) to $(2, 3)$ (the lower right) such that the labels along the path "spell" the alignment:

```
-TC
CTC
```

The first aligned pair is $\begin{bmatrix} - \\ C \end{bmatrix}$, so the first edge must be horizontal. The second pair is $\begin{bmatrix} T \\ T \end{bmatrix}$, so the second edge must be diagonal. The third pair is $\begin{bmatrix} C \\ C \end{bmatrix}$, so the third edge must be diagonal. Generally, when a path descends from row $i-1$ to row i, it picks up an aligned pair with top entry a_i. A path from $(0, 0)$ to (M, N) has zero or more horizontal edges, then a vertical or diagonal edges to row 1, then zero or more horizontal edges, then an edge to row 2, then $\cdots$, so the top entries of the labels along the path are a_1, a_2, $\cdots$, possibly with some interspersed dashes. Similarly, the bottom entries spell B if dashes are ignored, so the aligned pairs spell an alignment of A and B. Indeed, alignments are in general equivalent to paths, as we now state more precisely.

Fact: Let $G_{A,B}$ be the alignment graph for sequences A and B. With each path from $(0, 0)$ to (M, N) associate the alignment formed by concatenating the edge labels along the path, i.e.,

the alignment ''spelled'' by the path. Then every such path determines an alignment of A and B, and every alignment of A and B is determined by a unique path. In other words, there is a one-to-one correspondence between paths in $G_{A,B}$ from $(0,0)$ to (M,N) and alignments of A and B. Furthermore, if the score $\sigma(\pi)$ is assigned to each edge of $G_{A,B}$, where π is the aligned pair labeling that edge, then a path's score is exactly the score of the corresponding alignment.

At each node, the score is computed from the scores of immediate predecessors and of entering edges, which are pictured in Fig. 2. The procedure of Fig. 3 computes the maximum alignment score by considering rows of $G_{A,B}$ in order, sweeping left to right within each row. $S[i, j]$ denotes the maximum score of a path from $(0,0)$ to (i, j). Lines 7-10 mirror Fig. 2. In row 0 there is but a single edge entering a node (lines 2-3), and similarly for column 0 (line 5). This is a quadratic-space procedure since it uses the $(M+1)$-by-$(N+1)$ array S to hold all node-scores.

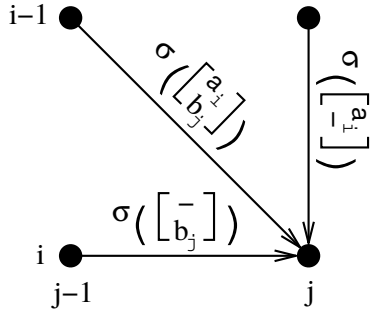

FIG. 2. Edges entering node (i, j) and their scores.

```
1.   S[0,0] ← 0
2.   for j ← 1 to N do
3.        S[0, j] ← S[0, j − 1] + σ(⎡ − ⎤)
                                   ⎣ b_j ⎦
4.   for i ← 1 to M do
5.        S[i,0] ← S[i − 1,0] + σ(⎡ a_i ⎤}
                                  ⎣ −   ⎦
6.        for j ← 1 to N do
7.             Vertical ← S[i − 1, j] + σ(⎡ a_i ⎤)
                                         ⎣ −   ⎦
8.             Diagonal ← S[i − 1, j − 1] + σ(⎡ a_i ⎤)
                                              ⎣ b_j ⎦
9.             Horizontal ← S[i, j − 1] + σ(⎡ − ⎤)
                                            ⎣ b_j ⎦
10.            S[i, j] ← max{Vertical, Diagonal, Horizontal}
11.  write "Maximum alignment score is" S[M, N]
```

FIG. 3. Quadratic-space, score-only alignment algorithm.

The next step is to see that the optimal alignment score for A and B can be computed in linear space. Indeed, it is apparent that the scores in row i of S depend only on those in row $i-1$. Thus, after treating row i, the space used for values in row $i-1$ can be recycled to hold values in row $i+1$. In other words, we can get by with space for two rows, since all that we ultimately want is the single score $S[M, N]$.

In fact, a single array, $S[0..N]$, is adequate. $S[j]$ holds the most recently computed value in column j, so that as values of S are computed, they overwrite old values. There is a slight conflict in this strategy, since two ''active'' values are needed in the current