# 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.

For the current discussion, we assume the following simple alignment-scoring scheme. For each possible aligned pair $\begin{bmatrix} x \\ y \end{bmatrix}$, where each of $x$ and $y$ is either a normal sequence entry or the symbol "–", there is an assigned score $\sigma(\begin{bmatrix} x \\ y \end{bmatrix})$. The score of a pairwise alignment is defined to be the sum of the $\sigma$-values of its aligned pairs (i.e., columns).

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. $(i-1, j) \to (i, j)$ 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.

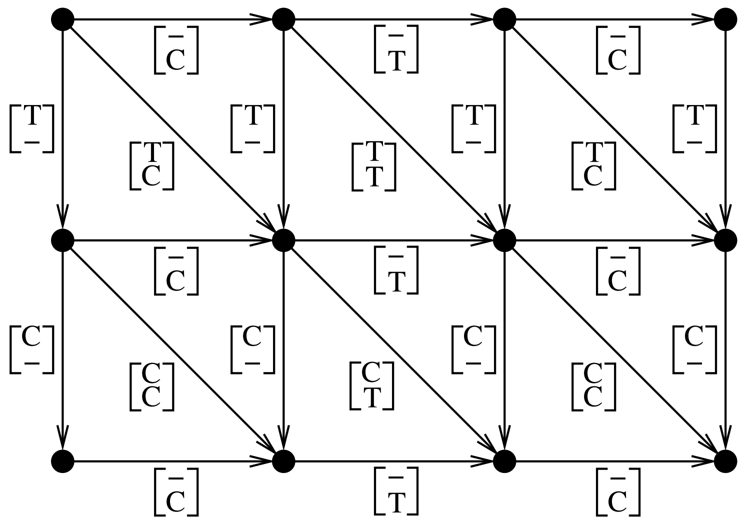


**FIG. 1.** Alignment graph $G_{A,B}$ for the sequences $A = TC$ and $B = CTC$.

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 $\pi$ 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.

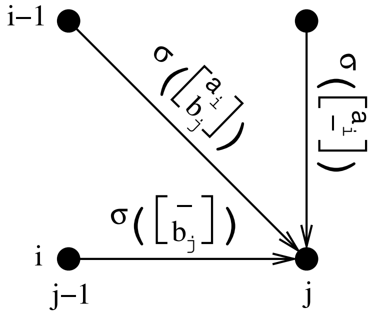


**FIG. 2.** Edges entering node $(i, j)$ and their scores.

```
S[0,0] ← 0
for j ← 1 to N do
        S[0, j] ← S[0, j − 1] + σ([− ; b_j])
for i ← 1 to M do
        S[i,0] ← S[i − 1,0] + σ([a_i ; −]}
        for j ← 1 to N do
                Vertical ← S[i − 1, j] + σ([a_i ; −])
                Diagonal ← S[i − 1, j − 1] + σ([a_i ; b_j])
                Horizontal ← S[i, j − 1] + σ([− ; b_j])
                S[i, j] ← max{Vertical, Diagonal, Horizontal}
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 column, necessitating an additional scalar, $s$, to hold one of them. Fig. 4 shows the grid locations of values in $S$ and of scalars $s$ and $c$ when $(i, j)$ is reached in the computation. $S[k]$ holds path scores for row $i$ when $k < j$, and for row $i-1$ when $k \geq j$. Fig. 5 is a direct translation of Fig. 3 using the memory-allocation scheme of Fig. 4.

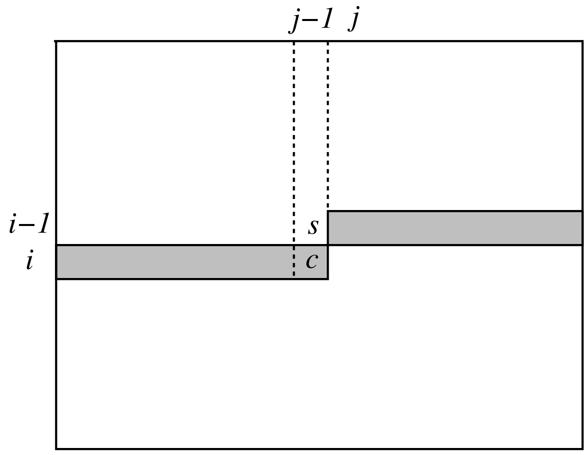


**FIG. 4.** Grid locations of entries of a vector of length $N+1$ just before the maximum path-score is evaluated at node $(i, j)$. Additionally, a scalar $s$ holds the path score at $(i-1, j-1)$ and $c$ holds the score at $(i, j-1)$.

$S[0] \leftarrow 0$
**for** $j \leftarrow 1$ **to** $N$ **do**
  $S[j] \leftarrow S[j-1] + \sigma(\left[\begin{smallmatrix} - \\ b_j \end{smallmatrix}\right])$
**for** $i \leftarrow 1$ **to** $M$ **do**
  $s \leftarrow S[0]$
  $S[0] \leftarrow c \leftarrow S[0] + \sigma(\left[\begin{smallmatrix} a_i \\ - \end{smallmatrix}\right])$
  **for** $j \leftarrow 1$ **to** $N$ **do**
    $c \leftarrow \max\{S[j] + \sigma(\left[\begin{smallmatrix} a_i \\ - \end{smallmatrix}\right]), s + \sigma(\left[\begin{smallmatrix} a_i \\ b_j \end{smallmatrix}\right]), c + \sigma(\left[\begin{smallmatrix} - \\ b_j \end{smallmatrix}\right])\}$
    $s \leftarrow S[j]$
    $S[j] \leftarrow c$
**write** "Maximum alignment score is" $S[N]$

**FIG. 5.** Linear-space computation of alignment scores.

We will soon need to perform this computation in the reverse direction. Here, the relevant edges are the ones *leaving* node $(i, j)$, as pictured in Fig. 6, and the quadratic-space algorithm is given in Fig. 7. A slight generalization of a linear-space version of Fig. 7 appears in lines 26-35 of Fig. 9; its derivation is left as an exercise for the reader.

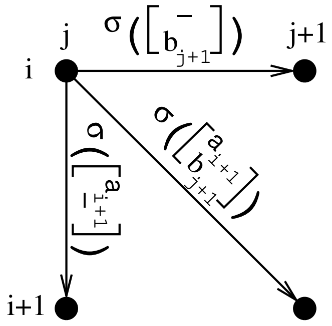


**FIG. 6.** Edges leaving $(i, j)$ and their scores.

$S[M, N] \leftarrow 0$
**for** $j \leftarrow N-1$ **down to** $0$ **do**
    $S[M, j] \leftarrow S[M, j+1] + \sigma(\begin{bmatrix} - \\ b_{j+1} \end{bmatrix})$
**for** $i \leftarrow M-1$ **down to** $0$ **do**
    $S[i, N] \leftarrow S[i+1, N] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix})$
    **for** $j \leftarrow N-1$ **down to** $0$ **do**

$$S[i, j] \leftarrow \max \begin{cases} S[i+1, j] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix}) \\ S[i+1, j+1] + \sigma(\begin{bmatrix} a_{i+1} \\ b_{j+1} \end{bmatrix}) \\ S[i, j+1] + \sigma(\begin{bmatrix} - \\ b_{j+1} \end{bmatrix}) \end{cases}$$

**write** "Maximum alignment score is" $S[0, 0]$

**FIG. 7.** Computation of alignment scores in the reverse direction.

**Hirshberg's Insight.** We are now ready to describe Hirschberg's linear-space alignment algorithm; the algorithm delivers an explicit optimal alignment, not merely its score. First, make a ''forward'' score-only pass (Fig. 5), stopping at the middle row, i.e., row $mid = \lfloor M/2 \rfloor$. Then make a backward score-only pass (the linear-space version of Fig. 7), again stopping at the middle row. Thus, for each point along the middle row, we now have the optimal score from $(0, 0)$ to that point and the optimal score from that point to $(M, N)$. Adding those numbers gives the optimal score over all paths from $(0, 0)$ to $(M, N)$ that pass through that point. A sweep along the middle row, checking those sums, determines a point $(mid, j)$ where an optimal path crosses the middle row. This reduces the problem to finding an optimal path from $(0, 0)$ to $(mid, j)$ and an optimal path from $(mid, j)$ to $(M, N)$, which is done recursively.

Fig. 8A shows the two subproblems and each of their ''subsubproblems''. Note that regardless of where the optimal path crosses the middle row, the total of the sizes of the two subproblems is just half the size of the original problem, where problem size is measured by the number of nodes. Similarly, the total sizes of all subsubproblems is a fourth the original size. Letting $T$ be the size of the original, it follows that the total sizes of all problems, at all levels of recursion, is at most $T + ½T + ¼T \cdots = 2T$. Since computation time is directly proportional to the problem size, this approach will deliver an optimal alignment in about twice the time needed to compute merely its score.

Fig. 8B shows a typical point in the alignment process. The initial portion of an optimal path will have been determined, and the current problem is to report the aligned pairs along an optimal path from $(i_1, j_1)$ to $(i_2, j_2)$. Fig. 9 provides detailed pseudo-code for the linear-space alignment algorithm.

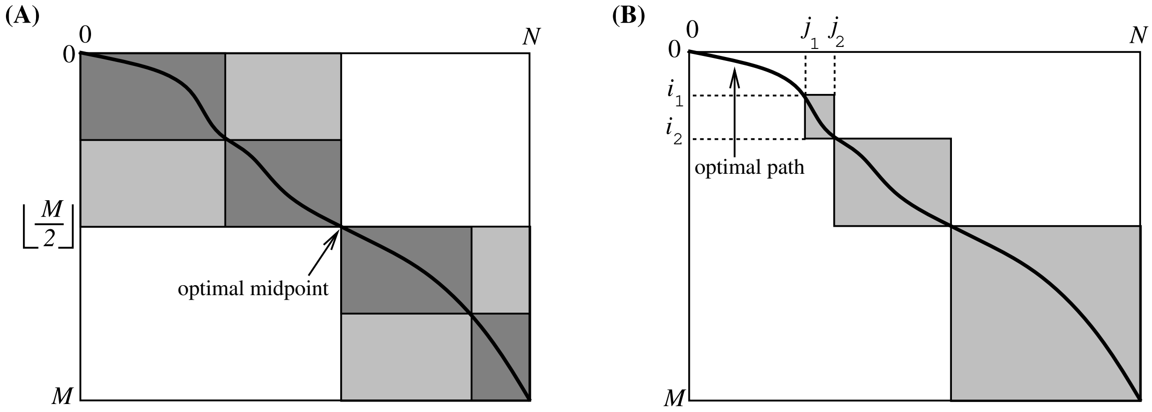


**FIG. 8. (A)** The two subproblems and four subsubproblems in Hirschberg's linear-space alignment procedure. **(B)** Snapshot of the execution of Hirschberg's algorithm. Shaded areas indicate problems remaining to be solved.

**shared strings** $a_1 a_2 \cdots a_M$, $b_1 b_2 \cdots b_N$
**shared temporary integer arrays** $S^-[0..N]$, $S^+[0..N]$

**procedure** $Align(M, N)$
  **if** $M = 0$ **then**
    **for** $j \leftarrow 1$ **to** $N$ **do**
      **write** $\left[\begin{smallmatrix} - \\ b_j \end{smallmatrix}\right]$
  **else**
    $path(0, 0, M, N)$

**recursive procedure** $path(i_1, j_1, i_2, j_2)$
  **if** $i_1 + 1 = i_2$ **or** $j_1 = j_2$ **then**
    **write** aligned pairs for maximum-score path from $(i_1, j_1)$ to $(i_2, j_2)$
  **else**
    $mid \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$
    /* find maximum path scores from $(i_1, j_1)$ */
    $S^-[j_1] \leftarrow 0$
    **for** $j \leftarrow j_1 + 1$ **to** $j_2$ **do**
      $S^-[j] \leftarrow S^-[j-1] + \sigma(\left[\begin{smallmatrix} - \\ b_j \end{smallmatrix}\right])$
    **for** $i \leftarrow i_1 + 1$ **to** $mid$ **do**
      $s \leftarrow S^-[j_1]$
      $S^-[j_1] \leftarrow c \leftarrow S^-[j_1] + \sigma(\left[\begin{smallmatrix} a_i \\ - \end{smallmatrix}\right])$
      **for** $j \leftarrow j_1 + 1$ **to** $j_2$ **do**
        $c \leftarrow \max\{S^-[j] + \sigma(\left[\begin{smallmatrix} a_i \\ - \end{smallmatrix}\right]), s + \sigma(\left[\begin{smallmatrix} a_i \\ b_j \end{smallmatrix}\right]), c + \sigma(\left[\begin{smallmatrix} - \\ b_j \end{smallmatrix}\right])\}$
        $s \leftarrow S^-[j]$
        $S^-[j] \leftarrow c$
    /* find maximum path scores to $(i_2, j_2)$ */
    $S^+[j_2] \leftarrow 0$
    **for** $j \leftarrow j_2 - 1$ **down to** $j_1$ **do**
      $S^+[j] \leftarrow S^+[j+1] + \sigma(\left[\begin{smallmatrix} - \\ b_{j+1} \end{smallmatrix}\right])$
    **for** $i \leftarrow i_2 - 1$ **down to** $mid$ **do**
      $s \leftarrow S^+[j_2]$
      $S^+[j_2] \leftarrow c \leftarrow S^+[j_2] + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ - \end{smallmatrix}\right])$
      **for** $j \leftarrow j_2 - 1$ **down to** $j_1$ **do**
        $c \leftarrow \max\{S^+[j] + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ - \end{smallmatrix}\right]), s + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ b_{j+1} \end{smallmatrix}\right]), c + \sigma(\left[\begin{smallmatrix} - \\ b_{j+1} \end{smallmatrix}\right])\}$
        $s \leftarrow S^+[j]$
        $S^+[j] \leftarrow c$
    /* find where maximum-score path crosses row $mid$ */
    $j \leftarrow$ value $x \in [j_1, j_2]$ that maximizes $S^-[x] + S^+[x]$
    $path(i_1, j_1, mid, j)$
    $path(mid, j, i_2, j_2)$

**FIG. 9.** Linear-space alignment algorithm.

**Local Alignment.** In many applications, a global (i.e., end-to-end) alignment of the two given sequences is inappropriate; instead, a local alignment (i.e., involving only a part of each sequence) is desired. In other words, one seeks a high-scoring path that need not terminate at the corners of the dynamic-programming grid (Smith and Waterman, 1981). The highest local alignment score can be computed as follows:

$$S[i,j] \leftarrow \max \begin{cases} 0 & \text{if } 0 \le i \le M \text{ and } 0 \le j \le N \\ S[i-1,j] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}) & \text{if } 1 \le i \le M \text{ and } 0 \le j \le N \\ S[i-1,j-1] + \sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix}) & \text{if } 1 \le i \le M \text{ and } 1 \le j \le N \\ S[i,j-1] + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix}) & \text{if } 0 \le i \le M \text{ and } 1 \le j \le N \end{cases}$$

A single highest-scoring alignment can be found by locating the alignment's end points (which is straightforward to do in linear space), then applying Hirschberg's strategy to the two substrings bracketed by those points.

Further complications arise when one seeks $k$ best alignments, where $k > 1$. For computing an arbitrary number of non-intersecting and high-scoring local alignments, Waterman and Eggert (1987) developed a very time-efficient method. Producing a linear-space variant of their algorithm requires ideas that differ significantly from those presented in previous sections (Huang and Miller, 1991; Huang *et al*., 1990).

## REFERENCES


Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162**, 705-708.

Hirschberg, D. S. (1975) A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, **18**, 341-343.

Huang, X., R. Hardison and W. Miller (1990) A space-efficient algorithm for local similarities. *CABIOS* **6**, 373-381.

Huang, X. and W. Miller (1991) A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics* **12**, 337-357.

Myers, E. and W. Miller (1988) Optimal alignments in linear space. *CABIOS* **4**, 11-17.

Myers, E. and W. Miller (1989) Approximate matching of regular expressions. *Bull. Math. Biol.* **51**, 5-37.

Needleman, S. B. and C. D. Wunsch (1970) A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* **48**, 443-453.

Smith, T. F. and M. S. Waterman (1981) Identification of common molecular sequences. *J. Mol. Biol.* **197**, 723-728.

Waterman, M. S., T. F. Smith and W. A. Beyer (1976) Some biological sequence metrics. *Adv. Math.* **20**, 367-387.

Waterman, M. S. and M. Eggert (1987) A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *J. Mol. Biol.* **197**, 723-728.