Recent Developments in Linear-Space Alignment Methods: A Survey

KUN-MAO CHAO,[1] ROSS C. HARDISON,[2] and WEBB MILLER[1,†]

ABSTRACT

A dynamic-programming strategy for sequence alignment first proposed in 1975 by Dan Hirschberg can be adapted to yield a number of extremely space-efficient algorithms. Specifically, these algorithms align two sequences using only "linear space", i.e., an amount of computer memory that is proportional to the sum of the lengths of the two sequences being aligned. This paper begins by reviewing the basic idea, as it applies to the global (i.e., end-to-end) alignment of two DNA or protein sequences. Three of our recent extensions of the technique are then outlined. The first extension computes an optimal alignment subject to the constraint that each position, i, of the first sequence must be aligned somewhere between positions $L[i]$ and $U[i]$ of the second sequence, for given values of L and U. The second finds all aligned position pairs (i.e., potential columns of the alignment) that occur in an alignment whose score exceeds a given threshold. The third treats the case where each of the two sequences is allowed to be an alignment (e.g., a sequence of aligned pairs), using a sensitive scoring scheme. We also describe two linear-space methods for computing k best local (i.e., involving only a part of each sequence) alignments, where $k \geq 1$. One is a linear-space version of the algorithm of Waterman and Eggert (1987), and the other is based on the strategy proposed by Wilbur and Lipman (1983). Finally, we describe programs that implement various combinations of these techniques to provide a multi-sequence alignment method that is especially suited to handling a few very long sequences. The utility of these programs is illustrated by analysis of the locus control region of the β-like globin gene cluster of several mammals.

Key Words: sequence analysis, dynamic programming, linear-space algorithm, multiple 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.

[1]Department of Computer Science and Engineering, [2]Department of Biochemistry and Molecular Biology, The Pennsylvania State University, University Park, PA 16802.
[†]Corresponding author.

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.

Standard implementations of dynamic-programming alignment also require a quadratic amount of computer memory to explicitly produce a highest-scoring alignment. (Computing just the largest alignment score in less space is straightforward.) For certain applications, such as careful analysis of a few long DNA sequences, this space restriction is more important than the time constraint. For that reason, a number of methods have been proposed by biologists to decrease the amount of space allocated for each position pair, (i, j).

As it turns out, an alignment strategy that uses only linear space had earlier been published by a computer scientist. 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). We feel that the power of this approach is still not fully appreciated by biologists, and this survey is meant to assist in attracting the deserved recognition.

After developing the basic linear-space dynamic-programming alignment algorithm, we present the three recent extensions mentioned in the Abstract. Except for the last extension, the ideas are described for the simplification of affine gap costs where $g = 0$. Then two linear-space methods for computing k best non-intersecting local alignments for any $k \geq 1$ are outlined. These developments culminate in multi-sequence alignment algorithms that are the centerpiece of a software system we are building to analyze mammalian β-like globin gene clusters. This system, which is described near the end of the paper, is intended for eventual application to sequence data from other important genomic loci, when those data become available.

AVAILABILITY

The C source programs for our linear-space alignment methods are available by anonymous ftp from groucho.cse.psu.edu. Table 1 summarizes the available linear-space alignment programs mounted on the ftp site. The authors may be contacted by electronic mail to webb@groucho.cse.psu.edu.

Directory name	Function	References
band	Aligning within a diagonal band	Chao *et al*. (1992)
BMB	Constrained sequence alignment	Chao *et al*. (1993a)
robust	Locating well-conserved regions	Chao *et al*. (1993b)
sub	Computing all suboptimal alignments	Chao (1994)
sim	Local similarities	Huang and Miller (1991)
falign	Building local alignments from fragments	Chao and Miller (1994)
yama	Multiple sequence alignment	Hardison *et al*. (1993b, 1994)

Table 1. Summary of our linear-space alignment tools.

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.

In this and the next three sections 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 σ-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. $(i-1, j) \rightarrow (i, j)$ 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.

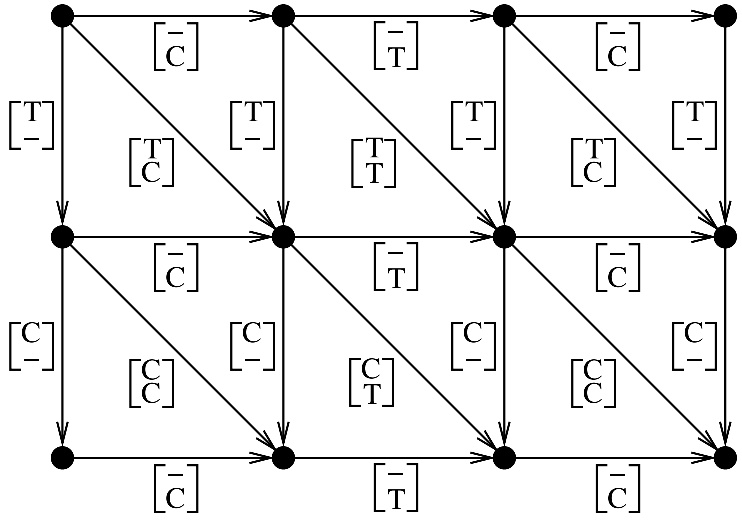

FIG. 1. Alignment graph $G_{A,B}$ for the sequences $A = TC$ and $B = CTC$.

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.

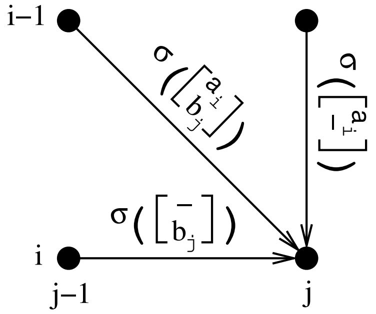

FIG. 2. Edges entering node (i, j) and their scores.

1. $S[0,0] \leftarrow 0$
2. **for** $j \leftarrow 1$ **to** N **do**
3. $\quad S[0, j] \leftarrow S[0, j-1] + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix})$
4. **for** $i \leftarrow 1$ **to** M **do**
5. $\quad S[i, 0] \leftarrow S[i-1, 0] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}\}$
6. $\quad$ **for** $j \leftarrow 1$ **to** N **do**
7. $\qquad Vertical \leftarrow S[i-1, j] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix})$
8. $\qquad Diagonal \leftarrow S[i-1, j-1] + \sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix})$
9. $\qquad Horizontal \leftarrow S[i, j-1] + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix})$
10. $\qquad S[i, j] \leftarrow \max\{Vertical, Diagonal, Horizontal\}$
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 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.

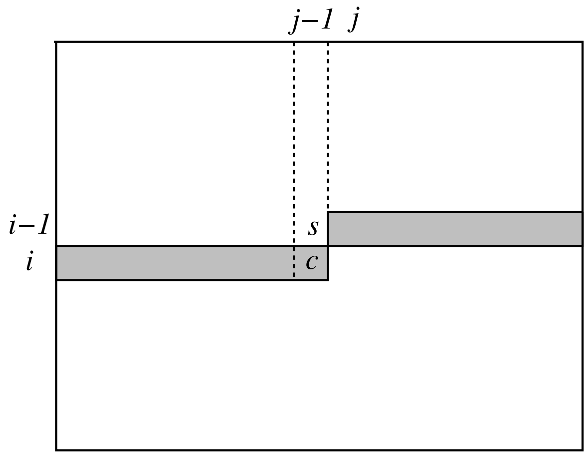

FIG. 4. Grid locations of entries of a vector of length $N+1$ just before the maximum path-score is evaluated at node (i, j). Additionally, a scalar s holds the path score at $(i-1, j-1)$ and c holds the score at $(i, j-1)$.

$S[0] \leftarrow 0$
for $j \leftarrow 1$ **to** N **do**
$\quad S[j] \leftarrow S[j-1] + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix})$
for $i \leftarrow 1$ **to** M **do**
$\quad s \leftarrow S[0]$
$\quad S[0] \leftarrow c \leftarrow S[0] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix})$
$\quad$ **for** $j \leftarrow 1$ **to** N **do**
$\quad\quad c \leftarrow \max\{S[j] + \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}),\ s + \sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix}),\ c + \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix})\}$
$\quad\quad s \leftarrow S[j]$
$\quad\quad S[j] \leftarrow c$
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.

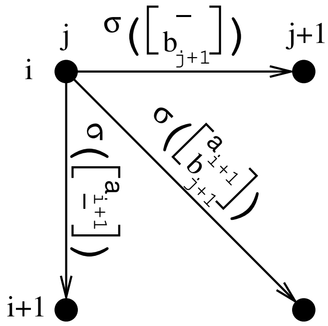

FIG. 6. Edges leaving (i, j) and their scores.

$$S[M,N] \leftarrow 0$$
$$\textbf{for } j \leftarrow N-1 \textbf{ down to } 0 \textbf{ do}$$
$$\qquad S[M,j] \leftarrow S[M,j+1] + \sigma(\begin{bmatrix} - \\ b_{j+1} \end{bmatrix})$$
$$\textbf{for } i \leftarrow M-1 \textbf{ down to } 0 \textbf{ do}$$
$$\qquad S[i,N] \leftarrow S[i+1,N] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix})$$
$$\qquad \textbf{for } j \leftarrow N-1 \textbf{ down to } 0 \textbf{ do}$$
$$\qquad\qquad S[i,j] \leftarrow \max \begin{cases} S[i+1,j] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix}) \\ S[i+1,j+1] + \sigma(\begin{bmatrix} a_{i+1} \\ b_{j+1} \end{bmatrix}) \\ S[i,j+1] + \sigma(\begin{bmatrix} - \\ b_{j+1} \end{bmatrix}) \end{cases}$$
$$\textbf{write} \text{ "Maximum alignment score is" } S[0,0]$$

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

HIRSCHBERG'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.

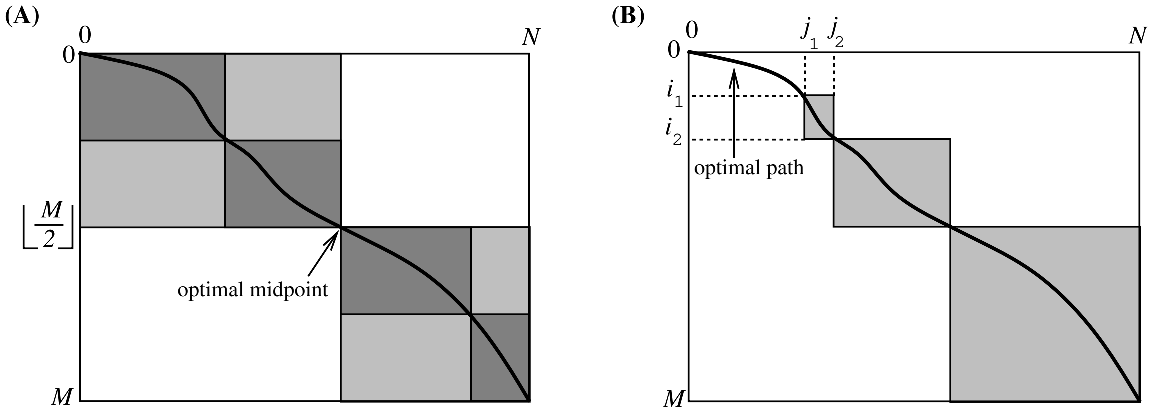

FIG. 8. (A) The two subproblems and four subsubproblems in Hirschberg's linear-space alignment procedure. **(B)** Snapshot of the execution of Hirschberg's algorithm. Shaded areas indicate problems remaining to be solved.

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.

ALIGNMENT WITHIN A NARROW REGION

Our first variant of Hirschberg's strategy computes an alignment when all operations are constrained to a narrow region of the dynamic-programming grid. In particular, suppose that $L[i]$ and $U[i]$ specify lower and upper bounds, respectively, on the columns to be considered in row i, for all relevant values of i. Of course, the algorithms of the preceding section can be modified to inspect only points satisfying these constraints, but the resulting methods are not as time-efficient as we would like.

The deficiency is most obvious in the "limiting case" where $L[i] = U[i] = i$ for all i, i.e., when the region degenerates to a diagonal line. The number of grid points is then just N, the sequence length. Splitting the problem along the middle row produces two subproblems of total size equal to that of the original problem, which is also the total of the sizes of the "grandchildren" problems. Since the recursion depth is $\log_2 N$, the running time of a straightforward application of Hirschberg's strategy exceeds the score-only time by the factor $\log N$.

We have developed several alignment-producing methods with time bound proportional to the region's area. One approach is based on the observation that score-only time is attained by Hirschberg's method if the region contains at least half of the dynamic-programming grid points. This holds since the number of point-inspections to produce the alignment is at most twice the number of dynamic-programming grid points, and hence at most four times the region's area.

Given a narrow region, R, we can proceed as follows. We may as well assume that L and U are non-decreasing since if, e.g., $L[i]$ were larger than $L[i+1]$, we could set $L[i+1]$ to equal $L[i]$ without affecting the set of constrained alignments. Enclose as many rows as possible from the top of the region in an upright rectangle, subject to the condition that the rectangle's area at most doubles the area of its intersection with R. Then starting with the first row of R not in the rectangle, cover additional rows of R with a second such rectangle, and so on. Fig. 10 illustrates this process of covering R with rectangles and Fig. 11 gives the algorithm for locating the partition line segments.

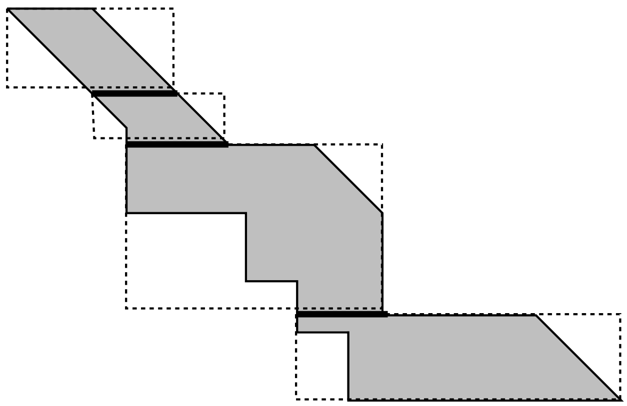

FIG. 10. Covering a narrow region, R, with non-intersecting rectangles, each of which has more points in R than outside R. The dark lines show where values of S^+ are saved during the backward pass described in the text.

```
I ← empty
i_0 ← 0
sum_widths ← U[0] − L[0] + 1
for i ← 1 to M do
    width ← U[i] − L[i] + 1
    sum_widths ← sum_widths + width
    if (U[i] − L[i_0] + 1) × (i − i_0 + 1) > 2 × sum_widths then
        I ← I ∪ {i}
        i_0 ← i
        sum_widths ← width
```

FIG. 11. Algorithm to partition a region of grid points.

A score-only backward pass is made over R, computing S^+. Values of S^+ are retained for the top line in every rectangle (the top rectangle can be skipped). See Fig. 10. It can be shown that the total length of these pieces cannot exceed three times the total number of columns, as required for a linear space bound. Next, perform a score-only forward pass, stopping at the last row in the first rectangle. A sweep along the boundary between the first and second rectangles locates a crossing edge on an optimal path though R. That is, we can find a point p on the last row of the first rectangle and a point q on the first row of the second rectangle such that there is a vertical or diagonal edge e from p to q, and e is on an optimal path. Such an optimal path can be found by applying Hirschberg's strategy to R's intersection with the first rectangle (omitting columns following p) and recursively computing a path from q through the remainder of R. This process inspects a grid point at most once during the backward pass, once in a forward pass computing p and q, and an average of four times for applying Hirschberg's method to R's intersection with a rectangle. Details of this approach can be found in Chao *et al.*, 1993b, while a substantially different method that attains a factor of essentially 2 is presented by Chao *et al.* (1993a).

FINDING ALIGNED PAIRS IN NEARLY-OPTIMAL ALIGNMENTS

Our second variant finds all edges that are contained in at least one path whose score exceeds a given threshold, τ. Again, a recursive subproblem will consist of applying the alignment algorithm over a rectangular portion of the original dynamic-programming matrix, but now it is necessary that we continue to work with values S^- and S^+ that are defined relative to the original problem. To accomplish this, each problem to be solved is defined by specifying values of S^- for nodes on the upper and left borders of the defining rectangle, and values of S^+ for the lower and right borders.

To divide a problem of this form, a forward pass propagates values of S^- to nodes in the middle row, and a backward pass propagates values of S^+ to the row that follows the middle row. This information allows us to determine all edges starting in the middle row that are contained in a path of score at least τ. We are left with two subproblems to solve, but their total area can be as large as that of the original problem. See Fig. 12A. For reasons that are analogous to ones explained in the previous section, this means that the divide-and-conquer process can require time *and space* exceeding the score-only requirements by the factor log M.

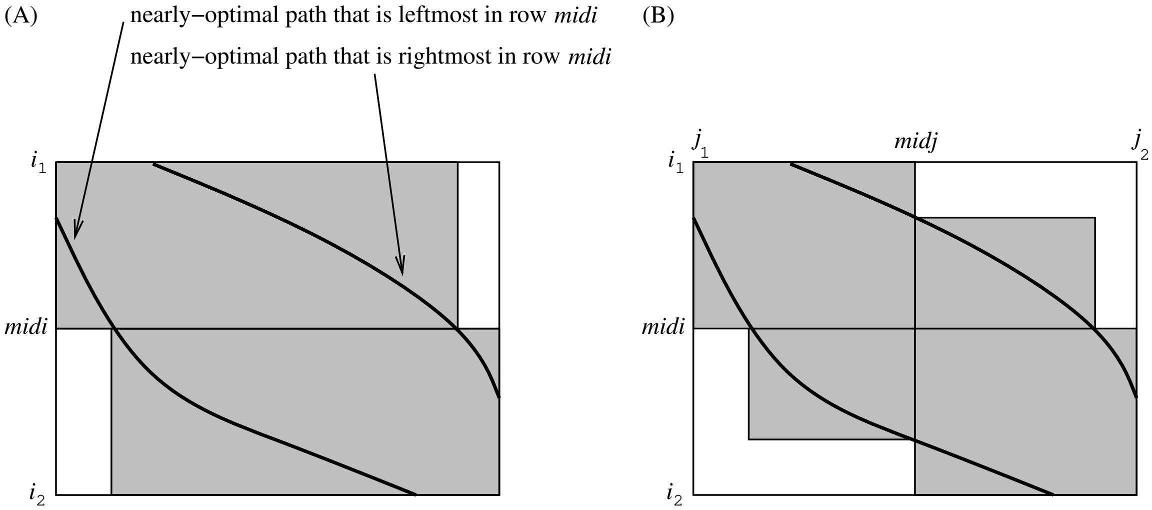

FIG. 12. Undesirable (A) and desirable (B) divisions into subproblems when computing all edges that lie in nearly-optimal paths, i.e., paths scoring at least τ.

The following strategy (Fig. 12B) reduces the space requirement to linear. A forward pass propagates values of S^- to nodes in the middle row *and the middle column*, and a backward pass propagates values of S^+ to those nodes. The data determining any one of the four subproblems, i.e., the arrays of S-values on its borders, is then at most half the size of the set of data defining the parent problem. The maximum total space requirement is realized when recursion reaches a directly solvable problem in the extreme upper left corner of the original grid; at that time there are essentially $2(M+N)$ S-values saved for borders of the original problem, $M+N$ values on the middle row and column of the original problem, $(M+N)/2$ values for the upper left subproblem, $(M+N)/4$ values for the upper-left-most subsubproblem, etc., giving a total of about $4(M+N)$ retained S-values. Fig. 13 outlines the approach.

recursive procedure *sub_opt*(i_1, j_1, i_2, j_2, boundary score vectors)

  if $i_1 + 1 \geq i_2$ **or** $j_1 + 1 \geq j_2$ **then**

    write all edges with score at least τ.

  else

    $midi \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$

    $midj \leftarrow \lfloor (j_1 + j_2)/2 \rfloor$

    A linear-space forward computation is performed to compute S^-:

      store $S^-[i, j]$ if $i = midi$ or $j = midj$;

    A linear-space backward computation is performed to compute S^+:

      store $S^+[i, j]$ if $i = midi$ or $j = midj$;

    /* Divide the problem by row *midi* and column *midj* */

    for each of the four subproblems **do**

      Let i'_1, j'_1, i'_2, and j'_2 be the shrunken boundary row and column indexes.

      sub_opt(i'_1, j'_1, i'_2, j'_2, new boundary score vectors)

FIG. 13. The algorithm for computing all edges that are contained in at least one path whose score exceeds a given threshold, τ.

The $\log M$ factor still afflicts the running time, though a precise analysis shows that it affects only grid points lying between nearly-optimal paths, and a more sophisticated algorithm reduces that penalty factor to $\log \log M$ (Chao, 1994).

ALIGNING TWO ALIGNMENTS

To obtain satisfactory alignments of DNA or protein sequences, one generally needs an alignment-scoring scheme that is more flexible than the ones treated so far in this paper (Fitch and Smith, 1983). Such schemes may assign a score for certain adjacent pairs of columns, in addition to a score for every individual column. A case in point concerns aligning two sequences that are themselves alignments (i.e., sequences of fixed-height columns), where the potential resulting alignments are scored using what is called the sum of pairwise scores with quasi-natural gap costs (Altschul, 1989).

To illustrate the basic idea behind these scores, suppose we are aligning A and B, where A is a three-way alignment of DNA sequences and B is a compatible two-way alignment. Thus a_i (i.e., the i^{th} entry of A) is a column containing three symbols, each being one of A, C, G, T, or "–". Given these five sequences (three from A and two from B), there are $\binom{5}{2}$ = 10 possible sequence pairs. With each such pair of DNA sequences, say sequences p and q with $1 \leq p < q \leq 5$, we have an associated set of scores $\sigma_{p,q}$ for aligned pairs. A five-column (i.e., with 5 entries) is assigned the score σ, defined as the sum of the ten scores for aligned pairs obtained by selecting two of the five entries. Moreover, a "gap cost" $g_{p,q}$ is given for each sequence pair. These costs are used to define a score for each adjacent pair of five-columns, as follows. Fix a pair of five-columns. For each sequence pair (p, q), the pair of five-columns is assessed a penalty $g_{p,q}$ if the p^{th} and q^{th} rows of the pair exhibit any one of the six patterns given in Fig. 14. These counts tend to over-estimate the actual number of gaps in the pairwise alignment induced by the five-way alignment, but their use is mandated by considerations of execution efficiency (Altschul, 1989).

For $\begin{bmatrix} x & - \\ x & x \end{bmatrix}$, $\begin{bmatrix} x & x \\ x & - \end{bmatrix}$, $\begin{bmatrix} x & - \\ - & x \end{bmatrix}$ or $\begin{bmatrix} - & x \\ x & - \end{bmatrix}$ count 1 gap start.

For $\begin{bmatrix} - & x \\ - & - \end{bmatrix}$ or $\begin{bmatrix} - & - \\ - & x \end{bmatrix}$ count 1 gap end.

FIG. 14. Rules for counting gaps and quasi-gaps given a pair of adjacent columns of a multi-sequence alignment. A pair of rows has been extracted, giving a pair of aligned pairs. Here x denotes an entry other than "−". The total count for the adjacent multi-sequence columns is the number of gap starts and gap ends, summed over all pairs of rows.

Such alignment problems can be modeled as follows. With each dynamic-programming grid point (i, j), associate three nodes, denoted $(i, j)_V$, $(i, j)_D$ and $(i, j)_H$. The V (vertical) node has edges entering from each of the three nodes above it, the D node has three entering diagonal edges and the H node has three entering horizontal edges. (Nodes in row 0 or column 0 may have fewer entering edges.) Scores are assigned to nodes and edges as depicted in Fig. 15. Again, alignments correspond to paths in the graph, but now a path's score is the sum of the scores attached to its edges *and its nodes*.

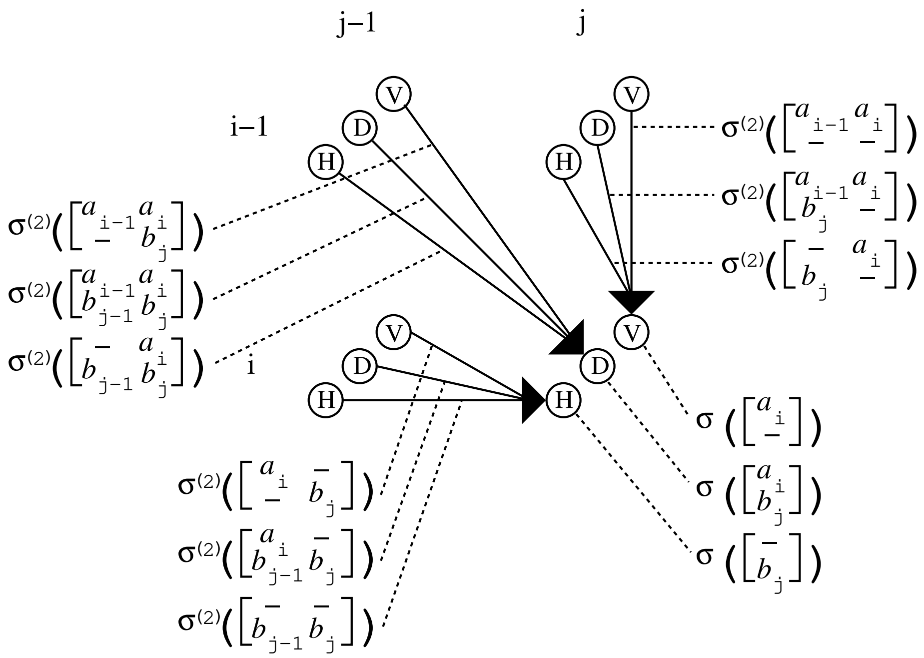

FIG. 15. The three nodes at grid point (i, j), the edges entering those nodes, and the scores associated with those nodes and edges. This model assigns a score $\sigma^{(2)}(c_1 c_2)$ to every adjacent column-pair $c_1 c_2$.

Given this model of the alignment problem, one can proceed as before. For example, to determine the score, $D[i, j]$, of an optimal path ending at $(i, j)_D$, maximize over the three entering edges, i.e.,

$$D[i,j] \leftarrow \sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix}) + \max \begin{cases} V[i-1,j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ - & b_j \end{bmatrix}) \\ D[i-1,j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ b_{j-1} & b_j \end{bmatrix}) \\ H[i-1,j-1] + \sigma^{(2)}(\begin{bmatrix} - & a_i \\ b_{j-1} & b_j \end{bmatrix}) \end{cases}$$

Also, Hirschberg's strategy extends to this model. That is (1) a score-only computation can be performed using essentially just the space to store values in one row and (2) forward and backward passes to the middle row allow one to recursively construct an optimal path in approximately twice the score-only time. Fig. 16 provides detailed pseudocode for the linear-space alignment algorithm.

shared string arrays $a_1 a_2 \cdots a_M,\ b_1 b_2 \cdots b_N$
shared temporary integer arrays $V^-[0..N], D^-[0..N], H^-[0..N], V^+[0..N], D^+[0..N], H^+[0..N]$
procedure $Align(M, N)$
    if $M = 0$ **then**
        write $\begin{bmatrix} - \\ b_1 \end{bmatrix} \begin{bmatrix} - \\ b_2 \end{bmatrix} \cdots \begin{bmatrix} - \\ b_N \end{bmatrix}$
    else
        $path(0, 0, 'D', M+1, N+1, 'D')$

recursive procedure $path(i_1, j_1, type_1, i_2, j_2, type_2)$
    if $i_1 + 1 = i_2$ or $j_1 = j_2$ **then**
        write aligned pairs for maximum-score path from $(i_1, j_1)_{type_1}$ to $(i_2, j_2)_{type_2}$
    else
        $mid \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$
            /* A linear-space forward computation is performed to compute V^- and D^- */
        $V^-[j] \leftarrow$ maximum path score from $(i_1, j_1)_{type_1}$ to $(mid, j)_V$, for $j \in [j_1, j_2]$
        $D^-[j] \leftarrow$ maximum path score from $(i_1, j_1)_{type_1}$ to $(mid, j)_D$, for $j \in [j_1, j_2]$
            /* A linear-space backward computation is performed to compute V^+ and D^+ */
        $V^+[j] \leftarrow$ maximum path score from $(mid, j)_V$ to $(i_2, j_2)_{type_2}$, for $j \in [j_1, j_2]$
        $D^+[j] \leftarrow$ maximum path score from $(mid, j)_D$ to $(i_2, j_2)_{type_2}$, for $j \in [j_1, j_2]$
            /* find where maximum-score path crosses mid column */
        $(J, X) \leftarrow$ values $J \in [j_1, j_2]$ and $X \in \{'V', 'D'\}$ that maximize $X^-[J] + X^+[J]$
        $path(i_1, j_1, type_1, mid, J, X)$
        $path(mid, J, X, i_2, j_2, type_2)$

FIG. 16. Algorithm for delivering an optimal alignment in the model of aligning two alignments.

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). In brief, the strategy is to record the interesting parts of S in a space-efficient fashion and to perform local recomputation each time a best alignment is reported.

For each vertex $v = (i, j)$, define $First(v)$ to be the last vertex in some fixed topological order such that an alignment beginning at $First(v)$ and ending at v has score $S[i, j]$. It can be shown that if vertices u and v satisfy $First(u) \ne First(v)$, then an optimal alignment from $First(u)$ to u and an optimal alignment from $First(v)$ to v have no vertex in common. Vertices are partitioned into equivalence classes by their $First$-values; vertices u and v are in the same class if and only if $First(u) = First(v)$.

The algorithm works as follows. A forward pass is made through all vertices to construct k best equivalence classes. After delivering the best alignment (which is in the equivalence class with the highest score), a backward pass is performed to locate the region that has to be recomputed in order to update the recorded k best equivalence classes. A forward pass is then made over the region to find newly-exposed equivalence classes. This process is repeated until the k^{th} best alignment is delivered. Fig. 17 gives the outline. This algorithm has also been adapted to run efficiently on a parallel computer (Huang *et al*, 1992).

```
1.  Compute k best equivalence classes in a single sweep
2.  for i ← 1 to k do
3.      Deliver an optimal alignment in the equivalence class with the highest score
4.      if i ≠ k then
5.          Determine a rectangle to be recomputed
6.          Obtain k − i best equivalence classes by recomputing the rectangle
```

FIG. 17. Algorithm for computing k best local alignments.

FAST LOCAL ALIGNMENT

To attain greater speed for local alignment, Wilbur and Lipman (1983, 1984) employed the strategy of building alignments from "fragments", such as short runs of exact matches. This strategy has been very successful as an initial phase for database search. With protein sequences, it might work better to begin with inexact but high-scoring matches, such as those used by the *blast* program (Altschul *et al*., 1990). In general, a full-resolution alignment process can be made more efficient by restricting the search to a "neighborhood" of the alignment-from-fragments (Pearson and Lipman, 1988; Pearson, 1990; Chao *et al*., 1994).

It is straightforward to construct a highest-scoring alignment from fragments in $O(F^2)$ time, where F is the total number of fragments (Wilbur and Lipman, 1983). Based on the "candidate-list paradigm", Eppstein *et al*. (1992) developed an algorithm that runs in $O(F \log\log F)$ time. (Strictly speaking, these times should involve the two sequence lengths as additive terms, and the "log log F" can be improved slightly.) However, the data structure employed to obtain this theoretical efficiency is unusable in practice. With a practical data structure, the complexity becomes $O(F \log F)$, which is still a great improvement over $O(F^2)$ for problems of the size we regularly solve.

When long DNA sequences are aligned, it may be impractical to store all of the fragments. The algorithm of Eppstein *et al*. (1992) can be easily adapted to compute merely the optimal score of a local alignment in $O(M+N)$ space by keeping only those fragments in current (four) candidate lists, which are all of size $O(M+N)$. To deliver a local alignment in $O(M+N)$ space, Chao and Miller (1994) extend the approach of Hirschberg (1975) to identify one or two fragments near the middle of an optimal alignment, then recursively compute the alignment's remaining prefix and suffix. Several possibilities arise when identifying the middle fragments from the information retained in the candidate lists created by the forward and backward passes to the middle rows. Chao and Miller (1994) show that all possibilities can be checked in $O(c)$ time where there are c columns in the subproblem. The resulting algorithm runs in $O((M+N+F \log N) \log M)$ time and $O(M+N)$ space. The "log M" factor from the $F \log N \log M$ term arises in the analysis because in theory the two subproblems could contain almost all fragments of the parent problem.

Chao and Miller (1994) also give a time-efficient, linear-space algorithm for constructing k best non-intersecting alignments from fragments, following the strategy of Huang and Miller (1991). A forward pass is made through the entire set of fragments to find the first and last fragments on an optimal alignment. As paths to fragments f are discovered, they are divided into equivalence classes according to the first fragment on a highest-scoring alignment ending at f, and information about k best pairwise non-equivalent paths is retained. When fragments of a highest-scoring alignment are discarded, it is sufficient to recompute scores for fragments in the equivalence class containing the alignment's fragments. It is shown that this recomputation can be carried out efficiently.

MULTIPLE ALIGNMENT TOOLS

We have implemented a family of multi-sequence alignment programs, based on the algorithms described in this paper. The user is required to specify an aligning tree that indicates an appropriate order for progressive alignment; typically this tree will mirror the sequences' phylogenetic tree. Each program then operates progressively (Feng and Doolittle, 1987), i.e., by repeatedly aligning two sequences or smaller alignments to build up the final alignment. At each progressive step the sum of pairwise scores with quasi-natural gap costs is maximized (subject to the constraint that the step leaves columns of its input alignments intact). Thus, each step involves a pairwise alignment process that may be equivalent to a generalized (three nodes per grid point) optimal path problem.

For example, an evolutionary tree for (homologous sequences from) human, galago, rabbit and mouse is given in panel (A) of Fig. 18. The corresponding order for progressive alignment would be to align sequences from human and galago, then to align the resulting pairwise alignment with the rabbit sequences, and finally to align the 3-way alignment with the mouse sequence, denoted:

((human galago)rabbit)mouse

For human, galago, lemur and rabbit, the evolutionary tree in panel (B) suggests that one align the prosimian primate sequences (galago and lemur), then align the human sequence and the prosimian alignment, and finally align the 3-way alignment with the rabbit sequence, denoted:

(human(galago lemur))rabbit

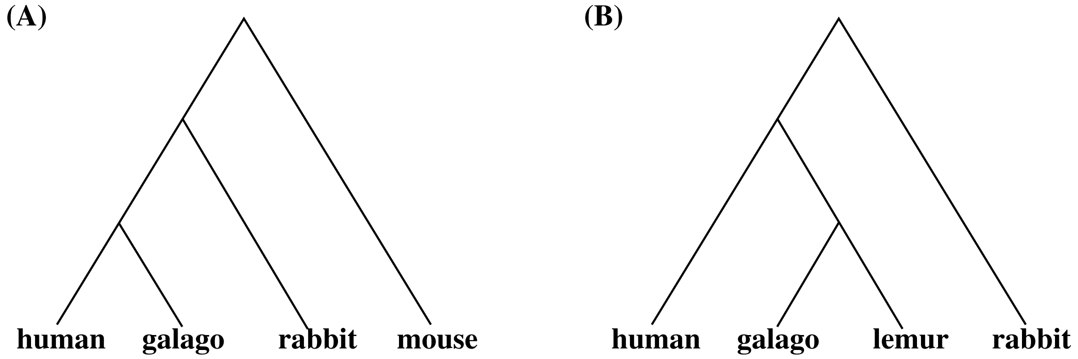

FIG. 18 Evolutionary trees. Branch lengths in these two idealized trees are not related to evolutionary time.

Different substitution scores $\sigma_{p,q}$ are permitted for each sequence pair. Moreover, the programs support a flexible scheme for scoring gaps. In general, a gap of length k at the left end of sequence p is penalized $g_p^L + e_p^L k$, a gap of length k at the right end of sequence p is penalized $g_p^R + e_p^R k$, and an internal gap or progressive quasi-gap of length k in the pairwise alignment of sequences p and q is penalized $g_{p,q} + e_{p,q} k$. Different scores for each sequence pair are appropriate when aligning sequences at varying phylogenetic distances, and we need flexible end-gap scores (in particular, the option of setting them to 0) for aligning sequences of vastly different lengths.

One program, called *yama* (*y*et *a*nother *m*ultiple *a*ligner), implements the generalized Hirschberg's linear-space algorithm (Fig. 16). The time taken to align alignments A and B of lengths M and N, respectively, is analyzed as follows. First, a pass over A determines, for each column of A, the sum of all pairwise σ scores (i.e., for each pair of entries in that column), the number of pairwise gap starts and the number of pairwise gap ends (using the rules of Fig. 14). This requires time $O(r_A^2 M)$, where r_A is the number of rows in A. (If we wanted to penalize all quasi-gaps, not just the *progressive* quasi-gaps, we would also compute the number of additional gap ends that would be introduced by placing a null column before each column.) A similar pass is made over B. Let $k_{A,B}$ be the number of pairs of basic sequences, one from A and one from B, i.e., $k_{A,B} = r_A r_B$. Each of the σ and $\sigma^{(2)}$ values in Fig. 15 costs $O(k_{A,B})$ time to compute. Since there are $O(1)$ such values per grid point and $O(MN)$ grid points, the total time to align A and B, including the initial passes over A and B, is $O(r_A^2 M + r_B^2 N + k_{A,B} MN)$. Under any normal conditions (e.g., $r_A \le N$), the time is $O(k_{A,B} MN)$. The computation time for delivering an optimal alignment is approximately twice the time needed to compute merely its score. Thus computing an optimal alignment of A and B takes time $O(k_{A,B} MN)$. Finally, consider the time needed to align n sequences progressively. Let k_i denote $k_{A,B}$ at the i^{th} step, where alignments A and B are aligned. There are $n-1$ such steps, and the sum of the k_i is $\frac{1}{2}n(n-1)$, since every pair of basic sequences is counted exactly once. Thus, if the final multiple alignment has length L, then the total time complexity is:

$$\sum_{i=1}^{n-1} O(k_i L^2) = O(n^2 L^2)$$

Assuming that most of the basic sequences have length nearly L, the time for progressive multiple alignment is thus roughly equal to the time for computing all pairwise alignments. We remark that several other programs save time in each progressive step by aligning consensus sequences that summarize alignments A and B. We typically align only a few sequences (i.e., n is small) under conditions where execution time is not a concern, so the n^2 factor is quite affordable.

To increase efficiency, a step of the progressive alignment algorithm can be constrained to the portion of the dynamic-programming grid lying between two boundary lines. Another program, called *yama2*, performs constrained alignment using lower and upper bounding functions $L_{p,q}$ and $U_{p,q}$ for some or all sequence pairs. When a progressive step aligns alignments A and B, each pair (p, q), such that sequence p appears in A and sequence q appears in B, creates a constraint on the step, and all such constraints are enforced.

Finally, *yama3* considers constrained alignments consisting of aligned pairs in nearly-optimal alignments. In our intended usage, a set of "patterns" is specified (perhaps the consensus sequences for transcription factor binding sites); *yama3* selects, from among all alignments with the highest score, an alignment with the largest number of conserved blocks that match a pattern.

For the alignment discussed in the next section, we began with a 73,326-basepair sequence that covers most of the β-like globin gene locus of humans. Twenty-six shorter sequences, with lengths from a few hundred to over 10,000, were extracted from the corresponding sequence data for five other mammals. These supposedly homologous sequences were identified by applying a pairwise (k-best local) alignment program to compare the sequence from human with that from the other species and, for each of the shorter sequences, the corresponding alignment was used to define bounding functions $L_{p,q}$ and $U_{p,q}$, where sequence p is the human sequence. The process of selecting sequences and bounding functions is described in detail by Hardison *et al*. (1994). Incidentally, computing the multi-sequence alignment took about 45 minutes on a Sun SPARCstation 2 workstation (the initial pairwise alignments took much longer).

What is important for the current discussion is that a long sequence was aligned with a number of much shorter sequences. Work was required to do this in a manner that efficiently uses both computer memory to compute the alignment and file space to store it.

Letting n denote the number of sequences being aligned, and letting S be the sum of the lengths of the input sequences, computer memory requirements are as follows. Storing inputs for *yama*, *yama*2 and *yama*3 requires at most S bytes for input sequences, plus $\frac{1}{2}n(n-1)$ tables of σ values, $n(n-1)$ integers for the $g_{p,q}$ and $e_{p,q}$ penalties, and $4n$ integers for the g_p^L, e_p^L, g_p^R and e_p^R penalties. (For the β-globin example, we can use $n = 6$ for these bounds, assuming that the alignment-scoring scheme depends only on the species-pair.) Also, *yama2* and *yama3* use at most S integers for the $L_{p,q}$ and $U_{p,q}$. For intermediate computed values and the final alignment, *yama* uses space for $8S$ integers, *yama2* uses $8S$ characters and $12S$ integers, while *yama3* uses $2S$ characters, $25S$ integers and the space for storing the aligned pairs that occur in a nearly-optimal alignment for some progressive step.

In the β-globin example, there are 82,418 columns in the resulting *yama2* alignment (i.e., the one discussed in the next section). If file-system storage for this 27-way alignment is column-by-column, we need to retain more than 2 million entries. A more concise representation is to store the alignment as a chain of blocks with different patterns of non-dashes (i.e., A, C, G or T) and dashes; about 680K bytes would be needed in this example. Better yet, for each sequence we actually store the lengths and positions of all gaps in the corresponding row of the alignment. With this representation, 16K bytes of disk space are enough for storing the alignment of β-like globin gene clusters.

AN EXAMPLE: THE β-GLOBIN LOCUS CONTROL REGION

As already mentioned, the multi-sequence alignment programs described in the previous section have been applied to DNA sequences from the β-like globin gene clusters of several mammals (Hardison *et al*., 1993b). The resulting alignment is the backbone of an E-mail server that we are building to provide the annotated alignment in register with a compilation of experimental data. This Globin Gene Server (Hardison *et al*., 1994) is essentially a hypothesis-generating machine. Currently, users can readily ascertain whether a particular region of interest is conserved in representatives of several mammalian orders, and we intend that they will soon be able to view summaries of previous experimental work on homologous genes in mammals (e.g. results on the β-globin gene promoter from human, mouse and rabbit will be compiled in register, as dictated by the sequence alignment). This will make it immediately apparent whether previous results are congruent toward a given conclusion, or if some data on a particular region are in conflict, or if some particularly well-conserved region has not been tested for function. We refer to this process of examining the multiple alignment (with inferences about conservation of the sequences) in register with experimental data as "electronic genetic analysis" and consider it to be a prototype for the type of analysis that will become the norm for any locus as more data on large regions of mammalian genomes becomes available.

To illustrate this process, we show in Fig. 19 the annotated alignment of a section, called HS2, of the locus control region (LCR) of mammalian β-like globin gene clusters. The LCR is a dominant control region located at the 5′ end of the gene cluster. It is required for the high-level expression in red cells of any gene within the cluster, and it confers the ability for globin genes randomly integrated in a chromosome to be expressed, regardless of their position of integration. These properties have led to the proposal of several functions for the LCR, including activation of a chromosomal domain, establishment of a domain boundary, and enhancement of expression of genes within that domain (Grosveld *et al*., 1993). Several functional regions within the LCR are marked as sites that are very sensitive to DNase digestion in nuclei; these are referred to as hypersensitive sites, or HS. The second HS, or HS2, will enhance expression of linked globin genes in transfected cells (Tuan *et al*., 1989) and can confer position-independent expression in some assays (Talbot *et al*., 1990; Caterina *et al*., 1991). This region of the LCR has been very intensively studied, but with only modest attention paid to the level of sequence conservation. Fig. 19 shows very strong conservation in the HS2 region, particularly from positions 8648 to 8677 in the human sequence. The region preceding this core has several conserved blocks of sequence separated by more variable sequences. The repeating AT motif after position 8882 is unique to humans. In fact the sequences for several hundred base pairs after the conserved core of HS2 are unique to each mammalian species (Hardison *et al*., 1993a).

We have compiled the results from a number of detailed studies of HS2 and examined them in register with the conserved sequences identified in the multiple alignment. These data include precise localization of the DNase hypersensitive sites, mapping of the binding sites for proteins in nuclei of cells, and the effects of fragments of HS2 on expression of a linked human β-globin gene both in transgenic mice and in erythroleukemia cells containing the test DNA integrated into a chromosome. Many points can be made on the basis of this view of HS2, but a few selected observations (Fig. 20) will illustrate the congruence of data demonstrating a key role for NFE2, and will show one example of the many new avenues of inquiry that this dataset suggests.

The two tandem binding sites for the erythroid nuclear protein NFE2 are critical for the function of HS2. A 374 bp fragment extending from the HindIII site (positions 8486-8491) to the XbaI site (8860-8865) functions to activate the β-globin gene in the assays summarized in Fig. 20, although the magnitude of the effect varies from report to report (lines 1, 5, 9). All transgenic mice carrying this construct express the gene, indicating that the expression is independent of position of integration (second data column for lines 1-8). This fragment has four DNase hypersensitive sites (line 14; all seven on this line comprise HS2 as defined by a lower resolution assay), one of which is close to the NFE2 binding sites. Deletion of fragments that contain the NFE2 binding sites causes a substantial reduction in level of expression (lines 3, 6, 10). Addition of short fragments containing the NFE2 binding sites confers some activity in transgenic mice (line 7) and other studies show this region will give considerable enhancement in transfected cells (Ney *et al*., 1990). These binding sites are occupied in cells, as shown by the *in vivo* footprints (lines 15 and 19). Given the preponderance of data for the importance of these binding sites, it is not surprising that these sequences are very highly conserved in several mammals (Fig. 19, Fig. 20 top line). Thus all investigations show that this region is critical for function of HS2.

However, NFE2 binding to its cognate sites does not explain the full function of HS2. Even when the NFE2 binding sites are removed, the remaining region of HS2 will confer position-independent expression in transgenic mice (Fig. 20, line 3). Addition of short fragments containing the NFE2 binding sites causes at best a limited increase in expression of the β-globin gene in transgenic mice (line 7) and has almost no effect in the pools of stable MEL cell transformants (line 11). Several other DNase hypersensitive sites are apparent (line 14), as are several other footprints both *in vivo* (lines 16, 17, 18, and 20) and *in vitro* (Talbot *et al*., 1990; Caterina *et al*., 1991). Thus HS2 doubtless uses several other protein-binding sites for full function.

Candidates for these other sites are in the conserved regions listed on the top line of Fig. 20. Some of these match or overlap with regions that have been examined experimentally. For instance, the GATA1 binding site at 8730 to 8735 has been observed both *in vitro* (Talbot *et al*., 1990) and *in vivo* in one experiment (Fig. 20, line 16) but not in another (lines 18-20). A DNase HS maps close to the GATA1 binding site (line 14). This is a strong candidate for an important element of HS2, and further experiments to test this hypothesis can clearly be done. Likewise, the footprinting experiments listed in Fig. 20 show that a USF binding site is occupied in cells (lines 17 and 20), but a deletion of this region has very little effect on the function of the rest of HS2 in transgenic mice (line 2).

Many other sites of interest are apparent in the conserved regions. We are intrigued by the highly conserved region between the binding sites for NFE2 and GATA1, and recently we demonstrated specific protein binding *in vitro* (L. Elnitski and R. Hardison, unpublished data). Many other candidates for functional sites within HS2 remain to be tested. Incorporation of all these data into a rational model will be a challenge, but the multiple alignment and the linked database of experimental results will be a very helpful tool as many laboratories continue to work on this problem.

FIG. 19. Portion of an alignment of a 73,326-basepair sequence (GenBank locus HUMHBB) covering most of the β-like globin gene cluster of humans with homologous sequences of various lengths from several other mammals. This particular portion contains DNase I hypersensitive site HS2. The alignment was computed by the program *yama2* according to a strategy described by Hardison *et al*. (1993b, 1994), and stored in a compressed form. Arbitrary portions of the alignment can be drawn by a program called *lat*. For example, a PostScript file for Fig. 19 can be obtained by sending electronic mail to `globin@groucho.cse.psu.edu` with the one-line message "`send lat 8400 9000`". Boxes were automatically drawn by *lat* around regions of six or more consecutive columns with at most one mismatch per column.

FIG. 20. Summary of selected data on HS2 of the β-globin locus control region. The top and bottom lines show the conserved segments that are boxed in Fig. 19. Commonly used restriction endonuclease cleavage sites are indicated. Demonstrated or proposed binding sites for proteins are listed below the conserved segments; the gata1 site in lower case is not as strong a match to the consensus site as is the GATA1 site (upper case). Lines 1-8 show selected results of tests of the ability of DNA fragments from HS2 to confer high-level, position independent expression of the human β-globin gene in transgenic mice, as described by Caterina *et al*., 1991 (ref. 1) and Liu *et al*., 1992 (ref. 2). The region of HS2 included in the construct is shown as a horizontal line. Internal deletions are indicated by a space with a Δ above it. Lines 9-11 show measurements of the ability of the restriction fragments to enhance expression of integrated copies of the human β-globin gene in induced MEL cells as described by Talbot *et al*., 1990 (ref. 3). The detailed mapping of the sites of DNase I cleavage in nuclei is also from ref. 3. The contact points between DNA sequences and bound proteins *in vivo*, or *in vivo* footprints, (lines 15-20) are reported by Reddy and Shen, 1993 (ref. 4) and Ikuta and Kan, 1991 (ref. 5). The boxes denoting footprints are shaded like the corresponding boxes indicating conserved segments.

OTHER LINEAR-SPACE DYNAMIC-PROGRAMMING ALIGNMENT TECHNIQUES

Chao *et al* (1993b) extended Hirschberg's approach to compute additional information that in some sense indicates the degree to which the aligned residues are conserved. One method computes the left and right boundaries of the portion of the grid containing paths that come within a specified tolerance of the optimal score. Another method determines, for each aligned pair (i.e., column) of an optimal alignment, the amount by which the optimal score must be lowered before reaching an alignment not containing that pair. As a special case, this value tells whether the pair is in all optimal alignments (namely, the pair is in all optimal alignments if and only if its associated value is non-zero). All these methods run in linear space and score-only time, i.e., execution time that is at most a constant multiple of the time to produce the alignment's score.

Several other alignment problems of interest for molecular biology can be tackled with Hirschberg's divide-and-conquer strategy, but where non-trivial data structures resulting from the forward and backward passes have to be merged. An example concerns "concave gap penalties" (Waterman, 1984), which provide a more flexible scoring scheme than treated here. Miller and Myers (1988) describe a "linear space" approach, though in this case the result necessarily holds only for the expected (as opposed to worst-case) space consumption (Rabani and Galil, 1992).

It is worth noting that Hirschberg's strategy is not the only way to subdivide a dynamic-programming matrix to achieve linear-space performance. Chao *et al*. (1992) and Chao *et al*. (1993a) utilize a strategy that subdivides a problem into a variable number of subproblems, as opposed to Hirschberg's two equal-height subproblems.

DISCUSSION

Simultaneous alignment of several sequences is among the most important problems in computational molecular biology. In particular, accurate multiple alignment is critical for our use of DNA sequence comparisons to study gene regulation (Hardison and Miller, 1993; Miller *et al*., 1994). In spite of the plethora of existing ideas and methods for multiple alignment (surveyed by Chan *et al*., 1992), no available program seemed well-suited to our needs.

A critical requirement for our work is the ability to compare sequences that are thousands of nucleotides long, which limits us to "linear-space" comparison algorithms, i.e., methods that use computer space proportional to the sum of the sequence lengths. Thus we have been motivated to develop a variety of space-efficient alignment methods. Another requirement is tight control over the alignment-scoring scheme used by the alignment procedure. Such control is needed because we compare sequences from species at a variety of evolutionary distances, and the settings for alignment-scoring parameters that optimally expose conserved features vary from case to case. Moreover, because we often want to align sequences of substantially different lengths, flexible penalizing schemes for end-gaps are critical.

It should be noted that recently Higgins *et al*. (1992) announced new multiple-alignment software. Their strategy for aligning two alignments uses, in effect, the following special case of our node and edge scores: σ for a D node is the average pairwise score, σ for V and H nodes is the pairwise gap extension pair penalty (i.e., $\begin{bmatrix} - \\ x \end{bmatrix}$, where x is a sequence entry), $\sigma^{(2)}$ for an edge from a D or V node to an H node, or from a D or an H node to a V node, is the gap penalty g, and $\sigma^{(2)}$ for other types of edges is 0. Actually, with these scores the number of edges per grid point can be reduced from nine to seven (Myers and Miller, 1988).

ACKNOWLEDGMENTS

We thank Michael Waterman for making several useful comments. This work was supported in part by grant R01 LM05110 from the National Library of Medicine to W. M. and by PHS grant R01 DK27635 to R. C. H.

REFERENCES

Altschul, S. F. (1989) Gap costs for multiple sequence alignment. *J. theor. Biol.* **138**, 297-309.

Altschul, S., W. Gish, W. Miller, E. Myers and D. Lipman (1990) A basic local alignment search tool. *J. Mol. Biol.* **215**, 403-410.

Caterina, J. J., T. M. Ryan, K. M. Pawlik, R. D. Palmiter, R. L. Brinster, R. R. Behringer and T. M. Townes (1991) Human β-globin locus control region: Analysis of the 5′ DNase I hypersensitive site HS2 in transgenic mice. *Proc. Natl. Acad. Sci. USA* **88**, 1626-1630.

Chan, S. C., A. K. C. Wong and D. K. Y. Chiu (1992) A survey of multiple sequence comparison methods. *Bull. Math. Biol.* **54**, 563-598.

Chao, K.-M., W. R. Pearson and W. Miller (1992) Aligning two sequences within a specified diagonal band. *CABIOS* **8**, 481-487.

Chao, K.-M., R. Hardison and W. Miller (1993a) Constrained sequence alignment. *Bull. Math. Biol.* **55**, 503-524.

Chao, K.-M., R. Hardison and W. Miller (1993b) Locating well-conserved regions within a pairwise alignment. *CABIOS* **9**, 387-396.

Chao, K.-M. (1994) Computing all suboptimal alignments in linear space. *Combinatorial Pattern Matching '94*, Lecture Notes in Computer Science **807**, 31-42.

Chao, K.-M. and W. Miller (1994) Linear-space algorithms that build local alignments from fragments. To appear in *Algorithmica*.

Chao, K.-M., J. Zhang, J. Ostell and W. Miller (1994) A fast local alignment tool for very long DNA sequences. Submitted.

Doolittle, R. F. (1990) *Methods in Enzymology, Vol. 183*, "Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences," R. F. Doolittle, ed.

Eppstein, D., Z. Galil, R. Giancarlo and G. F. Italiano (1992) Sparse dynamic programming. I. Linear cost functions. *Jour. Assoc. Comput. Mach.* **39**, 519-545.

Feng, D.-F. and R. F. Doolittle (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* **25**, 351-360.

Fitch, W. M. and T. F. Smith (1983) Optimal sequence alignments. *Proc. Natl. Acad. Sci. USA* **80**, 1382-1386.

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

Grosveld, F., M. Antoniou, M. Berry, E. de Boer, N. Dillon, J. Ellis, P. Fraser, O. Hanscombe, J. Hurst, A. Imam, M. Lindenbaum, S. Philipsen, S. Pruzina, J. Strouboulis, S. Raguz-Bolognesi and D. Talbot (1993) The regulation of human globin gene switching. *Phil. Trans. R. Soc. Lond. B* **339**, 183-191.

Hardison, R. C. and W. Miller (1993) Use of long sequence alignments to study the evolution and regulation of mammalian globin gene clusters. *Molecular Biology and Evolution* **10**, 73-102.

Hardison, R. C., J. Xu, J. Jackson, J. Mansberger, O. Selifonova, B. Grotch, J. Biesecker, H. Petrykowska and W. Miller (1993a) Comparative analysis of the locus control region of the rabbit β-like globin gene cluster: HS3 increases transient expression of an embryonic ε-globin gene. *Nucleic Acids Res.* **21**, 1265-1272.

Hardison, R. C., K.-M. Chao, M. Adamkiewicz, D. Price, J. Jackson, T. Zeigler, N. Stojanovic and W. Miller (1993b) Positive and negative regulatory elements of the rabbit ε-globin gene revealed by an improved multiple alignment program and functional analysis. *DNA Sequence* **4**, 163-176.

Hardison, R. C., K.-M. Chao, S. Schwartz, N. Stojanovic, M. Ganetsky and W. Miller (1994) Globin Gene Server: a prototype E-mail database server featuring extensive multiple alignments and data compilation for electronic genetic analysis. *Genomics* **21**, 344-353.

Higgins, D. G., A. J. Bleasby and R. Fuchs (1992) CLUSTAL V: improved software for multiple sequence alignment. *CABIOS* **8**, 189-191.

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.

Huang, X., W. Miller, S. Schwartz and R. Hardison (1992) Parallelization of a local similarity algorithm. *CABIOS* **8**, 155-165.

Ikuta, T. and Y. W. Kan (1991) *In vivo* protein-DNA interactions at the β-globin locus. *Proc. Natl. Acad. Sci. USA* **88**, 10188-10192.

Liu, D., J. C. Chang, P. Moi, W. Jiu, Y. W. Kan and P. T. Curtin (1992) Dissection of the enhancer activity of β-globin $5'$ DNase I-hypersensitive site 2 in transgenic mice. *Proc. Natl. Acad. Sci. USA* **89**, 3899-3903.

Miller, W. and E. Myers (1988) Sequence comparison with concave weighting functions. *Bull. Math. Biol.* **50**, 97-120.

Miller, W., S. Schwartz and R. Hardison (1994) A point of contact between computer science and molecular biology. *Computational Science and Engineering* **1**, 69-78.

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.
Ney, P., B. Sorrentino, K. McDonagh and A. Nienhuis (1990) Tandem AP-1-binding sites within the human β-globin dominant control region function as an inducible enhancer in erythroid cells. *Genes Devel.* **4**, 993-1006.
Pearson, W. R. (1990) Rapid and sensitive sequence comparison with FASTP and FASTA. In Doolittle (1990), 63-95.
Pearson, W. R. and D. J. Lipman (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci.* **85**, 2444-2448.
Rabani, Y., and Z. Galil (1992) On the space complexity of some algorithms for sequence comparison. *Theoretical Computer Science* **95**, 231-244.
Reddy, P. M. S. and C.-K. J. Shen (1993) Erythroid differentiation of mouse erythroleukemia cells results in the reorganization of protein-DNA complexes in the mouse βmaj globin promoter but not its distal enhancer. *Mol. Cell. Biol.* **13**, 1093-1103.
Smith, T. F. and M. S. Waterman (1981) Identification of common molecular sequences. *J. Mol. Biol.* **197**, 723-728.
Talbot, D., S. Philipsen, P. Fraser and F. Grosveld (1990) Detailed analysis of the site 3 region of the human β-globin dominant control region. *EMBO J.* **9**, 2169-2178.
Tuan, D., W. Solomon, I. London and D. Lee (1989) An erythroid-specific, developmental-stage-independent enhancer far upstream of the human "β-like globin" genes. *Proc. Natl. Acad. Sci. USA* **86**, 2554-2558.
Waterman, M. S., T. F. Smith and W. A. Beyer (1976) Some biological sequence metrics. *Adv. Math.* **20**, 367-387.
Waterman, M. S. (1984) Efficient sequence alignment algorithms. *J. theor. Biol.* **108**, 333-337.
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.
Wilbur, W. J. and D. J. Lipman (1983) Rapid similarity searches of nucleic acid and protein sequence data banks. *Proc. Natl. Acad. Sci. U.S.A.* **80**, 726-730.
Wilbur, W. J. and D. J. Lipman (1984) The context dependent comparison of biological sequences. *SIAM J. Appl. Math.* **44**, 557-567.