MULTIPLE ALIGNMENT IN LINEAR SPACE

□ KUN-MAO CHAO,[1] ROSS C. HARDISON[2,3] and WEBB MILLER[1,3]

[1]Department of Computer Science,
[2]Department of Molecular and Cell Biology and
[3]Center for Gene Regulation
The Pennsylvania State University
University Park, PA 16802

(*E.mail: webb@cs.psu.edu*)

This paper develops algorithms for simultaneous alignment of more than two sequences. The algorithms' strengths are that they (1) require only space proportional to the size of the computed alignment and (2) treat gap penalties in a careful manner proposed by Altschul. For the most efficient use of computer time, computation can be constrained to a region of the dynamic-programming matrix. Utility of these techniques is illustrated by aligning sequences that include the ε-globin genes of four mammals, which locates conserved regulatory signals.

1. Introduction. 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). 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. Indeed, one of our major research themes has been the development of such space-efficient alignment methods (e.g., Myers and Miller, 1988; Huang *et al*., 1990; Chao *et al*., 1993a). 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.

These considerations led us to develop multiple-sequence alignment algorithms that run in linear space and afford the desired control over alignment scores. Section 2 describes precisely what we compute. Section 3 covers the graph model underlying the algorithms. The model is somewhat more general than we have needed before, though many techniques for linear-space alignment extend to this generalized framework, e.g., methods to delimit the range of suboptimal alignments (Chao *et al*., 1993b). (An inessential difference is that here we draw edges from lower left to upper right, for compatibility with our software package.) The basic linear-space multiple alignment

Work 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.

algorithm is developed in Section 4. Section 5 discusses the determination of constraints for the multiple-alignment process from constraints on the implied pairwise alignments; these ideas permit a straightforward adaptation of techniques of Chao *et al*. (1993a) to progressive multiple alignment.

Implementation of these alignment algorithms is described in Section 6, and Section 7 illustrates their application with an analysis of mammalian ε-globin genes. The C source code described in the Implementation section can be obtained over the Internet via anonymous ftp from groucho.cs.psu.edu.

2. Alignments and their Scores. An *alignment* of two sequences is obtained by inserting some number (perhaps 0) of dashes in each sequence to yield padded sequences of equal length, then placing the first padded sequence above the second. Fig. 2.1 gives an example. To emphasize that all sequence entries are required to appear in the alignment, we use the term *global* (as opposed to *local*). Each column of an alignment is called an *aligned pair*. For example, $\begin{bmatrix} C \\ T \end{bmatrix}$ is the second aligned pair in Fig. 2.1, and $\begin{bmatrix} - \\ A \end{bmatrix}$ is the third. In general, we require that an alignment not contain $\begin{bmatrix} - \\ - \end{bmatrix}$, which we call the *null column*. In contexts where null columns are permitted, we use the term *quasi-alignment* to emphasize that the ban on null columns has been temporarily lifted.

Two given sequences can be aligned in many different ways. To make explicit our criteria for preferring one alignment over another, we define a score for each alignment — the higher the score, the better the alignment. To do this, we assign a score denoted $\sigma(\begin{bmatrix} x \\ y \end{bmatrix})$ to each aligned pair $\begin{bmatrix} x \\ y \end{bmatrix}$, including the cases where x or y is a dash, and assess a constant *gap penalty*, denoted g, for each gap in the alignment; a *gap* is defined as a run of dashes in a row of the alignment that is terminated by either a non-dash symbol or an end of the row. Gap penalties are charged so that a single gap of length, say, k will be preferred to several gaps of total length k, which is desirable since a gap can be created in a single evolutionary event. The score of an alignment is defined as the sum of σ values for all aligned pairs, minus g times the number of gaps. Fig. 2.1 gives an example of an alignment's score.

```
AC--GTTCAA
ATACG---TA
```

Figure 2.1. A global alignment of ACGTTCAA and ATACGTA. If the score of an aligned pair is 1 for a match (i.e., whenever the upper and lower entries are identical) and −1 otherwise (including the case where one entry is a dash), and if the gap penalty is $g = 3$, then the alignment's score is −10. (There are 3 matches, 7 mismatches and 2 gaps.) There exist higher scoring alignments of these two sequences.

Selection of the scoring parameters σ and g is often a major factor affecting the usefulness of the computed alignments, since it determines which sequence regions will be considered non-aligning (e.g. because of negative scores) and what relationships will be assigned between aligning regions. Appropriateness of scoring parameters depends on several factors, including evolutionary distance between the species being compared. For example, after some experimentation we settled on the following scores for aligning the

β-like globin gene clusters of human and rabbit: matching nucleotides score +1, mismatches score −1, and a gap of length k is penalized $6 + 0.2 \times k$. In other words, $\sigma(\begin{bmatrix} x \\ x \end{bmatrix})$ = 1 when x is any of A, C, G or T, $\sigma(\begin{bmatrix} x \\ y \end{bmatrix}) = -1$ when $x \neq y$ and neither x nor y is a dash, $\sigma(\begin{bmatrix} x \\ - \end{bmatrix}) = \sigma(\begin{bmatrix} - \\ x \end{bmatrix}) = -0.2$ when x is not a dash, and g = 6. However, for comparing the same gene cluster between humans and the prosimian primate galago, whose ancestors diverged from human ancestors more recently than the rabbit-human split (roughly 55 million years ago, compared to 80 million), these parameters result in alignments that span some clearly non-homologous regions; changing the σ-score for pairs with one dash from −0.2 to −0.3 gives much better results. Another factor affecting parameter choice is that different regions of a genome may evolve at different rates. For example, we obtained much better results when comparing the α-like globin gene clusters of human and rabbit, which appear to be evolving at a significantly higher rate than the β-like globin gene clusters, by changing the gap cost from $6 + 0.2 \times k$ to $4 + 0.4 \times k$ (Hardison *et al*., 1991).

Scoring a multiple alignment

When simultaneously aligning more than two sequences, we want knowledge of appropriate parameters for pairwise alignment to lead immediately to appropriate settings for the multiple-alignment scoring parameters. Thus, we desire a scoring scheme for multiple alignments that is intimately related to our pairwise alignment scores. Of course, it is also necessary that the approach be amenable to a multiple-alignment algorithm that is reasonably efficient with computer resources, i.e., time and space.

To attain this tight coupling of pairwise and multiple alignment scores at reasonable computational expense, we have adopted the sum-of-pairs substitution scores and quasi-natural gap costs, as described by Altschul (1989). Some notation will help for a precise description of these ideas. Assume we are given $S_1, S_2, \cdots, S_m$, each of which is a sequence of "letters." A *quasi-alignment* of those sequences is an $m \times n$ array of letters and dashes, such that removing dashes from row i leaves the sequence S_i for $1 \leq i \leq m$. An *alignment* is a quasi-alignment containing no null columns, i.e., columns consisting entirely of dashes. For each pair of sequences, say S_i and S_j, rows i and j of the m-way alignment constitute a pairwise quasi-alignment of S_i and S_j; removing any null columns produces a pairwise alignment of these sequences.

To score an m-way alignment α given appropriate parameters for the score, say $Score_{i,j}$, for pairwise alignments between each sequence pair (S_i, S_j), it is natural to define

$$Score(\alpha) = \sum_{i<j} Score_{i,j}(\alpha_{i,j}) \qquad \text{(SP)}$$

where $\alpha_{i,j}$ is the pairwise alignment of S_i and S_j induced by α. However, as explained in detail by Altschul (1989), this scoring scheme results in undesirable algorithmic complexity. Altschul further observed that this complexity can be reduced dramatically if for every pair of rows of the m-way alignment we assess an additional gap penalty for each "quasi-gap", defined as follows. Fix a pair of rows and consider a gap, G, in the corresponding pairwise quasi-alignment, i.e., a run of consecutive gap symbols occurring in one of the rows (the run should be extended in both directions until it hits a letter or the end of the sequence). If at least one dash in G is aligned with a letter in the other row, then G corresponds to a gap in the pairwise alignment (i.e., after discarding null columns), and hence is penalized. The other possibility is that every dash in G is aligned with a dash in the other sequence. If the gap in the other sequence starts strictly before and ends strictly after G, then G is called a *quasi-gap* and may be penalized; if either end of G is aligned to an end of the gap in the other sequence, then G is not penalized. See Fig. 2.2.

In summary, we score a multiple alignment as follows. For each pair of rows, say rows i and j, fix appropriate substitution scores $\sigma_{i,j}$ and a gap cost $g_{i,j}$. Then the score for the multiple alignment is determined by equation (SP), where each $Score_{i,j}(\alpha_{i,j})$ is found by adding the σ values for non-null columns of the pairwise quasi-alignment, and subtracting a gap penalty g for each gap and for each "progressive" quasi-gap (defined below).

```
seq1 :  G C – T C
seq2 :  A – – – C
seq3 :  G – A T C
```

	match	mismatch	gap pair	gap	quasi–gap	Score
seq1 : **G C – T C** *seq2* : **A – – – C**	1	1	2	1	1	–7
seq1 : **G C – T C** *seq3* : **G – A T C**	3	0	2	2	0	–4
seq2 : **A – – – C** *seq3* : **G – A T C**	1	1	2	1	0	–4
						–15

Figure 2.2. Scoring a multiple alignment. All pairs of rows are scored as described in the text (all quasi-gaps are penalized) and the multiple alignment's score is the sum of the pairwise scores. The notion of a "quasi-gap" is defined in the text; it is introduced to reduce the complexity of computing a highest-scoring multiple alignment. For this example, pairwise alignments are scored by: match = 1, mismatch = –1, gap pair (one symbol is "–") = –0.5 and gap penalty = –3. Null columns (both entries "–") always score 0. (In general, a different scoring scheme can be used for each pair of rows.) The alignments shown are not intended to be optimal.

End-gaps

Occasionally, different scoring criteria are applied to *end-gaps*, i.e., gaps that are terminated by an end of the row, as pictured in Fig. 2.3. For example, when constructing a three-way alignment between two long sequences that match from end to end and a shorter sequence that matches only a portion of the longer ones, it might be appropriate to not penalize end-gaps in the short sequence. In other cases, it might be appropriate to penalize only end-gaps lying to the right of a particular sequence.

–	–	–	–	x	x	–	–	–
x	x	x	–	x	x	–	x	x
–	–	x	x	x	x	x	x	x
–	x	x	x	–	–	x	x	–

Figure 2.3. End-gaps, as indicated by shaded regions, where x stands for a non-dash letter.

Let us be more precies about a point that has been glossed over so far. An aligned pair actually consists of paired sequence positions, where a dash symbol denotes the position between two specific sequence entries (or at an end of the sequence). Thus the first and last aligned pairs in Fig. 2.1 are distinct, though they are both represented by $\begin{bmatrix} A \\ A \end{bmatrix}$. For our work to date, it has been adequate to use scores σ that depend only on the contents of the two locations, not their positions within the sequences. Thus, $\sigma(\begin{bmatrix} A \\ A \end{bmatrix})$ does not depend on where the particular instances of A occur. However, there are situations where position-dependent scores are quite appropriate (Gribskov *et al*., 1987). Similarly, the algorithms we use could handle position-dependent gap penalties. However, the only form of position-dependence that we regularly use concerns handling of end-gaps.

Progressive alignment

For computing a multiple alignment, we adopt the approach of "progressive" pairwise alignments (Feng and Doolittle, 1987). Thus, later steps of the process align two "sequences", one or both of which can themselves be an alignment (i.e., sequence of fixed-height columns), as in Fig. 2.4.

$$\begin{bmatrix} - \\ - \end{bmatrix}$$

$$\begin{array}{cccc} \mathbf{A} & - & - & \mathbf{A} \\ \mathbf{C} & \mathbf{G} & \mathbf{T} & \mathbf{A} \end{array} + \begin{array}{ccccc} \mathbf{A} & \mathbf{G} & \mathbf{T} & \mathbf{G} & \mathbf{G} \\ \mathbf{A} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{A} \end{array} = \begin{array}{ccccc} \mathbf{A} & - & - & - & \mathbf{A} \\ \mathbf{C} & \mathbf{G} & - & \mathbf{T} & \mathbf{A} \\ \mathbf{A} & \mathbf{G} & \mathbf{T} & \mathbf{G} & \mathbf{G} \\ \mathbf{A} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{A} \end{array}$$

Figure 2.4. Aligning two pairwise alignments. The columns of each given pairwise alignment are treated as "symbols", and these sequences of symbols are aligned by inserting null columns. In this example, inserting a null column in the first alignment creates a quasi-gap relative to the first two rows of the 4-way alignment; this quasi-gap is *not* a progressive quasi-gap (for this step of a progressive alignment).

The notion of a *progressive quasi-gap* can now be defined. Consider aligning alignments A and B. The progressive quasi-gaps (relative to that step of the progressive alignment process) are all quasi-gaps in the resulting alignment *except* those that appear between two rows within an input alignment (A or B) as a result of inserting a null column. See Fig. 2.4 for an example. Thus, any quasi-gap relative to a row of A and a row of B is a progressive quasi-gap.

Each step of the progressive alignment process is done so as to minimize the sum-of-pairs score, where progressive quasi-gaps (but not other quasi-gaps) are penalized. With a little extra work, all quasi-gaps could be counted, but we prefer to not do so since quasi-gaps are merely an annoying side-effect of the alignment algorithm. Though each step is performed optimally, there is no guarantee that the final multiple alignment is highest-scoring over all possible ways of aligning the given sequences.

Since the relationships between entries of two original sequences are fixed at the first step that aligns those sequences (or alignments containing those sequences), it is wise to first compute the pairwise alignments that warrant the most confidence, then combine those into multiple alignments. An appropriate order for progressive alignment can be inferred directly from an evolutionary tree for those sequences. For example, an evolutionary tree for (homologous sequences from) human, galago, rabbit and mouse is given in panel (A) of Fig. 2.5. 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

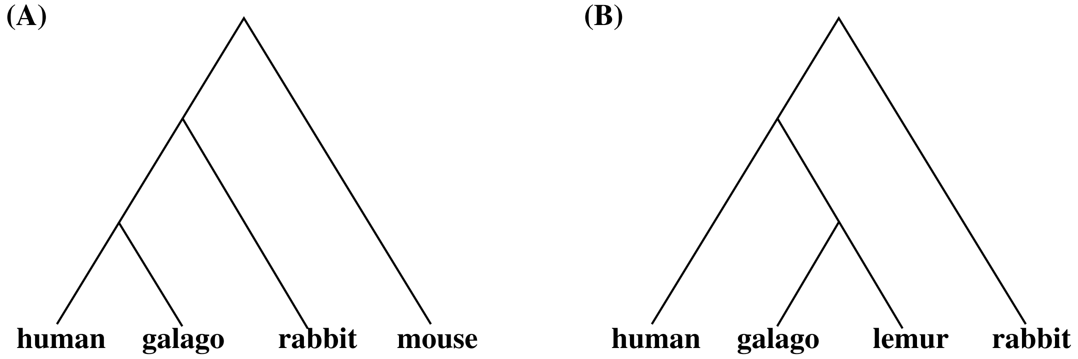

Figure 2.5. Evolutionary trees. Branch lengths in these two idealized trees are not related to evolutionary time.

3. A Graph Model of the Alignment Process. 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 alignment algorithms. This section describes a graph model that is appropriate for progressive multiple alignment with quasi-natural gap costs.

Consider sequences $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$. The *second-order alignment graph* for A and B, denoted $G^{(2)}_{A,B}$, is a directed graph with the following nodes, edges, and node labels. At each grid point (i, j) satisfying $i \in [1, M]$ and $j \in [1, N]$, there are three labeled nodes, as specified in Fig. 3.1 and pictured in Fig. 3.2. Grid points $(i, 0)$ for $i \in [1, M]$, or $(0, j)$ for $j \in [1, N]$, have one node, again specified in Fig. 3.1. In addition, there is a node $(0, 0)_D$ labeled with π_{start} and having edges to $(1, 0)_H$, $(1, 1)_D$ and $(0, 1)_V$, and a node $(M+1, N+1)_D$ labeled π_{end} and with entering edges from all three nodes at (M, N).

node	*condition*	*label*	*entering edges from*
$(i, j)_H$	$i > 0$	$\begin{bmatrix} a_i \\ - \end{bmatrix}$	all nodes at $(i-1, j)$
$(i, j)_V$	$j > 0$	$\begin{bmatrix} - \\ b_j \end{bmatrix}$	all nodes at $(i, j-1)$
$(i, j)_D$	$i, j > 0$	$\begin{bmatrix} a_i \\ b_j \end{bmatrix}$	all nodes at $(i-1, j-1)$

Figure 3.1. The nodes at a general grid point (i, j) in the graph model for a second-order alignment graph $G^{(2)}_{A,B}$. H, V and D stand for horizontal, vertical and diagonal, respectively, which are the orientations of the entering edges.

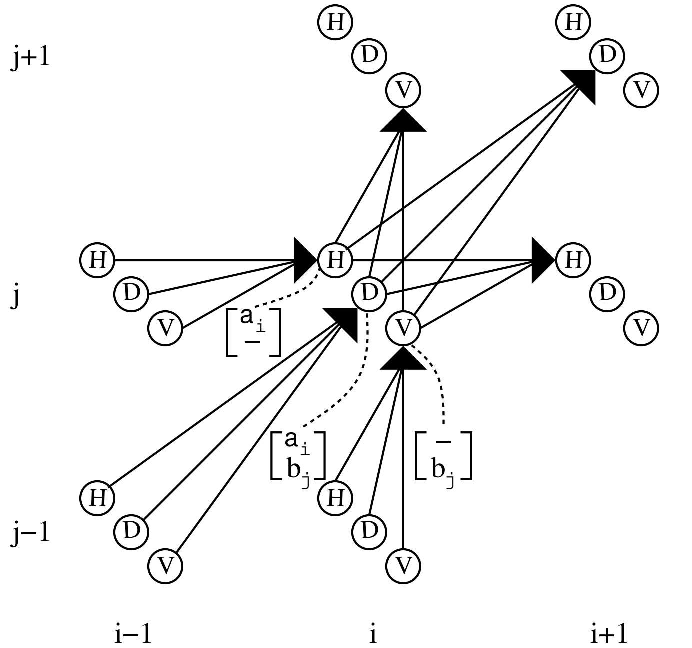

Figure 3.2. Edges entering and leaving the three nodes at grid point (i, j) in a second-order alignment graph, together with labels on those nodes. The case where $0 < i < M$ and $0 < j < N$ is pictured; the boundary cases are simplified in obvious ways.

It is readily seen that alignments of A and B are in one-to-one correspondence with paths from $(0, 0)_D$ to $(M+1, N+1)_D$. For instance, when a path crosses from column $i-1$ to column i, it picks up an aligned pair with top entry a_i. A path from $(0, 0)$ to (M, N) has zero or more vertical edges, then a horizontal or diagonal edges to column 1, then zero or more vertical edges, then an edge to column 2, then $\cdots$, so the top entries of the labels along the path are a_1, a_2, $\cdots$, 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.

Graphs $G^{(2)}_{A,B}$ are natural for dealing with a "second-order" alignment-scoring scheme, which is defined as follows. In addition to a score $\sigma(\pi)$ for each possible aligned pair π, a score is now added for each pair $\pi_i \pi_{i+1}$ of aligned pairs that are adjacent in the alignment. To include measurements of such dependencies at the start and the end of the alignment, we utilize special "start-of-alignment" and "end-of-alignment" pairs, π_{start} and π_{end}. Thus, we assume that a score $\sigma^{(2)}(\pi, \Pi)$ is given for every possible pair of aligned pairs, π and Π. Then the *second-order alignment score* of an alignment with n aligned pairs π_i is defined as

$$\sum_{i=1}^{n} \sigma(\pi_i) + \sum_{i=0}^{n} \sigma^{(2)}(\pi_i, \pi_{i+1})$$

where $\pi_0 = \pi_{start}$ and $\pi_{n+1} = \pi_{end}$.

Scores can be assigned in a natural way to make an alignment's score identical to the sum of edge and node scores on the corresponding path. The following theorem states our conclusions about $G^{(2)}_{A,B}$ more precisely. The proof is left as an exercise.

Theorem 3.1. Let $G^{(2)}_{A,B}$ be the second-order alignment graph for sequences A and B. With each path from $(0,0)$ to $(M+1, N+1)$ associate the alignment formed by concatenating the node labels along 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. Furthermore, suppose scores are assigned to nodes and edges by the rules: a node labeled with the aligned pair π scores $\sigma(\pi)$ and the edge from a node labeled π_p to a node labeled π_q scores $\sigma^{(2)}(\pi_p, \pi_q)$. Then the sum of the scores of a path's nodes and edges is exactly the score of the corresponding alignment for second-order alignment scores.

Second-order alignment scores include the handling of quasi-natural gap costs when the sequences being aligned are themselves alignments (sequences of aligned pairs, triples, etc.). To understand the problems that arise when scoring gaps in multiple alignments, and to see how we solve them, consider two fixed rows, say rows r and s, in a multiple alignment. Ideally, we would like to count the number of gaps in the pairwise alignment implied by those two rows (i.e., after removal of $\begin{bmatrix} - \\ - \end{bmatrix}$ pairs). However, we are limited by the alignment process to gathering information from "windows" of only two adjacent columns of rows r and s. The rules we use are given in Fig. 3.3. In words, at a transition point where a gap starts in just one sequence, count one gap start; at a transition from apparent gaps in both sequences to a gap in just one sequence, count one gap end.

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.

Gap count = number of gap starts + number of gap ends.

Figure 3.3. Gap-counting rules, where x stands for a non-dash letter or an entry of either π_{start} or π_{end}.

Applications of the rules of Fig. 3.3 are given in Fig. 3.4. To correctly count implied pairwise gaps for the third case of Fig. 3.4 requires comparison of parts of the quasi-alignment that are separated by several columns. While this is easy to do when scoring a given alignment, it is prohibitively expensive when computing an optimal alignment. Thus, we use the gap-counting rules of Fig. 3.3, which correctly count the number of quasi-gaps. Indeed, this is exactly how the notion a "quasi-gap" arises; we do the best we can at counting gaps, subject to the constraints that every gap gets penalized and that only adjacent aligned pairs are inspected to compute the alignment's score, then look to see what is actually counted.

```
x x − − − − x x
x x x x − − x x
 ∧ ∧ ∧ ∧ ∧ ∧ ∧
 0 1 0 0 0 0 0   ⇒ 1 gap  (correct)

x x − − x x x x
x x x x − − x x
 ∧ ∧ ∧ ∧ ∧ ∧ ∧
 0 1 0 1 0 0 0   ⇒ 2 gaps  (correct)

x x − − − − x x
x x x − − x x x
 ∧ ∧ ∧ ∧ ∧ ∧ ∧
 0 1 0 0 1 0 0   ⇒ 2 gaps  (incorrect)
```

Figure 3.4. Three applications of Fig. 3.3's gap-counting rules.

The precise relationship between second-order alignment scores and our approach to multiple alignments is as follows. Each stage of the progressive alignment process computes an alignment between a p-way alignment A and a q-way alignment B. (A DNA or protein sequence is considered a 1-way alignment.) Thus, $A = a_1 a_2 \cdots a_M$, where each a_i is a column of p letters, $B = b_1 b_2 \cdots b_N$ has N columns of q letters each and, e.g., the "aligned pair" $\begin{bmatrix} a_i \\ - \end{bmatrix}$ is a column of height $p+q$ whose last q entries are "–". The σ-value for a column of the resulting $(p+q)$-way alignment is the sum of σ-values for all pairs of its entries that differ from $\begin{bmatrix} - \\ - \end{bmatrix}$. The $\sigma^{(2)}$ value for adjacent columns is the sum of counts obtained by the rules of Fig. 3.3 for each pair of rows. Fig. 3.5 gives an example.

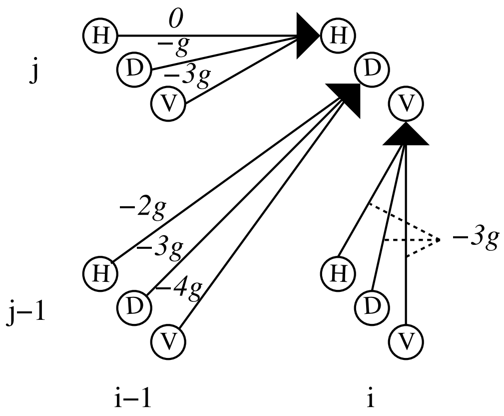

Figure 3.5. An example of scores on edges entering nodes at grid point (i, j) for progressive alignment using quasi-natural gap costs. The sequences $A = a_1 a_2 \cdots a_M$ and $b_1 b_2 \cdots b_N$ are each a pairwise alignment of "basic" sequences (e.g., DNA), with $a_{i-1} = a_i = b_{j-1} = \begin{bmatrix} x \\ - \end{bmatrix}$ and $b_j = \begin{bmatrix} - \\ x \end{bmatrix}$, where x denotes a non-dash basic sequence entry. As an example, consider the edge from $(i-1, j-1)_H$ to $(i, j)_D$. The H node is labeled $\begin{bmatrix} a_{i-1} \\ - \end{bmatrix}$ and the D node is labeled $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$. Thus the edge score is:

$$\sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ - & b_j \end{bmatrix}) = \sigma^{(2)}(\begin{bmatrix} x & x \\ - & - \\ - & - \\ - & x \end{bmatrix}) = -2g$$

4. Alignment Algorithm. This section describes a linear-space algorithm for pairwise alignment with second-order scores (and hence for progressive multiple alignment with quasi-natural gap costs). Following the strategy originated by Hirschberg (1975), the algorithm's first component is a procedure that computes maximal path scores from node $(i_1, j_1)_{type}$, where *type* is either D or H, to every vertex in column *mid* lying between rows j_1 and j_2. Here, the score of such a path is the sum of its node and edge scores *except for the initial node*, $(i_1, j_1)_{type}$. The algorithm of Fig. 4.2 is straightforward if one keeps in mind the weights on nodes at grid point (i, j) and on their entering edges, as pictured in Fig. 4.1.

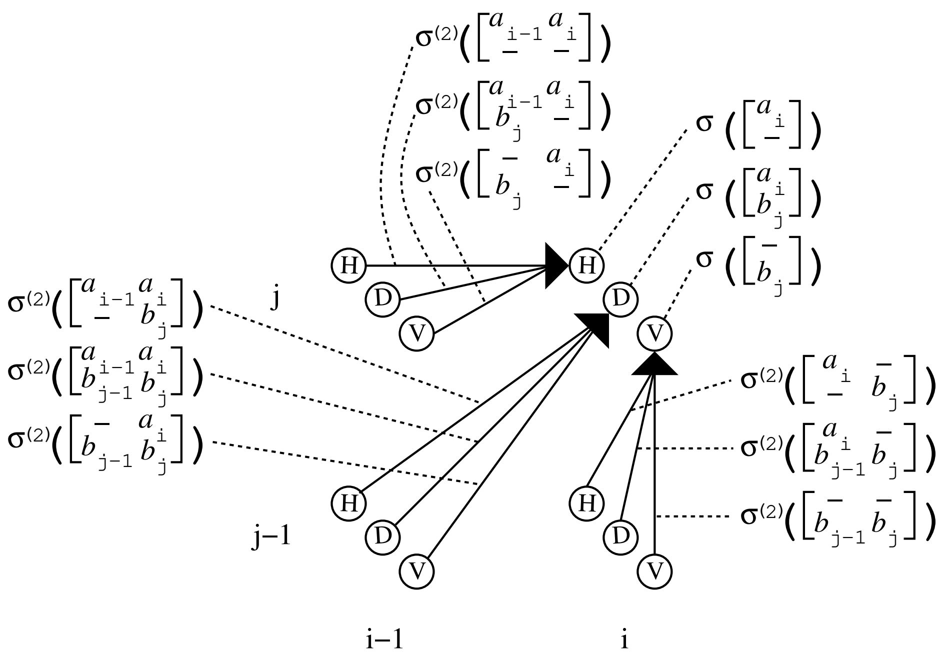

Figure 4.1. Edges entering the nodes at grid point $(i,\ j)$ of $G^{(2)}_{A,\,B}$ and their scores.

procedure *forward*$(i_1, j_1, type, mid, j_2)$
  for X in $\{'D', 'H', 'V'\}$ **do**
    $X[i_1, j_1] \leftarrow$ **if** $X = type$ **then** 0 **else** $-\infty$
  for $j \leftarrow j_1 + 1$ **to** j_2 **do**
    $D[i_1, j] \leftarrow H[i_1, j] \leftarrow -\infty$

$$V[i_1, j] \leftarrow \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix}) + \begin{cases} D[i_1, j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i_1} & - \\ b_{j-1} & b_j \end{bmatrix}) & \text{if } j = j_1 + 1 \text{ and } type = 'D' \\ H[i_1, j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i_1} & - \\ - & b_j \end{bmatrix}) & \text{if } j = j_1 + 1 \text{ and } type = 'H' \\ V[i_1, j-1] + \sigma^{(2)}(\begin{bmatrix} - & - \\ b_{j-1} & b_j \end{bmatrix}) & \text{if } j > j_1 + 1 \end{cases}$$

  for $i \leftarrow i_1 + 1$ **to** *mid* **do**
    $D[i, j_1] \leftarrow V[i, j_1] \leftarrow -\infty$

$$H[i, j_1] \leftarrow \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}) + \begin{cases} D[i-1, j_1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ b_{j_1} & - \end{bmatrix}) & \text{if } i = i_1 + 1 \text{ and } type = 'D' \\ H[i-1, j_1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ - & - \end{bmatrix}) & \text{otherwise} \end{cases}$$

    for $j \leftarrow j_1 + 1$ **to** j_2 **do**

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

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

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

Figure 4.2. Quadratic-space, score-only alignment algorithm for second-order scores.

The next step is to see that the algorithm of Fig. 4.2 can be revised to run in linear space. Indeed, it is apparent that the scores in column i depend only on those in column $i-1$. Thus, after treating column i, the space used for values in column $i-1$ can be recycled to hold values in column $i+1$. In other words, we can get by with space for two columns, since ultimately we want only the scores $D[mid, j]$ and $H[mid, j]$. In fact, arrays for one column's scores are adequate. Arrays $D[j]$, $H[j]$ and $V[j]$ hold the most recently computed value in row j, so that as new values are computed, they overwrite old values. There is a slight conflict in this strategy, since two "active" values are needed for D, H and V in the current row, necessitating additional scalars, d, h and v. Fig. 4.3 shows the grid locations of values in D, H and V, and of scalars d, h and v, when (i, j) is reached in the computation of Fig. 4.4. For example, $D[k]$ holds path scores to D nodes for column i when $k < j-1$, and for column $i-1$ when $k \geq j-1$.

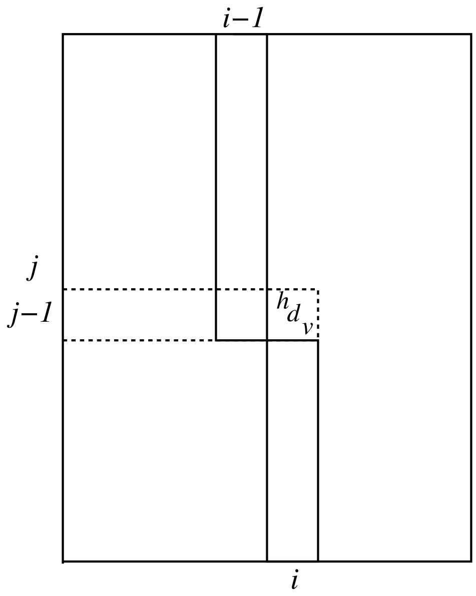

Figure 4.3. Grid locations of entries of vectors D, H and V of length $j_2 - j_1 + 1$ just before the maximum path scores are evaluated at grid point (i, j). Additionally, scalars d, h and v hold the path scores at $(i, j-1)_D$, $(i, j-1)_H$ and $(i, j-1)_V$, respectively.

procedure *linear_forward*$(i_1, j_1, type, mid, j_2)$
  for X in $\{'D', 'H', 'V'\}$ **do**
    $X[j_1] \leftarrow$ **if** $X = type$ **then** 0 **else** $-\infty$
  for $j \leftarrow j_1 + 1$ **to** j_2 **do**
    $D[j] \leftarrow H[j] \leftarrow -\infty$

$$V[j] \leftarrow \sigma(\begin{bmatrix} - \\ b_j \end{bmatrix}) + \begin{cases} D[j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i_1} & - \\ b_{j-1} & b_j \end{bmatrix}) & \text{if } j = j_1 + 1 \text{ and } type = 'D' \\ H[j-1] + \sigma^{(2)}(\begin{bmatrix} a_{i_1} & - \\ - & b_j \end{bmatrix}) & \text{if } j = j_1 + 1 \text{ and } type = 'H' \\ V[j-1] + \sigma^{(2)}(\begin{bmatrix} - & - \\ b_{j-1} & b_j \end{bmatrix}) & \text{if } j > j_1 + 1 \end{cases}$$

  for $i \leftarrow i_1 + 1$ **to** *mid* **do**
    $d \leftarrow v \leftarrow -\infty$

$$h \leftarrow \sigma(\begin{bmatrix} a_i \\ - \end{bmatrix}) + \begin{cases} D[j_1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ b_{j_1} & - \end{bmatrix}) & \text{if } i = i_1 + 1 \text{ and } type = 'D' \\ H[j_1] + \sigma^{(2)}(\begin{bmatrix} a_{i-1} & a_i \\ - & - \end{bmatrix}) & \text{otherwise} \end{cases}$$

    for $j \leftarrow j_1 + 1$ **to** j_2 **do**

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

      $D[j-1] \leftarrow d$
      $H[j-1] \leftarrow h$
      $V[j-1] \leftarrow v$
      $d \leftarrow t$

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

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

    $D[j_2] \leftarrow d$
    $H[j_2] \leftarrow h$
    $V[j_2] \leftarrow v$

Figure 4.4. Linear-space, score-only alignment algorithm for second-order scores.

The next component of Hirschberg's approach is computation of second-order alignment scores in the reverse direction. The algorithm of Fig. 4.6 computes the maximum path score from each vertex lying on column *mid* between rows j_1 and j_2, over all paths ending at node $(i_2, j_2)_{type}$, where *type* is either D or H. Now, the score of such a path is the sum of its node and edge scores *except for the initial node*. For "backwards" computation, the relevant edges are those *leaving* nodes at grid point (i, j); see Fig. 4.5. The derivation of a linear-space version of Fig. 4.6 is left as an exercise for the reader.

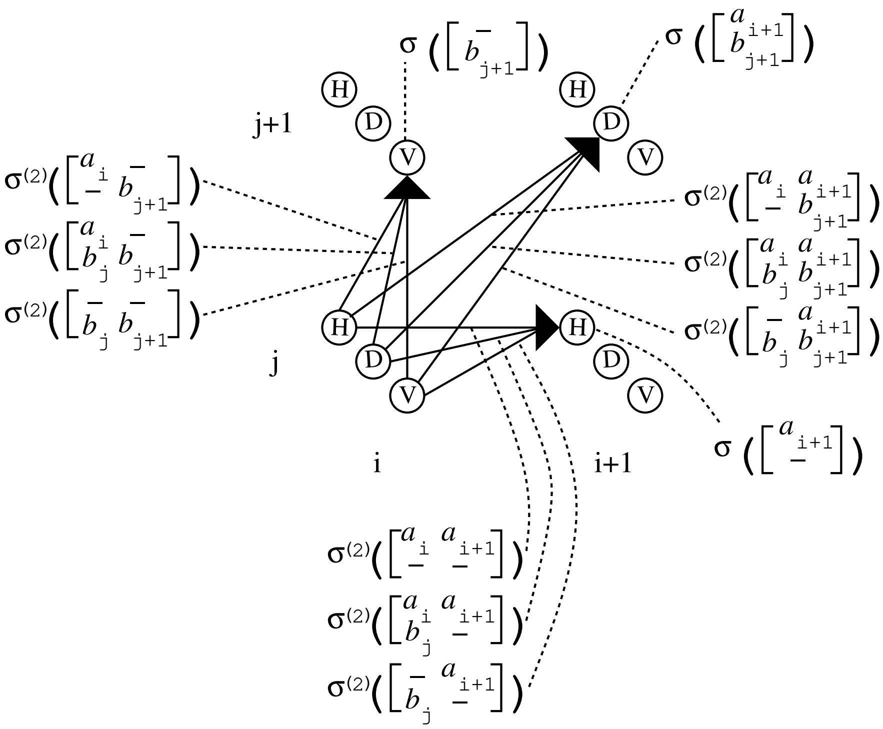

Figure 4.5. Edges leaving the nodes at grid point (i, j) and their scores.

procedure $backward(mid, j_1, i_2, j_2, type)$

  for X in $\{'D', 'H', 'V'\}$ **do**

    $X[i_2, j_2] \leftarrow$ **if** $X = type$ **then** 0 **else** $-\infty$

  for $j \leftarrow j_2 - 1$ **down to** j_1 **do**

    $D[i_2, j] \leftarrow H[i_2, j] \leftarrow V[i_2, j] \leftarrow -\infty$

  for $i \leftarrow i_2 - 1$ **down to** mid **do**

    if $type = H$ **then**

$$
\begin{aligned}
D[i, j_2] &\leftarrow H[i+1, j_2] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix}) + \sigma^{(2)}(\begin{bmatrix} a_i & a_{i+1} \\ b_{j_2} & - \end{bmatrix}) \\
H[i, j_2] &\leftarrow H[i+1, j_2] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix}) + \sigma^{(2)}(\begin{bmatrix} a_i & a_{i+1} \\ - & - \end{bmatrix}) \\
V[i, j_2] &\leftarrow H[i+1, j_2] + \sigma(\begin{bmatrix} a_{i+1} \\ - \end{bmatrix}) + \sigma^{(2)}(\begin{bmatrix} - & a_{i+1} \\ b_{j_2} & - \end{bmatrix})
\end{aligned}
$$

    else

      $D[i, j_2] \leftarrow H[i, j_2] \leftarrow V[i, j_2] \leftarrow -\infty$

    for $j \leftarrow j_2 - 1$ **down to** j_1 **do**

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

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

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

Figure 4.6. Computation of second-order alignment scores in the reverse direction.

Consider the problem of finding an optimal path from $(i_1, j_1)_{type_1}$ to $(i_2, j_2)_{type_2}$. Note that it is immaterial whether path scores include scores at the first or last vertex, so long as all path scores follow the same convention. To divide the problem, forward and reverse score-only passes are performed, both stopping at the middle column, i.e., $mid = \lfloor (i_1 + i_2)/2 \rfloor$. Thus, for each node along the middle column, we have the optimal score from $(i_1, j_1)_{type_1}$ to that node and the optimal score from that node to $(i_2, j_2)_{type_2}$. The specific scores used in the reverse pass omit the score of nodes on column mid, so the sum of these two scores at a node at grid point (mid, j) is the maximum score of a path from $(i_1, j_1)_{type_1}$ to $(i_2, j_2)_{type_2}$ and passing through that node. Thus we can find a node $(mid, J)_X$ where an optimal path crosses the middle column, and where, without loss of generality, X is D or H. This reduces the problem to finding an optimal path from $(i_1, j_1)_{type_1}$ to $(mid, J)_X$ and an optimal path from $(mid, J)_X$ to $(i_2, j_2)_{type_2}$. Performing this computation recursively finds all nodes on an optimal path.

Fig. 4.7(A) shows the two subproblems and each of their ‘‘subsubproblems’’. Note that regardless of where the optimal path crosses the middle column, the total size 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 X 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 problem size, this approach will deliver an optimal alignment in about twice the time needed to compute merely its score.

Fig. 4.7(B) 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. 4.8 provides detailed pseudocode for the linear-space alignment algorithm.

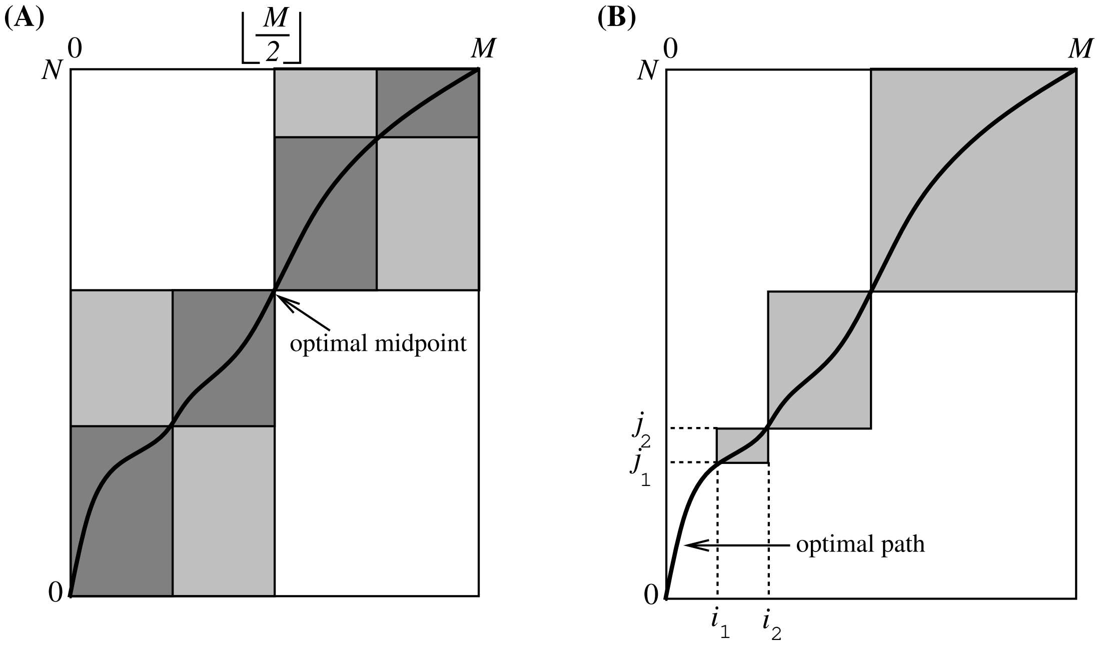

Figure 4.7. (A) The two subproblems and four subsubproblems in Hirschberg's linear-space alignment procedure. (B) Shapshot of the execution of Hirschberg's algorithm, where shaded areas indicate problems remaining to be solved.

shared string arrays $a_1 a_2 \cdots a_M$, $b_1 b_2 \cdots b_N$
shared temporary integer arrays $D^-[0..N]$, $H^-[0..N]$, $V^-[0..N]$, $D^+[0..N]$, $H^+[0..N]$, $V^+[0..N]$
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
    $mid \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$
      /* call the linear-space version of $forward(i_1, j_1, type_1, mid, j_2)$ */
    $D^-[j] \leftarrow$ maximum path score from $(i_1, j_1)_{type_1}$ to $(mid, j)_D$ $j \in [j_1, j_2]$
    $H^-[j] \leftarrow$ maximum path score from $(i_1, j_1)_{type_1}$ to $(mid, j)_H$ $j \in [j_1, j_2]$
      /* call the linear-space version of $backward(mid, j_1, i_2, j_2, type_2)$ */
    $D^+[j] \leftarrow$ maximum path score from $(mid, j)_D$ to $(i_2, j_2)_{type_2}$ $j \in [j_1, j_2]$
    $H^+[j] \leftarrow$ maximum path score from $(mid, j)_H$ to $(i_2, j_2)_{type_2}$ $j \in [j_1, j_2]$
      /* find where maximum-score path crosses mid column */
    $(J, X) \leftarrow$ values $J \in [j_1, j_2]$ and $X \in \{'D', 'H'\}$ that maximize $X^-[J] + X^+[J]$
    $path(i_1, j_1, type_1, mid, J, X)$
    $path(mid, J, X, i_2, j_2, type_2)$

Figure 4.8. Algorithm for delivering an optimal alignment with second-order alignment scores.

The algorithm can accommodate alternative ways of scoring end-gaps in a progressive multiple alignment. Computing $\sigma(\begin{bmatrix} a_i \\ b_j \end{bmatrix})$, where a_i and b_j are columns of alignments, involves adding values that may have the form, say, $\sigma(\begin{bmatrix} - \\ x \end{bmatrix})$, where x is a sequence entry. The alignments being aligned can be consulted to see if the dash is in an end-gap of the corresponding (non-alignment) sequence, and a special end-gap penalty can be assessed if appropriate. Similar remarks hold for the assessment of penalties g when computing $\sigma^{(2)}$ values.

Consider the time taken to align alignments A and B of lengths M and N, respectively. The score of an optimal alignment of A and B can be determined by running Fig. 4.2 with $i_1 = j_1 = 0$, $type$ = 'D', $mid = M$ and $j_2 = N$. 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. 3.3). 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. 4.2 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 execute Fig. 4.2 or 4.4, 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 (Fig. 4.8) is approximately twice the time needed to compute merely its score (Fig. 4.2); see Myers and Miller (1988) for a complete explanation. 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 ½$n(n-1)$, 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.

5. Constrained Alignment. 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. Chao *et al*. (1993a) described a linear-space algorithm, substantially different from Hirschberg's strategy, for computing such pairwise, constrained alignments. Although the graph model used here is somewhat different from that used by Chao *et al*., a straightforward modification of their method works for progressive multiple alignment with quasi-natural gap costs, and modified penalties for end-gaps can again be accommodated. However, it should be noted that our approach does not guarantee that an optimal solution to the constrained problem is optimal for the original (unconstrained) problem.

The remainder of the section discusses one way that constraints on each step of a progressive alignment can be determined from constraints on the implied pairwise alignments. Denote the sequences being aligned as *seq* 1, *seq* 2, $\cdots$, *seq* n, in the order they are to appear in the multiple alignment, and denote the k^{th} entry of *seq* p by a_k^p. Assume that for each pair (p, q) with $1 \le p < q \le n$ we are given lower and upper bounding functions $L_{p,q}$ and $U_{p,q}$ determining a constraining region for the pairwise alignment of *seq* p with *seq* q. Informally, we are concerned only with alignments that pair a_k^p with a_j^q where $L_{p,q}[k] \le j \le U_{p,q}[k]$. Fig. 5.1 depicts the situation, and precise requirements for L and U are given by Chao *et al*. (1993a).

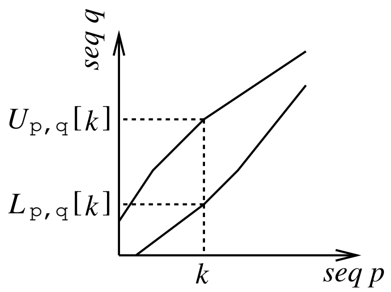

Figure 5.1. Constraining region for pairwise alignment.

Consider one step of progressive multiple alignment, in which alignment A is to be aligned with alignment B, and consider the i^{th} column of A. Pick any non-dash entry of that column (there must be at least one), say a^p_k where $k \le i$. Let *seq q* be any of the sequences involved in alignment B, which implies that $p < q$. The entry $a^q_{L_{p,q}[k]}$ of *seq q* must appear in exactly one column of B; let $L^*_{p,q}[i]$ denote the position of that column in B. Similarly, $a^q_{U_{p,q}[k]}$ appears in column $U^*_{p,q}[i]$ of B. Fig. 5.2 depicts the situation.

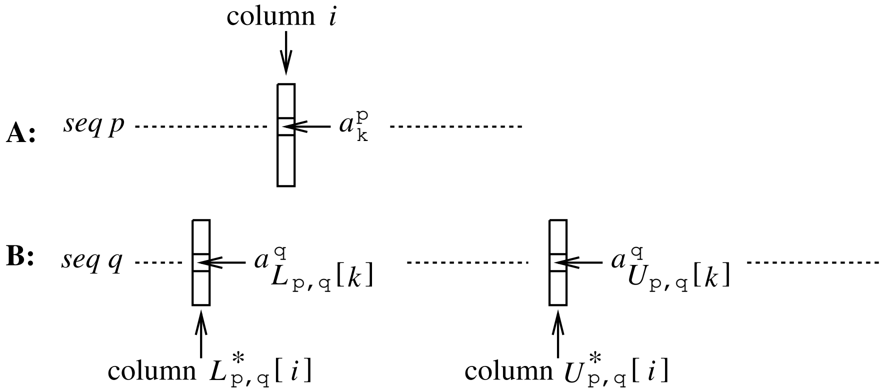

Figure 5.2. Constraining conditions for one step of a progressive multiple alignment.

Define $L[i]$ to be the maximum of the $L^*_{p,q}[i]$, taken over all non-dash entries of column i of A and all sequences entering into B. Determination of an upper bound $U[i]$ is similar. These functions need the following minor "normalizations". First, let $L'[i] = \min_{i \le j} L[j]$ (which can be computed in a linear-time backward pass, since $L'[i] = \min\{L[i], L'[i+1]\}$) and let $U'[i] = \max_{j \le i} U[j]$. In effect, the constraining region is expanded minimally to make the boundary lines non-decreasing functions of i. Then, if $L'[i] > U'[i]$, we replace both $L'[i]$ and $U'[i]$ by their average; this guarantees that there exists an alignment satisfying the constraints. Finally, since the regions may be overly restrictive, it is sometimes desirable to have a user-defined "expansion width" w that is subtracted from $L'[i]$ and added to $U'[i]$ for all i.

6. Implementation. We have implemented the above methods for progressive multiple alignment with quasi-natural gap costs. One program, called *yama* (*y*et *a*nother *m*ultiple *a*ligner), implements Hirschberg's linear-space algorithm for second-order alignment scores, as described in Section 4. Another program, called *yama2*, performs constrained progressive alignment as described in Section 5.

Both *yama* and *yama2* allow different substitution scores σ for each sequence pair. Moreover, both 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 $g_{p,q} + e_{p,q} k$. The programs also allow users to specify an aligning tree that indicates an appropriate order for progressive alignment. Finally, *yama2* reads lower and upper bounding functions $L_{p,q}$ and $U_{p,q}$ for each sequence pair, plus an ‘‘expansion width’’ w, as described in Section 5.

Let n denote the number of sequences being aligned and let L be the length of the resulting alignment. Storing inputs for *yama* and *yama*2 requires at most nL 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. Also, *yama2* uses at most $n(n-1)L$ integers for the $L_{p,q}$ and $U_{p,q}$. For intermediate computed values and the final alignment, *yama* uses space for $2nL$ characters and $8L$ integers, while *yama2* uses $(2n+9)L$ characters and $14L$ integers.

7. An Example. To illustrate use of the programs *yama* and *yama2*, we aligned the ε-globin gene regions of human, galago, rabbit and mouse. We began by computing the highest-scoring local alignment for each of the six sequence pairs, using the *sim* program (Huang *et al*., 1990). For each run, a match scored +1 and a mismatch score −1. The score for an aligned pair with one dash was −0.3 for human versus galago, and −0.2 otherwise. The gap-open penalty g was 5 for comparisons involving mouse, and 6 otherwise. (Mouse diverged earlier and is evolving faster.) In each case, the alignment covered all three exons and at least 300 bp of the 5′ flank.

The left-most and right-most positions that align consistently in all six pairwise alignments (as determined by the program of Miller, 1993) were used for the start and end positions of the multiple alignments. This gives a 4-way alignment that spans 1851 basepairs of the human sequence. For both *yama* and *yama2*, alignment scores were the same as for the *sim* alignments, end-gaps were penalized the same as internal gaps, and the progressive alignment was guided by the evolutionary tree of Fig. 2.5(A). For constraining regions for sequence pairs (Fig. 5.1), we simply used the *sim* alignments as both the lower and upper bound. Thus $L_{p,q}[i] = U_{p,q}[i]$ except when there is a vertical gap in column i of the pairwise alignment between *seq p* and *seq q*. This gives an extremely narrow region, which necessitates the use of a non-zero ‘‘expansion width’’ w to get satisfactory results. Table 7.1 gives alignment scores and execution times for various values of w. Even with $w = 2$, *yama2* locates almost all of the well-conserved (i.e., boxed) regions of Fig. 7.1; the only difference is a minor change in the first two boxes.

w	score	time (sec.)
0	−16333	0.9
1	14099	1.4
2	23129	2.1
3	25445	2.5
4	26843	3.4
5	28213	3.7
6	29019	4.4
8	29461	5.6
10	30107	6.9
12	30267	8.0
16	30551	10.3
20	30765	12.9
30	30803	18.5
50	30891	30.1
500	30891	342
yama	30929	709

Table 7.1. Alignment scores and execution times (on a Sun SparcStation II) for *yama2* with various value of *w* (expansion width). The last line gives the results for *yama*, which performs unconstrained alignment. On certain other examples, *yama2* produces a higher score than does *yama*.

The portion of the multiple alignments containing the proximal 5′-flanking region of the four ε-globin genes is shown in Fig. 7.1. This region extends for almost 400 bp 5′ to the major start site for transcription (the cap site) at 19486 in the human sequence. Highly conserved segments, indicating the presence of potential regulatory sequences, are seen throughout this region. Although some of these aligning blocks had been observed in previous alignments by inspection (Bailey *et al*., 1992; Hardison, 1983), others are revealed by the *yama* program. Some of the conserved sequences are well-known from previous functional and binding assays. For example, the sequences around the motifs TTATCA, CTCCACCCC, and CCAAT in the first 160 bp 5′ to the cap site are all well conserved. These correspond to the binding sites for transcription factors GATA1, Sp1 (or a related protein) and one of the CCAAT-binding factors (CBF), respectively. These proteins bind specifically to their cognate sites 5′ to the ε-globin gene, and each site has been implicated in either basal promoter function or in the response to an erythroid enhancer element by mutating this sequence and testing its expression in cultured cells (Gong and Dean, 1993; Gong *et al*., 1991; Motamed *et al*., 1993; Yu *et al*., 1991). Although it has not been studied extensively in this gene, the AATAAA motif located about 30 bp 5′ to the cap site matches the binding site for TBP (TATA-binding protein), the DNA-binding protein in TFIID, which is a complex required for assembly of the general transcription complex including RNA polymerase II (Nakajima *et al*., 1988). Thus demonstrably important sequences are readily identified in the multiple alignments generated by *yama*. Other segments that are as well conserved, such as the sequences between the CBF and TBP sites, just 5′ to the Sp1 site, and 5′ to the YY1 sites, are strong candidates for other regulatory elements and warrant further study.

Additionally, the multiple alignments can provide important insight in resolving apparently contradictory results. For example, a segment of about 30 bp located approximately 300 bp 5′ to the cap site has been implicated as a negative transcriptional control element (Cao *et al*., 1989). The conserved sequences in this region are labeled YY1, YY1 and "silencer" in Fig. 7.1. Gumucio *et al*. (1992) present evidence that the abundant nuclear protein YY1 binds in the region indicated in Fig. 7.1, whereas Peters *et al*. (1993) assign YY1 binding to the region labeled "silencer." Furthermore, on the basis of binding studies with the human sequence, Peters *et al*. (1993) argue that the proximal YY1 site (following the labeling in Fig. 7.1) and the region between it and the "silencer" contain two binding sites for GATA1. The multiple alignment supports the assignments of Gumucio *et al*. (1992) for the location of YY1 sites and shows that the GATA1 sites are not well conserved. However, the region marked "silencer" is conserved, and likely does play a role in negative control of transcription, but it is not clear that the protein YY1 is involved at this site. The multiplicity of conserved sequences and binding sites reflects complexity in the regulation of the ε-globin gene (Hardison *et al*., submitted), and further study combining both the multiple alignment data, protein binding studies and functional analysis should lead to a comprehensive model for its regulation.

Figure 7.1 goes here.

Figure 7.1. Multiple alignment of sequences around the promoters of mammalian ε-globin genes. The portion of the multiple alignment described in the text and extending for about 400 bp 5′ to the cap site is shown. Boxes are drawn around blocks of sequences with very few substitutions. The consensus sequences for the binding sites of relevant nuclear proteins are underlined, as is the cap site itself, and proteins demonstrated to bind these sequences are identified above the boxes. Other notable features, such as a polypyrimidine-polypurine conserved block (YRC), a "silencer" element, and a secondary promoter at position −200 are also labeled.

8. Discussion. We have described multiple alignment algorithms that are efficient in their use of computer time and space. Moreover, these methods permit flexible schemes for penalizing end-gaps and internal gaps.

It should be noted that recently Higgins *et al*. (1992) announced new multiple-alignment software. Their strategy for aligning two alignments can be viewed as a special case of our linear-space algorithm for second-order alignment scores. In particular, their approach amounts to using the following 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).

For our work, additional flexibility is needed, especially for handling gap penalities. We need to align rather long DNA sequences at various evolutionary distances, so sequence-specific pairwise scores are appropriate. Moreover, the aligned sequences are frequently of substantially different lengths, perhaps because only a limited amount of sequence data has been generated for a given species, and in such cases end-gaps need careful handling to produce sensible alignments.

We close by proposing an open problem. Chao *et al*. (1993b) give a pairwise alignment algorithm that computes the smallest region containing all alignments that come within a given amount of the optimal alignment score. Knowledge of this region can be used to locate the most highly conserved portions of the aligning sequences. Presumably, this method can be extended to progressive multiple alignment, and it might be interesting to do so.

LITERATURE

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

Bailey, W. J., J. L. Slightom and M. Goodman. 1992. Rejection of the "flying primate" hypothesis by phylogenetic evidence from the ε-globin gene. *Science* **256**, 86-89.

Cao, S. X., P. D. Gutman, H. P. G. Dave and A. N. Schechter. 1989. Identification of a transcriptional silencer in the 5′-flanking region of the human ε-globin gene. *Proc. Natl. Acad. Sci. USA.* **86**, 5306-5309.

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., R. C. Hardison and W. Miller. 1993a. Constrained sequence alignment. *Bull. Math. Biol.* **55**, 503-524.

Chao, K.-M., R. C. Hardison and W. Miller. 1993b. Locating well-conserved regions within a pairwise alignment. *CABIOS*, in press.

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

Gong, Q.-H., and A. Dean. 1993. Enhancer-dependent transcripion of the ε-globin promoter requires promoter-bound GATA-1 and enhancer-bound AP-1/NF-E2. *Mol. Cell. Biol.* **13**, 911-917.

Gong, Q.-H., J. Stern and A. Dean. 1991. Transcriptional role of a conserved GATA-1 site in the human ε-globin gene promoter. *Mol. Cell. Biol.* **11**, 2558-2566.

Gribskov, M., A. McLachlan and D. Eisenberg. 1987. Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA* **84**, 4355-4358.

Gumucio, D. L., H. Heilstedt-Williamson, T. A. Gray, S. A. Tarle, D. A. Shelton, D. Tagle, J. Slightom, M. Goodman and F. S. Collins. 1992. Phylogenetic footprinting reveals a nuclear protein which binds to silencer sequences in the human γ and ε globin genes. *Mol. Cell. Biol.* **12**, 4919-4929.

Hardison, R. C. 1983. The nucleotide sequence of the rabbit embryonic globin gene b4. *J. Biol. Chem.* **258**, 8739-8744.

Hardison, R. C., and W. Miller. 1993. Use of long sequence alignments to study the evolution and regulation of mammalian globin gene clusters. *Mol. Biol. Evol.* **10**, 73-102.

Hardison, R. C., D. Krane, D. Vandenbergh, J.-F. Cheng, J. Mansberger, J. Taddie, S. Schwartz, X. Huang and W. Miller. 1991. Sequence and comparative analysis of the rabbit α-like globin gene cluster reveals a rapid mode of evolution in a G+C rich region of mammalian genomes. *J. Mol. Biol.* **222**, 233-249.

Hardison, R., K.-M. Chao, M. Adamkiewicz, D. Price, J. Jackson, T. Zeigler and W. Miller. 1993. Positive and negative regulatory elements of the rabbit ε-globin gene revealed by an improved multiple alignment program and functional analysis. Submitted.

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. C. Hardison and W. Miller. 1990. A space-efficient algorithm for local similarities. *CABIOS* **6**, 373-381.

Miller, W. 1993. Building multiple alignments from pairwise alignments. *CABIOS* **9**, 169-176.

Motamed, K., C. Bastiani, Q. Zhang, A. Bailey and C.-K. J. Shen. 1993. CACC box and enhancer response of human embryonic ε globin promoter. *Gene*, in press.

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

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

Yu, C.-Y., K. Motamed, J. Chen, A. D. Bailey and C.-K. J. Shen. 1991. The CACC box upstream of human embryonic ε globin gene binds Sp1 and is a functional promoter element *in vitro* and *in vivo*. *J. Biol. Chem. 266*, 8907-8915.

Nakajima, N., M. Horikoshi and R. Roeder. 1988. Factors involved in specific transcription by mammalian RNA polymerase II: purification, genetic specificity and TATA box-promoter interactions of TFIID. *Mol. Cell. Biol.* **8**, 4028-4040.

Peters, B., N. Merezhinskaya, J. F. X. Diffley and C. T. Noguchi. 1993. Protein-DNA interactions in the ε-globin gene silencer. *J. Biol. Chem.* **268**, 3430-3437.