Linear-Space Algorithms that Build Local Alignments from Fragments[1]

Kun-Mao Chao[2] and Webb Miller[2]

Abstract. This paper presents practical algorithms for building an alignment of two long sequences from a collection of "alignment fragments," such as all occurrences of identical 5-tuples in each of two DNA sequences. We first combine a time-efficient algorithm developed by Galil and coworkers with a space-saving approach of Hirschberg to obtain a local alignment algorithm that uses $O((M+N+F\log N)\log M)$ time and $O(M+N)$ space to align sequences of lengths M and N from a pool of F alignment fragments. Ideas of Huang and Miller are then employed to develop a time- and space-efficient algorithm that computes n best non-intersecting alignments for any $n>1$. An example illustrates the utility of these methods.

Key Words. Sequence comparison, Local alignment, Dynamic programming, Candidate-list paradigm, Linear-space algorithm.

1. Introduction. The unfortunate scarcity of interaction between biologists and computer scientists is well illustrated by the parallel developments of dynamic-programming methods for comparing sequences. Such methods were independently discovered by biologists (Needleman and Wunsch, 1970), computer scientists (Wagner and Fischer, 1974), and workers in other fields. (For a survey of the history, see Sankoff and Kruskal, 1983.) The precise relationship between the methods developed by the two communities is somewhat obscured by notational differences, most but not all of which are insignificant. Computer scientists typically make explicit the relationship between two sequences by producing a list of editing operations that converts one sequence to the other, while biologists prefer to see an alignment of the sequences. Moreover, mathematically-oriented researchers find it natural to compare sequences using a distance "metric", i.e., where a sequence is at distance zero from itself and large values of the measure correspond to highly divergent sequences. On the other hand, biologists tend to think in terms of similarity scores, i.e., a sequence has a very high similarity score when compared with itself, similar sequences have a smaller, but still positive, similarity score, and a pair of unrelated sequences has a negative score. It is tempting to believe that there is some kind of "duality" between minimizing a distance measure and maximizing a similarity score that makes it immaterial which one is adopted. Indeed, this is true under certain circumstances (Smith *et al*., 1981), though not for "local" alignments (see below).

[1]This work was supported in part by grant R01 LM05110 from the National Library of Medicine.
[2]Department of Computer Science, The Pennsylvania State University, University Park, PA 16802, USA.

Biologists need more general measurements of sequence relatedness than are typically considered by computer scientists. The most popular formulation in the computer science literature is the "longest common subsequence problem," which is equivalent to scoring alignments by simply counting the number of exact matches. For comparing protein sequences, it is important to permit the bonus awarded for aligning two symbols to depend on the particular symbol pair (Feng *et al*., 1985). For both DNA and protein sequences, it is standard to penalize a long gap (i.e., deletion from one of the sequences) less than the sum of the penalties for a set of shorter gaps of the same total length (Fitch and Smith, 1983). This is usually accomplished by charging $g + t \times e$ for a gap of length t. Thus the "gap-open penalty" g is assessed for every gap, regardless of length, and an additional "gap-extension penalty" e is charged for every sequence entry in the gap. Such penalties are called *affine* gap penalties. Gotoh (1982) showed how to efficiently compute optimal alignments under such scoring rules.

Even more general models for quantifying sequence relatedness have been proposed. For example, it is sometimes useful to have the penalty for adding a symbol to a gap depend on the position of the gap within the sequence (Gribskov *et al*., 1990), which is motivated by the observation that insertions in certain regions of a protein sequence can be much more likely than at other regions. Another generalization is to let the incremental gap cost $\delta_i = c_{i+1} - c_i$, where a k-symbol gap costs c_k, be a monotone function of i, e.g., $\delta_1 \geq \delta_2 \geq \cdots$ (Waterman, 1984; Miller and Myers, 1988; Galil and Giancarlo, 1989). In this paper we model only affine gap penalties since we are concerned mainly with DNA sequences (for which position-dependent penalties are not so useful as for proteins) and since concave or convex gap penalties have yet to be found truly useful. Indeed, there is some evidence that monotonic gap-extension penalties incorrectly model nature in certain circumstances (Pascarella and Argos, 1992).

Besides needing flexible ways of scoring alignments, biologists typically want to compute kinds of alignments that are not equivalent to the models studied by computer scientists. Computer scientists generally consider a problem equivalent to *global* alignment, i.e., computing an optimal alignment that is required to extend from the starts of the given sequences to their ends. Biologists frequently find it more useful to seek an alignment that is highest-scoring among all alignments between an arbitrary section of the first sequence and an arbitrary section of the second sequence (Smith and Waterman, 1981), which is called the *local* alignment problem. Probably the most useful of these variations for aligning biological sequences is that of computing "n best non-intersecting" local alignments. Care must be taken to formalize this notion in a way that allows subtle matches lying near much stronger (but perhaps less interesting) matches to be found, while still permitting efficient computation (Waterman and Eggert, 1987; Waterman, 1989). Earlier attempts to define the "n best local alignments problem" in terms of minimizing a measure of the distance between two sequences led to substantially more cumbersome algorithms (Goad and Kanehisa, 1982; Sellers, 1984).

On the computer science side, Hirschberg (1975) discovered a method for computing longest common subsequences using only linear space (space proportional to the sum of the sequence lengths) rather than the naive quadratic space (space proportional to the product of the sequence lengths). Although space, rather than time, is often the constraining factor when applying dynamic-programming techniques to biological sequences (e.g., with a DNA sequence of length 50,000, a quadratic-space method uses billions of computer memory locations), biologists didn't discover the technique for themselves. While Hirschberg's original formulation was for alignment scores that are unrealistically simple for applications in biology, Myers and Miller (1988) (also Miller and Myers, 1988) extended the approach to affine gap costs. Moreover, linear-space methods have been developed for the "*n* best local alignment problem" (Huang *et al*., 1990; Huang and Miller, 1991).

To attain greater speed, biologists have employed the strategy of building alignments from alignment fragments (Wilbur and Lipman, 1983, 1984). For example, one could specify some fragment length $k \geq 1$ and work with fragments consisting of a segment of length at least k that occurs in both sequences. With protein sequences, it might well work better to begin with inexact but high-scoring matches, such as those used by the *blast* program (Altschul *et al*., 1990) for other purposes. In any case, algorithms that optimize the score over alignments constructed from fragments can run faster that algorithms than optimize over all possible alignments.

Alignments constructed from fragments (or often just the alignments' scores) have been very successful in initial filtering criteria within programs that search a sequence database for matches to a query sequence; database sequences whose alignment score with the query sequence falls below a threshold are ignored, and the remainder are subjected to a slower but higher-resolution alignment process. Moreover, the high-resolution 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*., 1992).

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). Eppstein *et al*. (1992a) develop such an alignment 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. It should be noted that the basic approach of Eppstein *et al*. was discovered independently by Myers and Huang (1992) in solving a different problem; they present an alternative and very useful graphical description of how the method works.

By adapting a number of existing ideas and proposing a few new ones, this paper develops a method for constructing n best non-intersecting local alignments from given alignment fragments. Moreover, the algorithm runs in linear space and utilizes alignment scoring schemes that are appropriate for biological problems. We begin with a review of the algorithm of Eppstein *et al*., reformulated to compute the score of the best *local* alignment with affine gap costs. The next step, which is again straightforward, is to utilize Hirschberg's approach to find an optimal alignment (not merely its score) using only linear space. The final step, and the one involving a serious extension of what was already known, is to compute n best local alignments from fragments, following the general outline of Huang and Miller (1991).

The strength of this paper lies in the utility of the algorithm developed. The algorithm is central to our plans for dealing with DNA sequences of lengths between 10^5 and

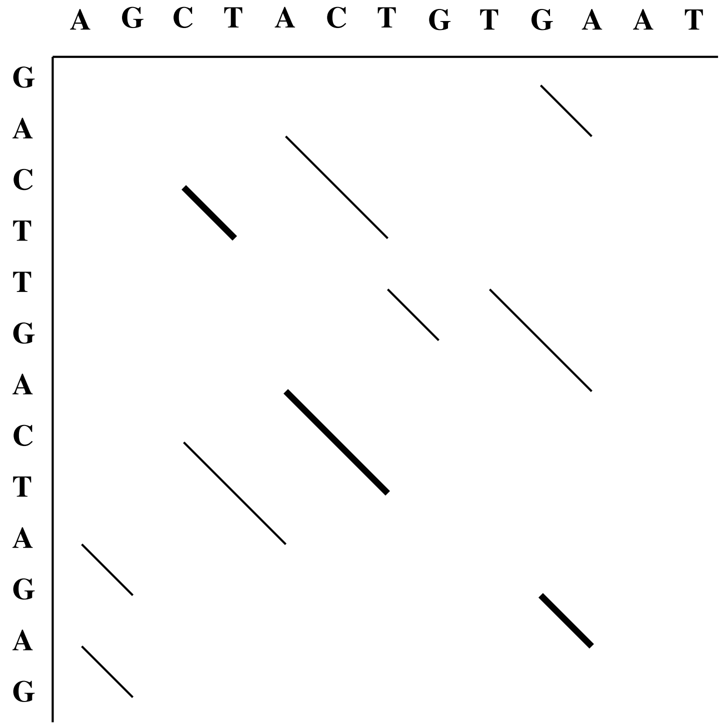
10^6, an area that will soon be important. Whereas alignments built from, say, exact matches of length 5, are not sufficient, in themselves, for our studies of gene regulation and molecular evolution (Hardison and Miller, 1993), this algorithm provides a sensitive filtering procedure for locating regions whose similarity can then be measured by a more accurate technique, like that of Chao *et al*. (1993). Moreover, its rather considerable algorithmic complexity is appropriate; it is expected to substantially outperform simpler methods.

A useful attribute of the algorithm given here is that its position on the sensitivity-vs-speed spectrum is tuned by the choice of initial fragments. For instance, in an example at the end of the paper, using 6-words (exact matches of length at least 6) permits adequate detection of fairly subtle similarities with a 16-fold speedup compared to full-resolution alignment. Near the other extreme, Chao *et al*. (1993) use 8-words to attain a 1000-fold speedup when finding an alignment of length over 100,000 (between chloroplast genomes of tobacco and liverwort).

Unfortunately, several desirable topics must be omitted from this report. A thorough evaluation of the method's effectiveness for its intended uses would require specification of the accompanying high-resolution alignment technique and a careful discussion of biological properties of sequences that are only now starting to become available; hence it lies beyond the scope of this paper. Also, only an incomplete theoretical analysis is offered since no one has as yet shown how to give a satisfying performance analysis of even the simplest realistic n-best alignment method (Waterman and Eggert, 1987).

2. The basic definitions. Let the given sequences be $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$. The discussion is simplified by specifying a set of alignment fragments and their scores, though it is straightforward to apply the algorithms described below to other classes of alignment fragments. With this in mind, define a *fragment* to be a triple (i, j, k) such that $a_i = b_j$, $a_{i+1} = b_{j+1}$, $\cdots$, $a_{i+k-1} = b_{j+k-1}$ and $k \geq k\_min$, for fixed minimum fragment length k_min. Moreover, a fragment is to be maximal, i.e., not properly contained in another fragment. Fragment $f' = (i', j', k')$ is said to be *above* fragment $f = (i, j, k)$ if $i' + k' \leq i$ and $j' + k' \leq j$, and f is then *below* f'. Notice that this defines a partial-ordering relation. An alignment is defined as a sequence of fragments $f_1, f_2, \cdots, f_l$ such that f_i is above f_{i+1} for $i = 1, \cdots, l-1$. (See Figure 1.) Henceforth, the terms "alignment" and "path" are used interchangeably. It should be noted that these definitions originate from Eppstein *et al*. (1992a).

(A)



(B)

C T T G A C T – A G A
C T – – A C T G T G A

Figure 1. (A) Graphical representation of the alignment fragments consisting of exact matches of length 2 or more between *GACTTGACTAGAG* and *AGCTACTGTGAAT*. (B) Traditional representation of the local alignment generated from the three dark diagonal line segments of Part A.

The score of fragment $f = (i, j, k)$ is defined as k and denoted $sc(f)$. Penalties for connecting fragments are needed for scoring alignments. Let the nonnegative constant g and positive constants e and r be the penalties for opening up a gap (horizontal or vertical displacement), for extending a gap by one symbol, and for replacing one symbol by another, respectively. The affine function $gap(t) = g + te$ is charged for a gap of length t. Define the *diagonal* of fragment $f = (i, j, k)$ to be $j - i$, denoted by $Diag(f)$. Suppose $f' = (i', j', k')$ is above $f = (i, j, k)$. The penalty of connecting f' and f, denoted by *Connect*(f', f), is defined as follows (see Figure 2):

Case 1, $Diag(f) = Diag(f')$:
$$Connect(f', f) = (i - i' - k')r.$$
Case 2, $Diag(f) > Diag(f')$:
$$Connect(f', f) = gap(Diag(f) - Diag(f')) + (i - i' - k')r.$$
Case 3, $Diag(f) < Diag(f')$:
$$Connect(f', f) = gap(Diag(f') - Diag(f)) + (j - j' - k')r.$$

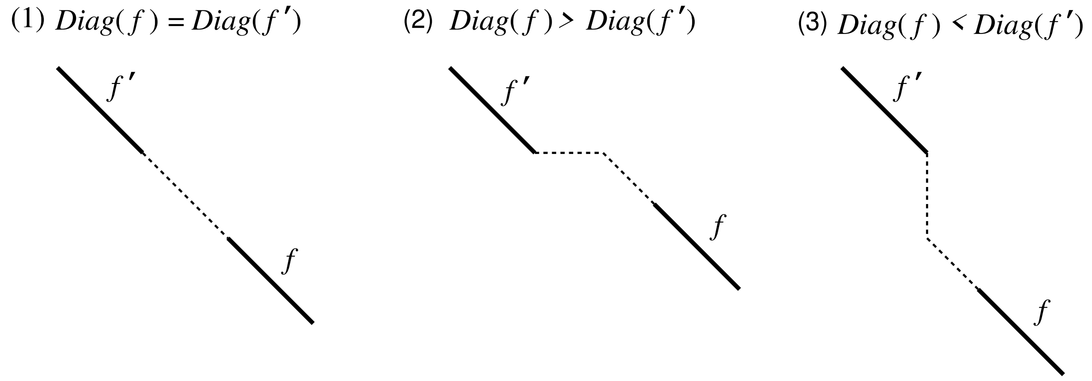


Figure 2. Connecting two fragments.

We assume that $r \le 2e$, guaranteeing that the penalty for connecting f' and f is the minimum among all possible ways of connection. The *score* of an alignment $(f_1, f_2, \cdots, f_l)$ is defined as the sum of the fragment scores, minus the connection penalties of adjacent fragments, i.e.:

$$\sum_{i=1}^{l} sc(f_i) - \sum_{i=1}^{l-1} Connect(f_i, f_{i+1})$$

For example, if we take $g = e = r = 1$, then the score of the alignment in Figure 1B is 7 − 6 = 1. Notice that all symbol replacements are considered equal; there are no provisions of utilizing identical or similar sequence elements between fragments. It is this idealization that permits efficient computation of maximum-score alignments.

The local alignment problem is thus to find an alignment with the highest score. For a global alignment problem, one needs to assess additional penalties for connecting the alignment to $(0, 0)$ and $(M+1, N+1)$.

Let $Score(f)$ be the maximum score over all alignments ending at f, so that $\max_f \{Score(f)\}$ gives the score of the best local alignment. Because alignments must use whole fragments, the principle of optimality yields the recurrence relation:

$$Score(f) = \max\{0, \max_{f' \text{ above } f} \{Score(f') - Connect(f', f)\}\} + sc(f)$$

A straightforward dynamic-programming method, essentially just an optimal-path algorithm for directed acyclic graphs, solves the recurrence in $O(F^2)$ time, where F is the number of fragments (Wilbur and Lipman, 1984).

3. The algorithm of Eppstein et al. The algorithm of Eppstein *et al*. (1992a) is based on the "candidate-list paradigm" (Miller and Myers, 1988; Galil and Park, 1992), i.e., we keep lists of all fragments f' that are candidates for maximizing $Score(f') - Connect(f', f)$ for some fragment f at which $Score(f)$ will later be evaluated. The computation proceeds by rows. When f's starting position is reached, the fragment above it that determines $Score(f)$ can be found by searching through some candidate lists. Once f's end position is reached, f is added to any candidate lists that should contain it.

To facilitate the process, the region below a fragment is divided into three disjoint subregions, as depicted in Figure 3. Suppose $f' = (i', j', k')$ is above $f = (i, j, k)$. Then f is said to be under the *right influence* of f' if $Diag(f') < Diag(f)$, while f is under the *left influence* of f' if $Diag(f') > Diag(f)$. In case $Diag(f') = Diag(f)$, f is said to be under the *diagonal influence* of f'. The *right-influence region* (*left-influence region*, *diagonal-influence region*) of f' is defined to be the range consisting of all points (x, y) such that $i' + k' \le x \le M$, $j' + k' \le y \le N$, and $y - x > Diag(f')$ ($y - x < Diag(f')$, $y - x = Diag(f')$, respectively). Let

$$RI(f) = \max\,\{Score(f') - Connect(f', f) \text{ such that } f \text{ is under the right influence of } f'\}$$
$$LI(f) = \max\,\{Score(f') - Connect(f', f) \text{ such that } f \text{ is under the left influence of } f'\}$$
$$DI(f) = \max\,\{Score(f') - Connect(f', f) \text{ such that } Diag(f) = Diag(f')\}$$

Then $Score(f) = \max\{0, RI(f), LI(f), DI(f)\} + sc(f)$.

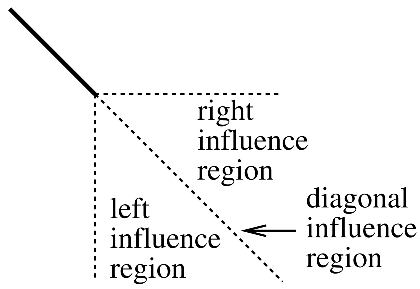


Figure 3. Influence regions.

$DI(f)$ can be determined by considering only the nearest previous fragment on the same diagonal. $RI(f)$ requires more effort, but is similar to, and somewhat simpler than, $LI(f)$. Thus, we discuss only computation of $LI(f)$; the interested reader can refer to Eppstein *et al*. (1992a) for the algorithm's complete description. Define $Decay(f : x, y)$ as $Score(f) - Connect(f, h)$ where h is an imaginary fragment starting at (x, y).

Lemma 1. If $Decay(f : x, y) \ge Decay(f' : x, y)$ for some (x, y) in the common left-influence region of fragments f and f', then the inequality holds for *all* entries in that region.

Proof. It suffices to show that $Decay(f : x, y) - Decay(f' : x, y)$ is independent of x and y. If $f = (i, j, k)$, then

$$Decay(f : x, y) = Score(f) - gap(Diag(f) + x - y) - (y - j - k)r$$

where r is the symbol-replacement penalty. An analogous equation holds for $f' = (i', j', k')$, so

$$\begin{aligned} &Decay(f : x, y) - Decay(f' : x, y) \\ &\quad = Score(f) - Score(f') + (Diag(f') - Diag(f))e + (j + k - j' - k')r \end{aligned}$$

where e is the gap-extension penalty. $\square$

Lemma 1 says that if f is superior to f' for some point in their common left-influence region (Figure 4), it is superior for all points in that region.

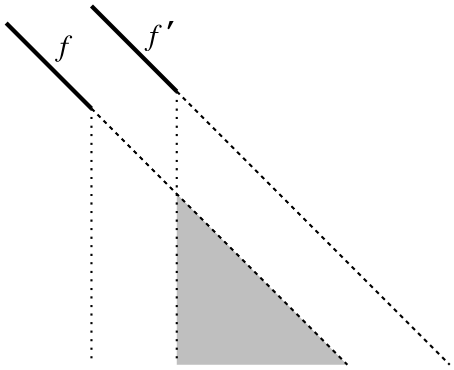


Figure 4. Common left-influence region of f and f'.

A fragment f' is said to *left-dominate* the region if for any fragment f starting in that region $LI(f) = Score(f') - Connect(f', f)$. Figure 5 depicts the regions that are left-dominated by each fragment. (Ties can be broken arbitrarily.) Each row is divided into intervals by the vertical and diagonal lines that separate regions, and the same fragment can left-dominate more than one of these intervals (e.g. fragment f_1 in Figure 6.2). Two candidate lists are used for computing $LI(f)$. One sorted list, denoted by LC, gives the columns where vertical region boundaries intersect the current row. Given a point p in the current row, LC is searched for the largest entry strictly less than p, to which is attached a pointer to the ''active'' fragment that left-dominates the interval immediately to the entry's right. The other sorted list, denoted by LD, gives the diagonals where region boundaries intersect the current row. Given p, LD is searched for the largest entry not exceeding p's diagonal, to which is attached a pointer to the active fragment that left-dominates the interval beginning at that diagonal. If p is the initial point of fragment f, then $LI(f)$ is easily calculated since one of the two fragments found by the searches determines $LI(f)$.

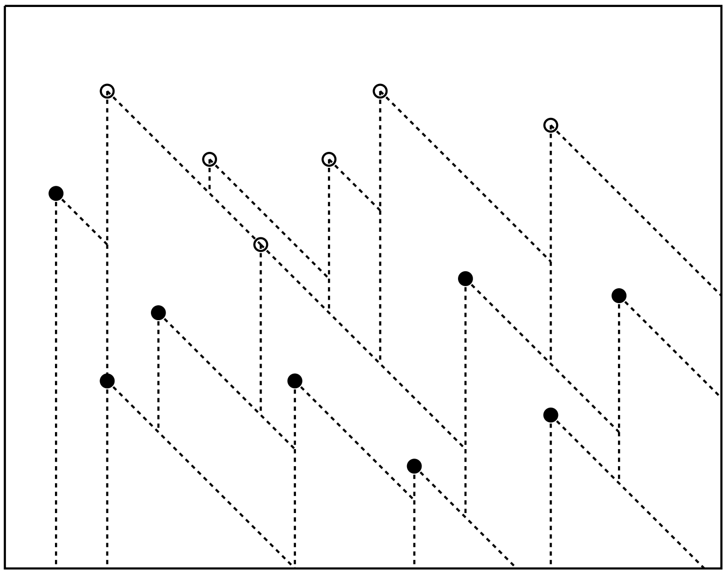


Figure 5. Regions left-dominated by some fragments. Points indicate the lower end of a fragment, and dark points specify those fragments that are still ''active'' at the last row.

Figure 6 illustrates the process of searching *LC* and *LD* to find the active fragment that left-dominates the interval containing p. Searching *LC* finds c and returns f. In Figure 6.1, p is in the interval left-dominated by f. In Figure 6.2, p is not under the left influence of f, but searching *LD* finds d and returns f_3 which left-dominates d. (Notice that d is not in the left-influence region of f_2.) Incidentally, Figure 6.2 indicates why it will not work to search *LD* for the first entry *larger* than p's diagonal.

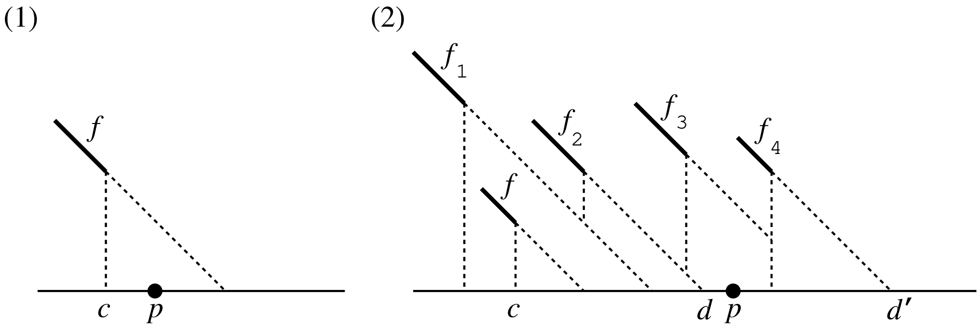


Figure 6. Locating the fragment that left-dominates the interval containing p.

Once $LI(f)$ has been computed for all f's starting at the current row, *LC* and *LD* are updated for the next row as follows. For each fragment f ending at the current row, first determine if it left-dominates some region. In general (except when f ends on some column boundary in *LC* or some diagonal boundary in *LD*), this is done by searching *LC* and *LD* to find the fragment, say f', that left-dominates the end-point of f, and then computing $Decay(f : x, y) - Decay(f' : x, y)$ for any (x, y) in their common left-influence region. If f is superior to f', then modify *LC* and *LD* to reflect the new region left-dominated by f. Several cases arise when f is added to LC and LD. Here we take one case as an example. Readers can refer to Eppstein *et al.* for more details.

Suppose that the closest boundary to the left of f's end-point is a diagonal and the closest boundary to the right is a column (see Figure 7). We keep intersection lists, denoted by $CUT(i)$, giving the columns where two boundary lines intersect in row i. In the case at hand, the borders on either side of f's end-point intersect at a point a in the CUT list for some later row (Figure 7), and a must be removed from that list. Two new intersection points, b and c, must be added to the appropriate lists (if they fall within the grid). Moreover, f now left-dominates the region R_2 and f' left-dominates the regions R_1 and R_3. Thus, we add one column left-dominated by f to LC and one diagonal left-dominated by f' to LD.

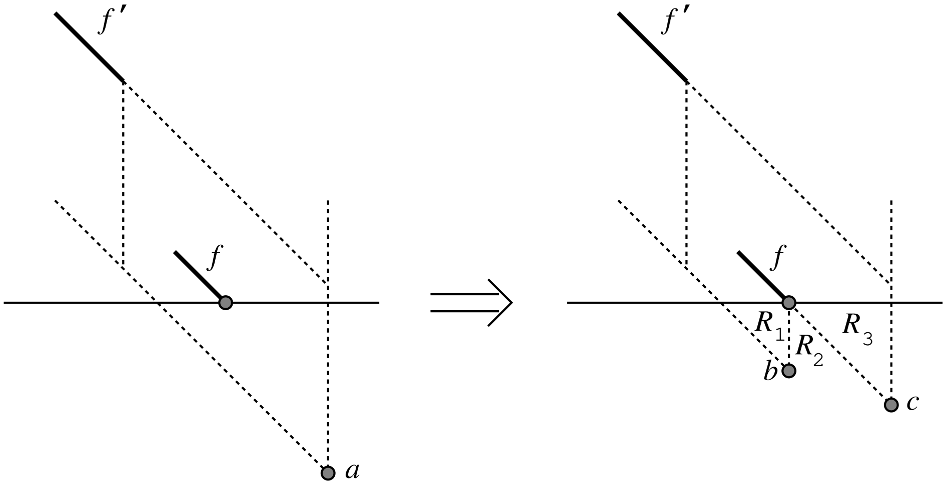


Figure 7. Case when the end-point of f lies between a diagonal and a column boundary.

Before leaving row i, points saved in $CUT(i)$ need to be treated. For each intersection point, we extend either the column or diagonal, in the following sense. At a given intersection point, three regions come together (see Figure 8). Let f_1, f_2 and f_3 be the left-dominating fragments for regions R_1, R_2 and R_3, respectively. If $Decay(f_1\colon x, y) \geq Decay(f_3\colon x, y)$ for some (x, y) in their common left-influence region, then we (1) remove from LC the column where f_3 ends, (2) change the pointer for diagonal $Diag(f_1)$ from f_2 to f_3 and (3) if the interval just right of the intersection point is terminated by a column boundary, we add an entry to the CUT list for the row where diagonal $Diag(f_1)$ crosses that column. Otherwise, we (1) remove diagonal $Diag(f_1)$ from LD and (2) if the interval just left of the intersection point begins at a diagonal boundary, we add an entry to the appropriate CUT list.

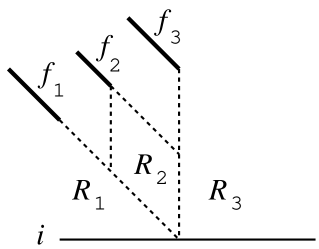


Figure 8. An intersection point.

The following pseudo-code gives the outline for computing $LI(f)$ and updating LC, LD and the CUT lists.

for $i \leftarrow 1$ **to** M **do**
{ **for** each fragment f starting at row i **do**
{ $f' \leftarrow SEARCH_L(LC, LD, f)$
$LI(f) \leftarrow Score(f') - Connect(f', f)$
}
for each fragment f ending at row i **do**
$UPDATE_L(LC, LD, CUT, f)$
Handle points in $CUT(i)$.
}

Consider now the running time of the complete alignment algorithm. The diagonal influence $DI(f)$ can be computed in $O(1)$ time since it involves merely finding the nearest fragment above f and on f's diagonal. $RI(f)$ can be computed in a simpler way than $LI(f)$ because the right-influence regions are bounded by rows (instead of columns) and diagonals. Therefore, the intersection lists are no longer needed. Furthermore, only one list, sorted by diagonals, needs to be maintained. Suppose that the candidate lists (LC, LD and the list for the right influence) are implemented as balanced search trees, so that each search or update operation takes $O(\log t)$ time, where the size of the tree is $t \leq N$. The fragments can be generated in $O(M+N+F)$ time using a suffix tree, so the total time for the above algorithm, ignoring the updating of CUT lists, is $O(M+N+F \log N)$. Note that the total number of CUT points handled is most $2F$, since a fragment determines at most two boundary lines (Figure 5) and a boundary line dies at each intersection point. Each CUT point can be handled in $O(\log N)$ time, so the total time for a balanced-tree implementation is $O(M+N+F \log N)$.

4. A linear-space algorithm for local alignment. When long DNA sequences are aligned, it may be impractical to store all of the fragments. For example, there are 3,504,057 maximal fragments of length at least 5 between a 73,326-symbol DNA sequence containing the human β-like globin gene cluster and a corresponding 44,594-symbol sequence for rabbit.

To compute the optimal score of a local alignment, it is sufficient to keep only those fragments in current candidate lists (e.g., LC and LD). Fragments that start in row i are generated when the algorithm reaches row i. Also, for each fragment f, the number of candidate lists containing f is maintained, and f is deleted when the number reaches zero. Since the size of a candidate list is $O(M + N)$, the total space requirement for this score-only method is $O(M + N)$.

Explicitly producing a best local alignment in linear space is more difficult. Our approach works in two phases. In outline, we locate first and last fragments of a best local alignment, then use a linear-space global alignment algorithm to compute an optimal global alignment for the sequence segments bounded by these two fragments. In order to locate the first and last fragments, we compute for each f the first fragment on a path to f of score $Score(f)$. When $Score(f)$ is determined, this first fragment is either f itself (if $Score(f) = sc(f)$) or equal to the first fragment for the f' determining $Score(f)$. A somewhat more efficient strategy for locating the first and last fragments can be found in Huang *et al*. (1990).

Locating the first and last fragments of a best local alignment reduces the problem to generating a global alignment in the region bounded by these two fragments. Specifically, if the first fragment is $f' = (i', j', k')$ and the last fragment is $f = (i, j, k)$, then the local alignment we seek consists of f', followed by a global alignment of $a_{i'+k'}a_{i'+k'+1} \cdots a_{i-1}$ and $b_{j'+k'}b_{j'+k'+1} \cdots b_{j-1}$ (discarding fragments that expand to larger fragments in the complete sequences), followed by f. To solve the global problem, we use the strategy devised by Hirschberg (1975). Begin by applying a global, cost-only variant of the algorithm of Eppstein *et al*. It differs from the local alignment algorithm described above in that the 0 term in the equation

$$Score(f) = \max\{0, RI(f), LI(f), DI(f)\} + sc(f)$$

is replaced by a penalty for reaching f from the starts of the sequence segments defining the global problem. Let $Score^{-}(f)$ denote this global ''backwards'' score at f. The process is stopped after processing the middle row, i.e., row $m = \lfloor (i' + k' + i - 1)/2 \rfloor$ where i' etc. are as above, and the current candidate lists are saved. Then an ''inverted'' version of the algorithm is applied to compute $Score^{+}(f)$ defined as the maximum score of an alignment from f to the ends of the segments defining the global problem. This backward pass is stopped after processing row $m + 1$.

The goal is now 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 (see Figure 9). As will now be shown, each possibility can be checked in $O(c)$ time where there are $c = j - j' - k'$ columns in the subproblem, so we need only check them all and pick the one yielding the best alignment.

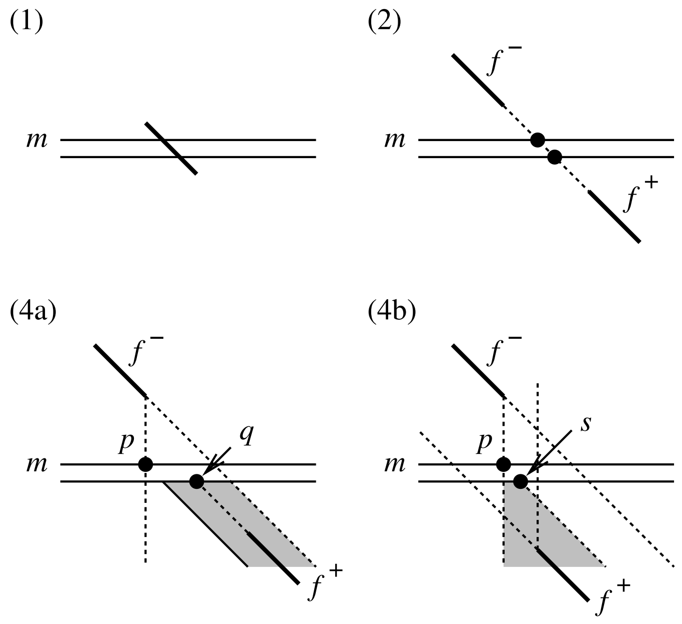


Figure 9. Cases that arise when dividing a global alignment problem.

The first case is that an optimal alignment uses a fragment that includes rows m and $m+1$ (Figure 9.1). For each such crossing fragment f, $Score^{-}(f)$ + $Score^{+}(f)$ − $sc(f)$ gives the best score of all (global for the subproblem) alignments using f, and all of these values can be inspected in $O(c)$ time.

For all the remaining cases, fix an optimal alignment, let f^{-} be the last fragment on the alignment that lies on or above row m, and let f^{+} be the first fragment on or after row $m+1$. (Actually, f^{-} and f^{+} could be "pseudo-fragments" corresponding to the upper left or lower right corners of the subproblem, which are added to create a *global* alignment problem.) Our second case is where f^{-} and f^{+} lie on the same diagonal. In $O(c)$ time we can loop over all fragments associated with lists for *DI* in the upper and lower problems, in order of increasing diagonal, and determine all potential pairs (f^{-}, f^{+}). The optimal score over all paths that jump from f^{-} to f^{+} is $Score^{-}(f^{-})$ + $Score^{+}(f^{+})$ − *Connect*(f^{-}, f^{+}).

Case 3 is where $Diag(f^{+}) > Diag(f^{-})$. As before, our decision to compute by rows makes handling right influences simpler than left influences, so we omit explicit treatment of Case 3.

When $Diag(f^{+}) < Diag(f^{-})$, it is impossible for both f^{-} to intersect row m and f^{+} to intersect row $m+1$. (This is because f^{+} has to start strictly after the column where f^{-} ends.) Without loss of generality, suppose that f^{-} does not intersect row m. Project the lower end of f^{-} vertically onto row m, obtaining point p of Figures 9.4a and 9.4b. Thus if $f^{-} = (i^{-}, j^{-}, k^{-})$, then $p = (m, j^{-} + k^{-} - 1)$. There are two subcases.

The first subcase is when the diagonal containing p does not exceed $Diag(f^{+})$. For each fragment f^{+} associated with the list for *DI* of the lower problem, let f^{+}'s diagonal intersect the row $m+1$ at point q. Recall that the candidate lists for the upper problem were updated to be ready for row $m+1$ when treating row m. Point q is under the left influence of f^-, so f^- can be determined from q in $O(\log c)$ time. The best score for a path jumping from f^- to f^+ is again $Score^-(f^-) + Score^+(f^+) - Connect(f^-, f^+)$. Better yet, we can arrange that the possible fragments f^+ and the left-influence intervals in row $m+1$ can be enumerated in order of increasing diagonal in $O(c)$ time, which gives a linear-time treatment of Case 4a.

The other subcase is when the diagonal containing p exceeds $Diag(f^+)$. Let point s be the intersection of row $m+1$ and the diagonal containing p. Then s is under the left influence of f^- and p is under the backward left influence of f^+. Therefore, given p, we could determine s, f^-, f^+, and the highest score of a global alignment that jumps from f^- to f^+, all in $O(\log c)$ time. Actually, we can arrange that the left-influence intervals in row $m+1$ and the backward left-influence intervals in row m can be enumerated from left to right in $O(c)$ time, which covers Case 4b in linear time.

The fragments f^- and f^+ (or f in Case 1) reduce the problem to two subproblems (Figure 10), which are solved recursively. Problems with fewer than k_min rows or columns, where k_min is the minimum fragment length, are handled by doing nothing. Clearly an optimal alignment is computed by this method using only $O(M+N)$ space.

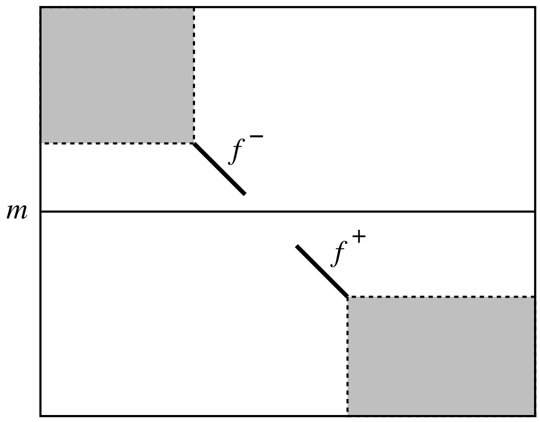


Figure 10. The two subproblems.

Let $T(M, N, F)$ denote the worst-case time to apply the global "divide-and-conquer" alignment algorithm to sequences of lengths M and N with F fragments. Our next goal is to show that

$$T(M, N, F) = O((M + N + F \log N) \log M)$$

It then follows immediately that the same time bound holds for the local alignment algorithm (which performs an additional $O(M + N + F \log N)$-time pass to find the first and last fragment).

First, consider subproblem sizes when a problem is divided (Figure 10). Both subproblems have fewer than $M/2$ rows. If the upper subproblem has N^- columns and F^- fragments and the lower subproblem has N^+ columns and F^+ fragments, then $N^- + N^+ <$ N and $F^- + F^+ < F$. Previous considerations show that the time to split a problem with size-parameters M, N and F into those subproblems (not counting the time to solve the subproblems) is bounded by $\tau(M + N + F \log N)$ for some constant τ, where we interpret $\log x$ to mean $\max\{\log_2 x, 1\}$. Then $T(M, N, F) \le \tau(M + N + F \log N) \log M$. To see this, first note that it holds if the problem is such that no recursive calls are made. For other problems, by induction:

$$\begin{aligned}
&T(M, N, F) \\
&\quad \le \tau(M + N + F \log N) + T(M/2, N^-, F^-) + T(M/2, N^+, F^+) \\
&\quad \le \tau(M + N + F \log N) + \tau(M/2 + N^- + F^- \log N^-) \log M/2 \\
&\qquad + \tau(M/2 + N^+ + F^+ \log N^+) \log M/2 \\
&\quad \le \tau(M + N + F \log N) + \tau(M + N + F \log N) \log M/2 \\
&\quad = \tau(M + N + F \log N) \log M
\end{aligned}$$

5. The n best local alignments. A pair of long and related sequences will often exhibit a number of important local similarities. To discover them, it is inadequate to determine a highest-scoring alignment, a second-highest, a third-highest, and so on, since trivial perturbations of the highest-scoring alignment will often dominate the list. The following strategy yields far more useful results. First, compute a highest-scoring alignment. Remove all fragments in that alignment and find a highest-scoring alignment from the remaining fragments. Remove all fragments in that second alignment and find a highest-scoring alignment from the remaining fragments. And so on until n alignments have been reported. We refer to this process as computing n best non-intersecting local alignments. Formally speaking, two local alignments of sequences A and B *intersect* if they share a fragment. A list $\alpha_1, \alpha_2, \cdots, \alpha_n$ of alignments of A and B is referred to as *n best local alignments* of A and B if α_1 is a highest-scoring local alignment and if $2 \le i \le n$, then α_i is highest-scoring among all local alignments that do not intersect $\alpha_1, \alpha_2, \cdots, \alpha_{i-1}$. It should be noted that different tie-breaking rules may result in n best local alignments with a different sequence of scores. (see Huang and Miller (1991) for an analogous example.)

A straightforward implementation of computing n best non-intersecting local alignments, which starts anew with each reduced set of fragments, is unnecessarily inefficient. Typically, most or all of the computed alignments will be far shorter than the underlying sequences, and discarding an alignment's fragments affects $Score(f)$ only for fragments f lying near the alignment. The idea, then, is to develop an incremental approach that repeats only those parts of the computation where results may change. For traditional sequence alignment via dynamic programming, Waterman and Eggert (1987) developed a quadratic-space algorithm, and a linear-space algorithm was given by Huang and Miller (1991).

We next present a time-efficient, linear-space algorithm for constructing n best non-intersecting alignments from fragments, following the strategy of Huang and Miller (1991). It is assumed that n is known *a priori*. In outline, the algorithm works as follows. A forward pass is made through the entire set of fragments to find the first and last fragments on an optimal alignment. This pass differs from the earlier procedure in that 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 n 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 (Lemma 2, below).

5.1. Equivalence classes. In general, we use G to denote a set of fragments. Specifically, G_1 is the original set of maximal fragments between sequences A and B, and G_m for $m > 1$ is obtained from G_{m-1} by removing the fragments of a highest-scoring local alignment. Let $Score_m(f)$ be the maximum score over all alignments from G_m ending at f, and let $<_G$ denote any topological order on the fragments in G_1 (relative to the "above" relation). $First_m(f)$ is defined to be the last fragment in this ordering such that there is an alignment of score $Score_m(f)$ from that fragment to f using only fragments in G_m (i.e., the topological order is used to break ties).

Lemma 2. Fix $m \geq 1$ and let u be the fragment such that G_{m+1} is formed from G_m by removing the fragments of an optimal alignment from $First_m(u)$ to u. If v is a fragment with $First_m(v) \neq First_m(u)$, then $Score_{m+1}(v) = Score_m(v)$ and $First_{m+1}(v) = First_m(v)$.

Proof. The critical observation is that an optimal path (alignment) from $First_m(v)$ to v cannot share a fragment with an optimal path from $First_m(u)$ to u. To see this, suppose that f occurred on both paths. Without loss of generality, $First_m(v)$ follows $First_m(u)$ in the chosen topological order, and it follows readily that there exists an optimal path from $First_m(v)$ to u, a contradiction. (For more details, see the proof of Lemma 1 of Huang and Miller, 1991). $\Box$

Define a relation E_m over the fragments in G_m by: $u\ E_m\ v$ if and only if $First_m(u) = First_m(v)$. E_m is an equivalence relation, and hence partitions the fragments in G_m into equivalence classes. For each equivalence class C of E_m, define $Score_m(C) = \max\{Score_m(f): f \in C\}$.

Let W be the $n - m + 1^{th}$ highest equivalence-class score in G_m. (As alignments are reported and the equivalence classes are refined, W will in general increase.) The *effective region* of f is chosen so that if f' starts outside of the effective region then $Score(f) - Connect(f, f') \leq W$. The following lemma explains how to determine the effective region.

Lemma 3. Suppose $Score_m(f) > W$ and define

$$h = \lceil \max\{ (Score_m(f) - W - g)/e), (Score_m(f) - W)/r \} \rceil$$

where g, e and r are the penalties for connecting fragments. If f' is a fragment lying below f and starting more than h rows or more than h columns after the end of f, then $Score_m(f) - Connect(f, f') \leq W$.

Proof. First suppose that $Diag(f') = Diag(f)$. If more than h rows separate f and f', then $Connect(f, f') \geq hr \geq Score(f) - W$, i.e., $Score(f) - Connect(f, f') \leq W$. Otherwise, suppose without loss of generality that f' is under the left influence of f. Let $f = (i, j, k)$ and $f' = (i', j', k')$. $Connect(f, f') = gap(Diag(f) - Diag(f')) + (j' - j - k)r = g + (j - i - j' + i')e + (j' - j - k)r = g + h_1 e + h_2 r$, where $h_1 + h_2 = i' - i - k \geq h$. If $r \geq e$, then $Connect(f, f') \geq g + he \geq g + Score(f) - W - g = Score(f) - W$. Otherwise, $Connect(f, f') \geq g + hr \geq hr \geq Score(f) - W$. $\Box$

Let $Box(T, L, B, R)$ denote the rectangle whose upper left corner is (T, L) and lower right corner is (B, R), and let $f = (i, j, k)$. If $Score(f) > W$, then the effective region of f is the rectangle $Box(i + k - 1,\ j + k - 1,\ \min\{M, i + k - 1 + h\},\ \min\{N, j + k - 1 + h\})$

where h is defined in Lemma 3. In general, these regions are square (see Figure 11), except when truncated at an edge of the dynamic-programming grid. If $Score(f) \le W$, then f's effective region is empty.

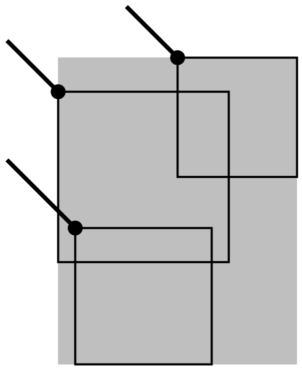


Figure 11. A rectangle containing all effective regions of an equivalence class.

Each of the retained equivalence classes C is represented by a 7-tuple:

$< S, F, u, T, L, B, R >$ where
$S = Score_m(C)$,
$F = First_m(f)$ for all $f \in C$,
$First_m(u) = F$ and $Score_m(u) = Score_m(C)$, and
$Box(T, L, B, R)$ contains all effective regions of fragments in C.

Henceforth, we use *tuple* to designate such a 7-tuple, and refer to the entries of tuple C by $C.S, C.F, \cdots, C.R$.

5.2. Algorithm outline. We are now ready to discuss the algorithm outline of Huang and Miller (1991) in more detail.

Algorithm outline
1. Compute n best tuples $<S, F, u, T, L, B, R>$ in G_1 in a single sweep
   for $m \leftarrow 1$ **to** n **do**
2. { $C \leftarrow$ a maximum-score tuple in G_m
3. Construct an optimal alignment from $C.F$ to $C.u$
   if $m \ne n$ **then**
4. { Determine $T \le C.T$ and $L \le C.L$ so that no alignment from G_{m+1} starting outside $Box(T, L, C.B, C.R)$ and ending inside $Box(C.T, C.L, C.B, C.R)$ has score greater than W
5. Obtain $n-m$ best tuples in G_{m+1} by recomputing $Box(T, L, C.B, C.R)$
   }
   }

Once a best class C from G_m has been located and its optimal alignment reported, we need to discover any high-scoring alignments that were hidden by that alignment. To do so, perform a backward computation to locate row T and column L such that it is sufficient to recompute the region $Box(T, L, C.B, C.R)$. (Any alignment starting outside

$Box(T, L, C.B, C.R)$ and ending in $Box(C.T, C.L, C.B, C.R)$ has score at most W, and hence can be ignored.) That is, a forward pass will then be performed inside $Box(T, L, C.B, C.R)$ to look for fragments where $Score_{m+1}(f)$ exceeds W, in which case the set of retained equivalence classes is altered. It should be noted that potentially more efficient methods exist for delimiting a recomputation region that is not necessarily rectangular, but keeping to rectangular regions simplifies the discussion.

For step 1, the algorithm of Section 3 is employed to find n best tuples, which are maintained in a list. When a better-score tuple is found, it replaces a minimum-score tuple in the list. Step 3 is accomplished by applying the linear-space method discussed in Section 4. Since the (forward) recomputation in step 5 is straightforward and similar to step 1, we leave it as an exercise to the reader.

Step 4 is more involved. In Section 5.3, we discuss a backward computation to locate T and L in step 4. Finally, Section 5.4 describes a variant of the algorithm of Eppstein *et al.* that is required in the backward computation.

5.3. The backward computation. In the following, we describe how to locate T and L in step 4. Let $\overline{Score}(f)$ denote the maximum score over all alignments starting at $f = (i, j, k)$ and ending in $Box(T, L, C.B, C.R)$, and let $Last(f)$ be the obvious analog of $First(f)$. Define

$$Ext_t(f) = \max\left\{0,\ i - \left\lceil \max\{\overline{Score}(f)/e,\ \overline{Score}(f)/r\} \right\rceil\right\}$$

$$Ext_l(f) = \max\left\{0,\ j - \left\lceil \max\{\overline{Score}(f)/e,\ \overline{Score}(f)/r\} \right\rceil\right\}$$

Notice that opening up a gap is not penalized. Roughly speaking, $Ext_t(f)$ and $Ext_l(f)$ are the boundary row and column, respectively, such that extending from row $\le Ext_t(f)$ or column $\le Ext_l(f)$ to f will not gain any additional score. The following two lemmas pave the way for termination conditions. (Only the alignments ending in $Box(T, L, C.B, C.R)$ are considered.)

Lemma 4. If f is a fragment lying above f' and ending above row $Ext_t(f')$, such that the effective region of f does not intersect any row $\ge Ext_t(f')$, then the score of any alignment in which f and f' are adjacent is at most W.

Proof. First we show $Connect(f, f') \ge Score(f) - W + \overline{Score}(f')$. Let $f = (i, j, k)$ and $f' = (i', j', k')$.

Case 1, $Diag(f') = Diag(f)$. By definition,

$$\begin{aligned}
&Connect(f, f') \\
&\quad = (i' - i - k)r \\
&\quad = (Ext_t(f') - i - k)r + (i' - Ext_t(f'))r \\
&\quad = Connect(f, f'') + \left(\left\lceil \max\{\overline{Score}(f')/e, \overline{Score}(f')/r\} \right\rceil\right)r \\
&\qquad\qquad \text{where } f'' \text{ is a pseudo fragment starting at } p = (Ext_t(f'), Ext_t(f') + Diag(f')) \\
&\quad \ge Score(f) - W + \overline{Score}(f') \quad \text{(by Lemma 3)}
\end{aligned}$$

Case 2, $Diag(f') > Diag(f)$. The proof for this case is similar to Case 1.

Case 3a, $Diag(f') < Diag(f)$ and $j + k - 1 < Ext_t(f') + Diag(f')$. The proof for this case is similar to Case 1.

Case 3b, $Diag(f') < Diag(f)$ and $j + k - 1 \geq Ext_t(f') + Diag(f')$. Since the effective region of f does not intersect any row $\geq Ext_t(f')$, it is easy to see that $(Score(f) - W - g)/e \leq Diag(f) - (j + k - Ext_t(f'))$. Thus, $Score(f) - W \leq g + (Diag(f) - (j + k - Ext_t(f'))e$. By definition,

$$
\begin{aligned}
&Connect(f, f') \\
&\quad = gap(Diag(f) - Diag(f')) + (j' - j - k)r \\
&\quad = g + (Diag(f) - Diag(f'))e + (j' - j - k)r \\
&\quad = g + (Diag(f) - (j + k - Ext_t(f'))e + (j + k - Ext_t(f') - Diag(f'))e + (j' - j - k)r \\
&\quad \geq Score(f) - W + (j + k - Ext_t(f') - j' + i'))e + (j' - j - k)r \\
&\quad = Score(f) - W + h_1 e + h_2 r \\
&\qquad\quad \text{where } h_1 + h_2 = i' - Ext_t(f') = \left\lceil \max\{\overline{Score}(f')/e, \overline{Score}(f')/r\} \right\rceil \\
&\quad \geq Score(f) - W + \overline{Score}(f')
\end{aligned}
$$

It follows that the score of an optimal alignment in which f and f' are adjacent

$$
\begin{aligned}
&= Score(f) + \overline{Score}(f') - Connect(f, f') \\
&\leq W
\end{aligned}
$$

□

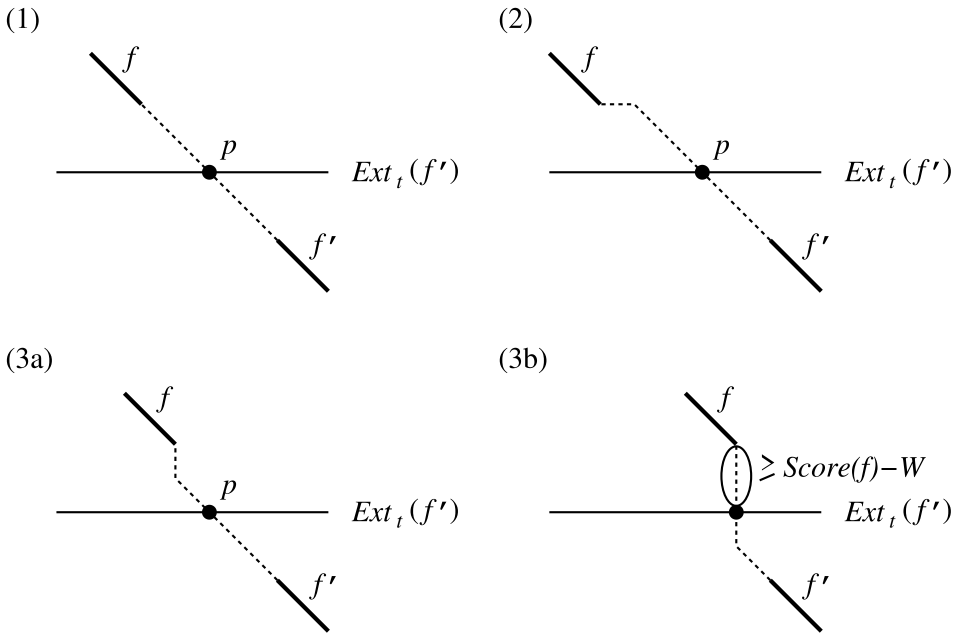


Figure 12. Divide $Connect(f, f')$ at row $Ext_t(f')$.

Lemma 5. If f is a fragment lying above f' and ending left of column $Ext_l(f')$, such that the effective region of f does not intersect any column $\geq Ext_l(f')$, then the score of any alignment in which f and f' are adjacent is at most W.

Proof. The proof is symmetrical to that for Lemma 4. □

Now, termination conditions for a backward computation are given. It involves that we simultaneously determine an intermediate rectangle $Box(T', L', C.B, C.R)$. The backward computation is terminated when it meets the following two conditions (see Figure 13):

(i) If f is still active after we have treated row T and column L, and $Last(f)$ ends in $Box(T', L', C.B, C.R)$, then $Ext_t(f) \geq T$ and $Ext_l(f) \geq L$. (By "active" we mean those fragments associated with some candidate list or those ending closest to T and L on their diagonal.)

(ii) No rectangle for a saved class intersects the Γ-shaped shaded region. ($T < T'$ unless $T' = 0$, and $L < L'$ unless $L' = 0$.)

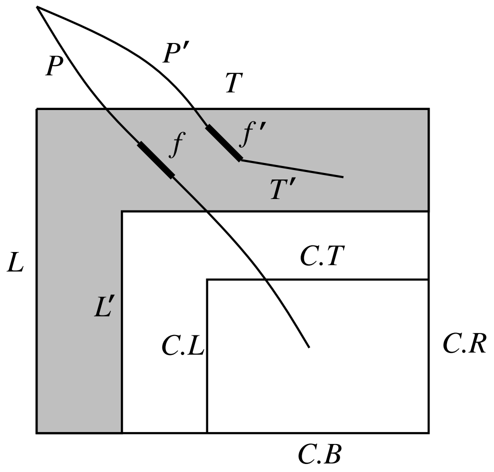


Figure 13. Subcase 2 of Lemma 6.

The following lemma shows that when the backward computation meets the termination conditions, it fulfills the need in step 4 of the algorithm outline given above.

Lemma 6. Let P be an alignment that starts outside $Box(T, L, C.B, C.R)$ and ends in $Box(C.T, C.L, C.B, C.R)$. Then $Score(P) \leq W$.

Proof. Assume $T \neq 0$ and $L \neq 0$ (cases when $T = 0$ and/or $L = 0$ can be handled similarly). Let f be the first fragment of P that ends in $Box(T, L, C.B, C.R)$. There are two subcases.

Subcase 1: f starts outside $Box(T, L, C.B, C.R)$. By condition (i), it is clear that $Last(f)$ must end in the shaded region, which implies $Score(P) \leq W$ by condition (ii).

Subcase 2: f starts in $Box(T, L, C.B, C.R)$ (Figure 13). In this case, there exists some alignment P' aligning an active fragment (possibly f itself), say f', such that f' is the first fragment of P' that starts in $Box(T, L, C.B, C.R)$, and $Score(P) \leq Score(P')$. We now show that $Score(P') \leq W$. If $Last(f')$ ends outside $Box(T', L', C.B, C.R)$, it is easy to show that $Score(P') \leq Score(Last(f')) \leq W$. Otherwise, $Last(f')$ must end in $Box(T', L', C.B, C.R)$, which means $Ext_t(f') \geq T$ and $Ext_l(f') \geq L$ by condition (i). Together with condition (ii), this guarantees that the effective region of any fragment in P' that is above f' does not intersect any row $\geq Ext_t(f')$ or any column $\geq Ext_l(f')$. By

Lemma 4 and Lemma 5, we have $Score(P') \leq W$. Thus $Score(P) \leq W$. □

Figure 14 gives the backward computation for locating T and L. Assume that *LIST* stores $n-m+1$ best tuples for G_m, and only fragments in G_{m+1} are considered. When *locate* stops, the two termination conditions are met. Furthermore, T is guaranteed to be strictly less than T' unless $T' = 0$, and similarly for L. Section 5.4 explains how to update lists affected by adding row t and column l.

```
procedure locate
    Perform a backward computation in region Box(C.T, C.L, C.B, C.R).
    t ← T′ ← C.T
    l ← L′ ← C.L
    T ← max{0, min{T′ − 1, min{Ext_t(f) : f ends in Box(C.T, C.L, C.B, C.R)}}}
    L ← max{0, min{L′ − 1, min{Ext_l(f) : f ends in Box(C.T, C.L, C.B, C.R)}}}
    repeat
      {  while t > T or l > L do
           {  while t > T do
                {  t ← t − 1
                   for each fragment f ending at row t within columns l and C.R do
                     {  Compute Score‾(f)
                        if Last(f) ends in Box(T′, L′, C.B, C.R) then
                          {  T ← min{T, Ext_t(f)}
                             L ← min{L, Ext_l(f)}
                          }
                     }
                   Update lists affected by adding row t
                }
              while l > L do
                {  l ← l − 1
                   for each fragment f ending at column l within rows t and C.B do
                     {  Compute Score‾(f)
                        if Last(f) ends in Box(T′, L′, C.B, C.R) then
                          {  T ← min{T, Ext_t(f)}
                             L ← min{L, Ext_l(f)}
                          }
                     }
                   Update lists affected by adding column l
                }
           }
      } until disjoint(T, L, T′, L′, C.B, C.R) or T = L = 0


boolean disjoint( var T, L, T′, L′; b, r)
    for each c in LIST do
        if c.T ≤ b and c.L ≤ r and c.B ≥ T and c.R ≥ L and (c.T < T′ or c.L < L′) then
          {  T′ ← min{T′, c.T}
             L′ ← min{L′, c.L}
             T ← max{0, min{T, T′ − 1}}
             L ← max{0, min{L, L′ − 1}}
             for each active fragment f do
                 if Last(f) ends in Box(T′, L′, b, r) then
                   {  T ← min{T, Ext_t(f)}
                      L ← min{L, Ext_l(f)}
                   }
             return false
          }
    return true
```

Figure 14. Algorithm for computing T and L.

5.4 Interleaving computations in the row and column directions. Since earlier discussions concerned top-to-bottom computations, the following description is couched in those terms, though in actuality computation of T and L runs in the reverse direction (Figure 14). Computation of T and L differs from the computation of the first and last fragments of an optimal local alignment in the following respects.

1. Fragments contained in alignments reported earlier are not considered.
2. Fragments that cross column $C.R$ or row $C.B$ are ignored.
3. Columns and rows are added to the computed region (in an effort to satisfy condition (i), above), so row and column lengths vary in an unpredictable manner.
4. If extending T and/or L causes the region to intersect a rectangle associated with another equivalence class, C', then T' and L' must be extended to guarantee $T' \le C'.T$ and $L' \le C'.L$ (Figure 13).

Additions 1 and 2 do not warrant further discussion here, and the procedure *disjoint* of Figure 14 covers condition 4. The only important complication caused by property 3 is that candidate lists for rows might be affected while computing in the column direction, and vice versa. For example, in Figure 15, adding more columns to the region might cause q to right-dominate some interval of row $t+1$ (the candidate lists for row $t+1$ remain after treating row t), though q is currently not represented as influencing row $t+1$. Note that computation of $RI(f)$ in the column direction works like computation of $LI(f)$ in the row direction.

5.4.1 Updating the right-influence candidate list for row $t+1$ when adding column l. Consider the effect upon the right-influence candidate list for row $t+1$ from the fragments ending at column l before or at row t. There are two phases in the updating procedure. The first phase deletes those ignorable fragments. Let p and q be fragments ending at column l where q ends above p. Fragment q is said to be ignorable if $Decay(p: x, y) \ge Decay(q: x, y)$ for some (x, y) in their common right-influence region (see Figure 15). This is because the right-influence region of q after row t is contained in p's right-influence region and the inequality holds for all entries in that region (analogous to Lemma 1). Let $c_1, c_2, \cdots, c_h$ be the list of fragments ending at column l before row $t+1$ in decreasing row order. The following pseudo-code removes the ignorable fragments from the list.

```
u ← 1
v ← 2
while v ≤ h do
   {  if Decay(c_u: x, y) ≥ Decay(c_v: x, y) for some (x, y) in their common
         right-influence region then
              remove c_v from the list
      else
              u ← v
      v ← v+1
   }
```

After the removal of those ignorable fragments, the remaining fragments have the property that if one fragment, say f, ends above another, say f' in column l before row $t+1$, then $Decay(f: x, y) > Decay(f': x, y)$ for all (x, y) in their common right-influence region.

Phase 2 determines which of the remaining fragments in phase 1 could possibly right-dominate some region after row t. Let *Rlist* be the right-influence candidate list, sorted by diagonals, for row $t+1$ before adding column l. For each of the remaining fragments in phase 1, say f, search *Rlist* with $Diag(f)$ to find the right-dominating fragment, say f'. If $Decay(f: x, y) > Decay(f': x, y)$ for some (x, y) in their common right-influence region, f is added to *Rlist*. Adding f might cause the deletion of some fragments in *Rlist*. Those fragments can be detected by sweeping *Rlist* from $Diag(f)$ to the right until reaching the end of Rlist, or some fragment, say f'', such that $Decay(f: x, y) <$ $Decay(f'': x, y)$ for some (x, y) in their common right-influence region.

The time spent in phase 1 is $O(h)$, which can be charged as $O(1)$ per fragment. For each remaining fragment, phase 2 performs one search operation, one possible insertion, and some possible deletions caused by adding that fragment. Suppose that the right-influence candidate list is implemented as a balanced search tree, so that each of the search, insertion, and deletion operations can be done in time $O(\log S_R)$, where S_R is the maximum size of the right-influence candidate list. Charge the search and insertion cost to the fragment itself. However, the deletion cost is charged to the deleted fragment. Since each fragment can be deleted from the right-influence candidate list at most once, it follows that for each fragment we charge $O(\log S_R)$ cost in these two phases.

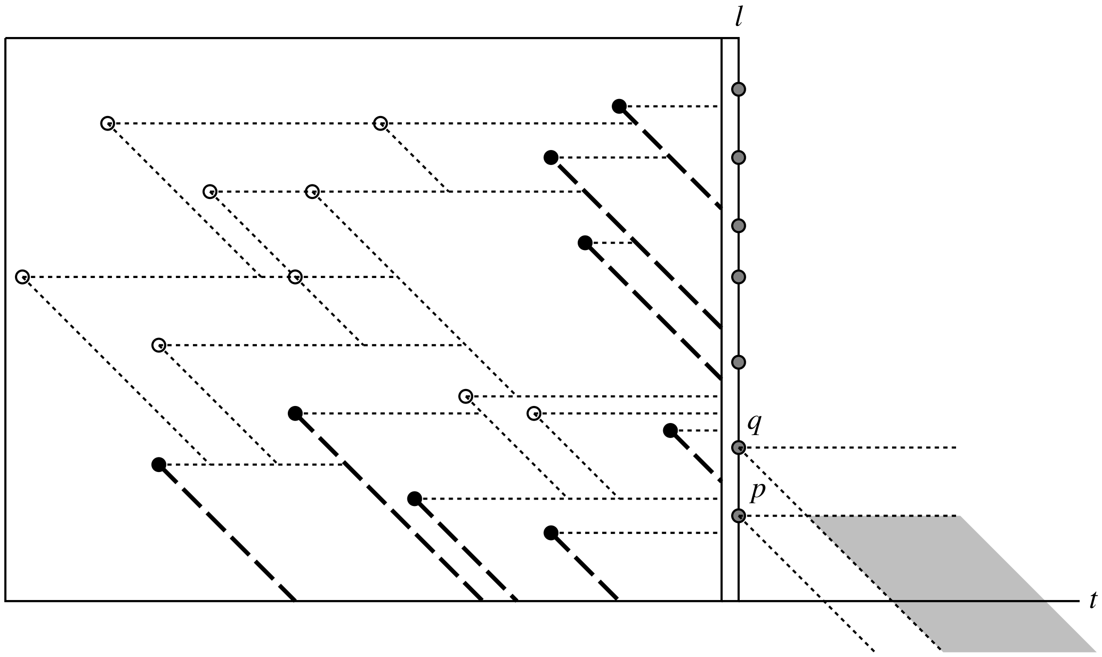


Figure 15. The effect on *Rlist* when adding one more column.

5.4.2 Updating the left-influence candidate lists for row $t+1$ *when adding column* l . Let LC and LD be the left-influence candidate lists for row $t+1$ before adding column l. Let CUT be the intersection lists for the row direction before adding column l. There are two phases in updating LC, LD and the CUT lists for the row direction. The first phase deletes those ignorable fragments. Let p and q be fragments ending at column l where q ends above p. Fragment p is said to be ignorable if $Decay(q: x, y) \geq Decay(p: x, y)$ for some (x, y) in their common left-influence region (see Figure 16). This is because the left-influence region of p after row t is contained in q's left-influence region and the inequality holds for all entries in that region (Lemma 1). The method for this phase is similar to phase 1 in Section 5.4.1. After the removal of those ignorable fragments, the remaining fragments have the property that if one fragment, say f, ends above another, say f' in column l before row $t+1$, then $Decay(f\colon x, y) < Decay(f'\colon x, y)$ for all (x, y) in their common left-influence region.

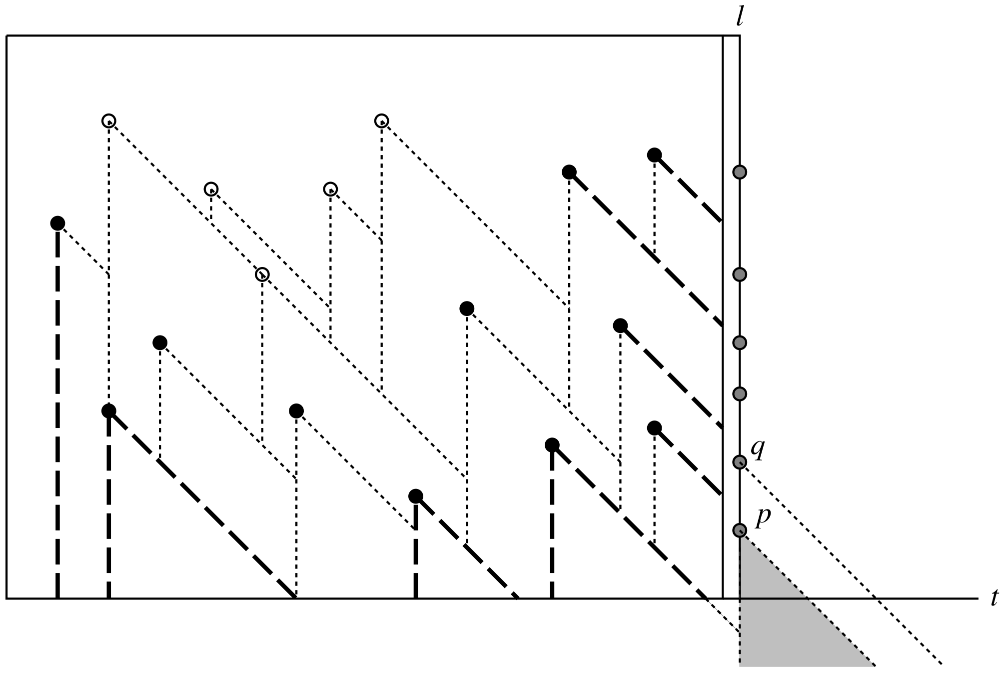


Figure 16. The effect on *LC*, *LD*, and *CUT* when adding a column.

Phase 2 updates *LC*, *LD* and the *CUT* lists. Notice that diagonals may be added to and/or deleted from *LD*. Also, some left-dominating fragments for diagonals may be changed. However, at most one column is added to *LC* and one intersection point is added to the *CUT* list.

Process those remaining fragments upward from row t. Let p be the fragment at hand and let b be the intersection of row $t+1$ and $Diag(p)$ (see Figure 17). First, search *LC* and *LD* with b to find its left-dominating fragment, say s. If $Decay(p\colon x, y) \geq Decay(s\colon x, y)$ for some (x, y) in their common left-influence region, add $Diag(p)$ to *LD* if it is not already there. Moreover, the left-dominating fragment for $Diag(p)$ has to be determined. Let q be the nearest fragment among the remaining fragments that end above p at column l. If $Decay(q\colon x, y) \geq Decay(s\colon x, y)$ for some (x, y) in their common left-influence region, q is the left-dominating fragment for $Diag(p)$. Otherwise, s is the one left-dominating $Diag(p)$.

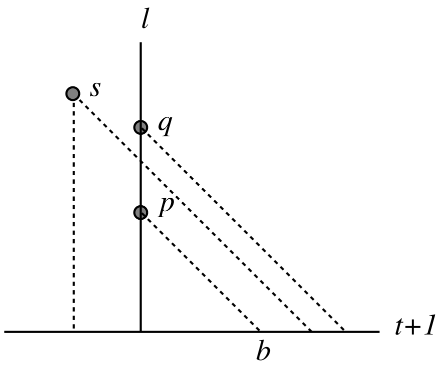


Figure 17. Determining the effect of a remaining fragment.

After adding p, it may be necessary to remove some diagonals from *LD* and change the left-dominating fragment for one diagonal. To do so, delete the *LD*'s diagonal boundaries leftwards from $Diag(p) - 1$ until reaching a diagonal boundary started by a fragment, say p', such that $Decay(p'\colon x, y) > Decay(p\colon x, y)$ for some (x, y) in their common left-influence region, or passing diagonal $l - t$. In the former case, change the left-dominating fragment for diagonal $Diag(p')$ to p. In the latter case, add column l left-dominated by p to LC. Furthermore, if the closest boundary to the left of (t, l) is a diagonal, add an intersection point to the CUT list.

Again, in phase 1 we charge $O(1)$ cost for each fragment. For each remaining fragment, the second phase does two search operations (one for LC and the other for *LD*), at most three insertions (one to LC, another to *LD*, and the other to some CUT list), and some possible deletions from *LD* caused by adding that fragment. Suppose that LC, *LD*, and the CUT lists are implemented as balanced search trees, so that each of the search, insertion, and deletion operations can be done in time $O(\log S_L)$, where S_L is the maximum of the sizes of LC, *LD*, and the CUT lists. Charge the search and insertion cost to the fragment itself. However, the deletion cost will be charged to the fragment ending at the start of the deleted diagonal boundary. Since each diagonal boundary can be deleted from *LD* at most once, it follows that for each fragment we charge $O(\log S_L)$ cost in these two phases.

6. Discussion. We implemented the algorithm for n best local alignments as a C program, called *falign*. Fragments are found using hashing (not suffix trees) and candidate lists are implemented as skip lists (Pugh, 1990).

The tests performed on *falign* included comparison of a 73326-symbol sequence containing the human β-like globin gene cluster and an analogous 44594-symbol sequence from rabbit. As described by Hardison and Miller (1993), this comparison is typical of those that we perform to study gene regulation and molecular evolution. The upper panel of Figure 18 shows the positions of alignments computed by our ‘‘highest-resolution’’ alignment program, called *sim* (Huang *et al*., 1990; Huang and Miller, 1991). (Alignments are drawn from lower left to upper right so that features displayed on the vertical axis follow biological conventions.) *Sim* computes n best local alignments for traditional sequence comparison (not fragment-based). The lower panel of Figure 18 shows the positions of alignments computed by *falign* with fragment length $k = 7$. Two regions of particular interest for us are indicated on Figures 18 and 19; they play prominent roles in regulating these genes. Table 1 reports execution times for *falign*, with three values of k, and for *sim*.

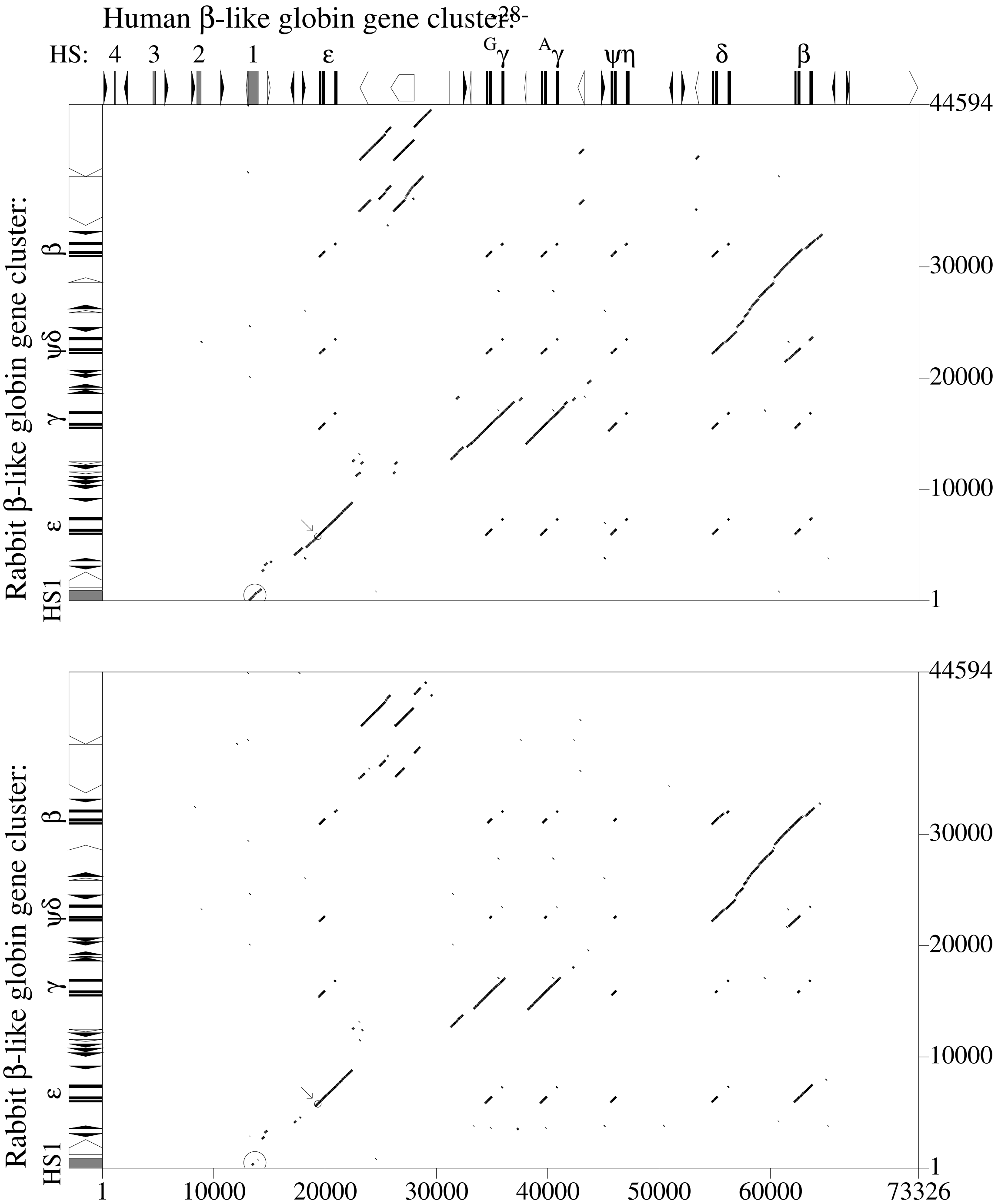


Figure 18. Graphical representations of the positions of local alignments produced by two alignment programs. Regions discussed in the text are indicated by circles and an arrow. The alignments were drawn by the *laps* program (Schwartz *et al*., 1991; Boguski *et al*., 1992). (Upper panel) Alignments found by *sim*, scoring match = 1, mismatch = −1, gap open = 4.0, gap extension = 0.4. An alignment is shown if and only if its score exceeds τ = 23, which was chosen so that 5% of random sequence-pairs of this length and composition have a gap-free local alignment of score at least τ; 190 local alignments met this criterion. (Lower panel) Positions of alignments computed by *falign* with k (the fragment size) set to 7. Matches were scored 1, replacements 0.1, gap open = 3.0 and gap extension = 0.2, and 200 alignments were computed. To get *falign* alignments to approximate *sim* alignments, it is necessary to use more lenient scores, since *falign* has no mechanism to award matches that occur in runs of length less than k. For this plot, a line segment was drawn from the start of each fragment except the last in an alignment until reaching the row or column of the next fragment. Individual fragments are too small to

be seen at this resolution.

	pass 1		*total*	
k	*fragments*	*time*	*fragments*	*time*
8	79,708	0.66	111,844	1.65
7	276,725	2.06	504,964	4.41
6	974,316	7.72	4,086,890	34.5
sim	—	—	—	569

Table 1. Statistics concerning *falign* and *sim* for computing 200 non-intersecting alignments of the sequences shown in Figure 18. The third column gives execution times for the initial pass, as described in Section 3. The fifth column gives the time to compute 200 best local alignments. Times were measured in minutes on a Sun SPARCstation 2 workstation. Notice that with $k = 6$, each fragment is encountered more than four times on average. (Most of this revisiting happens because the rectangle that is recomputed when discarding the best local alignment nearly equals the original problem.)

We investigated how well *falign* would have worked as a substitute for *sim* in two of our recent projects. A study of regulation of the ε-globin gene (Hardison *et al*., 1993a) dealt exclusively with a region of length about 400 in each species, which is indicated by the arrows in Figures 18 and 19. As shown in Figure 19, *falign* detects this region even with $k = 8$. Hardison *et al*. (1993a) used *sim* alignments to delimit the sequence regions that were then submitted to a program that simultaneously aligned sequences from five species, and the *falign* alignments with $k = 8$ would have sufficed for that purpose. The region indicated by the larger circle at the very bottom of Figure 18 is part of the Locus Control Region, or *LCR*, which was studied in Hardison *et al*. (1993b). In the portion of the LCR encompassed by the rabbit sequence used here, namely the region denoted HS1 in Figures 18 and 19, *falign* output was inadequate with $k = 7$. However, with $k = 6$, the alignments produced by *falign* agree very closely with *sim* alignments in the two regions discussed here (Figure 19). Thus, with that fragment size, *falign* (perhaps coupled with the program of Chao *et al*., 1993) would work about as well as *sim* for studies of gene regulation in that region, while running 16 times faster (Table 1).

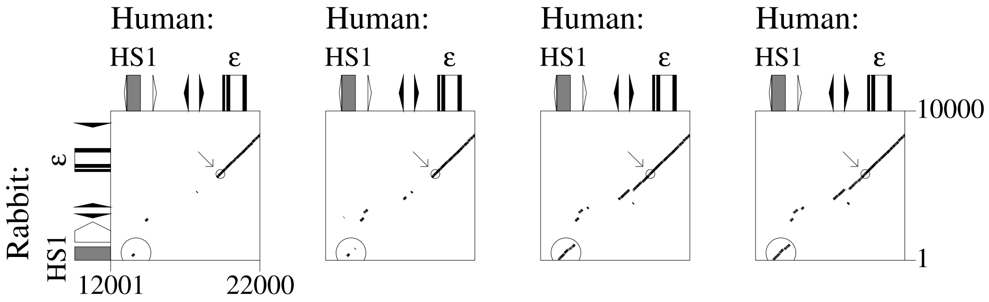


Figure 19. Closeups of *falign* and *sim* alignments in the regulatory region at the lower left of Figure 18. The first three plots depict *falign* alignments with k set at 8, 7 and 6, respectively, and the last shows *sim* alignments. Alignment scores were as described in the legend of Figure 18.

We close the paper by mentioning a few open problems. First, one might extend the method from affine gap penalties to concave or convex penalties. More interesting to us would be the removal of the ''$\log M$'' factor from the $F \log N \log M$ term of our time bound for linear-space alignment. The factor arises in our analysis because in theory the two subproblems (Figure 10) could contain almost all fragments of the parent problem. One might show that the $\log M$ factor disappears in an *expected*-time analysis. Better yet, it might be possible, with an algorithm modification, to prove such a worst-case bound. Finally, we would like to see a tight time analysis for the n best local alignments problem.

7. Acknowledgement. We thank the referees for two of the most thorough and insightful reviews that we have seen recently. Their comments resulted in numerous improvements in the presentation. We also thank Scott Schwartz and Zheng Zhang for helpful conversations.

References

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

Boguski, M., R. Hardison, S. Schwartz and W. Miller (1992) Analysis of conserved domains and sequence motifs in cellular regulatory proteins and locus control regions using new software tools for multiple alignment and visualization. *The New Biologist* **4**, 247-260.

Chao, K.-M., R. C. Hardison W. Miller (1993) Constrained sequence alignment. To appear in *Bull. Math. Biol.*

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

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 (1992a) Sparse dynamic programming. I: linear cost functions. *J. Assoc. Comput. Mach.* **39**, 519-545.

Eppstein, D., Z. Galil, R. Giancarlo and G. F. Italiano (1992b) Sparse dynamic programming. II: convex and concave cost functions. *J. Assoc. Comput. Mach.* **39**, 546-567.

Feng, D. F., M. S. Johnson and R. F. Doolittle (1985) Aligning amino acid sequences: comparison of commonly used methods. *J. Mol. Evol.* **21**, 112-125.

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

Galil, Z., and R. Giancarlo (1989) Speeding up dynamic programming with applications to molecular biology. *Theor. Comput. Sci.* **64**, 107-118.

Galil, Z., and K. Park (1992) Dynamic programming with convexity, concavity and sparsity. *Theor. Comput. Sci.* **92**, 49-76.

Goad, W. B. and M. I. Kanehisa (1982) Pattern recognition in nucleic acid sequences I: a general method for finding local homologies and symmetries. *Nucleic Acids Research* **10**, 247-263.

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

Gribskov, M., R. Luthy and D. Eisenberg (1990) Profile analysis. In Doolittle (1990), 146-159.

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., J. Xu, J. Jackson, J. Mansberger, O. Selifonova, B. Grotch, H. Petrykowska, J. Biesecker 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. To appear in *Nucleic Acids Res.*.

Hardison, R. C., K.-M. Chao, M. Adamkiewicz, J. Jackson, D. Price and W. Miller (1993b) Upstream regulatory elements of the rabbit embryonic ε-globin gene revealed by an improved multiple alignment program and by functional analysis. Submitted.

Hirschberg, D. S. (1975) A linear space algorithm for computing maximal common subsequences. *Comm. Assoc. Comput. Mach.* **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.

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

Myers, E. and X. Huang (1992) An $O(N^2 \log N)$ restriction map comparison and search algorithm. *Bull. Math. Biol.* **54** 599-618.

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

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.

Pascarella, S., and P. Argos (1992) Analysis of insertions/deletions in protein structures. *J. Mol. Biol.* **224**, 461-471.

Pearson, W. R. (1990) Rapid and sensitive sequence comparison with FASTP and FASTA. In Doolittle (1990), 63-95.

Pearson, W. R. and D. Lipman (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci USA* **85**, 2444-2448.

Pugh, W. (1990) Skip lists: a probabilistic alternative to balanced trees. *Comm. Assoc. Comput. Mach.* **33**, 668-676.

Sankoff, D. and J. B. Kruskal (eds) (1983) *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparisons*. Addison-Wesley, Reading, Massachusetts.

Schwartz, S., Miller, W., C.-M. Yang and R. Hardison (1991) Software tools for analyzing pairwise sequence alignments. *Nucleic Acids Research* **19**, 4663-4667.

Sellers, P. H. (1984) Pattern recognition in genetic sequences by mismatch density. *Bull. Math. Biol.* **46**, 501-514.

Smith, T. F. and M. S. Waterman (1981) Identification of common molecular sequences. *J. Mol. Biol.* **147**, 195-197.

Smith, T. F., M. S. Waterman and W. M. Fitch (1981) Comparative biosequence metrics. *J. Mol. Evol.* **18**, 38-46.

Wagner, R. A. and M. J. Fischer (1974) The string-to-string correction problem. *J. Assoc. Comput. Mach.* **21**, 168-173.

Waterman, M. S. (1984) Efficient sequence alignment algorithms. *J. theor. Biol.* **108**, 333-337.

Waterman, M. S. (1989) Chapter 3. Sequence alignments. *Mathematical Methods for DNA Sequences*, M. S. Waterman, ed., CRC Press, Boca Raton, FL., pp. 53-92.

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., and D. Lipman (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc. Nat. Acad. Sci. USA* **80**, 726-730.

Wilbur, W., and D. Lipman (1984) The context dependent comparison of biological sequences. *SIAM J. Appl. Math.* **44**, 557-567.