Aligning two sequences within a specified diagonal band

Kun-Mao Chao, William R. Pearson[1] *and Webb Miller*[2]

Abstract

We describe an algorithm for aligning two sequences within a diagonal band that requires only $O(NW)$ computation time and $O(N)$ space, where N is the length of the shorter of the two sequences and W is the width of the band. The basic algorithm can be used to calculate either local or global alignment scores. Local alignments are produced by finding the beginning and end of a best local alignment in the band, and then applying the global alignment algorithm between those points. This algorithm has been incorporated into the FASTA program package, where it has decreased the amount of memory required to calculate local alignments from $O(NW)$ to $O(N)$ and decreased the time required to calculate optimized scores for every sequence in a protein sequence database by 40%. On computers with limited memory, such as the IBM-PC, this improvement both allows longer sequences to be aligned and allows optimization within wider bands, which can include longer gaps.

Introduction

Rigorous sequence alignment algorithms compare each of the residues in one sequence to each of the residues in the other, a process that requires computation time that is proportional to the product of the lengths of the two sequences ($O(MN)$). Biologically relevant sequence alignments, however, usually extend from the beginning of both sequences to the end of both sequences and thus the rigorous approach is needlessly time consuming; significant sequence similarities are rarely found by aligning the end of one sequence with the beginning of the other. As a result of the biological constraint, it is frequently

Department of Computer Science, The Pennsylvania State University, University Park, PA 16802
[1] Department of Biochemistry, University of Virginia, Charlottesville, VA 22908
[2] To whom reprint requests should be sent.

possible to calculate an optimal alignment between two sequences by considering only those residues that can be aligned within a diagonal band that is W in width. With sequences $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$, one can specify constants $L \le U$ such that aligning a_i with b_j is permitted only if $L \le j - i \le U$, as in Figure 1. (Thus, an alignment of an opsin sequence with β-adrenergic receptor could have any number of insertions and deletions, but for $W = 32$ the difference between the number of insertions and the number of deletions would be limited to 15 or less at any point in the alignment. A more precise definition of the problem is given later.) For example, it rarely takes a dozen insertions or deletions to align any two members of the globin superfamily; thus, an optimal alignment of two globin sequences can be calculated in $O(NW)$ time that is identical to the rigorous alignment that requires $O(NM)$ time. In this case, $N \approx M \approx 150$, so the alignment in a $W = 32$ band is five times faster.

Various strategies for alignment along a diagonal band have been described. The problem is discussed by Sankoff and Kruskal (1983, pp. 276-281) under the rubric "cutting corners," and was also proposed by Ukkonen (1985) and Fickett (1984). Recently Spouge (1991) surveyed this approach for the calculation of optimal distances between sequences. Alignment within a band is used in the final stage of the FASTA program for rapid searching of protein and DNA sequence databases (Pearson and Lipman, 1988; Pearson, 1990). Pearson (1991) has recently shown that for 32 of the 34 protein superfamilies with 20 or more members that are annotated in release 23 of the PIR protein sequence database, searches that use a 31-residue-wide band centered on the diagonal with the highest approximate similarity score perform as well as a rigorous Smith-Waterman calculation.

As with rigorous $O(NM)$ sequence comparisons, one can calculate measures of sequence similarity in a band that reflect either a "global" or a "local" similarity score. Global similarity scores were first described by Needleman and Wunsch (1970); they require that the alignment extend from the beginning of both sequences to the end of both sequences (Waterman, 1989; Pearson and Miller, 1992). For optimization in a band, the requirement to "start at the beginning, end at the end" is reflected in the $L \le \min(0, N - M)$ and $U \ge max(0, N - M)$ constraints. "Local" sequence alignments do not require that the beginning and end of the alignment correspond to the beginning and end of the sequence (i.e., the aligned sequences can be arbitrary substrings of the given sequences, A and B); they simply require that the alignment have the highest similarity score. For a "local" alignment in a band, it is natural to loosen the requirement to simply $L \le U$. Algorithms for computing an optimal local alignment can utilize a global alignment procedure to perform subcomputations: once locally optimal substrings A' of A and B' of B are found, which can be done by any of several available methods, then a global alignment procedure is called with A' and B'. Appropriate values of L' and U' for the global problem are inferred from the L and U of the local problems. In other situations, a method to find unconstrained local alignments (i.e., without band limits) might determine appropriate values of L and U before invoking a global alignment procedure within a band.

While the application of rigorous alignment algorithms to long sequences can be quite time-consuming, it is often the space requirement that is limiting in practice. Straightforward implementations of the dynamic programming algorithm use $O(MN)$ space to produce alignments. Thus, aligning two sequences of length $M = N = 5{,}000$ might take a few minutes on a workstation, which is affordable though annoying, but the computation might be impossible because the computer possesses insufficient memory. When M and N approach 50,000, these methods cannot be run on even the largest supercomputer. This limitation can be removed with space-saving strategies. Several recent algorithms (Hirschberg, 1975; Myers and Miller, 1988; Huang *et al.*, 1990) use only "linear space", i.e, space proportional to the sum of the sequences' lengths, which allows them to be applied to sequences of length 100,000 on a workstation (though the computation might run for several days). Remarkably, these linear-space solutions usually require only about twice as much computation time as the $O(MN)$-space methods (Myers, 1986; Miller and Myers, 1988; Huang and Miller, 1991).

The global alignment algorithm of Myers and Miller (1988), which permits both a gap-open penalty and a gap-extension penalty, can be easily modified to constrain the solution to a band. Unfortunately, the resulting time required to produce the alignment can exceed that of the score-only calculation by a substantial factor. If T denotes the number of entries in the band of the dynamic programming matrix that are filled in ($T = N \times W$), then producing the alignment involves computing as many as $T \times \log_2 N$ entries (including recomputations of entries evaluated at earlier steps). Thus, the time to deliver an alignment exceeds that for computing its score by a log factor; for $N = 1000$, the simple linear-space strategy requires 10-times longer to construct the alignment than the $O(NW)$-space approach.

This paper presents algorithms for aligning sequences within a band that require linear space and $O(NW)$ time to calculate either the similarity score or the actual alignment. The score-only algorithms are straightforward extensions of algorithms described elsewhere (e.g., Huang *et al*., 1990), and we only sketch their derivation. The focus of this paper is on a linear-space, $O(NW)$ time algorithm to produce an optimal global alignment within a specified band. The reader familiar with dynamic programming methods for sequence alignment is invited to skip over the subsection entitled ‘‘Computing the optimal score’’ and proceed directly to ‘‘Delivering the alignment.’’

System and Methods

The program described in this paper is written in C and was developed on Sun workstations running SunOS Unix. The code is portable and has been implemented on IBM-PC (DOS) and Macintosh personal computers.

Algorithm

Consider the problem of *locally* aligning within a band. We are given sequences $A = a_1 a_2 \cdots a_M$ and $B = b_1 b_2 \cdots b_N$ and band limits $L \le U$. The goal is to compute a highest-scoring alignment of a substring $A' = a_c a_{c+1} \cdots a_s$ of A and a substring $B' = b_d b_{d+1} \cdots b_t$ of B subject to the following constraint. Any non-empty prefix of the alignment must constitute an alignment of $a_c a_{c+1} \cdots a_i$ with $b_d b_{d+1} \cdots b_j$ where $L \le j - i \le U$. Alignments are scored by adding a bonus $V(a, b)$ for aligning symbol a with b and subtracting a penalty $Q + R \times k$ for a gap of k symbols. The substitution scores V, the gap-open penalty $Q \ge 0$ and the gap-extension penalty $R \ge 0$, where at least one of Q or R must be non_zero, are given as part of the problem statement. In FASTA, the PAM250 matrix (Dayhoff *et al.*, 1978) usually provides the substitution scores V for protein sequence comparisons; FASTA uses a gap-open penalty of $Q = 8$ and a gap-extension penalty of $R = 4$. (The condition that one of Q or R must be non-zero guarantees that a locally optimal alignment cannot begin with a gap, which justifies the above definition's implicit assumption that any non-empty prefix involves entries of both A and B.)

Computing the optimal score.

The approach taken by Huang *et al*. (1990, Figure 1A) can be modified to produce an algorithm for the score-only version of this problem. See Figure 2, where three optimal scores are computed at each grid-point (i, j), *viz*., $E(i, j)$ (for alignments ending with a deletion), $F(i, j)$ (for alignments ending with an insertion), and $H(i, j)$ (for arbitrary alignments). The main modification is a "shearing" operation on the score matrices to save space. The left side of Figure 1 depicts the usual dynamic-programming matrix and shows the band of width $W = U - L + 1$ that needs to be filled in. The right side shows the actual storage arrangement. The position at the intersection of row i and column j of the right side corresponds to row i and column $j + L + i - 1$ on the left. It is straightforward (though tedious) to work out all the details. We have simplified treatment of the boundary values by setting them all to 0, whereas Figure 1A of Huang *et al*. (1990) sets the boundary values of E and F to $-Q$. The net result is that a gap at the beginning of an alignment is not charged a gap-open penalty, but this does not affect an optimal local alignment.

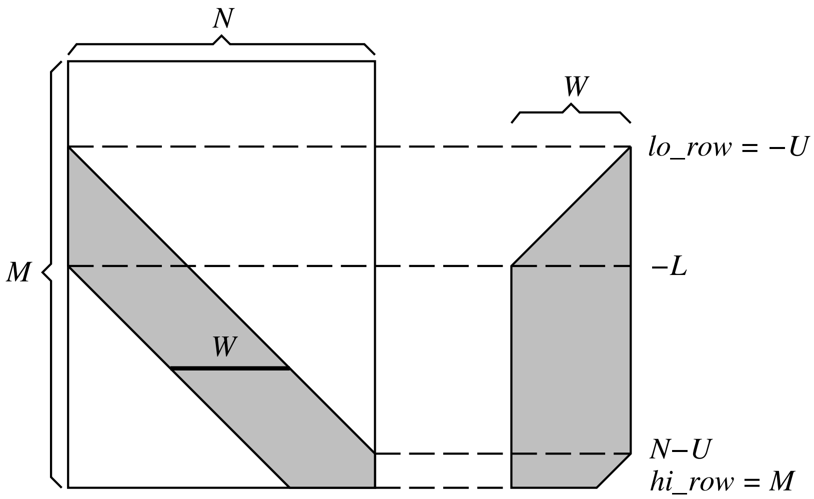

Fig. 1. Case where $U < 0$ and $N - U < M \leq N - L$.

```
scalar arrays H[0..M, 0..W + 1], E[0..M, 1..W + 1], F[0..M, 0..W]
scalar score
integers lo_row, hi_row, lo_diag, hi_diag

hi_diag ← U − L + 1
if L > 0 then lo_diag ← 1
else if U < 0 then lo_diag ← hi_diag
else lo_diag ← 1 − L
lo_row ← max(0, − U)
hi_row ← min(M, N − L)
score ← 0
H(lo_row, lo_diag − 1) ← 0
for j ← lo_diag to hi_diag + 1 do
      H(lo_row, j) ← E(lo_row, j) ← 0
for i ← lo_row + 1 to hi_row do
   {  if lo_diag > 1 then lo_diag ← lo_diag − 1
      if i > N − U then hi_diag ← hi_diag − 1
      H(i, lo_diag − 1) ← F(i, lo_diag − 1) ← 0
      for j ← lo_diag to hi_diag do
         {  F(i, j) ← max{F(i, j − 1), H(i, j − 1) − Q} − R
            E(i, j) ← max{E(i − 1, j + 1), H(i − 1, j + 1) − Q} − R
            H(i, j) ← max{H(i − 1, j) + V(a_i, b_{j+L−1+i}), E(i, j), F(i, j), 0}
            score ← max{score, H(i, j)}
         }
   }
write "best score is" score
```

Fig. 2. $O(MW)$-space, score-only algorithm for local alignment within a band.

Passing from the quadratic-space version of Figure 2 to the linear-space version in Figure 3 is again straightforward. Storing scores by diagonal instead of column slightly simplifies the algorithm; the variable h of Figure 1A of Huang *et al.* is unnecessary.

```
scalar vectors HH[0..W + 1], EE[1..W + 1]
scalars score, f
integers lo_row, hi_row, lo_diag, hi_diag

hi_diag ← U − L + 1
if L > 0 then lo_diag ← 1
else if U < 0 then lo_diag ← hi_diag
else lo_diag ← 1 − L
lo_row ← max(0, − U)
hi_row ← min(M, N − L)
score ← 0
HH(lo_diag − 1) ← 0
for j ← lo_diag to hi_diag + 1 do
      HH(j) ← EE(j) ← 0
for i ← lo_row + 1 to hi_row do
  {  if lo_diag > 1 then lo_diag ← lo_diag − 1
     if i > N − U then hi_diag ← hi_diag − 1
     HH(lo_diag − 1) ← f ← 0
     for j ← lo_diag to hi_diag do
       {  f ← max{f, HH(j − 1) − Q} − R
          EE(j) ← max{EE(j + 1), HH(j + 1) − Q} − R
          HH(j) ← max{HH(j) + V(a_i, b_{j+L−1+i}), EE(j), f, 0}
          score ← max{score, HH(j)}
       }
  }
write "best score is" score
```

Fig. 3. Linear-space, score-only algorithm for local alignment within a band.

Later, when we consider the problem of producing an optimal local alignment (not just its score), we will have several options for determining the start and the end of an optimal local alignment (i.e., the positions (c, d) and (s, t) mentioned in the first paragraph of this section). One approach is to embellish the algorithm of Figure 3 so that it retains the start points of optimal local alignments (see Huang and Miller (1991) for more details). However, doing so roughly doubles the cost of the computation. The alternative that we prefer here is to perform the ‘‘forward pass’’ of Figure 3 to compute the end point, then use a reverse pass from the end point to find the start point. This approach is developed further by Huang *et al*. (1990, Figure 1B), and we do not give the details in this paper.

The next component of our linear-space algorithms for delivering alignments is an algorithm for computing the optimal score of *global* alignments within a band, Figure 4. For this problem, we require that $L \le \min(0, N - M)$ and $U \ge \max(0, N - M)$, i.e., the band must contain the upper left and the lower right corners of the dynamic programming matrix. Failing this condition, the global alignment cannot lie between diagonals L and U. No major new ideas are required, though some minor observations are helpful. Performing the computation bottom-up, rather than top-down, results in a minor economy when producing an explicit alignment (see below). Certain boundary values are set to minus infinity to prevent alignments from straying outside the band. Initialization of row M follows Myers and Miller (1988). Moreover, unlike Figure 3, the HH values can be negative.

```
scalar vectors HH[0..W + 1], EE[0..W]
scalars t, f
integers lo_diag, hi_diag

lo_diag ← 1
hi_diag ← N − M − L + 1
HH(hi_diag) ← 0
HH(hi_diag + 1) ← HH(lo_diag − 1) ← EE(lo_diag − 1) ← −∞
EE(hi_diag) ← t ← −Q
for j ← hi_diag − 1 down to lo_diag do
  {  HH(j) ← t ← t − R
     EE(j) ← t − Q
  }
for i ← M − 1 down to 0 do
  {  if i < −L then lo_diag ← lo_diag + 1
     if hi_diag < W then hi_diag ← hi_diag + 1
     HH(hi_diag + 1) ← f ← −∞
     for j ← hi_diag down to lo_diag do
       {  f ← max{f, HH(j + 1) − Q} − R
          EE(j) ← max{EE(j − 1), HH(j − 1) − Q} − R
          HH(j) ← max{HH(j) + V(a_i, b_{j+L+i}), EE(j), f}
       }
  }
write "score is" HH(lo_diag)
```

Fig. 4. Linear-space, score-only backward algorithm for global alignment within a band.

Delivering the alignment.

We now describe a method for delivering alignments using linear space in $O(NW)$ time. To begin, we review the method of Hirschberg (1975) and note where it fails for alignments within a band. A more complete discussion of Hirschberg's approach and its generalizations is given by Pearson and Miller (1992).

Consider the global alignment problem without band constraints. An alignment can be interpreted as a path through the rectangular dynamic programming matrix, from the upper left to lower right. The time required by the linear-space, score-only algorithm is interpreted as the matrix's area (the unit time is that required to evaluate the matrix at one point, (i, j)). To construct an alignment in linear space, the score-only algorithm is modified so that it determines the point where an optimal path crosses the middle row (i.e., it finds the aligned pair involving $a_{M/2}$), which leaves two subproblems to be solved to complete the alignment. The critical observation for the time analysis is that the total area (i.e., time) of the subproblems is always exactly half that of the original problem.

Repeating this procedure on each of the subproblems produces more aligned pairs and generates more "subsubproblems" that total one fourth of the original problem's area, as depicted in Figure 5. Letting T denote the score-only time, the time for solving all the problems that arise in this "divide-and-conquer" approach, and hence produce the complete alignment, is $T \times (1 + ½ + ¼ + \cdots) \approx 2T$.

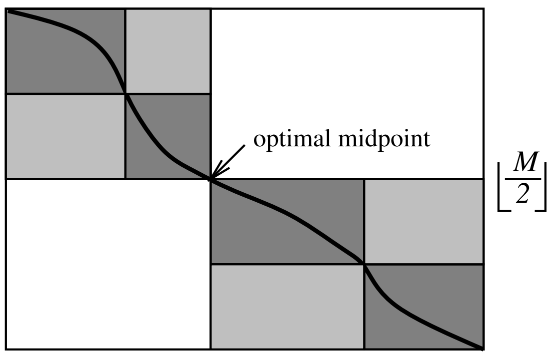

Fig. 5. Hirschberg's divide-and-conquer approach.

To see what goes wrong with a simplistic implementation of band constraints, suppose $L = U$ (which constitutes the "worst case"). Dividing the problem in half produces two subproblems of total size *equal to* the original problem. Similarly, the total size of the subsubproblems equals the original problem. Thus the time for the complete computation is $T \times (1 + 1 + 1 \cdots)$, and we need to determine the length of the string of 1's. The problem reduces to the following. Suppose we have a line of length N (let both sequences have length N). We divide that line in half, divide a sub-line in half, divide a subsub-line in half, $\cdots$. How many divisions before we reach a line of length 1? The answer is $\log_2 N$ (rounded upward) which implies that the total time is $T \times (\log_2 N)$. This additional log factor can be substantial (e.g., for N = 50,000, $\log_2 N$ is 16); fortunately it is unnecessary.

To avoid the log factor, we need a new way to subdivide the problem that limits the subproblems to some fraction, $\alpha < 1$, of the band. Figure 6 summarizes the idea. The score-only backward pass (Figure 4) is augmented so that at each point it computes the next place where an optimal path crosses the mid-diagonal (i.e., diagonal $\frac{1}{2}(L+U)$). Using only linear space, we can save this information at every point on the ''current row'' or on the mid-diagonal. (More details are provided below.) When this pass is completed, we can use the retained information to find the sequence of points where an optimal solution crosses the mid-diagonal, which splits the problem into some number of subproblems. The total area of these subproblems is roughly half of the original area for a narrow band with widely-space crossing points; in other cases it is even less. Figure 6 depicts the original problem and the set of subproblems.

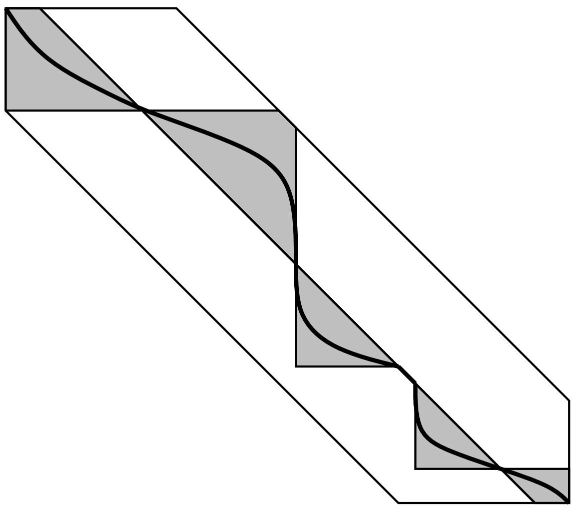

Fig. 6. Splitting the problem into subproblems.

These observations yield the following outline of a recursive procedure to deliver an optimal alignment.

```
procedure
    if M = 0 or N = 0 or L = U then
        Generate the obvious alignment.
    else
      { Determine the nodes where an optimal path crosses the mid-diagonal.
        from the first subproblem to the last subproblem do
            Recursively solve the subproblem.
      }
end
```

A more thorough discussion of the algorithm presupposes an intimate knowledge of dynamic-programming sequence alignment with gap-open penalty $Q \neq 0$. Specifically, we assume familiarity with the graph model that we have employed in earlier papers (Myers and Miller, 1989, pp. 7-11; Huang and Miller, 1991, pp. 339-343). Our discussion will be couched in terms of the normal "upright" dynamic programming matrix. Our actual implementation of these ideas applies the shearing transformation illustrated in Figure 1.

At each grid-point (i, j) there are three nodes, *viz.*, the E node (for alignments ending with a deletion), the F node (for alignments ending with an insertion), and the H node (for arbitrary alignments). The presence of three nodes at each grid-point, which arises because $Q \neq 0$, is troublesome when we seek the intersections of an optimal path and the mid-diagonal; we need to determine more than just i and j. Of course, later computations require that we also determine which side of the mid-diagonal contains the subproblem.

In the score-only backward pass we retain, for each of these three nodes at each grid-point on the current row, the optimal score and the row index where an optimal path from that node crosses the mid-diagonal. If (i, j) is on the mid-diagonal, then for each node we retain these same two values plus the specific edge by which an optimal path leaves the node.

The row index of the next crossing point for the E node at (i, j) can be computed as follows. (F and H nodes are handled in a similar way.) Examining the max operation at the E node tells us which node contributes the maximum value to the current E node. Suppose $HH(j-1) - Q > EE(j-1)$, i.e., the H node at $(i+1, j)$ is the contributing node. (Notice that the operation is sheared, see Figure 4.) Then the following pseudo-code computes the row index of the next crossing point.

if $(i+1, j)$ is on the mid-diagonal **then**
  $Next(j)_E \leftarrow i+1$
else
  $Next(j)_E \leftarrow Next(j-1)_H$

A similar approach handles the case when $HH(j-1) - Q < EE(j-1)$. In the case of a tie, any tie-breaking rule can be applied.

When the backward pass reaches the upper left corner, this information allows us trace along an optimal path to determine all the crossing points and each one's in-edge and out-edge, as follows. Without loss of generality, suppose that the source node lies above the mid-diagonal. Information retained at the source node gives us its next crossing point (i, j), and the edge entering (i, j) must be a vertical edge. Thus this edge starts at the E node at $(i-1, j)$ and ends at either the E node or the H node at (i, j). Given the scores at the two possible ending nodes, we can decide which case holds. Working from the desired node at (i, j), we use information retained there to determine the leaving edge, the location of the next crossing point, and the direction of the edge entering it. Figure 7 illustrates this computation. Iterating this process takes us to the goal node.

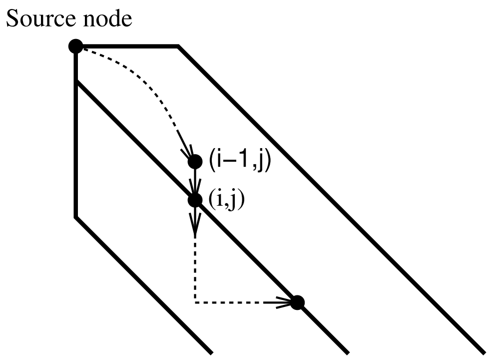

Fig. 7. Finding successive crossing points.

It should be noted that this band-aligning algorithm could be considered to generalize Hirschberg's approach by rotating the matrix partition line. The idea of partition line rotation has been exploited by Edmiston *et al*. (1988) and others in devising parallel sequence comparison algorithms. Nevertheless, the dividing technique proposed in this paper, which can produce more than two subproblems, reveals a new paradigm for space-saving strategies.

Finally, we make two observations about the process of ''solving a subproblem.'' First notice that our recursive procedure must solve a slightly more general problem than aligning within a band; we must be able to tell it which kind of node (E, F, or H) to start at and which kind to end at. (The assumption implicit in Figure 4 is that we start and end at H nodes.)

Second, we utilize knowledge of the direction of the first or last edge in the subproblem's solution to decrease the band-width by 1. For example, in the prototypical subproblem depicted in Figure 8, we emit the aligned pair that deletes a_{k+1} (corresponding to the dark vertical edge), recursively solve the shaded subproblem, then emit the aligned pair that inserts b_{l+mid_d} to return the path to the mid-diagonal.

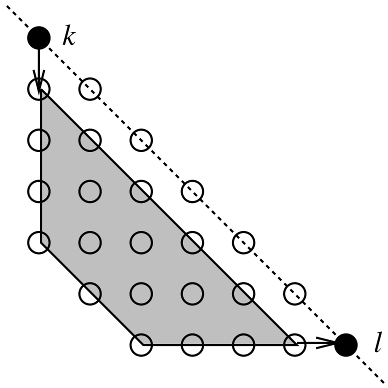

Fig. 8. Closer view of a subproblem.

Our explanation of the alignment-delivering algorithm is complete. Development of pseudo-code is straightforward from this point. However, a complete algorithm (initialization, coverage of all subcases, application of the shearing transformation, etc.) at the level of detail in Figures 2-4, constitutes around 175 lines of pseudo-code.

Execution time

A thorough time analysis of our alignment-delivering algorithm runs into trouble with the odd-shaped first and last subproblems in Figure 6, but a bound of $4T$ can be rigorously proved. Here T is the time for the score-only algorithm (Figure 4) as augmented to compute crossing points. In experiments that we conducted, Figure 4 averaged 1.4 times faster than its augmented variant, which itself ran 1.7 time faster than the alignment-generating strategy of Figure 6. Thus, the alignment-generating procedure ran in 2.4 times the score-only time.

Implementation

These algorithms for local scoring in a band have been incorporated into a new version of the FASTA package (1.6), replacing the previous algorithms for alignments in a band. Two forms of the algorithms are used. FASTA and TFASTA use a rapid lookup-table-based method to identify the diagonal region that has the highest density of matches. The programs then calculate the best local alignment within a 31-residue band centered on that diagonal region, starting at the beginning of the projection of the band on the two sequences and stopping at the end of the projection of the band. This diagonal band can be quite asymmetrical, as Figure 1 shows. Alignments obtained with the new algorithm were identical to those obtained with the older version of the program with a few exceptions, where it appears that the older version of the program failed to extend the alignment properly. FASTA version 1.6 calculates local optimal scores in a band about 1.6 times as fast as earlier versions (Table I). As a result, it is now practical to routinely search the PIR protein sequence database using FASTA with *ktup=1* and calculate an optimized score for every sequence in the library, a comparison strategy that is as sensitive as the rigorous Smith-Waterman algorithm (Pearson, 1991, Table 5).

To reconstruct the alignment, earlier versions of FASTA required NW bytes on limited memory machines, or NW integers on larger machines. By incorporating the linear-space, global-band-alignment algorithm described here, the amount of space required has decreased to $4N$ bytes plus $4N$ integers. On an IBM-PC, this translates to $12N$ bytes, so sequences that are about three times as long can be aligned. The more dramatic effect of the linear-space algorithm results when wider bands are considered. Although the computation time increases for wider bands, the memory requirement does not.

The LFASTA program, which reports multiple local alignments between two sequences, uses a slightly different approach for defining the boundaries of the local alignment. In contrast to FASTA and TFASTA, which look for the best local alignment anywhere within the projected band, LFASTA starts at the end of an initial region and scans backwards until the local alignment score reaches zero. In this case the band is symmetric at the start: $L = -\frac{W-1}{2} < 0$ and $U = \frac{W-1}{2} > 0$ for L, U centered on the end of the initial region. LFASTA then scans forward from (c, d), where the best local alignment score was obtained, until the local score in the forward direction reaches zero. (s, t), the boundary found in this forward pass, is assigned at the position of the best local score in the forward direction. (c, d) and (s, t) are then used to produce a global sequence alignment.

Discussion

Programs for pairwise alignment within a band have a number of applications. Several uses are covered above; this section discusses a few others. Two of us (K.-M. C. and W. M.) are involved in the development of algorithms for sequence alignment and of software tools for analyzing those alignments (Schwartz *et al*., 1991; Boguski *et al*., 1992). The two pairwise alignment algorithms that we currently use, i.e., *blast* (Altschul *et al*., 1990) and *sim* (Huang *et al*., 1990), lie at opposite ends of the time-versus-sensitivity spectrum. For example, when comparing two 50 kilobase sequences, *blast* might run in 20 seconds (depending on how certain internal parameters are chosen) and *sim* might take 20 hours. LFASTA (Pearson and Lipman, 1988; Pearson, 1990) lies between these extremes, and could perform the alignments in about an hour. The main reason for *blast*'s reduced sensitivity (and the source of its speed) is that it produces only gap-free alignments. We are experimenting with several candidates for an alignment procedure lying at some intermediate point, i.e., with nearly the sensitivity of *sim*, but much more time-efficient. The general idea is to apply a fast method for identifying endpoints of desired alignments (i.e., for determining the substrings A' and B'), then apply dynamic programming to produce an alignment where gaps are handled optimally. One candidate, which preliminary trials indicate will do rather well at detecting meaningful alignments where the portions between gaps are individually low-scoring, extends interesting algorithmic idea developed recently by Galil and his coworkers (Eppstein *et al*., 1992). Like LFASTA, this alignment program uses a method other than the traditional dynamic-programming alignment algorithm to detect the presence of an interesting alignment, then applies dynamic programming in a band between appropriate diagonals.

Another approach involves user interaction with the graphical interface *lav* (Schwartz *et al*., 1991), which presents the user with a dotplot-like representation of a collection of local alignments. A box can be drawn around an interesting rectangular region, using the mouse, then *sim* can be run on a parallel computer over the indicated region (Huang *et al*., 1992). Often the natural regions that arise are band-shaped, and it would be time-efficient to align only within the band.

Many algorithms for aligning within a band remain to be developed, including aligning a short sequence to an arbitrary section of a long sequence (Sellers, 1980), efficient computation of k best local alignments (Huang and Miller, 1991), and methods for parallel computers. Another interesting problem is alignment within regions that are neither rectangles nor bands.

Availability

The C source code for our method for delivering alignments within a band, for both local and global alignments, can be obtained over the Internet via anonymous ftp from groucho.cs.psu.edu. The authors can be contacted by electronic mail at webb@cs.psu.edu. The latest version of the FASTA package can be obtained via anonymous ftp from uvaarpa.virginia.edu.

Acknowledgements

The referees provided many comments that improved the paper. Xiaoqiu Huang implemented a program for aligning within a band that incurred the unnecessary $\log_2 N$ time overhead. When we mentioned the problem of aligning within a band in score-only time to Gene Myers, he independently suggested that it might be possible to subdivide along a diagonal. K.-M.C. and W.M. are supported in part by grant R01 LM05110 from the National Library of Medicine. W.R.P. is supported by grant R01 LM04961 from the National Library of Medicine.

References

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

Boguski, M., Hardison, R. C., Schwartz, S., and Miller, W. (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. To appear in *The New Biologist*.

Dayhoff, M., Schwartz, R. M., and Orcutt, B. C. (1978) A model of evolutionary change in proteins *Atlas of Protein Sequence and Structure vol. 5, suppl. 3* National Biomedical Research Foundation, Silver Spring, MD 345-352.

Edmiston, E. W., Core, N. G., Saltz, J. H., and Smith, R. M. (1988) Parallel processing of biological sequence comparison algorithms. *International Journal of Parallel Programming*, **17**, 259-275.

Eppstein, D., Galil, Z., Giancarlo, R. and Italiano, G. F. (1992) Sparse dynamic programming. I. Linear cost functions. To appear in *Jour. Assoc. Comput. Mach.*.

Fickett, J. W. (1984) Fast optimal alignment. *Nucleic Acids Res.*, **12**, 175-180.

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

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

Huang, X. and Miller, W. (1991) A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math.*, **12**, 337-357.

Huang, X., Miller, W., Schwartz, S. and Hardison, R. C. (1992) Parallelization of a local similarity algorithm. To appear in *CABIOS*.

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

Myers, E. W. (1986) An $O(ND)$ difference algorithm and its variations. *Algorithmica*, **1**, 251-266.

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

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

Needleman, S. B. and Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* **48**, 444-453.

Pearson, W. R. (1990) Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods Enz.*, **183**, 63-98.

Pearson, W. R. (1991) Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, in press.

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

Pearson, W. R. and Miller, W. (1992) Dynamic programming algorithms for biological sequence comparison. To appear in *Meth. Enz.*.

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

Schwartz, S., Miller, W., Yang, C.-M. and Hardison, R. C. (1991) Software tools for analyzing pairwise alignments of long sequences. *Nucleic Acids Research*, **17**, 4663-4667.

Sellers, P. H. (1980) The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, **1**, 359-373.

Spouge, J. L. (1991) Fast optimal alignment. *CABIOS*, **7**, 1-7.

Ukkonen, E. (1985) Algorithms for approximate string matching. *Information and Control*, **64**, 100-118.

Waterman, M. S. (1989) Chapter 3. Sequence alignments. *Mathematical Methods for*

DNA Sequences, M. S. Waterman, ed., CRC Press, Boca Raton, FL., 53-92.

Table I

Search times with FASTA version 1.6

PIR Code		Length	FASTA v1.6 *ktup=1*	FASTA v1.6 optimized	FASTA v1.5 optimized	SSEARCH
cchu	Human cytochrome C	105	0.57	1.33	2.35	9.98
lcbo	Bovine Prolactin	229	1.07	2.37	4.05	21.77
a27366	Rat AMP deaminase	747	3.03	5.13	8.07	70.92
rnby3l	Yeast RNA polymerase	1460	5.98	8.37	11.60	138.58

Execution times (minutes) required to scan the PIR2.SEQ file (preliminary entries, 12,837 sequences containing 3,384,087 amino acids) of the NBRF Protein Identification Resource protein sequence database (release 30, September, 1991). The sequences shown were used to scan the database with FASTA version 1.6, *ktup=1*, with no optimized scores calculated during the scan, with *ktup=1*, calculating optimized scores for every sequence in the database, with FASTA version 1.5, calculating optimized scores for every sequence in the database, or with SSEARCH version 1.6. SSEARCH identifies the best local alignment using an $O(NM)$ implementation of the Smith-Waterman algorithm (Pearson, 1991; Pearson and Miller, 1992). Search times with FASTA version 1.5, *ktup=1*, with no optimized scores calculated during the database scan were indistinguishable from those obtained with version 1.6. Timings were performed on an IBM RS/6000 model 540.