JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 7, Numbers 1/2, 2000
Mary Ann Liebert, Inc.
Pp. 203–214

A Greedy Algorithm for Aligning DNA Sequences

ZHENG ZHANG,[1] SCOTT SCHWARTZ,[1] LUKAS WAGNER,[2] and WEBB MILLER[1]

ABSTRACT

For aligning DNA sequences that differ only by sequencing errors, or by equivalent errors from other sources, a greedy algorithm can be much faster than traditional dynamic programming approaches and yet produce an alignment that is guaranteed to be theoretically optimal. We introduce a new greedy alignment algorithm with particularly good performance and show that it computes the same alignment as does a certain dynamic programming algorithm, while executing over 10 times faster on appropriate data. An implementation of this algorithm is currently used in a program that assembles the UniGene database at the National Center for Biotechnology Information.

Key words: sequence alignment, greedy algorithms, dynamic programming.

1. INTRODUCTION

LET $A = a_1a_2 \ldots a_M$ AND $B = b_1b_2 \ldots b_N$ BE DNA SEQUENCES whose initial portions may be identical except for sequencing errors. Our goal is to see whether they are, in fact, extremely similar and, if so, how far that similarity extends. For instance, if one but not both of the sequences has had introns spliced out, the similarity might extend only to the end of the first exon. The fact that only sequencing errors are present, rather than evolutionary changes, means that we should not utilize "affine" gap costs (which penalize each run of consecutive gap columns an additional "gap open" penalty), and this simplifies the algorithms. For this discussion, let us assume an error rate of 3%.

To see how far the initial identity extends, we can set i and j to 0 and execute the following while-loop.

```
while $i < M$ and $j < N$ and $a_{i+1} = b_{j+1}$ do
    $i \leftarrow i + 1; j \leftarrow j + 1$
```

Let the loop terminate when $i = k$, i.e., where $k+1$ is the smallest index such that $a_{k+1} \neq b_{k+1}$, and suppose the mismatch is caused by a sequencing error. We expect that restarting the while-loop immediately beyond the error will determine another run of, say, 30 identical nucleotides (because of the 3% error rate). We consider three kinds of errors. The case when a_{k+1} and/or b_{k+1} is determined incorrectly (i.e., a substitution error) is handled by adding 1 to both i and j then restarting the while-loop; existence of an extraneous character in A is handled by raising i by 1 and restarting the loop; an extra nucleotide in B is treated by incrementing j and restarting the while-loop.

We expect that one of the three while-loops will iterate perhaps 30 times, whereas the other two will terminate almost immediately. Suppose for the moment that we're lucky and this happens. In effect, we've

[1] Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802.

[2] National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD 20894.

explored three adjacent diagonals of the dynamic programming grid (also called the edit graph). The search is faster than dynamic programming for two reasons. First, many grid points are entirely skipped, since only one of the diagonals was explored beyond a couple of points. Second, each inspected grid point only involves raising two pointers and comparing the characters they point to. In contrast, the dynamic programming algorithm inspects every grid point in the band and does so with a three-way comparison using arithmetic involving scoring parameters.

The faster approach, which we call the *greedy algorithm*, has been generalized and studied extensively by computer scientists (Miller and Myers, 1985; Myers, 1986; Ukkonen, 1985; Wu *et al.*, 1990). It has also been adapted to build several practical programs for comparing DNA sequences (Chao *et al.*, 1997; Florea *et al.*, 1998). The main contribution of this paper is a theoretically sound method for pruning the search region.

Returning to the above scenario, where we explore three adjacent diagonals, we see that two types of events can confound the identification of which sequencing error occurred and/or of the subsequent run of identical nucleotides. First, it can happen that all three while-loops terminate quickly, because the next sequencing error appears after only a few basepairs. Second, two of the while-loops may iterate an appreciable number of times, because the search has reached a low complexity sequence region, such as a mono- or dinucleoide repeat. In addition to these difficulties, there is the question of how to determine when the end of the similar region has been reached.

The X-drop approach (Altschul *et al.*, 1990; Altschul *et al.*, 1997; Zhang *et al.*, 1998a,b) provides a rather natural solution to these problems. The width of the region being searched, i.e., the number of adjacent diagonals, expands at regions of low-sequence complexity or concentrated sequencing errors. In addition, the mechanism that limits the band width will automatically provide an appropriate termination condition.

We begin by formulating an X-drop alignment algorithm that uses dynamic programming. We then develop a greedy X-drop algorithm that is guaranteed to compute the same results as does the dynamic programming algorithm, provided that the alignment scores satisfy $ind = mis - mat/2$, where $ind < 0$, $mis < 0$ and $mat > 0$ are the scores for an insertion/deletion, a mismatch, and a match, respectively. (A gap-open penalty is not allowed.) After briefly describing an application of the greedy algorithm to align very similar bacterial genomic sequences, we close by sketching several generalizations of our methods to more flexible alignment scoring schemes. These generalizations are appropriate when comparing very similar sequences that differ through evolution, such as homologous sequences from a human and a chimpanzee.

2. AN X-DROP ALGORITHM

Consider the problem of aligning initial portions of the sequences A and B. In practice, we will frequently want the alignment to immediately follow positions in A and B that have been matched by some other means, but that situation is not appreciably different. Our goal is to find a highest-scoring alignment of sequences of the forms $a_1a_2\ldots a_i$ and $b_1b_2\ldots b_j$, for some $i \le M$ and $j \le N$ that are chosen to maximize the score. For given i and j, define $S(i, j)$ to be the score of the best such alignment, where we add $mat > 0$ for each column aligning identical nucleotides, $mis < 0$ for each column aligning differing nucleotides, and $ind < 0$ for each column in which a nucleotide is paired with the gap symbol. Thus $S(0, 0) = 0$ (since the alignment with zero columns scores 0) and if $i > 0$ or $j > 0$, then $S(i, j)$ satisfies the following recurrence relation:

$$S(i, j) = \max \begin{cases} S(i-1, j-1) + mat & \text{if } i > 0,\ j > 0 \text{ and } a_i = b_j \\ S(i-1, j-1) + mis & \text{if } i > 0,\ j > 0 \text{ and } a_i \ne b_j \\ S(i, j-1) + ind & \text{if } j > 0 \\ S(i-1, j) + ind & \text{if } i > 0. \end{cases}$$

The recurrence relation immediately provides algorithms for computing $S(i, j)$. Positions (i, j) need to be visited in some order such that all positions $(i-1, j-1)$, $(i, j-1)$ and $(i-1, j)$ used in the definition of $S(i, j)$ are visited before (i, j) is considered. It is helpful to think of S as being defined on a rectangular grid, where $S(i, j)$ is the value at the grid point with horizontal coordinate i and vertical coordinate j; see Figure 1.

Figure 2 presents such a "dynamic programming" method. To achieve a symmetric treatment of rows and columns, we fill in the S-values along antidiagonals, where antidiagonal k is the set of all points (i, j)

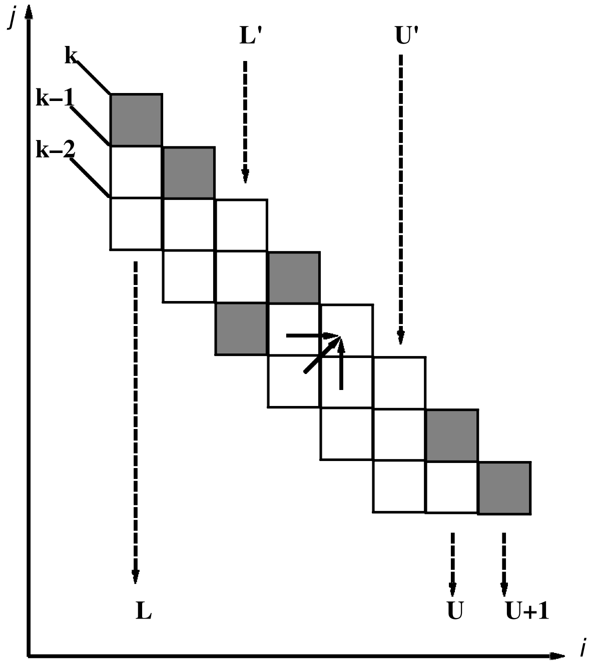

FIG. 1. Pruning of small S-values. The white boxes on antidiagonal $k-1$ designate points where $S(i,j) > -\infty$. $S(i,j)$ on antidiagonal k is computed for $i \in [L, U+1]$, after which entries whose scores are too small are reset to $-\infty$, and any such positions at the ends of the white boxes are pruned away giving L' and U'.

```
$T' \leftarrow T \leftarrow S(0,0) \leftarrow 0$
$k \leftarrow L \leftarrow U \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    for $i \leftarrow \lceil L \rceil$ to $\lfloor U \rfloor + 1$ in steps of $\frac{1}{2}$ do
        $j \leftarrow k - i$
        if $i$ is an integer then
            $S(i,j) \leftarrow \max \begin{cases} S(i-\frac{1}{2}, j-\frac{1}{2}) + mat/2 & \text{if } L \le i - \frac{1}{2} \le U \text{ and } a_i = b_j \\ S(i-\frac{1}{2}, j-\frac{1}{2}) + mis/2 & \text{if } L \le i - \frac{1}{2} \le U \text{ and } a_i \ne b_j \\ S(i, j-1) + ind & \text{if } i \le U \\ S(i-1, j) + ind & \text{if } L \le i - 1 \end{cases}$
        else
            $S(i,j) \leftarrow S(i-\frac{1}{2}, j-\frac{1}{2}) + \begin{cases} mat/2 & \text{if } a_{i+\frac{1}{2}} = b_{j+\frac{1}{2}} \\ mis/2 & \text{if } a_{i+\frac{1}{2}} \ne b_{j+\frac{1}{2}} \end{cases}$
        $T' \leftarrow \max\{T', S(i,j)\}$
        if $S(i,j) < T - X$ then $S(i,j) \leftarrow -\infty$
    $L \leftarrow \min\{i : S(i, k-i) > -\infty\}$
    $U \leftarrow \max\{i : S(i, k-i) > -\infty\}$
    $L \leftarrow \max\{L, k + 1 - N\}$
    $U \leftarrow \min\{U, M - 1\}$
    $T \leftarrow T'$
until $L > U + 1$
report $T'$
```

FIG. 2. A dynamic-programming X-drop algorithm.

such that $i + j = k$. We keep track of the best alignment score, denoted T, detected for a grid point lying on an antidiagonal before the current one. If we discover that $S(i, j) < T - X$, where $X \geq 0$ is user-specified, then we set $S(i, j)$ to $-\infty$ so as to guarantee that $S(i, j)$ will not play a role in subsequent evaluations of S. Informally, the rationale is that if the score of an optimal alignment of $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$ falls more than $-X$ below the best score seen so far, then we don't want to consider extensions of that alignment. For each antidiagonal, we determine the lower and upper bounds for the x-coordinate (horizontal position), denoted L and U, where finite values of S remain in that antidiagonal. Then we know a priori that any finite values in the next antidiagonal must lie at a position with X-coordinate between L and $U + 1$. The values in those positions are computed and tested for the "$T - X$" condition, and the new values of L and U are found by trimming points where $S = -\infty$ from the ends of the range L to $U + 1$. See Figure 1. Lines 15–16 of Figure 2 guarantee that the search will keep to legitimate points (i, j) where $i \leq M$ and $j \leq N$. (Lines 13–14 use the convention that $\min \emptyset = \infty$ and $\max \emptyset = -\infty$, where $\emptyset$ denotes the empty set.)

In a straightforward algorithm for antidiagonalwise computation, scores in antidiagonal k are computed from scores in antidiagonals $k-1$ and $k-2$. Our algorithm employs a device that reduces this dependence to just diagonal $k-1$. We add "half-nodes" at positions halfway along a diagonal line connecting $(i-1, j-1)$ to (i, j), where we assign a score that is halfway between $S(i - 1, j - 1)$ and $S(i, j)$ by adding half the cost of a match or mismatch. For the for-loop limits (line 5) we round positions of half-nodes to those of the appropriate regular node (i.e., upward for L and downward for U). For instance, suppose line 13 sets L to a half-value i, i.e., where $i = \lfloor i \rfloor + \frac{1}{2}$. Since $S(i, j)$ is used only to compute $S(i + \frac{1}{2}, j + \frac{1}{2})$, we want $S(i + \frac{1}{2}, j + \frac{1}{2})$ to be the first S-value found in the next loop iteration.

Note that the pruning rules of lines 13 and 14 do not change the set of (i, j) where a finite S-value is computed. Their role is to make the algorithm more efficient. Also note that after execution of lines 13–16 there may remain i between L and U where $S(i, k - i) = -\infty$, as indicated in Figure 1.

3. A GREEDY ALGORITHM

Greedy alignment algorithms work directly with a measurement of the difference between two sequences, rather than their similarity. In other words, near-identity of sequences is characterized by a small positive number instead of a large one. In the simplest approach, an alignment is assessed by counting the number of its differences, i.e., the number of columns that do not align identical nucleotides. The distance, $D(i, j)$, between the strings $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$ is then defined as the minimum number of differences in any alignments of those strings.

We will need the ability to translate back and forth between an alignment's score and the number of its differences, but this requirement constrains the alignment scoring parameters. Assume for the moment that any two alignments of $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$ having the same number of differences also have the same score, regardless of the particular sequence entries. Pick any such alignment that has at least two mismatch columns. One of those columns can be replaced by a deletion column followed by an insertion and, by changing one of the sequence entries, the other mismatch can be converted to a match. The transformation can be pictured as:

```
...A...G...      ...A-...G...
...C...T...      ...-C...G...
```

Here, dots mark alignment entries that are unchanged between the two alignments. This transformation leaves the number of differences, and hence the score, unchanged, from which it follows that $2 \times mis = 2 \times ind + mat$. In summary, the equivalence of score and distance implies that $ind = mis - mat/2$. As the following lemma shows (see also Smith *et al.*, 1981), that equality is sufficient to guarantee the desired equivalence and, furthermore, the formula to translate distance into score depends only on the antidiagonal containing the alignment end-point.

Lemma 1. *Suppose the alignment scoring parameters satisfy* $ind = mis - mat/2$. *Then any alignment of* $a_1a_2 \ldots a_i$ *and* $b_1b_2 \ldots b_j$ *with* d *differences has score* $S'(i + j, d) = (i + j) \times mat/2 - d \times (mat - mis)$.

Proof. Consider such an alignment with J matches, K mismatches and I indels, and let $k = i + j$. Then $k = 2J + 2K + I$, since each match or mismatch extends the alignment for two antidiagonals, while each indel extends it by one. The alignment has $d = K + I$ differences, while its similarity score is

$$\begin{aligned} mat \times J + mis \times K + ind \times I &= mat \times (k - 2K - I)/2 + mis \times K + (mis - mat/2) \times I \\ &= k \times mat/2 - d \times (mat - mis). \end{aligned}$$

■

For the remainder of this section, we assume that $ind = mis - mat/2$. Rather than determining values $S(i, j)$ in order of increasing antidiagonal, the values will be found in order of increasing $D(i, j)$. Note that Lemma 1 implies that minimizing the number of differences in an alignment is equivalent to maximizing its score (for a fixed point (i, j)), and that $S(i, j) = S'(i + j, D(i, j))$. In brief, knowing $D(i, j)$ immediately tells us $S(i, j)$.

The points where D equals 0 are just the (i, i) where $a_p = b_p$ for all $p \leq i$, since for other (i, j) an alignment of $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$ must contain at least one replacement or indel. Ignoring for the moment the X-drop condition, suppose we have determined all positions (i, j) where $D(i, j) = d - 1$. In particular, suppose we know the values $R(d - 1, k)$ giving the x-coordinate of the last position on diagonal k where the D-value is $d - 1$, for $-d < k < d$. (The kth diagonal consists of the points (i, j) where $i - j = k$.) Note that if $D(i, j) < d$, then (i, j) is on diagonal k with $|k| < d$.

Consider any (i, j) on diagonal k and where $D(i, j) = d$. Pick any alignment of $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$ having d differences. Imagine removing columns from the right end of the alignment that consist of a match. In terms of the grid, this moves us down diagonal k toward $(0, 0)$, through points with D-value d. The process stops when we hit a mismatch or indel column. Removing *that* column takes us to a point with D-value $d - 1$. We can reverse this process to determine the D-values in diagonal k or, to be more precise, to determine $R(d, k)$, the x-coordinate of the last position in diagonal k where $D(i, j) = d$. Namely, we find one position where $D(i, j) = d$ and $j = i - k$, as illustrated in Figure 3, then simultaneously increment i and j until $a_{i+1} \neq b_{j+1}$.

The correctness of the basic greedy alignment algorithm will not be discussed in further detail, because it is addressed by several previous publications (Miller and Myers, 1985; Myers, 1986; Ukkonen, 1985; Wu *et al.*, 1990). These earlier papers treat a slightly less general problem, in which mismatches are not considered (but rather replaced by insertion-deletion pairs); however, the difference is immaterial. On the other hand, the central step in a correctness proof of a more general algorithm is given below as Theorem 2.

What remains is to simulate line 12 of Figure 2. That is, we need a way to tell if $S(i, j) < T - X$, where T is the maximum $S(p, q)$ over all (p, q) where $p + q < i + j$. This was easy to do when

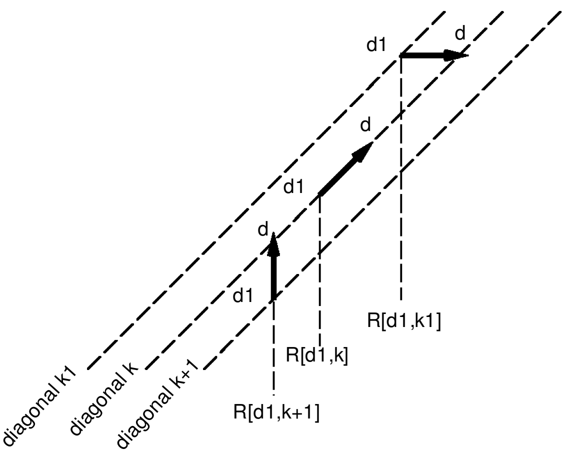

FIG. 3. Three cases for finding a point of distance d on diagonal k. The R-values give x-coordinates of the last positions on each diagonal with D-value $d - 1$. We can move right from a $d - 1$, or along a diagonal from a $d - 1$, or up from a $d - 1$, in order to find a point that can be reached with d differences. The furthest of these points along diagonal k must have D-value d. The first two moves raise the x-coordinate by 1. See line 10 of Figure 4.

the S-values were being determined in order of increasing antidiagonal, but now we are determining the S-values in a different order. It will be insufficient to simply keep the largest S-value determined so far in the computation, but we can get by with only a little more effort. (Our argument for this depends critically on our use of "half-nodes" in Figure 2.) Let "phase x" refer to the phase in the greedy algorithm that determines points where $D(i, j) = x$, and let $T[x]$ be the largest S-value determined during phases 0 through x.

Given the values $T[x]$ for $x < d$, and knowing that $D(i, j) = d$ (and hence that $S(i, j) = S'(i + j, d)$, where S' is as in Lemma 1), how can we determine the maximum S-value over all points on antidiagonals strictly before $i + j$? Note that there may well be points in those earlier antidiagonals whose S-value has not yet been determined. However, any such S-value will be smaller than $S(i, j)$, according to Lemma 1. In fact, any S-value on an earlier antidiagonal that exceeds $S(i, j)$ by more than X must have been determined a number of phases before phase d. The following result makes this precise.

Lemma 2. *Suppose $D(i, j) = d$, define*

$$d' = d - \left\lfloor \frac{X + mat/2}{mat - mis} \right\rfloor - 1$$

and let $T[d'] = \max\{S(p, q) : D(p, q) \le d'\}$. Then the following conditions are equivalent.

C1: $S(i, j) \ge T[d'] - X$.

C2: $S(i, j) \ge S(p, q) - X$ *for all* (p, q) *such that* $p + q < i + j$.

Here, (p, q) can be a "half-point," as used in Figure 2.

Proof. Our greedy algorithm treats only integer values of d. However, let us define $D(i + \frac{1}{2}, j + \frac{1}{2})$ to be $(D(i, j) + D(i + 1, j + 1))/2$, for all integers $i < M$ and $j < N$. Note that $T[d']$ is unchanged since d' is an integer. It is straightforward to verify that the formula for S in terms of D holds for half-points, i.e., $S(i + \frac{1}{2}, j + \frac{1}{2}) = (i + j + 1)mat/2 - D(i + \frac{1}{2}, j + \frac{1}{2})(mat - mis)$.

First we show that **C1** implies **C2** by assuming that **C2** is false and proving that **C1** is then false. Suppose $S(p, q) > S(i, j) + X$ for some (p, q) satisfying $p + q < i + j$. Then

$$\begin{aligned} D(p, q) &= [(p + q)mat/2 - S(p, q)]/(mat - mis) \\ &< [(p + q)mat/2 - S(i, j) - X]/(mat - mis) \\ &\le [(i + j)mat/2 - S(i, j) - X - mat/2]/(mat - mis) \\ &= D(i, j) - (X + mat/2)/(mat - mis). \end{aligned}$$

Without loss of generality, $D(p, q)$ is an integer, since otherwise (p, q) is a half-point on a mismatch edge, and $S(p - \frac{1}{2}, q - \frac{1}{2}) > S(p, q)$. Let V denote $(X + mat/2)/(mat - mis)$. Then $D(i, j) - D(p, q)$ is the first integer strictly larger than V (i.e., $\lfloor V \rfloor + 1$ or more). Hence $D(p, q) \le d'$. It follows that $S(i, j) + X < S(p, q) \le T[d']$.

For the converse, we assume that **C1** is false and prove that **C2** is then false. Suppose that $S(i, j) < T[d'] - X$. By definition, $T[d']$ equals some $S(p, q)$ where $D(p, q) \le d'$. If $p + q < i + j$, then we've found a witness that **C2** is false. But what if (p, q)'s antidiagonal is not smaller than (i, j)'s? Then pick an alignment ending at (p, q) and having at most d' differences. Consider the initial portion of that alignment ending on antidiagonal $i + j - 1$, say at (p', q'). (This is where half-points may be necessary; without half–points, the antidiagonal index would increase by 2 along match and mismatch edges.) That prefix has at most d' differences, and hence

$$\begin{aligned} S(p', q') - S(i, j) &= (p' + q')mat/2 - d'(mat - mis) \\ &\quad - (i + j)mat/2 + d(mat - mis) \\ &= -mat/2 + (d - d')(mat - mis) \\ &> X. \end{aligned}$$

■

In summary, comparing $S(i, j)$ with $T[d'] - X$ immediately tells us if $S(i, j)$ should be pruned. Note that this test need only be performed when we first find a point (i, j) on diagonal k and where $D(i, j) = d$, since if that point survives, then subsequent points on that diagonal with D-value d have their S-value increasing as fast as possible and hence won't be pruned. To be precise, suppose $S(i, j) \geq S(p, q) - X$ for all (p, q) satisfying $p + q < i + j$, and let $D(i + t, j + t) = D(i, j)$ for some $t > 0$. Then $S(i + t, j + t) = S(i, j) + t \times mat \geq S(p, q) - X + t \times mat \geq S(p + t, q + t) - X$ for all $(p + t, q + t)$ such that $p + q + 2t < i + j + 2t$.

Theorem 1. *Assume an alignment-scoring scheme in which $ind = mis - mat/2$, where $mat > 0$, $mis < 0$ and $ind < 0$ are the scores for a match, mismatch and insertion/deletion, respectively. The algorithm of Figure 4, with S' as defined in Lemma 1, always computes the same optimal alignment score as does the algorithm of Figure 2.*

Proof. First note that the previous discussion, including Lemma 2, implies the equivalence of the algorithms if no pruning is performed (lines 13–16 of Figure 2 and lines 19–22 of Figure 4). But pruning does not affect the computed result. In particular, consider lines 21–22 of Figure 4, which handle situations when the grid boundary is reached. If extension down diagonal k reaches $i = M$, then there is no need to consider diagonals larger than i, since positions searched at a later phase (larger d) will have smaller antidiagonal index, and hence smaller score. Thus we set $U \leftarrow k - 2$, so that when diagonals with index at most $U + 1$ are considered in the next phase, unnecessary work will be avoided. L is handled similarly. ■

Let d_{max} and $L \leq M + N$ denote the largest values of d and of $i + j$ attained when the algorithm of Figure 4 is executed. Then the algorithm's worst-case running time is $O(d_{max}L)$. The distinction between this algorithm and earlier greedy alignment methods is that another worst-case bound for the current method is $O(\sum_{d \leq d_{max}} U_d - L_d)$, where L_d and U_d denote the lower and upper bounds of $\{k : R(d, k) > -\infty\}$.

```
1.  $i \leftarrow 0$
2.  while $i < \min\{M, N\}$ and $a_{i+1} = b_{i+1}$ do $i \leftarrow i + 1$
3.  $R(0, 0) \leftarrow i$
4.  $T' \leftarrow T[0] \leftarrow S'(i + i, 0)$
5.  $d \leftarrow L \leftarrow U \leftarrow 0$
6.  repeat
7.      $d \leftarrow d + 1$
8.      $d' \leftarrow d - \left\lfloor \frac{X + mat/2}{mat - mis} \right\rfloor - 1$
9.      for $k \leftarrow L - 1$ to $U + 1$ do
10.         $i \leftarrow \max \begin{cases} R(d-1, k-1) + 1 & \text{if } L < k \\ R(d-1, k) + 1 & \text{if } L \leq k \leq U \\ R(d-1, k+1) & \text{if } k < U \end{cases}$
11.         $j \leftarrow i - k$
12.         if $i > -\infty$ and $S'(i + j, d) \geq T[d'] - X$ then
13.             while $i < M$, $j < N$ and $a_{i+1} = b_{j+1}$ do
14.                 $i \leftarrow i + 1$; $j \leftarrow j + 1$
15.             $R(d, k) \leftarrow i$
16.             $T' \leftarrow \max\{T', S'(i + j, d)\}$
17.         else $R(d, k) \leftarrow -\infty$
18.     $T[d] \leftarrow T'$
19.     $L \leftarrow \min\{k : R(d, k) > -\infty\}$
20.     $U \leftarrow \max\{k : R(d, k) > -\infty\}$
21.     $L \leftarrow \max\{L, \max\{k : R(d, k) = N + k\} + 2\}$
22.     $U \leftarrow \min\{U, \min\{k : R(d, k) = M\} - 2\}$
23. until $L > U + 2$
24. report $T'$
```

FIG. 4. Greedy algorithm that is equivalent to the algorithm of Figure 2 if $ind = mis - mat/2$.

```
procedure $script(d, k)$
    if $d > 0$ then
        $i \leftarrow \max\{R(d-1, k-1)+1, R(d-1, k)+1, R(d-1, k+1)\}$
        if $i = R(d-1, k-1)+1$ then
            $script(d-1, k-1)$
            print "delete $a_i$"
        else if $i = R(d-1, k)+1$ then
            $script(d-1, k)$
            print "replace $a_i$ by $b_{i-k}$"
        else
            $script(d-1, k+1)$
            print "insert $b_{i-k}$"
```

FIG. 5. An algorithm to determine an optimal edit script.

A more subtle analysis would show that under certain probabilistic assumptions, the expected running time is $O(d_{max}^2 + L)$ (Myers, 1986).

3.1. Determining the alignment

For some applications, a score-only alignment algorithm is adequate. That is, only the optimal score or number of differences needs to be reported, as in Figures 2 and 4. On the other hand, if certain intermediate values in Figure 4 are retained, then one can construct a minimum-sized set of replacement and indel operations that converts sequence A to B. Such a set is called an *optimal edit script*, and can readily be translated to a maximum-score alignment if the scoring scheme satisfies the condition of Lemma 1.

One approach to determining an optimal edit script is as follows. For each $d \geq 0$, again let L_d and U_d denote the lower and upper bounds of $\{k : R(d, k) > -\infty\}$. We store all $R(d, k)$ for $L_d - 2 \leq k \leq U_d + 2$; keeping a couple of $-\infty$ values on either end lets us avoid storing the L and U and dealing with them in the "trace-back" step. We also need to retain d_{best} and k_{best}, the values of variables d and k when the maximum value of T' is first assigned (line 16). Then execution of $script(d_{best}, k_{best})$ produces the desired result, where *script* is as in Figure 5.

Note that this approach requires $O(d_{max}C)$ space, where d_{max} is the value of d during the last iteration of the main loop in Figure 2, and C is the average value of $U_d - L_d$ for $0 \leq d \leq d_{max}$. It is straightforward to see that $C < d_{max}$. In cases where this space requirement causes problems, use of a more sophisticated algorithm (Myers, 1986) can reduce the worst-case space to $O(M + N)$, where M and N are the sequence lengths. In any case, the worst-case time required to execute $script(d_{best}, k_{best})$ is $O(M + N)$.

4. AN EXAMPLE

Implementations of the greedy algorithm have been incorporated into several alignment tools that are under development for use at the National Center for Biotechnology Information. Those tools and their intended uses will be described in a later publication. Here we mention a brief experiment that we performed, using a Blast-family tool that permits optional use of the greedy algorithm.

We used that program to align the genomic sequence of *Mycobacterium tuberculosis*, strain H37Rv (Cole *et al.*, 1998), with the sequence being generated at The Institute for Genomic Research from another *M. tuberculosis* strain that is 99% identical. A need to compare these sequences motivated others (Delcher *et al.*, 1999) to develop an alignment program capable of handling complete bacterial genomes. The H37Rv sequence is 4,441,529 nucleotides, while the other sequence is roughly the same length, though at the time we obtained the sequence (March 3, 1999), it consisted of 42 contigs. For one test we extracted the largest of those contigs (namely the reverse complement of contig 3732, which was 466,170 nucleotides). Part of the contig aligns with the start of the H37Rv, and part with the end (the genomes are circular, so the starting point for the sequence record is largely arbitrary).

In typical Blast fashion (Altschul *et al.*, 1990), the program begins by making a table of all 12-mers in the H37Rv sequence; this took about 15 seconds on our Sun Ultra-30 workstation (296 MHz). Then the TIGR contig was scanned, and matching 12-mers were inspected to see if they were included in a 30-bp exact match; this took an additional 45 seconds. Finally, an X-drop algorithm was applied to knit the 30-bp matches together into long gapped alignments. With both approaches (dynamic programming and greedy), this is done by recursively picking a longest exact match that does not intersect one of the earlier gapped alignments (initially the empty set) and extending in both directions until the X-drop condition terminates the extension.

With small values of X, say $2 \times ind$, the greedy algorithm ran in about a second, while the dynamic-programming algorithm took 15 seconds. The ratio, i.e., with the greedy algorithm 15 times faster than the dynamic programming algorithm, is typical in our experience; a thorough comparative analysis will appear later. When the same program was applied to align all 42 contigs (in both directions), it took 15 minutes to complete the task.

5. GENERALIZATIONS

For this development we have assumed that it is appropriate to penalize an indel about the same as, or slightly more than, a replacement. This seems consistent with published figures on the rates of actual errors in both single-pass (low accuracy) sequences (Ewing *et al.*, 1998; Hillier *et al.*, 1996) and high quality data.

On the other hand, it is widely appreciated that dynamic-programming alignment algorithms can guarantee a theoretically optimal alignment under a wide variety of scoring schemes. For instance, affine gap penalties (a gap of length k is penalized $\alpha + \beta k$ for fixed α and β) are required to accurately model sequence differences arising from evolution, and symbol-dependent replacement scores are needed for accurate protein alignments. More generally, position-dependent substitution and indel scores are possible and are frequently used in bioinformatics. All of these generalizations can be handled by appropriate modifications to Figure 2. Further generalizations are possible (e.g., to more general gap penalties (Miller and Myers, 1988)), but have yet to prove useful in practice.

Our discussion of the greedy algorithm has considered three operations, *viz.*, substitution, insertion and deletion of single nucleotides. Each operation was in essence assigned cost 1 when we chose to minimize the number of differences. Additional kinds of operations and/or more general costs can be contemplated, with the goal of developing an algorithm that finds a set of operations converting one given sequence to another at minimum total cost. When all costs are small nonnegative integers, it may be possible to develop a greedy-style algorithm that, at least in theory, is efficient if the two sequences are very similar. Myers and Miller (1989b) give results of this sort covering affine gap costs, as well as certain additional operations that appear inappropriate for bioinformatics applications. Symbol-dependent mismatch costs can also be handled, though we know of no published description, perhaps because the practical efficiency of greedy alignment algorithms degrades very quickly as generality is increased.

Here we briefly explore generalizations of the above results to wider classes of operations and costs. That is, assuming that the dynamic programming algorithm of Figure 2 is modified for a more general alignment-scoring scheme, when can an equivalent greedy algorithm be developed? In particular, what about more general scores for mismatches and gaps? We factor this into two extensions of our basic result and leave it to the industrious reader to combine those extensions. We close by mentioning alignment-scoring schemes that appear to defy the methods presented in this paper.

5.1. Arbitrary match, mismatch and indel scores

The dynamic-programming algorithm of Figure 2 works for arbitrary $mat > 0$, $mis < 0$ and $ind < 0$, whereas Figure 4 is equivalent to Figure 2 only if $ind = mis - mat/2$. However, with a more complex greedy algorithm, this constraint can be avoided. Moreover, the greedy method can handle cases where mismatches fall into classes that are scored differently (with DNA sequences this might be used to distinguish transitions $A \leftrightarrow G$ and $C \leftrightarrow T$ from transversions) and where insertions and deletions are treated separately (for instance when aligning a genomic sequence and a spliced gene sequence).

Suppose that a mismatch is either of type *MIS1* or type *MIS2* and that alignments are scored using integer constants $mat > 0$, $mis1 < 0$, $mis2 < 0$, $ins < 0$ and $del < 0$ for the five kinds of columns. Without loss of generality, mat is even, since all five values can be doubled without materially affecting the problem. Define $g = gcd(mat - mis1, mat - mis2, mat/2 - ins, mat/2 - del)$, and define $mis1' = (mat - mis1)/g$, $mis2' = (mat - mis2)/g$, $ins' = (mat/2 - ins)/g$ and $del' = (mat/2 - del)/g$. We define the *weighted difference cost* of an alignment with $M1$ mismatches of type *MIS1*, $M2$ mismatches of type *MIS2*, and

I insertions and D deletions to be $M1 \times mis1' + M2 \times mis2' + I \times ins' + D \times del'$. Mirroring Lemma 1, it is straightforward to show that any alignment of $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$ of weighted difference cost d has score $S''(i+j, d) = (i+j) \times mat/2 - d \times g$. Moreover, Lemma 2 carries over with d' replaced by $d'' = d - \lfloor (X + mat/2)/g \rfloor - 1$.

The next ingredient is a greedy algorithm for aligning under the weighted difference cost. Let $W(i, j)$ denote the minimum weighted distance cost of any alignment of $a_ia_2 \dots a_i$ and $b_1b_2 \dots b_j$. Theorem 2 is essentially the induction step for a correctness proof of the appropriate greedy alignment algorithm.

Theorem 2. *Fix $d > 0$ and k. Suppose for $c < d$ and for $t \in \{k-1, k, k+1\}$ that $Q(c, t) = \max\{i : W(i, i-t) \le c\}$, i.e., the x-coordinate of the last position on diagonal t having W-value at most c; we let $Q(c, t) = -\infty$ if $W(i, i-t) > c$ for all relevant i. Assume that $Q(d - del', k-1) < M$ and $Q(d - ins', k+1) \le N + k$. Let i be computed as follows.*

```
$i \leftarrow \max \begin{cases} Q(d - del', k-1) + 1 & \\ Q(d - mis1', k) + 1 & \text{if } (a_p, b_{p-k}) \in MIS1 \text{ for } p = Q(d - mis1', k) + 1 \\ Q(d - mis2', k) + 1 & \text{if } (a_p, b_{p-k}) \in MIS2 \text{ for } p = Q(d - mis2', k) + 1 \\ Q(d - ins', k+1) & \end{cases}$
if $Q(d-1, k) < i \le \min\{M, N+k\}$ then
    while $i < \min\{M, N+k\}$ and $a_{i+1} = b_{i+1-k}$ do
        $i \leftarrow i + 1$
else
    $i \leftarrow Q(d-1, k)$
```

The resulting value of i is $\max\{i : W(i, i-k) \le d\}$.

Proof. We first show that $W(i, i-k) \le d$. The contributions to the four-way maximization give the x-coordinate of the points reached by (1) adding a deletion column to an alignment of cost at most $d - del'$ ending on diagonal $k-1$, (2) adding a mismatch column of type *MIS1* to an alignment of cost at most $d - mis1'$ ending on diagonal k (3) adding a mismatch column of type *MIS2* to an alignment of cost at most $d - mis2'$ ending on diagonal k and (4) adding an insertion column to an alignment of cost at most $d - ins'$ ending on diagonal $k+1$. Thus the value i assigned by the *max* operation satisfies $W(i, i-k) \le d$. Progressing along the diagonal with the while-loop corresponds to adding match columns to the end of the alignment, so the final value of i satisfies $W(i, i-k) \le d$.

What remains is to show that any point $(p, p-k)$ on diagonal k where $W(p, p-k) \le d$ must satisfy $p \le i$. Pick an alignment of weighted cost at most d ending at $(p, p-k)$. Remove any match columns from the right end, until a mismatch or indel column is reached. First suppose *that* column is an insertion. Removing that column produces an alignment of cost at most $d - ins'$ ending on diagonal $k+1$, so the x-coordinate of the point reached is at most $Q(d - ins', k+1)$, which cannot exceed the value assigned to i by the four-way *max* operation. It follows readily that $p \le i$.

Another case is when the last nonmatching column is a mismatch of type *MIS1*. Removing that column gives an alignment of cost at most $d - mis1'$ ending at, say, $(x, x-k)$. Let $y = Q(d - mis1', k)$, whence $x \le y$. If $x = y$, then x is at most the value of the four-way *max*. Otherwise, $p \le y \le Q(d-1, k)$, since $a_{y+1} \ne b_{y-k+1}$. Thus, p cannot exceed the final value of i, which is at least $Q(d-1, k)$.

The other cases, namely a deletion column and the other kind of mismatch, are handled similarly. ■

The reader who has worked carefully through the development of the greedy algorithm for number-of-differences alignment should be able to fill in the details of a greedy algorithm for X-drop alignment using this more general alignment scoring scheme. The recurrence of Theorem 2 can be evaluated in time that is independent of the number of kinds of mismatches; verification of this observation is left to the reader.

5.2. Affine gap penalties

Suppose that an alignment with J matches, K mismatches, and G gaps of total length I is given score $J \times mat + K \times mis + I \times ind + G\alpha$; $\alpha < 0$ is frequently called the "gap-open penalty." A highest-scoring alignment of sequences A and B can be computed in time proportional to the product of the sequence lengths (Gotoh, 1982) by a dynamic programming algorithm that is equivalent to finding an optimal path in a graph that has three vertices per grid-point (i, j) (Myers and Miller, 1989a).

Let *mat*, *mis*, *ind* and α be integers, with *mat* even. Define $g = gcd(mat - mis, mat/2 - ind, -\alpha)$, $mis' = (mat - mis)/g$, $ind' = (mat/2 - ind)/g$ and $\alpha' = -\alpha/g$. The cost assigned to an alignment with K mismatches and G gaps of total length I is $d = K \times mis' + I \times ind' + G \times \alpha'$. Again, the alignment's score equals $S''(i + j, d) = (i + j) \times mat/2 - d \times g$, and setting $d'' = d - \lfloor (X + mat/2)/g \rfloor - 1$ gives the result analogous to Lemma 2. These observations, coupled with a greedy alignment algorithm for affine gap costs (Myers and Miller, 1989b), give a greedy algorithm equivalent to a dynamic-programming X-drop algorithm with affine gap penalties.

5.3. Symbol-dependent match scores

Symbol-dependent match scores are needed for accurate alignment of protein sequences, and the dynamic programming algorithm can deal with them easily. However, they present difficulties for the approach discussed in this paper, as we now sketch.

Consider an alignment of $a_1a_2 \ldots a_i$ and $b_1b_2 \ldots b_j$. Let $\sigma(a, b)$ denote the score awarded whenever a is aligned with b (a can be identical to b or distinct) and let *ind* be the indel score. Also, let *MAT* and *MIS* denote the set of matching and mismatching aligned pairs. Then the alignment's score is:

$$S(i, j) = \sum_{(a,b) \in MAT} \sigma(a, b) + \sum_{(a,b) \in MIS} \sigma(a, b) + I \cdot ind.$$

Give each mismatch (a, b) the cost $\sigma'(a, b) = (\sigma(a, a) + \sigma(b, b))/2 - \sigma(a, b)$, give an indel of c (where $c = a_p$ or $c = b_q$) the cost $\sigma'(c) = \sigma(c, c)/2 - ind$, and define the alignment's cost as:

$$D(i, j) = \sum_{(a,b) \in MIS} \sigma'(a, b) + \sum_{(-,c),(c,-) \in INDEL} \sigma'(c).$$

Then $S(i, j) = (\sum_{p=1}^{i} \sigma(a_p, a_p) + \sum_{p=1}^{j} \sigma(b_p, b_p))/2 - D(i, j)$. This observation reduces the score-maximizing problem to a cost-minimizing problem, which can be solved by dynamic programming.

At that point, the analogy with the above development for other scoring schemes breaks down. The formula for $S(i, j)$ in terms of $D(i, j)$ is not constant along each antidiagonal, spelling trouble for any attempt to generalize Lemma 2. More important, the existence of symbol-dependent indel costs seems at odds with development of a greedy strategy, since the loop

```
while $i < M$ and $j < N$ and $a_{i+1} = b_{j+1}$ do
    $i \leftarrow i + 1; j \leftarrow j + 1$
```

ignores the possibility of deleting or inserting a matched character under the assumption that a later deletion or insertion will work as well. We leave development of an efficient alignment algorithm for symbol-dependent match scores as an open problem.

ACKNOWLEDGMENTS

Z.Z., S.S. and W.M. were supported by grant LM05110 from the National Library of Medicine. We thank David Lipman for contributions to many aspects of this project and the Institute for Genomic Research for permitting access to unpublished sequence data.

REFERENCES

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

Altschul, S., Madden, T., Schäffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. 1997. Gapped BLAST and PSI-BLAST—a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402.

Chao, K.-M., Zhang, J., Ostell, J., Miller, W. 1997. A tool for aligning very similar DNA sequences. *CABIOS* 13, 75–80.

Cole, S.T., Brosch, R., Parkhill, J., Garnier, T., *et al.* 1998. Deciphering the biology of *mycobacterium tuberculosis* from the complete genome sequence. *Nature* 393, 537–544.

Delcher, A.L., S. Kasif, Fleishman, R.D., Peterson, J., White, O., and Salzberg, S.L. 1999. Global alignments of whole genomes. *Nucleic Acids Res.* 27, 2369–2376.

Ewing, B., Hillier, L., Wendl, M., and Green, P. 1998. Base-calling of automated sequencer traces using phred I. accuracy assessment. *Genome Research* 8, 175–185.

Florea, L., Hartzell, G., Zhang, Z., Rubin, G.M., and Miller, W. 1998. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Research* 8, 967–974.

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

Hillier, L., Lennon, G., Becker, M., Dietrich, N., DuBurque *et al.* 1996. Generation and analysis of 280,000 human expressed sequence tags. *Genome Research* 6, 807–828.

Miller, W., and Myers, E.W. 1985. A file comparison program. *Software—Practice and Experience* 15, 1025–1040.

Miller, W., and Myers, E.W. 1988. Sequence comparison with concave weighting function. *Bull. Math. Biol.* 50, 97–120.

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

Myers, E., and Miller, W. 1989a. Approximate matching of regular expressions. *Bull. Math. Biol.* 51, 3–37.

Myers, E., and Miller, W. 1989b. Row replacement algorithms for screen editors. *ACM Trans. Prog. Lang. and Sys.* 11, 33–56.

Smith, T.F., Waterman, M.S., and Fitch, W.M. 1981. Comparative biosequence metrics. *J. Mol. Biol.* 18, 38–46.

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

Wu, S., Manber, U., Myers, E., and Miller, W. 1990. An $O(NP)$ sequence comparison algorithm. *Information Processing Letters* 35, 317–323.

Zhang, Z., Berman, P., and Miller, W. 1998a. Alignments without low-scoring regions. *J. Comput. Biol.* 5, 197–210.

Zhang, Z., Schaffer, A.A., Miller, W., Madden, T.L., Lipman, D.J., Koonin, E.V., and Altschul, S.F. 1998b. Protein sequence similarity searches using patterns as seeds. *Nucleic Acids Res.* 26, 3986–3990.

Address correspondence to:
Webb Miller
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802

E-mail: webb@cse.psu.edu