A Greedy Algorithm for Aligning DNA Sequences

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. This may be the case when aligning an EST or mRNA with a larger genomic sequence. 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.

$$\begin{array}{ll} \text{while} & i < M \text{ and } j < N \text{ and } a_{i+1} = b_{j+1} \text{ do} \\ & i \leftarrow i+1; j \leftarrow j+1 \end{array}$$

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 re-starting 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} was 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 explored three adjacent diagonals of the dynamic-programming grid (also called the alignment 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). Here we describe the approach in detail, and show that for a simple class of alignment scores, the greedy algorithm is guaranteed to find a highest scoring alignment of the two given sequences. This observation provides a rationale for stopping the process of extending the alignment: when the alignment scores decrease more than a certain threshold, the alignment-extension process can be terminated.

The 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 (mismatches and gaps). 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. In terms of the graph that represents all alignments between sequences $a_1a_2\ldots a_M$ and $b_1b_2\ldots b_N$, we give each vertical and horizontal edge cost 1, and give the diagonal edge into vertex (i,j) cost 0 if $a_i = b_j$ and cost 1 otherwise. Using these edge costs, we want to find a path from $(0,0)$ to (M,N) of *lowest* total cost. (We say "cost" instead of "score" to reflect this difference.)

For each vertex (i,j), let $D(i,j)$ be the lowest cost over all paths from $(0,0)$ to that vertex. As is the case for any set of edge costs or scores, optimal paths in this graph can be found in a row-by-row manner using dynamic programming. However, the key idea of the greedy algorithm is to fill in the D-numbers in order of value. That is, we'll find all the vertices where D is 0, then all those where D is 1, and so on until we reach (M,N). 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 (insertion/deletion). In the following picture, which shows D-values at vertices of the alignment graph for `AAGCAAA` and `AGCTACA`, a dot denotes a value that is larger than those determined so far.

		A	G	C	T	A	C	A
	0	.	.	.	.	.	.	.
A	.	0	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.
C	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.

For $-M \leq k \leq N$, let *diagonal* k denote the set of grid points (i,j) such that $j - i = k$. Thus, all points with D-value 0 appear on diagonal 0, i.e., the main diagonal. Diagonal -1 is immediately below diagonal 0, and diagonal 1 is just above it.

Which positions have D-value 1? Note that any point with D-value 1 must appear in diagonal -1, 0, or 1, since reaching a point on another diagonal requires using at least 2 horizontal or vertical edges, each of which costs 1. Also, diagonal -1 and 1 have a point with D-value 1, since $D(0,1) = D(1,0) = 1$ (assuming that M and N are not 0). Similarly, diagonal 0 has such a point unless the two sequences are identical – it is the first point on diagonal 0 where the sequences contain a different letter. A little thought shows that in our example the 1's occur as follows:

		A	G	C	T	A	C	A
	0	1	.	.	.	.	.	.
A	1	0	1	.	.	.	.	.
A	.	1	1	.	.	.	.	.
G	.	.	1	.	.	.	.	.
C	.	.	.	1	.	.	.	.
A	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.
A	.	.	.	.	.	.	.	.

Similarly, the positions where $D(i, j) = 2$ are these:

		A	G	C	T	A	C	A
	0	1	2	.	.	.	.	.
A	1	0	1	2	.	.	.	.
A	2	1	1	2	.	.	.	.
G	.	2	1	2	.	.	.	.
C	.	.	2	1	2	.	.	.
A	.	.	.	2	2	2	.	.
A	.	.	.	.	.	2	.	.
A	.	.	.	.	.	.	.	.

To formulate an algorithm, we will not work directly with the D-matrix. Instead, we will compute the values $R(d, k)$, defined as the row index of the last position in diagonal k with D-value d, where the rows are numbered starting with 0. Thus, in the above example, $R(1, -1) = 4$, $R(1, 0) = 2$ and $R(1, 1) = 1$.

Fix $d > 0$, and suppose we have determined $R(d-1, k)$ for all k such that diagonal k contains a point with D-value $d - 1$. If diagonal k contains a D-value d (i.e., if $|k| \leq d$), then we think of the process of finding $R(d, k)$ as two steps: first finding *some* d on diagonal k, and then sliding down the diagonal along edges of cost 0 (i.e., matching sequence entries) to the last d (row $R(d, k)$).

Since each edge in the graph costs 0 or 1, moving right from a D-value of $d - 1$, along a diagonal from a $d - 1$, or down from a $d - 1$, leads to a grid point with D-value at most d. Moving right from the last $d-1$ in diagonal $k-1$ stays in row $R(d-1, k-1)$, whereas a diagonal move from the last $d - 1$ in diagonal k reaches row $R(d - 1, k) + 1$, and moving down from the last $d - 1$ in diagonal $k + 1$ reaches row $R(d - 1, k + 1) + 1$. Since the second of these positions in diagonal k cannot have D-value strictly less than d (why?), the D-value where diagonal k intersects row $\max\{R(d - 1, k - 1), R(d - 1, k) + 1, R(d - 1, k + 1) + 1\}$ must be d. Given this point on diagonal k, we then follow diagonal edges of cost 0 (if any exist) to determine $R(d, k)$.

Consider locating the last D-value of 2 on diagonal 0 in the above example. The move landing as far as possible down the diagonal from a 1 is a horizontal move from diagonal -1 along row 4. After that move is taken, one 0-cost edge can be followed, giving $R(2, 0) = 5$.

The algorithm of Figure 1 formalizes these steps. Line 11 finds the best move that can be made from a D-value of $d - 1$ on diagonals $k - 1$, k, or $k + 1$ to diagonal k, being careful when k is $-d$, $1 - d$, $d - 1$ and d. Lines 13 and 14 "slide down the diagonal" along edges of cost 0. The reader is invited to figure out how lines 16 and 17 deal with the bottom and right-hand boundaries of the grid. (*Hint:* L and U give the lower and upper bounds on the diagonals that can have a D-value of d.)

How do we know that the procedure correctly finds $R(d, k)$? We can suppose that it works correctly up to D-value $d - 1$. Consider any (i, j) on diagonal k and where $D(i, j) = d$. Pick any path from $(0, 0)$ to (i, j) having d differences (edges of cost 1). Imagine removing 0-cost edges from the end of the path. This moves us up diagonal k toward $(0, 0)$, through points with D-value d. The process stops when we hit a horizontal, vertical, or mismatch edge. Removing *that* edge 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 $R(d, k)$. Namely, we find one position where $D(i, j) = d$ and $j = i - k$, then simultaneously increment i and j until $a_{i+1} \neq b_{j+1}$.

$i \leftarrow 0$
while $i < \min\{M, N\}$ and $a_{i+1} = b_{i+1}$ do
    $i \leftarrow i + 1$
$R(0, 0) \leftarrow i$
$d \leftarrow L \leftarrow U \leftarrow 0$
repeat
    $d \leftarrow d + 1$
    $L \leftarrow L - 1$
    $U \leftarrow U + 1$
    for $k \leftarrow L$ to U do

$$i \leftarrow \max \begin{cases} R(d-1, k-1) & \text{if } L+1 < k \\ R(d-1, k)+1 & \text{if } L < k < U \\ R(d-1, k+1)+1 & \text{if } k < U-1 \end{cases}$$

        $j \leftarrow k - i$
        while $i < M$, $j < N$ and $a_{i+1} = b_{j+1}$ do
            $i \leftarrow i + 1$; $j \leftarrow j + 1$
        $R(d, k) \leftarrow i$
        if $i = M$ then $L \leftarrow k + 2$
        if $j = N$ then $U \leftarrow k - 2$
until $R(d, N - M) = M$
write "Number of differences is" d

Figure 1: Greedy algorithm for sequence alignment.

The greedy algorithm finds optimal alignments. We want to show that the greedy algorithm, though it attempts to minimize the number of differences (mismatches and gaps) in an alignment, can be thought of as maximizing the alignment score, provided that the score satisfies certain conditions. To do this, it is very helpful to have a formula that lets us translate back and forth between an alignment's score and the number of its differences. Let the alignment scoring scheme assign values *mat*, *mis* and *ind* to a match, mismatch, or insertion/deletion, respectively. We want to determine conditions on these three values under which the alignment score is closely related to the number of differences.

Assume for the moment that any two alignments of $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$ having the same number of differences also have the same score, regardless of the particular sequence entries, where i and j are fixed. 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:

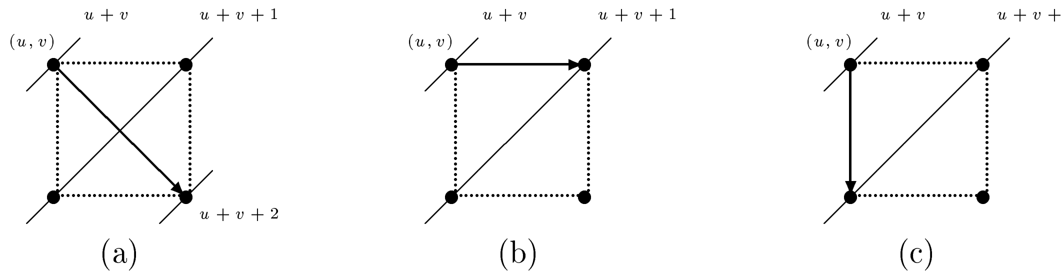
```
...A...G...     ...A-...G...
...C...T...     ...-C...G...
```

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 anti-diagonal containing the alignment end-point. (Point

(i, j) is in anti-diagonal $i + j$.)

Lemma 1. *Suppose the alignment scoring parameters satisfy* $ind = mis - mat/2$. *Then any alignment of* $a_1 a_2 \ldots a_i$ *and* $b_1 b_2 \ldots b_j$ *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 anti-diagonals, while each indel extends it by one. For instance, a substitution from a vertex (u, v) in the grid extends the alignment for the two anti-diagonals $u+v+1$ and $u+v+2$ (figure (a) below), while an indel from the same vertex extends it by the anti-diagonal $u+v+1$ only (figures (b) and (c)).



The alignment has $d = K + I$ differences, while its similarity score is

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

□

It follows that if $ind = mis - mat/2$, then minimizing the number of differences in an alignment is equivalent to maximizing its score, since $S(M, N) = S'(M + N, D(M, N))$, with S' defined as in the Lemma. In brief, knowing $D(i, j)$ immediately tells us $S(i, j)$.

1 References

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

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.

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

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

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. 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.