Bulletin of Mathematical Biology Vol. 48, No. 5/6, pp. 603–616, 1986.
Printed in Great Britain.

0092-8240/86$3.00 + 0.00
Pergamon Journals Ltd.

OPTIMAL SEQUENCE ALIGNMENT USING AFFINE GAP COSTS

■ STEPHEN F. ALTSCHUL*† and BRUCE W. ERICKSON*
*The Rockefeller University,
New York, NY 10021, U.S.A.
†Department of Applied Mathematics,
Massachusetts Institute of Technology,
Cambridge, MA 02139, U.S.A.

When comparing two biological sequences, it is often desirable for a gap to be assigned a cost not directly proportional to its length. If affine gap costs are employed, in other words if opening a gap costs v and each null in the gap costs u, the algorithm of Gotoh (1982, *J. molec. Biol.* **162**, 705) finds the minimum cost of aligning two sequences in order MN steps. Gotoh's algorithm attempts to find only one from among possibly many optimal (minimum-cost) alignments, but does not always succeed. This paper provides an example for which this part of Gotoh's algorithm fails and describes an algorithm that finds all and only the optimal alignments. This modification of Gotoh's algorithm still requires order MN steps. A more precise form of path graph than previously used is needed to represent accurately all optimal alignments for affine gap costs.

Introduction. The first widely used algorithm for aligning two biological sequences was developed by Needleman and Wunsch (1970). Sellers (1974) introduced mathematical rigor by finding optimal (minimum-cost) alignments, where substitution of element x by y is assessed a *substitution cost* $c(x, y)$, and aligning an element with a null (missing element) is charged a *null cost* w_1. As is frequently done when nucleic acid sequences are compared, in this paper $c(x, y)$ is chosen to be 0 if $x = y$ and 1 if $x \neq y$. For protein sequence comparison, other sets of substitution costs have proven useful (Schwartz and Dayhoff, 1978; Erickson and Sellers, 1983). The Sellers SS algorithm for finding all optimal alignments (Sellers, 1974) requires $O(MN)$ (on the order of MN) computational steps, where M and N are the lengths of the aligned sequences. In this approach, a *gap* of length k (k consecutive nulls of one sequence aligned with k consecutive elements of the other sequence) has cost $w_k = kw_1$. In other words, the gap costs are directly proportional to gap length. Ukkonen (1983) has described a fast algorithm that is applicable when substitution costs are equal to 1 and gap costs are proportional to gap length. His algorithm still requires $O(MN)$ steps in the worst case.

Since a single genetic event can insert or delete an entire segment of a genetic sequence, a long gap should arguably cost only slightly more than a shorter one. Waterman *et al.* (1976) have generalized the Sellers algorithm so that any

set of $w_k \leq kw_1$ can be used. In their algorithm, gap costs need not be directly proportional to gap length. Fitch and Smith (1983) have discussed a case in which such gap costs are necessary in order to produce the correct alignment. The major disadvantage of the algorithm of Waterman *et al.* (1976) is that it takes $O(M^2N)$ steps to find the minimal alignment cost, where M is the length of the longer sequence. Recently, Waterman (1984) has described an algorithm for concave gap costs that is conjectured to require $O(MN)$ steps.

The general approach of Waterman *et al.* (1976) allows each null in a gap to have a different cost. Gotoh (1982) considered the more restricted case of *affine gap costs*. Specifically, opening a gap costs v and each null in the gap costs u, so that $w_k = v + ku$, where $v, u > 0$. The major advantage of Gotoh's algorithm is that it finds the minimum cost of aligning two sequences in $O(MN)$ steps.

In addition, Gotoh's algorithm attempts to find just one (rather than all) of the optimal alignments. But the single alignment found occasionally fails to be optimal. Taylor (1984) has described a modification of Gotoh's algorithm that always finds at least one optimal alignment. Taylor's algorithm has the disadvantages that in the general case it does not find all and only the optimal alignments and that its storage requirements depend on the length of the longest gap to be allowed.

We describe here a modification of Gotoh's algorithm, called the SS-2 algorithm, that correctly finds all optimal alignments of two sequences in $O(MN)$ steps. We also present two sequences and a set of gap costs for which Gotoh's algorithm fails to find the single optimal alignment, and two sequences for which Taylor's algorithm cannot find all and only the optimal alignments. First, we introduce a more precise form of path graph than previously used, one which is needed to represent accurately all optimal alignments for affine gap costs.

Affine path graphs. Each alignment of two sequences can be represented in a two-dimensional graph by a *path*, which consists of a contiguous series of horizontal, diagonal and/or vertical edges between graph nodes from the upper left to lower right nodes of the graph. Figure 1a illustrates the complete path graph for sequences $X = x_1x_2 \ldots x_M$ and $Y = y_1y_2 \ldots y_N$. In general, diagonal edge $D_{i,j}$ aligns x_i with y_j, vertical edge $V_{i,j}$ aligns x_i with a null in Y, and horizontal edge $H_{i,j}$ aligns a null in X with y_j, as shown in Fig. 1b. If gaps are assigned a cost directly proportional to their length ($w_k = kw_1$), all optimal alignments of two sequences can simultaneously be represented in such a *linear path graph*. For example, if $w_k = k$, the five optimal alignments of sequences AGCCT and AGGTCC are represented by the five overlapping paths of the linear path graph in Fig. 1c. For affine gap costs, however, a linear path graph can be ambiguous in indicating precisely which paths are optimal. For example, if $w_k = 1 + k$, sequences AGT and TGAGTT have a minimum

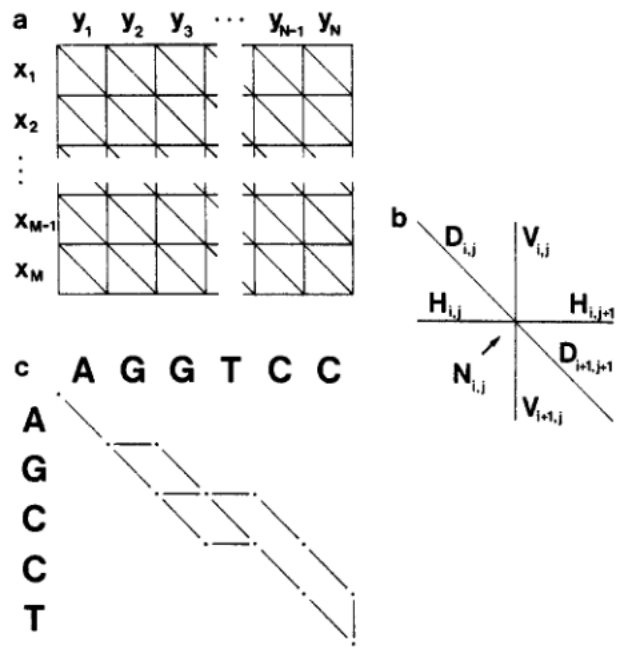

Figure 1. Linear path graphs. (a) The complete graph. (b) The edges adjacent to node $N_{i,j}$. (c) Paths representing the five optimal alignments of AGCCT and AGGTCC.

alignment cost of 5. Panels a–c of Fig. 2 show all three optimal alignments with this cost and the corresponding simple path graphs. In Fig. 3a these three optimal paths are combined to give a composite graph. This graph contains a fourth path (Fig. 2d) but fails to indicate that this fourth path is not optimal.

One way to solve this problem is to represent horizontal and vertical edges more precisely by the eight symbols shown in Fig. 4a. Their meanings are defined by the following four conventions, which are illustrated in Fig. 4b. (1) A path using a horizontal edge whose left half is bold must also use the horizontal edge to its left. (2) A path using a horizontal edge whose right half is bold must also use the horizontal edge to its right. (3) A path using a vertical edge whose top half is bold must also use the vertical edge above. (4) A path using a vertical edge whose bottom half is bold must also use the vertical edge below.

A path graph that employs these symbols and conventions is called an *affine path graph*. For example, the three optimal alignments in panels a–c of Fig. 2 are indicated unambiguously by the affine path graph shown in Fig. 3b. The SS-2 algorithm presented below actually produces the equivalent affine path graph shown in Fig. 3c.

When affine gap costs are used, the minimum cost of continuing a path from a given node to the lower right node of the graph may depend upon whether the

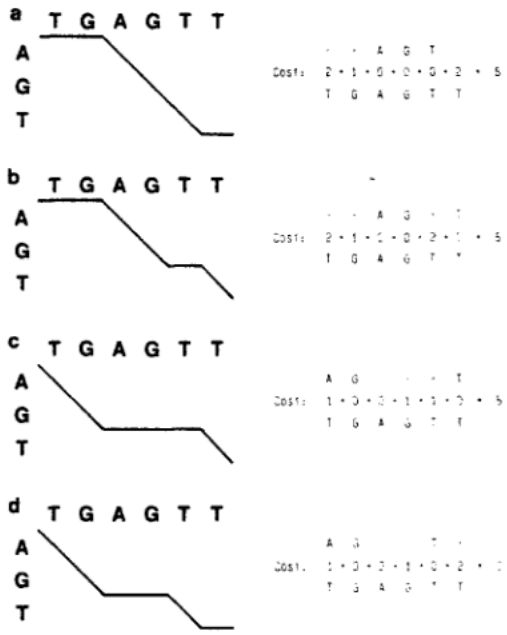

Figure 2. Four alignments of AGT and TGAGTT, and their path graphs. (a)–(c) Optimal alignments for $w_k = 1 + k$. (d) A non-optimal alignment.

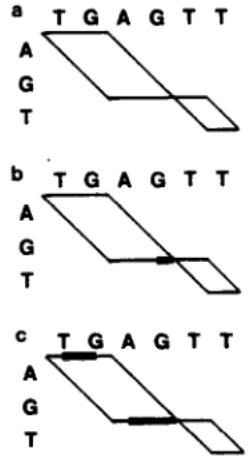

Figure 3. Composite path graphs representing the optimal alignments of AGT and TGAGTT for $w_k = 1 + k$. (a) The linear path graph. (b) An affine path graph representing only the optimal alignments. (c) The affine path graph produced by the SS-2 algorithm.

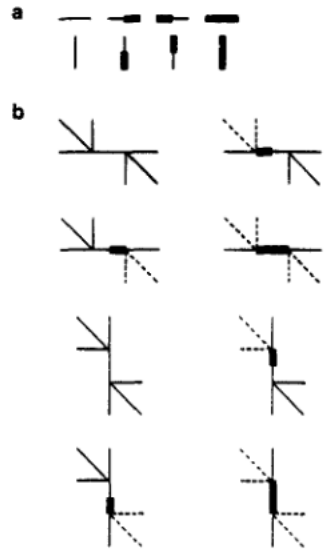

Figure 4. New edge symbols and their meaning. (a) Eight symbols for horizontal and vertical edges. (b) No path using the central edge may use the dotted edges.

given node was entered using a vertical, horizontal or diagonal edge. Paths that enter a node through different edges may have different optimal continuations. Thus new edge symbols, such as those in Fig. 4a, are necessary if all and only the optimal paths are to be represented in a single path graph. Using affine gap costs $w_k = v + ku$ is equivalent to charging $v+u$ for the first null in a gap and u for each subsequent null. Since any path that uses a vertical or horizontal edge has already opened a gap, each subsequent null in the gap will have identical cost u. Thus all paths that enter a node through a given edge will have the same optimal continuations. Therefore the edge symbols of Fig. 4a provide sufficient modification of the linear path graph to indicate precisely all and only the optimal alignments for affine gap costs.

When non-affine gap costs are employed, however, even an affine path graph will not suffice in the general case to represent accurately all and only the optimal alignments of two sequences. For example, if $w_1 = 1.2$ and $w_k = 0.7 + (0.7)k$ for $k > 1$, there are two optimal alignments of the sequences AGTCGA and GTTACCG (Fig. 5a). A linear path graph containing the paths that represent each of these alignments appears in Fig. 5c. It is not possible to use the horizontal edge symbols of Fig. 4a in this graph in a way that includes both optimal alignments but excludes the two non-optimal alignments shown in Fig. 5b. Further generalization of the vertical and horizontal edge symbols, however, would allow precise representation of the optimal alignments implied by these and more complicated sets of gap costs.

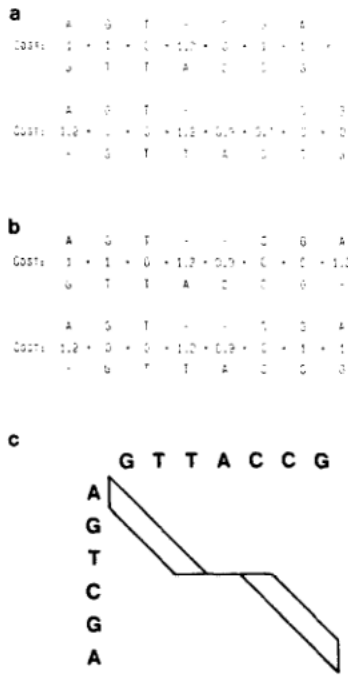

Figure 5. Non-affine gap costs. (a) The two optimal alignments of AGTCGA and GTTACCG for $w_1 = 1.2$ and $w_k = 0.7 + (0.7)k$ when $k > 1$. (b) Two non-optimal alignments. (c) A composite path graph representing the optimal alignments.

An Alignment Algorithm for Affine Gap Costs. An extension of Gotoh's algorithm is presented here that finds all and only the optimal alignments of two sequences for affine gap costs and represents them by an affine path graph in the manner described above. Let the two sequences be $X = x_1 x_2 \ldots x_M$ and $Y = y_1 y_2 \ldots y_N$ and the affine gap costs be $w_k = v + ku$. The algorithm uses three rectangular number arrays (P_{ij}, Q_{ij}, R_{ij}) to store the costs associated with the graph nodes N_{ij}. In place of arrays P and R two one-dimensional number arrays, and in place of array Q a variable, provide sufficient storage (Gotoh, 1982). The three rectangular arrays are used here for ease of exposition. For each array, index i ranges from 0 to M and index j from 0 to N. The algorithm also uses seven rectangular bit arrays ($a_{ij}, b_{ij}, \ldots, g_{ij}$) to store data associated with graph edges V_{ij}, H_{ij} and D_{ij}. For each array, index i ranges from 0 to $M+1$ and index j from 0 to $N+1$. It is possible to limit the first index to the range 0 to M and the second to the range 0 to N, but then dealing with the borders of the arrays during the edge assignment section of the algorithm becomes more complicated.

The SS-2 algorithm. The following 11 steps produce the minimum

alignment cost, $R_{M,N}$, and an affine path graph containing all and only those paths that represent optimal alignments of sequences X and Y. All statements involving a negative index should be omitted.

Initialization. Execute step {1}.

{1} Initialize the number and bit arrays:
For j from 0 to N, set $P_{0,j}$ to $+\infty$ and $R_{0,j}$ to $v+ju$.
For i from 0 to M, set $Q_{i,0}$ to $+\infty$ and $R_{i,0}$ to $v+iu$.
Set $R_{0,0}$ to 0.
Set bit arrays a–g uniformly to 0.
Set $c_{M+1,N+1}$ to 1.

Cost assignment. For i from 0 to M and j from 0 to N, execute steps {2}–{7}.

{2} Find the minimum cost of a path ending at node $N_{i,j}$ and using edge $V_{i,j}$:
Set $P_{i,j}$ to $u+\min(P_{i-1,j}, R_{i-1,j}+v)$.

{3} Determine if cost $P_{i,j}$ can be achieved using edge $V_{i-1,j}$ and if it can be achieved without using edge $V_{i-1,j}$:
If $P_{i,j}=P_{i-1,j}+u$, set $d_{i-1,j}$ to 1.
If $P_{i,j}=R_{i-1,j}+v+u$, set $e_{i-1,j}$ to 1.

{4} Find the minimum cost of a path ending at node $N_{i,j}$ and using edge $H_{i,j}$:
Set $Q_{i,j}$ to $u+\min(Q_{i,j-1}, R_{i,j-1}+v)$.

{5} Determine if cost $Q_{i,j}$ can be achieved using edge $H_{i,j-1}$ and if it can be achieved without using edge $H_{i,j-1}$:
If $Q_{i,j}=Q_{i,j-1}+u$, set $f_{i,j-1}$ to 1.
If $Q_{i,j}=R_{i,j-1}+v+u$, set $g_{i,j-1}$ to 1.

{6} Find the minimum cost of a path ending at node $N_{i,j}$:
Set $R_{i,j}$ to $\min(P_{i,j}, Q_{i,j}, R_{i-1,j-1}+c(x_i, y_j))$.

{7} Determine if cost $R_{i,j}$ can be achieved by using edge $V_{i,j}$, $H_{i,j}$ or $D_{i,j}$:
If $R_{i,j}=P_{i,j}$, set $a_{i,j}$ to 1.
If $R_{i,j}=Q_{i,j}$, set $b_{i,j}$ to 1.
If $R_{i,j}=R_{i-1,j-1}+c(x_i, y_j)$, set $c_{i,j}$ to 1.

Edge assignment. For i from M to 0 and j from N to 0, execute steps {8}–{11}.

{8} If there is no optimal path passing through node $N_{i,j}$ which has cost $R_{i,j}$ at node $N_{i,j}$, remove edges $V_{i,j}$, $H_{i,j}$ and $D_{i,j}$:
If ($a_{i+1,j}=0$ or $e_{i,j}=0$) and ($b_{i,j+1}=0$ or $g_{i,j}=0$) and ($c_{i+1,j+1}=0$).
Set $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ to 0.

{9} If no optimal path passes through node $N_{i,j}$, proceed to the next node:
If $a_{i+1,j}=b_{i,j+1}=c_{i+1,j+1}=0$, skip steps {10} and {11}.

{10} If edge $V_{i+1,j}$ is in an optimal path and requires edge $V_{i,j}$ to be in an optimal path, determine if an optimal path that uses edge $V_{i+1,j}$ must use edge $V_{i,j}$ and the converse:
If $a_{i+1,j}=1$ and $d_{i,j}=1$,
[set $d_{i+1,j}$ to $1-e_{i,j}$, set $e_{i,j}$ to $1-a_{i,j}$ and]
set $a_{i,j}$ to 1.
[Otherwise, set $d_{i+1,j}$ and $e_{i,j}$ to 0.]

{11} If edge $H_{i,j+1}$ is in an optimal path and requires edge $H_{i,j}$ to be in an optimal path, determine if an optimal path that uses edge $H_{i,j+1}$ must use edge $H_{i,j}$ and the converse:

If $b_{i,j+1}=1$ and $f_{i,j}=1$,
[set $f_{i,j+1}$ to $1-g_{i,j}$, set $g_{i,j}$ to $1-b_{i,j}$ and]
set $b_{i,j}$ to 1.
[Otherwise, set $f_{i,j+1}$ and $g_{i,j}$ to 0.]

Comments. The meaning of the bit arrays changes during the execution of the algorithm. Let an (i,j) *path* be a path from $N_{0,0}$ to $N_{i,j}$. After the cost assignment section is complete, the seven bit arrays store the following information:

$a_{i,j}=1$ iff an optimal (i,j) path uses $V_{i,j}$.
$b_{i,j}=1$ iff an optimal (i,j) path uses $H_{i,j}$.
$c_{i,j}=1$ iff an optimal (i,j) path uses $D_{i,j}$.
$d_{i,j}=1$ iff among $(i+1,j)$ paths through $N_{i,}$, an optimal one uses $V_{i,j}$.
$e_{i,j}=1$ iff among $(i+1,j)$ paths through $N_{i,j}$, an optimal one does not use $V_{i,j}$.
$f_{i,j}=1$ iff among $(i,j+1)$ paths through $N_{i,j}$, an optimal one uses $H_{i,j}$.
$g_{i,j}=1$ iff among $(i,j+1)$ paths through $N_{i,j}$, an optimal one does not use $H_{i,j}$.

After the edge assignment section is complete, the seven bit arrays store the affine path graph, as illustrated in Fig. 6 and described below:

$a_{i,j}=1$ iff an optimal (M, N) path uses $V_{i,j}$.
$b_{i,j}=1$ iff an optimal (M, N) path uses $H_{i,j}$.
$c_{i,j}=1$ iff an optimal (M, N) path uses $D_{i,j}$.
$d_{i,j}=1$ iff every optimal (M, N) path that uses $V_{i,j}$ also uses $V_{i-1,j}$. (The top half of $V_{i,j}$ is bold.)
$e_{i,j}=1$ iff every optimal (M, N) path that uses $V_{i,j}$ also uses $V_{i+1,j}$. (The bottom half of $V_{i,j}$ is bold.)
$f_{i,j}=1$ iff every optimal (M, N) path that uses $H_{i,j}$ also uses $H_{i,j-1}$. (The left half of $H_{i,j}$ is bold.)
$g_{i,j}=1$ iff every optimal (M, N) path that uses $H_{i,j}$ also uses $H_{i,j+1}$. (The right half of $H_{i,j}$ is bold.)

Note that if $a_{i,j}=0$, bits $d_{i,j}$ and $e_{i,j}$ are meaningless. Similarly, if $b_{i,j}=0, f_{i,j}$ and

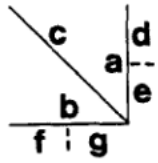

Figure 6. Bit array assignments. Arrays a–c correspond to full edges and d–g to half edges.

$g_{i,j}$ are meaningless. Table I shows the values of the number and bit arrays after the cost and edge assignment for sequences $X = \text{AGT}$ and $Y = \text{TGAGTT}$ where $w_k = 1 + k$. The affine path graph for this example is presented in Fig. 3c.

Since the algorithm involves a fixed number of steps for each node, its execution requires $O(MN)$ steps. In the initialization step, ∞ need only be a number larger than any number with which it will be compared during execution of the algorithm; the number $2v + \max(M, N)u + 1$ will suffice. The bit $c_{M+1,N+1}$ is initially set to 1 so that the conditions of steps {8} and {9} are false for node $N_{M,N}$. If the four expressions in brackets in steps {10} and {11} are

TABLE I

Number and Bit Arrays During Execution of the SS-2 Algorithm

Node index		After cost assignment										After edge assignment						
i	j	P	Q	R	a	b	c	d	e	f	g	a	b	c	d	e	f	g
0	0	∞	∞	0	0	0	0	0	1	0	1	0	0	0	*	*	*	*
0	1	∞	2	2	0	1	0	0	1	1	0	0	1	0	*	*	0	1
0	2	∞	3	3	0	1	0	0	1	1	0	0	1	0	*	*	1	0
0	3	∞	4	4	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	4	∞	5	5	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	5	∞	6	6	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	6	∞	7	7	0	1	0	0	1	0	0	0	0	0	*	*	*	*
1	0	2	∞	2	1	0	0	1	0	0	1	0	0	0	*	*	*	*
1	1	4	4	1	0	0	1	0	1	0	1	0	0	1	*	*	*	*
1	2	5	3	3	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	3	6	4	3	0	0	1	0	1	1	1	0	0	1	*	*	*	*
1	4	7	5	5	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	5	8	6	6	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	6	9	7	7	0	1	1	0	1	0	0	0	0	0	*	*	*	*
2	0	3	∞	3	1	0	0	1	0	0	1	0	0	0	*	*	*	*
2	1	3	5	3	1	0	1	1	0	0	1	0	0	0	*	*	*	*
2	2	5	5	1	0	0	1	0	1	0	1	0	0	1	*	*	*	*
2	3	5	3	3	0	1	0	0	1	1	0	0	1	0	*	*	0	1
2	4	7	4	3	0	0	1	0	1	1	1	0	1	1	*	*	1	1
2	5	8	5	5	0	1	0	0	1	1	0	0	1	0	*	*	0	0
2	6	9	6	6	0	1	0	0	1	0	0	0	0	0	*	*	*	*
3	0	4	∞	4	1	0	0	0	0	0	1	0	0	0	*	*	*	*
3	1	4	6	3	0	0	1	0	0	0	1	0	0	0	*	*	*	*
3	2	3	5	3	1	0	0	0	0	0	1	0	0	0	*	*	*	*
3	3	5	5	2	0	0	1	0	0	0	1	0	0	0	*	*	*	*
3	4	5	4	4	0	1	1	0	0	1	0	0	0	0	*	*	*	*
3	5	7	5	3	0	0	1	0	0	0	1	0	0	1	*	*	*	*
3	6	8	5	5	0	1	1	0	0	0	0	0	1	1	*	*	0	0

Starred bits are meaningless. Bits with index $i = 4$ or $j = 7$ do not change after initialization, and are not shown.

omitted, the linear path graph represented by bit arrays *a–c* contains all and only those edges that are part of *some* optimal path; such a graph often contains non-optimal paths. All seven bit arrays are still required to find these edges. While a specific set of substitution costs has been used in all examples in this paper, the SS-2 algorithm allows any set of substitution costs to be employed. Also, slight modification will allow it to employ different sets of affine gap costs for terminal and interior gaps. When the SS-2 algorithm is implemented on a computer lacking bit storage, the seven bits associated with a given node may conveniently be packed into a single byte.

Previous Algorithms. If $w_k = 5 + k$, a single optimal alignment exists for sequences AAAGGG and TTAAAAGGGGTT (Fig. 7). This alignment can not be found by Gotoh's algorithm. It saves only the edge bit arrays *a–c* during cost assignment in the forward direction. Steps corresponding to steps {3} and {5} of the SS-2 algorithm are absent. Gotoh's algorithm then repeats the cost assignment in the reverse direction, which saves additional edges, and finds a path with as few turns as possible that uses only saved edges. Since the two edges marked by arrows in the graph of Fig. 7 are not saved during either the forward or reverse cost assignment, neither edge can appear in the path found by Gotoh's algorithm. Thus the path shown in Fig. 7 is not found by Gotoh's algorithm even though it represents the only optimal alignment.

If $w_k = 1 + k$, sequences AGT and TGAGTT have only the three optimal alignments shown in panels a–c of Fig. 2. Taylor's algorithm for finding all optimal alignments (Taylor, 1984) saves at each node one vertical and one horizontal pointer for path traceback. Such a pointer system can find any pair of optimal alignments or all four alignments of Fig. 2, but it can not find precisely the three optimal alignments. In order to identify all and only the optimal alignments in the general case, a pointer system similar to that described by Smith *et al.* (1981) or Taylor (1984) must allow for an arbitrary

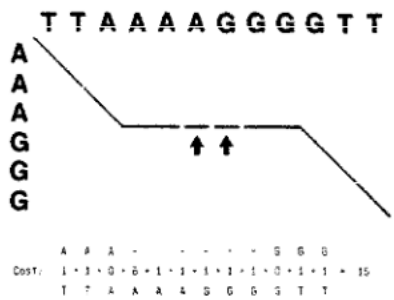

Figure 7. The optimal alignment of AAAGGG and TTAAAAGGGGTT for $w_k = 5 + k$. Arrows indicate two edges of the path graph not saved by Gotoh's algorithm.

number of pointers at each node, where many bits may be required to represent each pointer. In contrast, the system described above requires the storage of only seven bits per node even when gaps of arbitrary length are present. The optimal alignments can easily be represented by the corresponding affine path graph. Alternatively, all optimal alignments may be extracted from the bit arrays a–g and output linearly, as shown on the right of Fig. 2. When non-affine gap costs are employed, a generalization of this system may be inferior to a pointer system.

A Biological Example. The advantage of using affine gap costs when comparing biological sequences is illustrated by two DNA sequences for interleukin 2 (IL-2), an important regulator of T-cell clonal expansion. The DNA sequences code for human IL-2 (Taniguchi *et al.*, 1983) and murine IL-2 (Yokata *et al.*, 1985). The DD algorithm was used to search the diagonal path graph, which lacks all horizontal and vertical edges, for interesting gap-free subalignments of the two IL-2 sequences using a new similarity measure (Altschul and Erickson, 1986). Two of the four best subalignments found were human segment 65–107 with mouse segment 77–119 (43 nucleotides) and human segment 91–299 with mouse segment 133–341 (209 nucleotides). The ends of these subalignments overlap, as shown by the two paths in Fig. 8a for that part of the DD graph involving human segment 76–107 (H) and murine segment 88–149 (M). Joining these two subalignments requires a net deletion of 30 nucleotides from M, which can be achieved by inserting one or more gaps into segment H.

Using the SS algorithm (Sellers, 1974) and gap costs $w_k = 2k$ (Erickson and Sellers, 1983) to align segments H and M produces a large number of optimal subalignments (Fig. 8b). Because it costs nothing to open a gap, every optimal alignment contains at least 12 separate gaps in segment H, as illustrated by line H1 of Fig. 9. These alignments imply that at least 12 insertions or deletions are needed to explain the evolutionary divergence of segments H and M from a common ancestral gene segment. Such a large number of events is considered to be unlikely.

The number of insertions or deletions needed becomes smaller when a positive cost v is imposed for opening a gap (Figs 8c–e). For $v > 1$ and $u \geq 0.5$, all 30 nulls must be joined into a single gap, which can be inserted into any one of 11 different places in segment H (Fig. 8e). Specifically, it can be placed after nucleotide 90 (line H2 of Fig. 9), after nucleotide 100 (line H3 of Fig. 9), or after any nucleotide in between. Since murine segment 103–132 encodes 10 glutamine residues, we chose previously (Altschul and Erickson, 1986) the alignment of lines M and H2 in Fig. 9. Our experience in comparing nucleotide sequences using affine gap costs suggests that the costs $w_k = 2.5 + (0.5)k$ are useful with the SS-2 algorithm.

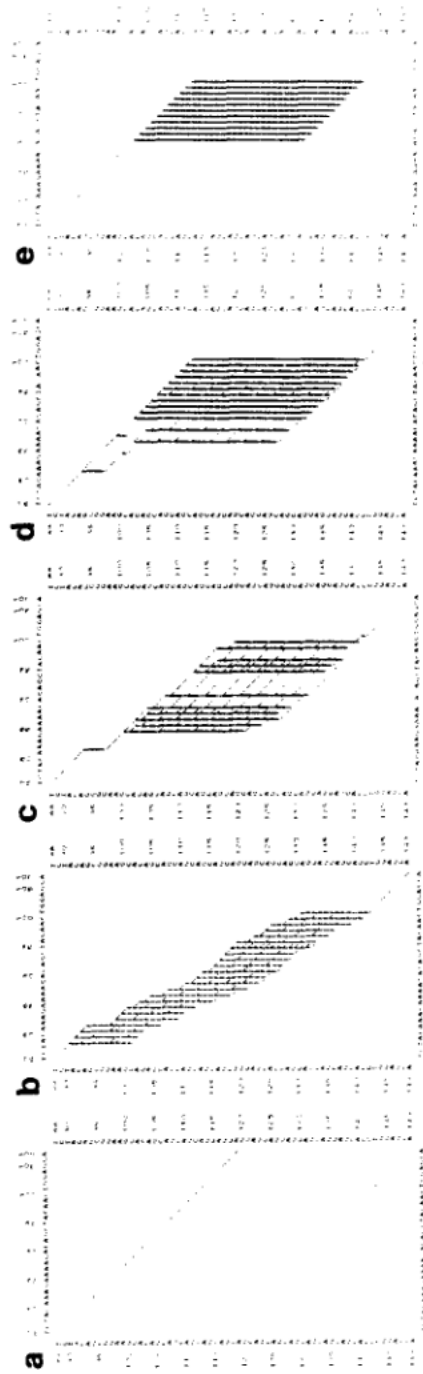

Figure 8. Five path graphs for two DNA sequences from interleukin 2. Vertical sequence M is a 62-nucleotide murine segment and horizontal sequence H is a 32-nucleotide human segment. (a) Part of a larger DD path graph. (b) The linear path graph for $w_k = 2k$; paths contain 12–18 gaps. (c) The affine path graph for $w_k = 0.5 + ku$ and $u \geq 1$; paths contain 3–11 gaps. (d) The affine path graph for $w_k = 1 + ku$ and $u \geq 1$; paths contain 1–3 gaps. (e) The affine path graph for $w_k = v + ku$, $v > 1$ and $u \geq 0.5$; paths contain only one gap.

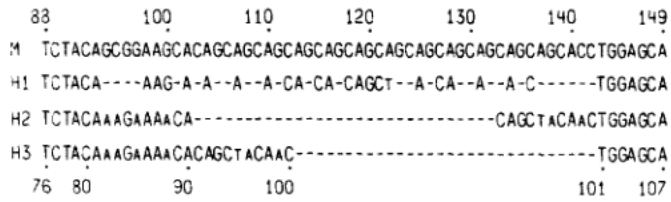

Figure 9. Three representative optimal alignments of murine segment M and human segment H from interleukin 2. Each of the lines H1–H3 is aligned with line M. Smaller letters are different from the corresponding letters in M.

The authors thank Dr Peter H. Sellers for encouragement and advice. This work was supported by research grant GM 32622 from the National Institute of General Medical Sciences and a research contract from the Office of Naval Research and the Army Research Office.

LITERATURE

Altschul, S. F. and B. W. Erickson. 1986. "A Nonlinear Measure of Subalignment Similarity and its Significance Levels." *Bull. math. Biol.* **48**, 617–632.

Erickson, B. W. and P. H. Sellers. 1983. "Recognition of Patterns in Genetic Sequences." In *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, D. Sankoff and J. B. Kruskal (Eds), pp. 55–91. Reading, MA: Addison-Wesley.

Fitch, W. M. and T. F. Smith. 1983. "Optimal Sequence Alignments." *Proc. natn. Acad. Sci. U.S.A.* **80**, 1382–1386.

Gotoh, O. 1982. "An Improved Algorithm for Matching Biological Sequences." *J. molec. Biol.* **162**, 705–708.

Needleman, S. B. and C. D. Wunsch. 1970. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins." *J. molec. Biol.* **48**, 443–453.

Schwartz, R. M. and M. O. Dayhoff. 1978. "Matrices for Detecting Distant Relationships." In *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 3, M. O. Dayhoff (Ed.), pp. 345–358. Washington, DC: National Biomedical Research Foundation.

Sellers, P. H. 1974. "On the Theory and Computation of Evolutionary Distances." *SIAM J. appl. Math.* **26**, 787–793.

Smith, T. F., M. S. Waterman and W. M. Fitch. 1981. "Comparative Biosequence Metrics." *J. molec. Evol.* **18**, 38–46.

Taniguchi, T., H. Matsui, T. Fujita, C. Takaoka, N. Kashima, R. Yoshimoto and J. Hamuro. 1983. "Structure and Expression of a Cloned cDNA for Human Interleukin-2." *Nature* **302**, 305–310.

Taylor, P. 1984. "A Fast Homology Program for Aligning Biological Sequences." *Nucl. Acids Res.* **12**, 447–455.

Waterman, M. S. 1984. "Efficient Sequence Alignment Algorithms." *J. theor. Biol.* **108**, 333–337.

——, T. F. Smith and W. A. Beyer. 1976. "Some Biological Sequence Metrics." *Adv. Math.* **20**, 367–387.

Ukkonen, E. 1983. "On Approximate String Matching." *Proc. Int. Conference on the Foundations of Computer Theory, Lecture Notes in Computer Science*, Vol. 158, pp. 487–496. Berlin: Springer-Verlag.

Yokota, T., N. Arai, F. Lee, D. Rennick, T. Mosmann and K. Arai. 1985. "Use of a cDNA Expression Vector for Isolation of Mouse Interleukin 2 cDNA Clones: Expression of T-Cell Growth Factor Activity After Transfection of Monkey Cells." *Proc. natn. Acad. Sci. U.S.A.* **82**, 68–72.

RECEIVED 2-17-86