Bulletin of Mathematical Biology Vol. 52, No. 3, pp. 359–373, 1990.
Printed in Great Britain.

0092-8240/90$3.00+0.00
Pergamon Press plc

OPTIMAL SEQUENCE ALIGNMENT ALLOWING FOR LONG GAPS*

■ OSAMU GOTOH
Department of Biochemistry,
Saitama Cancer Center Research Institute,
Ina-machi, Saitama 362, Japan

A new algorithm for optimal sequence alignment allowing for long insertions and deletions is developed. The algorithm requires $O((L+C)MN)$ computational steps, $O(LN)$ primary memory and $O(MN)$ secondary memory storage, where M and N ($M \geqslant N$) are sequence lengths, L (typically $L \leqslant 3$) is the number of segment specifying the gap weighting function, and C is a constant. We have also modified our earlier traceback algorithm so that it finds all and only the optimal alignments in a compact form of a directed graph. The current versions accept a set of aligned sequences as input, which facilitates multiple sequence alignment by some iterative procedures.

1. Introduction. Currently the most reliable and widely used methods for aligning a pair of biological sequences are those based on dynamic programming. Since the first development by Needleman and Wunsch (1970), these alignment methods have undergone great advances both in mathematics (Sellers, 1974; Waterman *et al.*, 1976; Smith *et al.*, 1981) and in algorithms (Sankoff, 1972; Gotoh, 1982; Wilbur and Lipman, 1983; Ukkonen, 1983; Fickett, 1984). Some earlier methods used a gap-weighting function $w(k)$ directly proportional to the gap length k, i.e. $w(k)=kw(1)$. However, accumulating experience with real biological sequences has revealed that better alignments are available when a length-independent term is supplemented to the gap-weighting function, i.e. $w(k)$ is linear as a function of k (Fitch and Smith, 1983). Gotoh (1982) first showed an $O(MN)$ algorithm with a linear (affine) gap weighting function, where M and N ($M \geqslant N$) are sequence lengths. Gotoh's algorithm has been further improved and generalized in several respects (Fredman, 1984; Taylor, 1984; Waterman, 1984; Altschul and Erickson, 1986; Gotoh, 1986; Myers and Miller, 1988).

Because a long stretch in a biological sequence can be lost or added by a single mutational event such as unequal crossing-over or transposition of a movable element, the probability of occurrence of a long gap seems almost independent of the gap length, while short insertions or deletions would occur in a length-dependent frequency. One way to incorporate this situation into

* Dedicated to Professor Akiyoshi Wada on the occasion of his 60th birthday.

alignment procedures is to use a "local" sequence matching method (Smith and Waterman, 1981; Gotoh, 1987). Another way is to use a global alignment method with concave gap-weighting functions. Waterman *et al.* (1976) presented an $O(MN(M+N))$ algorithm which allows any non-decreasing functions as the gap costs. Later Waterman (1984) proposed a more efficient algorithm, conjectured to require $O(MN \log(M))$ computational steps, with a concave gap-weighting function. Miller and Myers (1988) have recently improved Waterman's method so that both computation time and storage are considerably saved. However Miller and Myers' algorithm produces only one (or at most two) optimal alignment, even if there are many alternative optimal alignments. Moreover its implementation is sometimes difficult because many arrays of variable size must be allocated at run time.

We present here a much simpler algorithm when $w(k)$ is given by a piecewise linear function (see Fig. 1). The algorithm is a straightforward extension of our previous method with a linear gap-weighting function (Gotoh, 1982). We also demonstrate an efficient method to obtain a solution set which includes all and only the optimal alignments in a compact form of a directed graph. Although the accepted functional forms are somewhat restricted, the algorithm is simple and fast, and produces as good alignments as do more general methods.

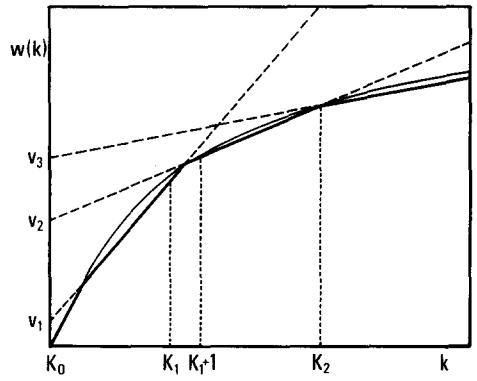


Figure 1. A piecewise linear gap-weighting function. The thick solid lines represent a piecewise linear gap-weighting function, $w(k)$, which approximates a smooth concave function (thin line).

2. Algorithm. To make the description clear, we briefly reproduce here the algorithm of Waterman *et al.* (1976). Let the two sequences under comparison be $\mathbf{a}=a_1a_2\ldots a_M$ and $\mathbf{b}=b_1b_2\ldots b_N$. The algorithm evaluates the elements of "distance matrix" $D(m, n)$ by induction:

$$D(m, n)=\min\{D(m-1, n-1)+d(a_m, b_n), F(m, n), G(m, n)\}, \qquad (1)$$

where:

$$F(m, n) = \min_{1 \leqslant k \leqslant m} \{D(m-k, n) + w(k)\}, \tag{2}$$

$$G(m, n) = \min_{1 \leqslant k \leqslant n} \{D(m, n-k) + w(k)\}, \tag{3}$$

and $d(a_m, b_n)$ is a measure of dissimilarity between residues a_m and b_n. Under certain initial conditions, the induction proceeds in any appropriate order from $m=1$ to M and $n=1$ to N. $D(M, N)$ is the desired distance between sequences **a** and **b**, and the path(s) through which the minimum value for $D(m, n)$ is chosen at each step gives the optimal alignment(s). The algorithm finishes after $O(MN(M+N))$ computational steps.

Now we consider the case of $w(k)$ being piecewise linear, i.e. $w(k) = w_i(k) = u_i k + v_i$ $(K_{i-1} < k \leqslant K_i$, for $1 \leqslant i \leqslant L)$, where $K_0 = 0$, $K_L = \infty$, $u_i > u_{i+1} \geqslant 0$, $w_i(K_i) \leqslant w_{i+1}(K_i)$, and $w_i(K_i + 1) > w_{i+1}(K_i + 1)$ $(1 \leqslant i < L)$. Then:

$$D(m-k, n) + w_i(k) \geqslant D(m-k, n) + w_{i-1}(k), \tag{4}$$

for $k \leqslant K_{i-1}$ $(1 < i \leqslant L)$ and:

$$D(m-k, n) + w_i(k) > D(m-k, n) + w_{i+1}(k), \tag{5}$$

for $k > K_i$ $(1 \leqslant i < L)$ because $w_i(k) \leqslant w_{i+1}(k)$ for $k \leqslant K_i$, and $w_i(k) > w_{i+1}(k)$ for $k > K_i$. These imply that outside the range where $w(k) = w_i(k)$, i.e. $k \leqslant K_{i-1}$ or $k > K_i$, there is at least one other term that is smaller than or equal to $D(m-k) + w_i(k)$. Hence, returning to equation (2), we get:

$$\begin{aligned} F(m, n) &= \min_{1 \leqslant i \leqslant L} \Big[\min_{K_{i-1} < k \leqslant \min(K_i, m)} \{D(m-k, n) + w_i(k)\} \Big] \\ &= \min_{1 \leqslant i \leqslant L} \{F_i(m, n)\}, \end{aligned} \tag{6}$$

where:

$$\begin{aligned} F_i(m, n) &\equiv \min_{1 \leqslant k \leqslant m} \{D(m-k, n) + w_i(k)\} \\ &= \min\{D(m-1, n) + v_i, F_i(m-1, n)\} + u_i. \end{aligned} \tag{7}$$

Note that the domain of $w_i(k)$ is extended in the definition of $F_i(m, n)$. The last equation has been derived in the same manner as described in Gotoh (1982). By symmetric logic, we get:

$$G(m, n) = \min_{1 \leqslant i \leqslant L} \{G_i(m, n)\}, \tag{8}$$

and:

$$G_i(m, n) = \min\{D(m, n-1) + v_i, G_i(m, n-1)\} + u_i. \tag{9}$$

If we define $F_0(m, n) \equiv D(m-1, n-1) + d(a_m, b_n)$, and $F_{i+L}(m, n) \equiv G_i(m, n)$, equation (1) is rewritten as:

$$D(m, n) = \min_{0 \leqslant i \leqslant 2L} \{F_i(m, n)\}. \tag{10}$$

Combination of equation (10) with equations (7) and (9) completes one cycle of the induction process. Because it takes $O(L)$ steps to calculate equation (10), and constant steps for equations (7) and (9), the total process finishes in $O((L+C)MN)$ computational steps. Moreover, because a single row, column, or reverse diagonal of each matrix of D, and F_i is enough to hold the information accessed later, the program will run with $O(LN)$ (or at most $O(L(M+N))$ memory.

3. Traceback

3.1. Abstract structure of optimal alignments. When two sequences are compared, possibly many alignments may have the minimum distance, i.e. multiple alignments may be "optimal". The set of these optimal alignments is called a solution set. Unfortunately, it is often impractical to enumerate all the optimal alignments in the standard manner. As discussed below, however, the complete structure of a solution set can be compactly represented in terms of a directed acyclic graph (solution graph). Our traceback algorithm is now adapted for producing the solution graph more efficiently and completely than those previously proposed (Gotoh, 1986).

For a representative example, we consider alignment between **a** = AGT and **b** = TGAGTT, the problem previously discussed by Altschul and Erickson (1986). There are three optimal alignments (Fig. 2) between **a** and **b** when $d(x, x) = 0$, and $u = v = d(x, y) = 1$ $(x \neq y)$.

Each alignment is also exhibited by a path in a two-dimensional Cartesian coordinate system (Fig. 2b). A diagonal stretch in a path represents a run of matches (a matching block), and a vertical or horizontal stretch represents a deletion in either sequence. It is obvious that there is a one-to-one correspondence between such a path (and hence an alignment) and the list of coordinates at the terminals of each stretch in the path. We call such a list a skeletal representation of the alignment (Fig. 2c). Adjacent elements, (i, j) and (m, n), in a skeletal representation have one of the three relations: (1)

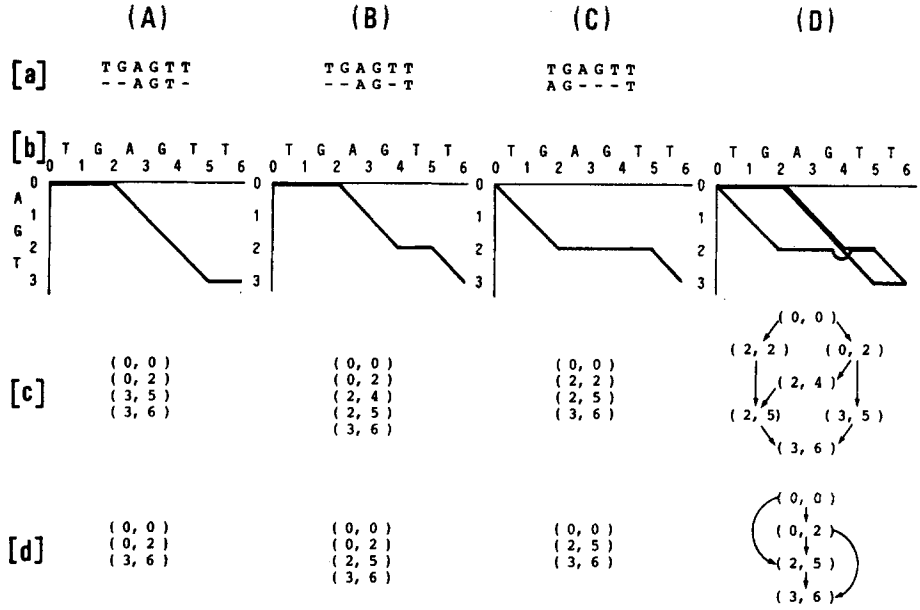


Figure 2. Various forms of representation of the three optimal alignments between **a** = AGT and **b** = TGAGTT. [a] Standard alignments, [b] path representations in a Cartesian coordinate system, [c] skeletal representations (A–C) and a solution graph (D), and [d] their reduced forms.

$i-j=m-n$, which corresponds to a matching block of $a_{i+1}a_{i+2}\ldots a_m \cdot b_{j+1}b_{j+2}\ldots b_n$; (2) $i=m$ and $j<n$, which corresponds to an insertion of $b_{j+1}b_{j+2}\ldots b_n$ opposite to the site between a_i and a_{i+1}; or (3) $i<m$ and $j=n$, which corresponds to an insertion of $a_{i+1}a_{i+2}\ldots a_m$ opposite to the site between b_j and b_{j+1}.

The composite structure of the optimal paths (Fig. 2D-b) is equivalent to the directed graph (Fig. 2D-c) which we will call the solution graph of optimal alignments between **a** and **b**. Each arc in a solution graph represents either a matching block (diagonal arc) or a deletion/insertion (non-diagonal arc), while a node corresponds to the coordinates at a terminal of such a block. A node is a branch node if it is a head of or a tail of more than one arc. A skeletal representation is a special solution graph with no branch node. A solution graph is acyclic because $i \leqslant m$ and $j \leqslant n$ for any arc $(i, j) \rightarrow (m, n)$.

Given a solution graph, it is easy to decompose it into individual skeletal representations by a depth-first search algorithm (Aho *et al.*, 1983). Variants of depth-first search algorithm are also useful to evaluate various interesting properties associated to the optimal alignments, e.g. the number of optimal alignments, the average percentage of sequence identities, etc.

3.2. Reduced form. We assume here that a deletion in one sequence is never immediately followed by a deletion in the other sequence. This is a desirable property for biological sequence alignment, and ensured if $u_L \geqslant \max d(x, y)/2$. Then, diagonal and non-diagonal arcs alternately appear in an optimal path. If an optimal path starts or ends with a non-diagonal arc, the source (0, 0) or the sink (M, N) is regarded as an empty diagonal arc that precedes or follows the terminal non-diagonal arc. In such a case, non-diagonal arcs and their tails are dispensable because they can be recovered by interpolation from the neighbouring nodes. Namely, if $(i,j)\rightarrow(k, l)$ and $(k, l)\rightarrow(m, n)$ are adjacent diagonal and non-diagonal arcs, respectively, then $k-l=i-j$ and either $k=m$ or $l=n$, and therefore (k, l) is obtained from (i,j) and (m, n) as:

$$i-j>m-n \Rightarrow k=m \quad \text{and} \quad l=m-i+j,$$

$$i-j<m-n \Rightarrow k=n+i-j \quad \text{and} \quad l=n.$$

Alternatively, a node (m, n) in a reduced solution graph may be interpreted to represent a matching block that starts with a_{m+1} and b_{n+1}, whereas an arc $(i,j)\rightarrow(m, n)$ represents an insertion that ends with a_m or b_n (depending on the sign of $i-j-m+n$). The output from our traceback routine described below is a reduced form of solution graph such as that shown in Fig. 2D-d.

3.3. Traceback. The principle of the traceback procedure is to construct the reduced form of solution graph for every subsequences $a_1a_2 \ldots a_m$ and $b_1b_2 \ldots b_n$ $(1 \leqslant m \leqslant M,\ 1 \leqslant n \leqslant N)$. We use two kinds of linked lists to implement these solution graphs. Each element of the first kind of list (primary list) is a "SAVE1-type" record consisting of four members, $\{m, n, p, q\}$, where (m, n) specifies a matching block that starts with a_{m+1} and b_{n+1}. The pointer variable p points to an previously stored record, $\{m', n', p', q'\}$, and indicates $(m', n')\rightarrow(m, n)$ is an arc in the reduced solution graph for subsequences $a_1a_2 \ldots a_m$ and $b_1b_2 \ldots b_n$. Variable q is a pointer to secondary list composed of "SAVE2-type" records of the form $\{p, q\}$. The role of the secondary list is to manage multiple paths from the same direction as discussed later in detail.

Figure 3 shows the outline of the algorithm. A Boolean variable $E_i(m, n)$ indicates whether $F_i(m, n)$ has been chosen, and $E_i(m, n)=$ true means that one or more optimal path comes to the point (m, n) from the direction indicated by i. $P_i(m, n)$ is a SAVE2-type record with two pointer variables. Note that an assignment to a record means a copy of the values for all members. Traceback information is stored when (m, n) represents a matching block, i.e. $E_i(m, n)=$ true for some $i>0$ and $E_0(m+1, n+1)=$ true. The latter condition is testable only after the induction proceeds to the point $(m+1, n+1)$, and hence the actual storing operation must be retarded until the function *adr*(m, n) is called at point $(m+1, n+1)$. This function refreshes the pointers after storing the

SAVE1-type records as indicated by $E_i(m, n)$. If $E_i(m, n)$=true for several i, (m, n) is a branch node where paths from different directions join.

Function *link*(m, n, i), on the other hand, handles joining of paths from the same direction, e.g. $(k, n)\rightarrow(m, n)$ and $(l, n)\rightarrow(m, n)$, when:

$$D(m, n)=F_i(m, n)=D(k, n)+w_i(m-k)=D(l, n)+w_i(m-l), \tag{11}$$

$1\leqslant i\leqslant L$, and $k<l<m$.

If equation (11) holds, then:

$$F_i(l, n)=D(l, n)+v_i, \tag{12}$$

because there is some x $(0\leqslant x<l)$ so that:

$$\begin{aligned} D(x, n)+w_i(l-x)&=F_i(l, n)\leqslant D(k, n)+w_i(l-k)\\ &=D(l, n)+v_i=F_i(m, n)-u_i\cdot(m-l)\leqslant D(x, n)+w_i(l-x). \end{aligned}$$

The last relation derives from $F_i(m, n)\leqslant D(x, n)+w_i(m-x)$ for any $x<m$. Of course similar arguments apply when $L<i\leqslant 2L$. A set of joining arcs from the same direction is therefore best implemented by a singly linked list (secondary list) which increases a new element when the two terms in the right member of equation (7) [or equation (9)] have the same value. (This is equivalent to equation (12) with $m-1$ substituted for l.) Function *link*(m, n, i) manages the addition of a new element. The entry to a secondary list is held in $P_i.q(m, n)$ while the terminal of each list is indicated by $q=$**nil**.

Figure 4 shows the results of a run of the algorithm for the example in Fig. 2. The arrows in Figs 4B and C indicate the routes of traceback. We extract only the nodes that contribute to the optimal alignments with the structural information about their links inherited. At this stage, the duplicate nodes which may appear when $w_i(K_i)=w_{i+1}(K_i)$ and the optimal alignment(s) happens to have a gap(s) of just the length of K_i are eliminated. The resulting directed graph is the desired solution graph in a reduced form (Fig. 2D-d) if the directions of arrows are reversed.

4. Comparison with Other Methods. Some years ago we showed an $O((L+C)MN)$ algorithm for optimal sequence alignment with piecewise linear gap-weighting functions (Gotoh, 1982). The present method is simpler and uses less memory space than the earlier version. Our algorithm imposes a limitation in the form of gap-weighting function as compared with the more general approach of Waterman (1984). However, the improved speed and reduced memory requirement well counterbalance the quality of the alignment which might be affected a little, if any, by the approximation in the gap-weighting function. Very recently, Miller and Myers (1988) have proposed two

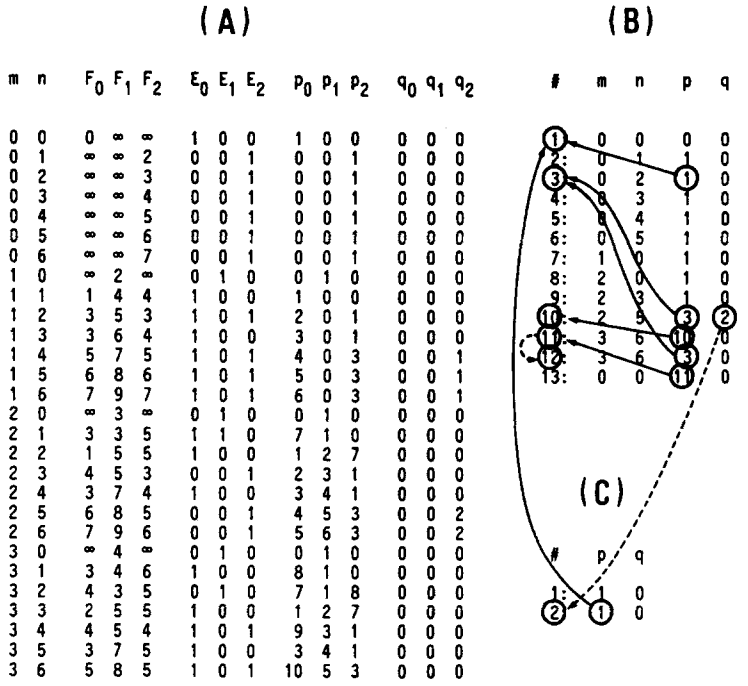
```
function adr(m,n): SAVE2-type;
    begin
        adr.p := Current position of the primary list;
        adr.q := nil;
        for i := 0 to 2L do
            if E_i(m,n) then
                write { m, n, P_i(m,n) } into the primary list;
        end;
    end;


function link(m,n,i): SAVE2-type;
    begin
        link.p := P_0(m,n).p;
        link.q := Current position of the secondary list;
        write { P_i(m,n) } into the secondary list;
    end;


for m := 1 to M do begin
    for n := 1 to N do begin
        for i := 1 to L do begin
            j := L + i;
            F_i(m,n) := Min { D(m-1,n) + v_i, F_i(m-1,n) } + u_i;
            F_j(m,n) := Min { D(m,n-1) + v_i, F_j(m,n-1) } + u_i;
            if D(m-1,n) + v_i < F_i(m-1,n) then
                P_i(m,n) := P_0(m-1,n) else
            if D(m-1,n) + v_i = F_i(m-1,n) then
                P_i(m,n) := link(m-1,n,i)
            else  P_i(m,n) := P_i(m-1,n);
            if D(m,n-1) + v_i < F_j(m,n-1) then
                P_j(m,n) := P_0(m,n-1) else
            if D(m,n-1) + v_i = F_j(m,n-1) then
                P_j(m,n) := link(m,n-1,j)
            else  P_j(m,n) := P_j(m,n-1);
        end;
        F_0(m,n) := D(m-1,n-1) + d(a_m,b_n);
        D(m,n)  := Min { F_i(m,n)  (0 ≤ i ≤ 2L) }
        for i := 0 to 2L do
            if D(m,n) = F_i(m,n) then E_i(m,n) := true
                                 else E_i(m,n) := false;
        if E_0(m,n) and E_i(m-1,n-1) for some i > 0 then
            P_0(m,n) := adr(m-1,n-1)
        else
            P_0(m,n) := P_0(m-1,n-1);
    end;
end;
P_last := adr(M,N);
```

Figure 3. Algorithm of the forward process for optimal sequence alignment with a piecewise linear gap-weighting function. This is not a program itself and does not use exactly correct PASCAL grammar.

related algorithms which have considerably better empirical performance than Waterman's algorithm. Miller and Myers' Algorithm 2 (MM-2) takes $O(MN\log(N+M))$ asymptotic calculation time for general concave gap-weighting functions, and $O(MN\log(L))$ time for piecewise linear functions. Thus the asymptotic performance of MM-2 is better than our algorithm that takes $O(MN(L+C))$ calculation time. For small values of L, however, our algorithm seems to be more efficient than MM-2, since the overhead in MM-2 due to management of a number of stacks and function calls of binary searches lowers the execution rate below that theoretically expected. We have long felt



m	n	F_0	F_1	F_2	E_0	E_1	E_2	p_0	p_1	p_2	q_0	q_1	q_2
0	0	0	∞	∞	1	0	0	1	0	0	0	0	0
0	1	∞	∞	2	0	0	1	0	0	1	0	0	0
0	2	∞	∞	3	0	0	1	0	0	1	0	0	0
0	3	∞	∞	4	0	0	1	0	0	1	0	0	0
0	4	∞	∞	5	0	0	1	0	0	1	0	0	0
0	5	∞	∞	6	0	0	1	0	0	1	0	0	0
0	6	∞	∞	7	0	0	1	0	0	1	0	0	0
1	0	∞	2	∞	0	1	0	0	1	0	0	0	0
1	1	1	4	4	1	0	0	1	0	0	0	0	0
1	2	3	5	3	1	0	1	2	0	1	0	0	0
1	3	3	6	4	1	0	0	3	0	1	0	0	0
1	4	5	7	5	1	0	1	4	0	3	0	0	1
1	5	6	8	6	1	0	1	5	0	3	0	0	1
1	6	7	9	7	1	0	1	6	0	3	0	0	1
2	0	∞	3	∞	0	1	0	0	1	0	0	0	0
2	1	3	3	5	1	1	0	7	1	0	0	0	0
2	2	1	5	5	1	0	0	1	2	7	0	0	0
2	3	4	5	3	0	0	1	2	3	1	0	0	0
2	4	3	7	4	1	0	0	3	4	1	0	0	0
2	5	6	8	5	0	0	1	4	5	3	0	0	2
2	6	7	9	6	0	0	1	5	6	3	0	0	2
3	0	∞	4	∞	0	1	0	0	1	0	0	0	0
3	1	3	4	6	1	0	0	8	1	0	0	0	0
3	2	4	3	5	0	1	0	7	1	8	0	0	0
3	3	2	5	5	1	0	0	1	2	7	0	0	0
3	4	4	5	4	1	0	1	9	3	1	0	0	0
3	5	3	7	5	1	0	0	3	4	1	0	0	0
3	6	5	8	5	1	0	1	10	5	3	0	0	0

Figure 4. Results of execution of the algorithm. (A) Matrix elements of F_i, E_i, $P_i \cdot p(p_i)$, and $P_i \cdot q(q_i)$. The values shown are at the end of each cycle in the induction. The constants **true** and **false** are represented by 1 and 0, respectively. (B) The primary and (C) the secondary lists. Arrows indicate the routes of traceback. Record Nos. 11 and 12 in the primary list realize a joining of arcs from different directions, while record No. 10 in the primary and No. 2 in the secondary lists realize a joining from the same direction.

no practical disadvantage even with two segments ($L=2$) of linear functions. For a fixed L, our algorithm may be implemented in a manner suitable for parallel processing. It is not immediately clear whether a complete solution set is easily available with a modification of MM-2 which is designed to get only one optimal alignment.

The traceback algorithm discussed above has fixed the incompleteness in those of Taylor (1984) and Gotoh (1986); Taylor's method might find non-optimal alignments (Altschul and Erickson, 1986) whereas Gotoh's might miss some optimal alignments like Fig. 2C since only a single path was taken from one direction at each node. Now we get all and only the optimal alignments with a single $O((L+C)MN)$ induction process and a traceback routine with $O(T)$ operations, where T is the total number of gaps in all the optimal alignments. Our traceback routine is much more efficient than that of Altschul and Erickson (1986) for the same purpose, because only those records that participate in the final solutions are read and processed, and because the data

structure is fully adapted for the subsequent depth-first-search routine. Altschul and Erickson's method takes additional $O(MN)$ steps for "edge assignment", and the final alignments are available after further bit manipulations the cost of which can be serious. To directly assess the performance, we implemented Altschul and Erickson's method with accommodation to a piecewise linear gap function. The results tested on an IBM PC-AT are shown in Table 1, which indicate that our method is really less time-consuming than Altschul and Erickson's, although the relative efficiency may be significantly machine-dependent. Compared with the method of Altschul and Erickson, the present method for linear gap-weighting functions ($L=1$) requires considerably larger storage, the precise size of which varies with the sequences and weight values as well as the sequence lengths. From a number of examinations we estimate that the storage requirement for alignment of protein sequences is about three times more than that needed by a bit map produced by Altschul and Erickson's method. For nucleotide sequences, the storage requirement amounts to six to nine times of Altschul and Erickson's bit map. However, the storage required in our method is only slightly dependent on L while it increases in proportion to L with a direct modification of Altschul and Erickson's method (examples are shown in Table 1). Hence, the required storage becomes comparable as L increases.

5. Implementation and an Example. We originally implemented the algorithm in C on a SONY NEWS-831 engineering workstation running under UNIX (4.2 BSD). The codes were then transferred to an IBM PC-AT, compiled with Turbo-C compiler (Borland International) and proven to be portable with minor changes. The sources, on an MS-DOS-format floppy disk, will be available from the author upon request.

Figure 5 shows alignments of human (Anderson *et al.*, 1981) and *X. laevis* (Roe *et al.*, 1985) mitochondrial 12s rRNAs at the 5′ portions. Each alignment is a representative of a solution set obtained with parameter sets in a domain of the same label (Fig. 6). A shadowed region in Fig. 6 has a complicated structure which is not detailed because the condition $u_1 \geqslant \max d(x, y)/2$ is violated or corresponding alignments have excessive gaps. All solutions shown are obtained with two- or three-segment gap functions, whereas solutions other than A, C, and K are not available with linear gap functions. Alignment F or one of its equivalents is most likely to be the correct one as judged from the proposed secondary structure models (Zwieb *et al.*, 1981; Roe *et al.*, 1985) and consistency of alignments with other mammalian sequences. The minute differences of solution D, G, I or J from F might be considered insignificant, but none of these plausible solutions is obtained with linear gap functions. The propriety of the use of piecewise linear gap-weighting function is also demonstrated by Monte Carlo simulations as the large values for normalized

Table 1. Dependence of calculation time and storage on gap weights

									This				AE	
L	u_1	v_1	u_2	v_2	u_3	v_3	Solution type	S (SD)	Time (sec)	SAVE1 (records)	SAVE2 (records)	Storage (bytes)	Time (sec)	Storage (bytes)
1	9	49	—	—	—	—	A	10.7	15	13 065	64	157 292	25	24 000
1	9	19	—	—	—	—	C	9.4	16	10 751	541	133 340	25	24 000
1	9	12	—	—	—	—	K	6.1	20	11 158	765	140 016	25	24 000
2	9	49	5	89	—	—	A	10.3	19	12 775	5 893	200 444	36	48 000
2	9	19	5	59	—	—	D	11.2	22	12 376	5 089	189 224	37	48 000
2	9	9	5	49	—	—	F	11.1	23	11 322	4 250	169 864	38	48 000
3	9	49	7	60	5	80	A	10.7	24	12 819	6 618	206 772	43	72 000
3	9	19	7	30	5	50	D	11.2	24	12 662	5 884	199 016	44	72 000
3	9	9	7	20	5	40	F	11.5	27	11 663	5 033	180 220	44	72 000

Solution types are shown in Fig. 5. Time and storage are those taken in getting the complete solution set with the present algorithm (This) or with Altschul and Erickson's method (1986) (AE). Unspecified parameter values are the same as those in the legend to Fig. 6. The sizes of a SAVE1-, a SAVE2-, and a SAVE-type records are 12, 8, and 8 bytes, respectively. $S = -(\text{distance} - \text{average})/(\text{standard deviation})$, where the average and the standard deviation were estimated by Monte Carlo method with 100 pairs of randomized sequences. All calculations were made on an IBM PC-AT personal computer. Calculation time for the present method includes that used for traceback but does not include that taken for Monte Carlo simulation. Calulation of the Altschul–Ericksons' method includes forward and edge-assignment routines but does not include further bit manipulations which were not clearly specified in their paper.

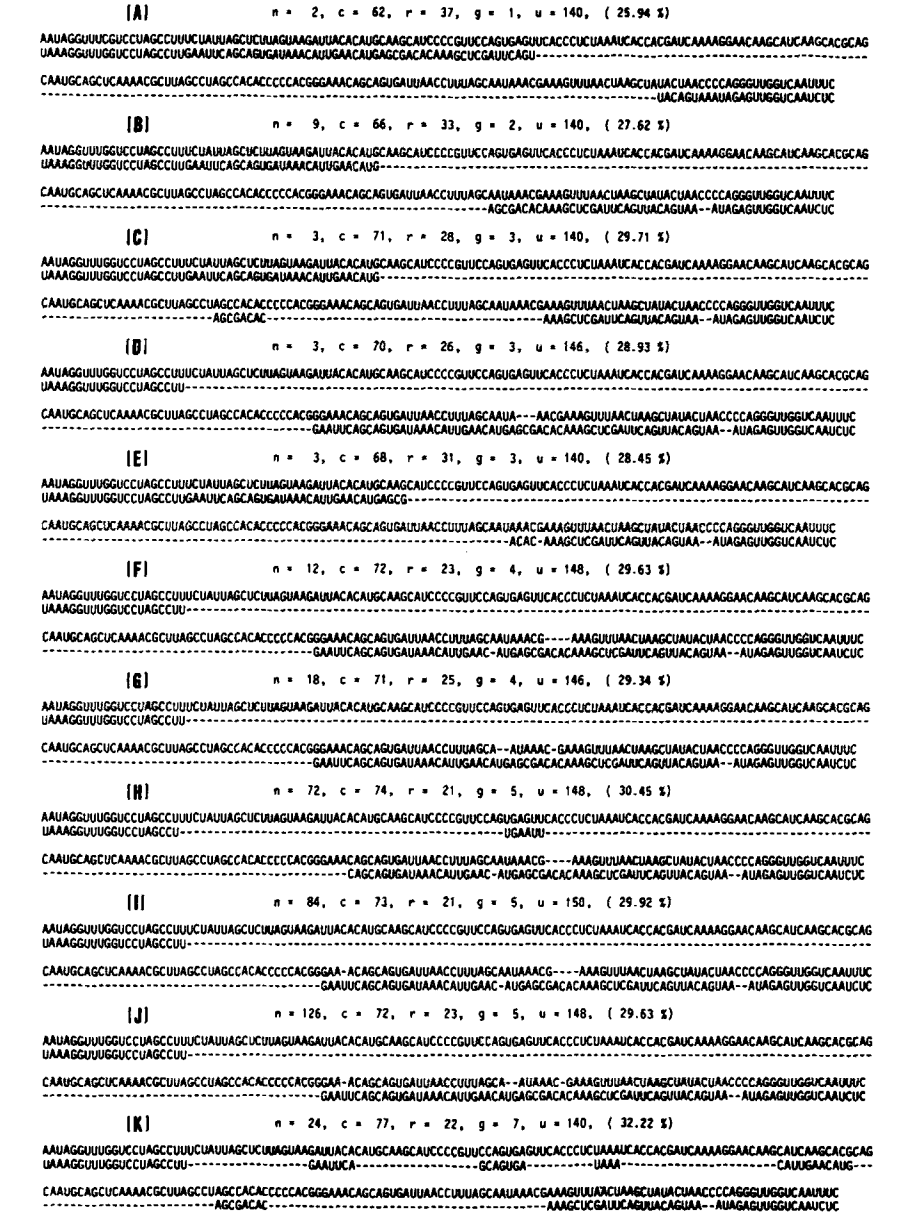


Figure 5. Optimal alignments at the 5′ portions of human (upper) and *X. laevis* (lower) mitochondrial 12S rRNAs. Each alignment is a representative of a solution set obtained with sets of parameter values specified in Fig. 6 by a domain with the same label. The numbers of equivalent solutions (n), matched identical (c) or non-identical (r) residues, unpaired residues (u) and gaps (g) are shown above each alignment. Some statistics associated with these alignments are listed in Table 1.

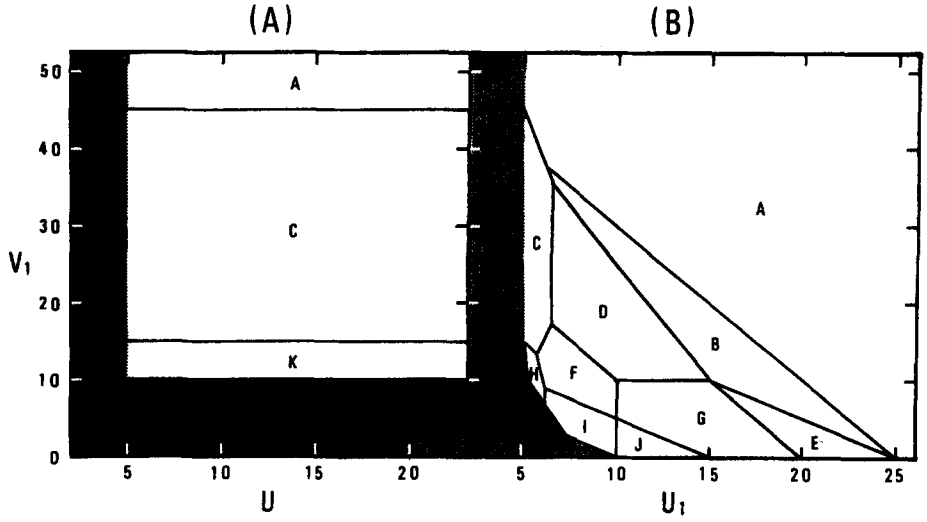


Figure 6. Parameter domains that give the same optimal solutions whose representatives are shown in Fig. 5. (A) Linear or (B) two-segment linear gap-weighting functions are used. Parameters u_1 and v_1 are varied with fixed values for $d(x, x)=0$, $d(x, y)=10$ $(x \neq y)$, $u_2=5$, and $K_1=10$. Shadowed regions are not examined because $u_1 < \max d(x, y)/2$, or are uninteresting because too many gaps are involved in the corresponding alignments.

deviation from the average (Table 1). Table 1 also shows that the value for constant C is approximately 2 to 3, though it actually depends on implementation.

In addition to the two major improvements described above, the current versions (**alp** for protein sequences and **aln** for nucleotide sequences) have been improved in several other points. First, a set of aligned sequences is accepted as input, which facilitate multiple-sequence alignment with some iterative procedure. Second, several command-line options are supported to make a number of calculations automatically. For example, "**alp/aln** -e *catalog_file*" performs alignment between every pair of sequences listed in the *catalog_file*. (Most of the options are common to those used in our local pattern matching programs, **psm/nsm** (Gotoh, 1987).) Similarly, "**alp/aln** -b *tree_file*" constructs a multiple-sequence alignment iteratively from closer pairs of sequences (or sets of aligned sequences) in much the same manner as noted by Waterman and Perlwitz (1984) and Feng and Doolittle (1987), where *tree_file* contains the sequence names and topological information about their relatedness. In combination with our three-sequence matching programs **alp3/aln3** (Gotoh, 1986), multiple-sequence alignments of good quality are obtainable. An alignment of cytochrome P450 proteins with nearly 40 members was obtained in this way, and evolutionary and structural implications derived from the alignment are discussed elsewhere (Gotoh and Fujii-Kuriyama, 1989).

The author thanks Dr Philip Taylor and Dr R. Allen White for suggestions on improvements of the representation of this paper.

LITERATURE

Aho, A. L., J. E. Hopcroft and J. D. Ullman. 1983. *Data Structure and Algorithms*. Reading, MA: Addison-Wesley.

Anderson, S., A. T. Bankier, B. G. Barrell, M. H. L. de Bruijn, A. R. Coulson, J. Drouin, I. C. Eperon, D. P. Nierlich, B. A. Roe, F. Sanger, P. H. Schreier, A. J. H. Smith, R. Staden and I. G. Young. 1981. Sequence and organization of the human mitochondrial genome. *Nature* **290**, 457–465.

Altschul, S. F. and B. W. Erickson. 1986. Optimal sequence alignment using affine gap costs. *Bull. math. Biol.* **48**, 603–616.

Feng, D.-F. and R. F. Doolittle. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. molec. Evol.* **25**, 351–360.

Fickett, J. W. 1984. Fast optimal alignment. *Nucleic Acids Res.* **12**, 175–179.

Fitch, W. M. and T. F. Smith. 1983. Optimal sequence alignments. *Proc. natl Acad. Sci. U.S.A.* **80**, 1382–1386.

Fredman, M. L. 1984. Algorithms for computing evolutionary similarity measures with length independent gap penalties. *Bull. math. Biol.* **46**, 553–566.

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

Gotoh, O. 1986. Alignment of three biological sequences with an efficient traceback procedure. *J. theor. Biol.* **121**, 327–337.

Gotoh, O. 1987. Pattern matching of biological sequences with limited storage. *Comput. Applic. Biosci.* **3**, 17–20.

Gotoh, O. and Y. Fujii-Kuriyama. 1989. Evolution, structure, and gene regulation of cytochrome P450. In *Frontier in Biotransformation*, K. Ruckpaul and H. Rein (eds), Vol. 1, pp. 165–243. Berlin: Akademie Verlag.

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

Myers, E. W. and W. Miller. 1988. Optimal alignments in linear space. *Comput. Applic. Biosci.* **4**, 11–17.

Needleman, S. B. and C. D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. molec. Biol.* **48**, 443–453.

Roe, B. A., D.-P. Ma, R. K. Wilson and J. F.-H. Wong. 1985. The complete nucleotide sequence of the *Xenopus laevis* mitochondrial genome. *J. biol. Chem.* **260**, 9759–9774.

Sankoff, D. 1972. Matching sequences under deletion/insertion constraints. *Proc. natl Acad. Sci. U.S.A.* **69**, 4–6.

Sellers, P. H. 1974. On the theory and computation of evolutionary distances. *SIAM J. appl. Math.* **26**, 787–793.

Smith, T. F. and M. S. Waterman. 1981. Identification of common molecular subsequences. *J. molec. Biol.* **147**, 195–197.

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

Taylor, P. 1984. A fast homology program for aligning biological sequences. *Nucleic Acids Res.* **12**, 447–455.

Ukkonen, E. 1983. On approximate string matching. *Proc. Int. Conf. Found. Comp. Theor. Lectures Notes Comp. Sci.* **158**, 487–496.

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

Waterman, M. S. and M. D. Perlwitz. 1984. Line geometries for sequence comparisons. *Bull. math. Biol.* **46**, 567–577.

Waterman, M. S., T. F. Smith and W. A. Beyer. 1976. Some biological sequence metrics. *Adv. Math.* **20**, 367–387.

Wilbur, W. J. and D. J. Lipman. 1983. Rapid similarity searches of nucleic acid and protein data banks. *Proc. natl Acad. Sci. U.S.A.* **80**, 726–730.

Zwieb, C., C. Glotz and R. Brimacombe. 1981. Secondary structure comparisons between small subunit ribosomal RNA molecules from six different species. *Nucleic Acids Res.* **9**, 3621–3640.

Received 25 April 1988
Revised 17 April 1989