*Bulletin of Mathematical Biology* Vol. 50, No. 2, pp. 97–120, 1988.
Printed in Great Britain.

0092-8240/88$3.00 + 0.00
Pergamon Press plc


# SEQUENCE COMPARISON WITH CONCAVE WEIGHTING FUNCTIONS

■ WEBB MILLER
Department of Computer Science,
The Pennsylvania State University,
University Park, PA 16802, U.S.A.
■ EUGENE W. MYERS†
Department of Computer Science,
University of Arizona,
Tucson, AZ 85721, U.S.A.

We consider efficient methods for computing a difference metric between two sequences of symbols, where the cost of an operation to insert or delete a block of symbols is a concave function of the block's length. Alternatively, sequences can be optimally aligned when gap penalties are a concave function of the gap length. Two algorithms based on the 'candidate list paradigm' first used by Waterman (1984) are presented. The first computes significantly more parsimonious candidate lists than Waterman's method. The second method refines the first to the point of guaranteeing $O(N^2 lg\ N)$ worst-case time complexity, and under certain conditions $O(N^2)$. Experimental data show how various properties of the comparison problem affect the methods' relative performance. A number of extensions are discussed, among them a technique for constructing optimal alignments in $O(N)$ space in expectation. This variation gives a practical method for comparing long amino sequences on a small computer.

*1. Introduction.* Sequences $A = a_1 a_2 \ldots a_M$ and $B = b_1 b_2 \ldots b_N$ can be compared by determining a sequence of 'basic operations' that converts $A$ to $B$ and minimizes the sum of the operations' weights. The most thoroughly studied set of basic operations are those that insert a single symbol, delete a symbol, or replace one symbol by another (Sankoff and Kruskal, 1983). Each such basic operation is assigned a nonnegative real number, called the operation's *weight* or *cost*, that may depend on the affected symbol or symbols.

An alternative approach is to treat an indel (i.e. insertion or deletion) of $k > 0$ consecutive symbols as an atomic operation, rather than $k$ operations on individual symbols. Traditionally, the operation is assigned a weight $w_k$ that depends only on the number of symbols inserted or deleted. Allowing 'block' indels is more natural than considering only single-symbol indels for certain applications in biology (Fitch and Smith, 1983) and computer science (Miller and Myers, 1986; Myers and Miller, 1988).

Unfortunately, determining a minimum-weight set of operations seems more expensive with multisymbol indels if the weights $w_k$ can be arbitrary. In

† This work was supported in part by NSF Grant DCR-8511455.

particular, the algorithm of Waterman *et al.* (1976) runs in time $O(N^3)$ (assuming $A$ and $B$ have approximately the same length, $N$), whereas sequence comparison with single-symbol indels can be performed in time $O(N^2)$ (Sankoff and Kruskal, 1983).

On the other hand, Gotoh (1982) showed how to perform sequence comparison with multisymbol indels in time $O(N^2)$ if the weighting function is affine, i.e. has the form $w_k = a + bk$ with $a, b$ constant. Fredman (1984) and Gotoh (1986) extended the method to simultaneously compare more than two sequences. Gotoh's result for two sequences has been generalized to a wider class of basic operations by Myers and Miller (1988). For the case that costs are integers and independent of the affected symbols, Myers and Miller (1988) also gave an $O(CN)$ 'greedy' algorithm. Here $C$ is the optimal cost of converting $A$ to $B$, so the algorithm is fast when $A$ and $B$ are similar. Miller and Myers (1986) developed a third algorithm, which has inferior worst case performance, but which works very efficiently in the context of screen updating applications.

This paper focuses on the efficient treatment of a wider class of 'concave' weighting functions. Waterman (1984) sketched an approach for concave weights that was conjectured to run in $O(N^2 lg\ N)$ expected-time. However, empirical tests reveal the algorithm to frequently exhibit $O(N^3)$ behavior, especially in cases where inputs are similar or $w_2 - w_1$ is larger than the average replacement cost. We present two new algorithms based on the 'candidate list paradigm'. The first algorithm is asymptotically faster in expectation than Waterman's and does not exhibit the same $O(N^3)$ weaknesses. The second algorithm refines the first to the point of guaranteeing $O(N^2 lg\ N)$ *worst-case* time complexity. Moreover, this method subsumes a number of additional results, e.g. it is $O(N^2)$ for affine weights, $O(N^2 lg\ P)$ for piece-wise affine weighting functions with $P$ pieces, and $O(N^2)$ if the equation $w_k = w_{k-x} + y$ is analytically solvable for arbitrary $x$ and $y$. In addition, application of the divide-and-conquer technique of Hirschberg (1975) to Methods 1 and 2 reduces their space complexity to $O(N)$ in expectation. A suite of empirical tests demonstrates the practical efficiency of our methods and confirms that Method 2 is the preferred method, both in theory and in practice. A software package written in *C* that delivers an optimal alignment in linear space using Method 2 is available from the authors upon request.

*2. The WSB Algorithm for Arbitrary Weighting Functions.* The starting point for this work is an $O(MN(M+N))$ algorithm by Waterman *et al.* (1976) for determining the minimum cost of converting $a_1 a_2 \ldots a_M$ to $b_1 b_2 \ldots b_N$ via symbol-dependent replacements and arbitrarily weighted block indels. Assume a $k$-symbol indel costs $w_k$, and the cost of replacing $a$ by $b$ is denoted $\delta(a, b)$. In the common case where $\delta$ is a metric, $\delta(a, a) = 0$ and $\delta(a, b) = \delta(b, a) \geqslant 0$. However, throughout this paper $\delta$ may be arbitrary. For $i \in [0, M]$ and

$j \in [0, N]$, let $C(i, j)$ denote the minimum cost of converting $a_1 a_2 \ldots a_i$ to $b_1 b_2 \ldots b_j$. The desired value is $C(M, N)$ and the algorithm proceeds by determining all $C(i, j)$ in row major order.

"The WSB Algorithm"

**define** $icost(i, j, p) \equiv C(i, p) + w_{j-p}$
**define** $dcost(i, j, p) \equiv C(p, j) + w_{i-p}$

$C(0, 0) \leftarrow 0$
**for** $j \leftarrow 1, 2, \ldots, N$ **do**
  $\{I(0, j) \leftarrow \min\{icost(0, j, p) : p \in [0, j-1]\}$
   $C(0, j) \leftarrow I(0, j)$
  $\}$
**for** $i \leftarrow 1, 2, \ldots, M$ **do**
  $\{D(i, 0) \leftarrow \min\{dcost(i, 0, p) : p \in [0, i-1]\}$
   $C(i, 0) \leftarrow D(i, 0)$
   **for** $j \leftarrow 1, 2, \ldots, N$ **do**
     $\{I(i, j) \leftarrow \min\{icost(i, j, p) : p \in [0, j-1]\}$
      $D(i, j) \leftarrow \min\{dcost(i, j, p) : p \in [0, i-1]\}$
      $C(i, j) \leftarrow \min\{I(i, j), D(i, j), C(i-1, j-1) + \delta(a_i, b_j)\}$
     $\}$
  $\}$ (1)

Informally, the computation of $C(i, j)$ for $i, j > 0$ in the innermost loop is justified as follows. An optimal conversion of $a_1 a_2 \ldots a_i$ to $b_1 b_2 \ldots b_j$ must either (i) insert $b_j$, (ii) delete $a_i$, or (iii) replace $a_i$ by $b_j$. The algorithm considers all three possibilities, then picks the best. Let $I(i, j)$ be the minimum cost over all conversions of $a_1 a_2 \ldots a_i$ to $b_1 b_2 \ldots b_j$ that insert $b_j$. The insertion must involve a block of $j-p$ symbols for some $p$, where the remaining operations optimally convert $a_1 a_2 \ldots a_i$ to $b_1 b_2 \ldots b_p$. Thus, defining $icost(i, j, p) = C(i, p) + w_{j-p}$ for $p \in [0, j-1]$, we have $I(i, j) = \min\{icost(i, j, p) : p \in [0, j-1]\}$. Similarly, $D(i, j)$, the minimum cost over all conversions of $a_1 a_2 \ldots a_i$ to $b_1 b_2 \ldots b_j$ that delete $a_i$, is $\min\{dcost(i, j, p) : p \in [0, i-1]\}$ where $dcost(i, j, p) = C(p, j) + w_{i-p}$. For the third possibility, $C(i-1, j-1) + \delta(a_i, b_j)$ is the minimum cost of a conversion that replaces $a_i$ with $b_j$. Thus $C(i, j) = \min\{I(i, j), D(i, j), C(i-1, j-1) + \delta(a_i, b_j)\}$.

*3. Concave Weighting Functions.* To see how the WSB algorithm can be made more efficient, consider the minimization operations,

$$I(i, j) \leftarrow \min\{icost(i, j, p) : p \in [0, j-1]\}, \tag{2}$$

for fixed row $i$. A given $p \in [0, N-1]$ enters into this minimization for all columns $j \in [p+1, N]$. It is helpful to visualize the '$p$-curve' $\{(x, icost(i, x, p))\}_{x=p+1}^{N}$, which shows $p$'s contribution to these later operations. Note that the $p$-curve has the same shape as the $q$-curve for any other $q \in [0, N-1]$, i.e. if $q < p$, then translating horizontally by $q-p$ and vertically by $C(i, q)-C(i, p)$ maps the $p$-curve into the $q$-curve.

It is often possible to determine that $p$ need not be considered in any subsequent minimization operations. For example, if there is a $q$ such that the $p$-curve lies above the $q$-curve for all columns $x$, then $p$ can be omitted. Comparing the $p$-curve and $q$-curve may be time-consuming if $w$ is arbitrary, but we will show that such comparisons are 'easy' for the class of concave curves. Formally, a weighting function $w$ is *concave* if and only if:

$$\Delta w_k \geq \Delta w_{k+1} \text{ for all } k \geq 1, \text{ where } \Delta w_k = w_{k+1} - w_k. \tag{3}$$

In words, $w$ is concave if the cost of inserting or deleting an additional symbol decreases with the size of the affected substring. Thus, every $p$-curve is concave downward since its domain is $[p+1, N]$. The least concave curve possible is a straight line ($\Delta w_k = \Delta w_{k+1}$ for all $k$), and since $\Delta w_k < 0$ is permitted, more concave curves can have a maximum as in Fig. 1. The concavity condition is equivalent to Waterman's (1984) condition: $w_{m+n+k} - w_{m+n} \leq w_{m+k} - w_m$ for all $m, n, k \geq 1$. If the concavity condition were strengthened to require $w_1 \geq \Delta w_1$, then concavity would imply that $w_{m+n} \leq w_m + w_n$ for all $m, n \geq 1$. As it stands, the two conditions are incomparable. The class of concave weighting functions was argued to be of biological interest by Waterman (1984). He cites the study of Fitch and Smith (1983) and suggests that concave functions such as $w_k = a + b \log k$ have intuitive appeal.

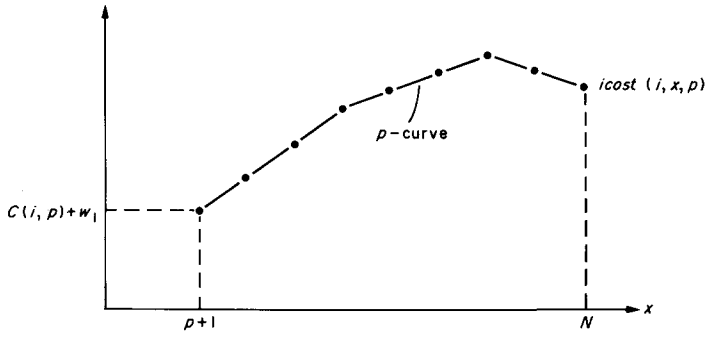


Figure 1. A sample $p$-curve.

The simple result given as Lemma 1 shows that concavity guarantees that a $p$-curve intersects a $q$-curve at most once.† This captures everything we need about concavity; subsequent proofs do not refer back to the original definition. In fact, there is a partial converse theorem that the reader is invited to formalize and prove, namely, if $w$ is neither concave nor convex ('convex' means that the sequence $\{\Delta w_k\}$ is nondecreasing), then there exist a $p$-curve and a $q$-curve that cross more than once.

Lemma 1, and all that follows, uses the notation $p <_x q$ and $p \leqslant_x q$ to mean that the $p$-curve lies strictly below (in the first case) or on-or-below (in the second case) the $q$-curve at point $x$. Formally:

$$\begin{aligned} p <_x q &\text{ iff } p, q \in [0, x-1] \text{ and } icost(i, x, p) < icost(i, x, q) \\ p \leqslant_x q &\text{ iff } p, q \in [0, x-1] \text{ and } icost(i, x, p) \leqslant icost(i, x, q) \end{aligned} \tag{4}$$

Keep in mind that a fixed $i$ is implicit in these definitions.

LEMMA 1. *If $w$ is concave and $p < q$ then:*
1. *If $p \leqslant_j q$ then $p \leqslant_x q$ for all $x \in [j, N]$;*
2. *If $p <_j q$ then $p <_x q$ for all $x \in [j, N]$;*
3. *If $q \leqslant_j p$ then $q \leqslant_x p$ for all $x \in [q+1, j]$;*
4. *If $q <_j p$ then $q <_x p$ for all $x \in [q+1, j]$.*

*Proof.* We only prove part 1; the other proofs are quite similar. Since $p < q$, $\Delta w_{y-q} \geqslant \Delta w_{y-p}$ for all $y > q$. Thus, for $x \in [j, N]$, $icost(i, x, p) = icost(i, j, p) + \sum_{y=j}^{x-1} \Delta w_{y-p} \leqslant icost(i, j, q) + \sum_{y=j}^{x-1} \Delta w_{y-p} \leqslant icost(i, j, q) + \sum_{y=j}^{x-1} \Delta w_{y-q} = icost(i, x, q)$, i.e. $p \leqslant_x q$. ∎

The notation $p \rightarrow_j q$, which can be read "$p$ dominates $q$ beyond $j$", signifies that the $p$-curve lies on or below the $q$-curve in all columns $x \in [j, N]$. Additionally, we require that either the $p$-curve is strictly below the $q$-curve for at least one point or, in case the curves are equal everywhere, that $p \geqslant q$. Formally:

$$p \rightarrow_j q \text{ iff } p \leqslant_x q \text{ for all } x \in [j, N] \text{ and either } p \geqslant q \text{ or } p <_x q \text{ for some } x \in [j, N]. \tag{5}$$

For concave weights, part 1 of the following lemma shows that to determine the $\rightarrow_j$-relationship between two curves, it is sufficient to compare them at the end points, $j$ and $N$. Part 2 of the lemma asserts that if $p$ dominates $q$ beyond $j$, then $p$ dominates $q$ beyond every $x \geqslant j$. This result illustrates the importance of breaking ties properly. For if we had defined $p \rightarrow_j q$ when $p =_x q$ for all $x \in [j, N]$ and $p < q$, then Lemma 2.2 and much that follows would fail. Part 3 guarantees that two mutually nondominating curves properly intersect, i.e. if $p < q$, then

† A contiguous set of points for which the curves are equal is considered a single intersection. This subtlety is important when, for example, the weighting function is a piece-wise affine curve.

the $p$-curve must begin strictly higher ($q <_j p$) and end strictly lower ($p <_N q$). Conversely, if two curves do not properly intersect, then one dominates the other.

LEMMA 2. *If $w$ is concave then:*

1. $p \rightarrow_j q$ *iff* $p < q$ *and* $p \leqslant_j q$ *and* $p <_N q$ *or* $p \geqslant q$ *and* $p \leqslant_N q$;
2. $p \rightarrow_j q$ *iff* $p \rightarrow_x q$ *for all* $x \in [j, N]$;
3. $p < q$ *and* $p \not\rightarrow_j q$ *and* $q \not\rightarrow_j p$ *iff* $q <_j p$ *and* $p <_N q$.

*Proof*

(1) First, suppose $p < q$. Then $p \rightarrow_j q \Rightarrow p \leqslant_x q$ for all $x \in [j, N]$ and $p <_x q$ for some $x \in [j, N] \Rightarrow_{Lemma1.2} p \leqslant_j q$ and $p <_N q$. ($\Rightarrow_{Lemma1.2}$ means "implies by part 2 of Lemma 1".) Conversely, $p \leqslant_j q$ and $p <_N q \Rightarrow_{Lemma1.1} p \leqslant_x q$ for all $x \in [j, N]$ and $p <_x q$ for $x = N \in [j, N] \Rightarrow p \rightarrow_j q$. Second, suppose $p \geqslant q$. Then $p \rightarrow_j q \Rightarrow p \leqslant_N q$ and, conversely, $p \leqslant_N q \Rightarrow_{Lemma1.3} p \leqslant_x q$ for all $x \in [p+1, N] \Rightarrow p \rightarrow_j q$.

(2) Supose $p \rightarrow_j q$ and $x \in [j, N]$. Then by part 1 either $p < q$, $p \leqslant_j q$, and $p <_N q$ or $p \geqslant q$ and $p \leqslant_N q$. Thus, either $p < q$, $p \leqslant_x q$ and $p <_N q$ or $p \geqslant q$ and $p \leqslant_N q$ by Lemma 1.1. But by part 1, this implies $p \rightarrow_x q$.

(3) First, suppose $p < q$, $p \not\rightarrow_j q$ and $q \not\rightarrow_j p$. Since $p \not\rightarrow_j q$, part 1 implies that either $q <_j p$ or $q \leqslant_N p$. Likewise, $q \not\rightarrow_j p$ implies $p <_N q$, ruling out the possibility that $q \leqslant_N p$. Thus, $q <_j p$ and $p <_N q$. Conversely, suppose that $q <_j p$ and $p <_N q$. Then $q <_j p \Rightarrow p \not\rightarrow_j q$ and $p <_N q \Rightarrow q \not\rightarrow_j p$. Moreover, if $p > q$, then $p <_N q \Rightarrow_{Lemma1.4} p <_j q$, a contradiction. Thus, $p < q$. ■

*4. The Candidate List Paradigm.* Two methods will be given to speed up the WSB algorithm when $w$ is concave. Both approaches maintain $N+2$ lists of 'candidates'. For a row, the candidates are column positions, and *vice versa*. One list, $R$, is for the current row of the $C$ matrix. $R$ is 'conservative' for position $(i, j)$ in the sense that $R$ excludes a value $q \in [0, j-1]$ only if $\min\{icost(i, x, p) : p \in R\}$ is unaffected for all $x \in [j, N]$. Similarly, a conservative list, $S(j)$, is maintained for each column $j \in [0, N]$.

Recently, Waterman (1984) presented a conservative candidate list algorithm. Both of the methods presented here compute significantly smaller candidate lists. Furthermore, the *min* computation for $I(i, j)$ and $D(i, j)$ is avoided by our methods since both guarantee that the first candidate in the list is optimal for position $(i, j)$, i.e.

$$I(i, j) = icost(i, j, R[1]) \text{ and } D(i, j) = dcost(i, j, S(j)[1]). \tag{6}$$

Methods 1 and 2 differ only in the procedures *Iupdate* and *Dupdate* for updating candidate lists and both have the overall form shown in (7).

"The General Structure of Methods 1 and 2"

$C(0,0) \leftarrow 0$
**for** $j \leftarrow 1, 2, \ldots, N$ **do**
  $\{R \leftarrow Iupdate(R, j)$
   $C(0,j) \leftarrow icost(0, j, R[1])$
  $\}$
**for** $i \leftarrow 1, 2, \ldots, M$ **do**
  $\{S(0) \leftarrow Dupdate(S(0), 0)$
   $C(i,0) \leftarrow dcost(i, 0, S(0)[1])$
   **for** $j \leftarrow 1, 2, \ldots, N$ **do**
    $\{R \leftarrow Iupdate(R, j)$
     $S(j) \leftarrow Dupdate(S(j), i)$
     $C(i,j) \leftarrow \min\{icost(i,j,R[1]), dcost(i,j,S(j)[1]), C(i-1,j-1)+\delta(a_i,b_j)\}$
    $\}$
  $\}$ (7)

Method 1 culls all positions that are dominated by another according to the relation $\rightarrow_j$. Method 2 goes further by computing lists that are minimal with respect to conservation, i.e. removal of any element gives a nonconservative list. Thus, Method 2's lists are in some sense optimal. Additionally, Method 2's lists are sublists of Method 1's lists, which are, in turn, sublists of Waterman's lists. Our discussions of the two methods focus exclusively on the computation of the $R$-lists for a given row $i$. The treatment of an $S(j)$-list for column $j$ is symmetric.

*5. Method 1.* As each of the two methods sweeps across row $i$, it culls unnecessary values from $R$. Method 1 is based on the informal observation that one of $p$ or $q$ can be discarded if the $p$-curve and the $q$-curve do not properly intersect between columns $j$ and $N$. To correctly handle such cases as two curves that are equal everywhere in $[j, N]$, Method 1 adopts the precise criterion that $p$ is eliminated if and only if $q \rightarrow_j p$ for some $q \in R$.

For $j \in [1, N]$ define the ordered list

$$R_j^1 = <p \in [0, j-1] : q \not\rightarrow_j p \text{ for all } q \in [0, j-1] - p>, \text{ in decreasing order.} \quad (8)$$

The relation $\rightarrow_j$ is reflexive, asymmetric, and transitive, i.e. a partial order. $R_j^1$ is the set of $p \in [0, j-1]$ having no $\rightarrow_j$-predecessors. It follows that if $q \in [0, j-1]$, then there exists a $p \in R_j^1$ such that $p \rightarrow_j q$. Moreover, Lemma 2.3 implies that every pair of curves in $R_j^1$ properly intersect on the interval $[j, N]$. Figure 2 gives an example.

In conformance with the previous section, we will show that $I(i,j) = icost(i, j, R_j^1[1])$ and that $R_j^1 = Iupdate^1(R_{j-1}^1, j)$ where $Iupdate^1$ is given below. In the procedure description, '$\cdot$' is list concatenation, '$\sim$' is deletion of

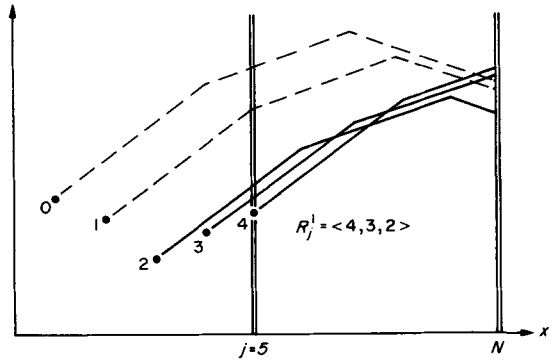


Figure 2. A sample Method 1 candidate list.

an element from a list, and $T[k]$ is the $k$th element of list $T$.

"The *Iupdate* Procedure for Method 1"

```
procedure Iupdate¹(T, j)
  {if j=1 then
     T←⟨0⟩
   else
     {T←⟨j−1⟩·T
      while |T|>1 and T[1]≤_N T[2] do
        T←T~T[2]
      k←2
      while k≤|T| do
        {while k>1 and T[k]≤_j T[k−1] do
           {k←k−1
            T←T~T[k]
           }
         k←k+1
        }
     }
   return T
  }                                                    (9)
```

LEMMA 3. *If w is concave and* $j \in [1, N]$ *then* $I(i,j) = icost(i, j, R_j^1[1])$.

*Proof.* First, suppose $q \in R_j^1 \sim R_j^1[1]$. Then $q < R_j^1[1]$, $q \not\to_j R_j^1[1]$, and $R_j^1[1] \not\to_j q$, so Lemma 2.3 implies that $R_j^1[1] <_j q$. For an arbitrary $q \in [0, j-1]$, there is a $p \in R_j^1$ satisfying $p \to_j q$, so $R_j^1[1] \leqslant_j p \leqslant_j q$. Thus, $R_j^1[1] \leqslant_j q$ for all $q \in [0, j-1]$, and consequently $I(i,j) = \min\{icost(i, j, q) : q \in [0, j-1]\} = icost(i, j, R_j^1[1])$. ■

THEOREM 1. *Let w be concave. Then* $R_1^1 = Iupdate^1(R, 1)$ *for any initial value of* $R$, *and* $R_j^1 = Iupdate^1(R_{j-1}^1, j)$ *for* $j \in [2, N]$.

*Proof.* $R_1^1 = \langle 0 \rangle = Iupdate^1(R, 1)$, so consider the call $Iupdate^1(R_{j-1}^1, j)$ for $j \in [2, N]$. We first show that $R_j^1 \subseteq T_{final}$, where $T_{final}$ denotes $T$'s value at the end of the procedure. After the assignment $T \leftarrow \langle j-1 \rangle \cdot T$, $T$ contains $R_{j-1}^1 \cup \{j-1\}$. But then $R_j^1 \subseteq T$, since if $q \in [0, j-2] - R_{j-1}^1$, then $p \rightarrow_{j-1} q$ for some $p \neq q$, hence Lemma 2.2 implies that $p \rightarrow_j q$ and $q \notin R_j^1$. If $q$ is eliminated by the first **while** loop, then $j-1 > q$ and $j-1 \leqslant_N q$, so Lemma 2.1 implies that $j-1 \rightarrow_j q$ and $q \notin R_j^1$.

If $p < q$ and $p, q \in T$ at the end of the first **while** loop, then $p <_N q$. To see this, first suppose $p, q \in R_{j-1}^1$. Then $p$ and $q$ are mutually nondominating, so Lemma 2.3 implies $p <_N q$. The other possibility is that $q = j-1$, in which case the first **while** loop guarantees $p <_N q$.

If $q$ is eliminated from $T$ by the second **while** loop, then $p \leqslant_j q$ for some $p < q$. Since $p <_N q$, as shown in the previous paragraph, Lemma 2.1 implies $p \rightarrow_j q$ and $q \notin R_j^1$. Thus, $R_j^1 \subseteq T_{final}$.

Suppose $p < q$ and $p, q \in T_{final}$. Then $q <_j p$. The second **while** loop enforces this condition for adjacent pairs $(T[k], T[k-1])$, which guarantees the ordering in general. Together with the condition $p <_N q$ (guaranteed by the first **while** loop), this implies $p \not\rightarrow_j q$ and $q \not\rightarrow_j p$ by Lemma 2.3. Now, if $q \notin R_j^1$ then there exists a $p \in R_j^1 \subseteq T_{final}$ such that $p \rightarrow_j q$, and this implies $q \notin T_{final}$. Thus, $R_j^1 \supseteq T_{final}$. ■

*6. Method 2.* Method 1 updates $R_{j-1}^1$ to $R_j^1$ by comparing curves at $j$ and at $N$. Tests for $x \in [j+1, N-1]$ are avoided, but two inefficiencies result. First, at each new column $j$, $R_j^1$ must be exhaustively checked for curves that cross between columns $j-1$ and $j$. Second, $R_j^1$ may contain values $p$ such that at every column $x \in [j, N]$ there is a $q \in [0, j-1]$ with $q <_x p$, e.g. the 3-curve in Fig. 3. The definition of $R_j^1$ merely guarantees that no single $q$ beats $p$ for all $x$.

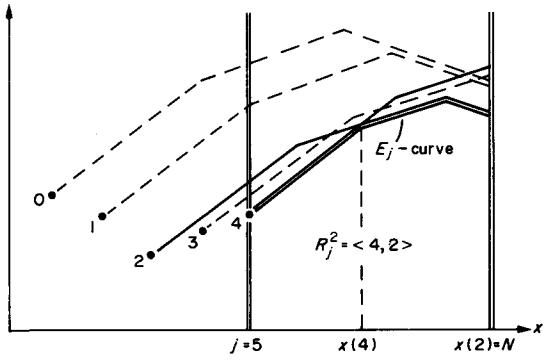


Figure 3. A sample Method 2 candidate list.

Method 2 avoids these problems by maintaining an explicit piece-wise representation of the 'minimum envelope' of the first $j$ curves, i.e. the curve $\{(x, E_j(x)\}_{x=j}^{N}$ where

$$E_j(x) = \min\{icost(i, x, p) : p \in [0, j-1]\}. \tag{10}$$

To insure that Method 2's candidate lists are always smaller than Method 1's, care is required when deciding which curve to let represent $E_j(x)$ for each $x$. Specifically, if more than one curve equals $E_j(x)$ at $x$ then the candidate $p$ that has no $\rightarrow_x$-predecessors must be chosen. In this case we say "$p$ represents the first $j$ curves at $x$" and write $p@_j x$. Formally:

$$p@_j x \text{ iff } p \in [0, j-1] \text{ and } x \in [j, N] \text{ and } (p \leqslant_x q \text{ and } q \not\rightarrow_x p) \text{ for every } q \in [0, j-1] - p. \tag{11}$$

The following argument shows that if $w$ is concave, then for each $x \in [j, N]$ there is exactly one $p$ representing the first $j$ curves at $x$. For $x \in [j, N]$, the set of curves that could represent $E_j(x)$ is $P_j(x) = \{p \in [0, j-1] : icost(i, x, p) = E_j(x)\}$. The relation $\rightarrow_x$ is a partial order on $P_j(x)$, and any $p$ in $P_j(x)$ having no $\rightarrow_x$-predecessors represents the first $j$ curves at $x$. If $p$ and $q$ both represented the first $j$ curves at $x$, with $p < q$, it would follow that $p \not\rightarrow_x q$ and $q \not\rightarrow_x p$. Lemma 2.3 would then imply that $q <_x p$, a contradiction. Thus, there is a unique maximal element of $P_j(x)$ under $\rightarrow_x$, and this element uniquely represents the first $j$ curves at $x$.

For $j \in [1, N]$ define the ordered candidate list

$$R_j^2 = \langle p \in [0, j-1] : p@_j x \text{ for some } x \in [j, N] \rangle, \text{ in decreasing order.} \tag{12}$$

Figure 3 gives an example. Suppose $p \in R_j^2$. For all $q \in [0, j-1] - p$, $q \not\rightarrow_x p$ for some $x$, so $q \not\rightarrow_j p$ by Lemma 2.2. Hence $p \in R_j^1$, proving that $R_j^2 \subseteq R_j^1$. Thus, by breaking ties with $\rightarrow_x$ we guarantee that Method 2's candidate list is a subset of Method 1's. Moreover, the following result shows that the first element of the candidate list is optimal for position $(i, j)$.

LEMMA 4. *If $w$ is concave and $j \in [1, N]$, then $I(i, j) = icost(i, j, R_j^2[1])$.*

*Proof.* Lemma 3 implies that $I(i, j) = icost(i, j, R_j^1[1])$, so we need only show that $R_j^2[1] = R_j^1[1]$. $R_j^1[1]@_j j$ because $R_j^1[1] \leqslant_j q$ for all $q \in [0, j-1]$, and $q \not\rightarrow_j R_j^1[1]$ whenever $q \neq R_j^1$ by the definition of $R_j^1$. Thus, $R_j^1[1] \in R_j^2$. But $R_j^2 \subseteq R_j^1$ so $R_j[1]$, being the largest element of $R_j^1$, must be the largest member of $R_j^2$. ■

We now turn to showing that the $p$-curves in $R_j^2$ form a piece-wise representation of the $E_j$-curve. Since $R_j^2 \subseteq R_j^1$, every curve in $R_j^2$ properly intersects every other in the interval $[j, N]$. For $p > q$ whose curves properly intersect on $[j, N]$, define the 'crossing point of $p$ and $q$' as follows:

$$p \times q = \max\{x \in [j, N] : p <_x q\}. \tag{13}$$

Note that $p \times q \in [j, N-1]$ and by Lemma 1 $p <_x q$ for $x \leqslant p \times q$ and $q \leqslant_x p$ for $x > p \times q$. Suppose $R_j^2 = \langle p_1, p_2, \ldots, p_n \rangle$, i.e. $p_k = R_j^2[k]$. Define the partition point, $\chi(p_k)$, of $p_k$ for all $k$ as follows:

$$\chi(p_k) = \begin{cases} p_k \times p_{k+1} & \text{if } k < n \\ N & \text{if } k = n \end{cases}. \tag{14}$$

Since every pair of curves in $R_j^2$ properly intersect on $[j, N]$ it follows that $\chi(p_k)$ is properly defined and an element of $[j, N]$. Figure 3 illustrates the definition of $\chi$.

LEMMA 5. *Suppose that w is concave, $j \in [1, N]$, and $R_j^2 = \langle p_1, p_2, \ldots, p_n \rangle$. Let $\tau(p_1) = j$ and $\tau(p_k) = \chi(p_{k-1}) + 1$ for $k > 1$. Then:*

1. *For $k \in [1, n-1]$, $\chi(p_k) < \chi(p_{k+1})$.*
2. *For $k \in [1, n]$, $p_k @_j x$ for all $x \in [\tau(p_k), \chi(p_k)]$.*

*Proof.* For $k > 1$, $p_{k-1} <_{\chi(p_{k-1})} p_k$ and Lemma 1.1 imply that $p_{k-1} <_x p_k$ for $x < \chi(p_{k-1})$. Thus $p_k$ *cannot* represent the first $j$ curves for all $x < \tau(p_k)$. For $k < n$, $p_{k+1} <_{\chi(p_k)+1} p_k$ and Lemma 1.1 imply that $p_{k+1} \leqslant_x p_k$ for $x > \chi(p_k)$. Moreover, since $p_{k+1} <_N p_k$ ($p_k > p_{k+1}$ properly intersect), it follows that $p_{k+1} \rightarrow_x p_k$ for $x > \chi(p_k)$. Thus $p_k$ *cannot* represent the first $j$ curves for any $x > \chi(p_k)$. Now $p_k \in R_j^2$, so it must represent the first $j$ curves at some $x$. By the above it must be that $\tau(p_k) \leqslant x \leqslant \chi(p_k)$ for such an $x$. Thus Lemma 5.1 follows as $\chi(p_{k-1}) < \tau(p_k) \leqslant \chi(p_k)$ and Lemma 5.2 follows as $p_k$ is the only candidate that could represent the curves in the interval $[\tau(p_k), \chi(p_k)]$. ■

Lemma 5.1 shows that the intervals $[\tau(p_k), \chi(p_k)]$ form a partition of $[j, N]$ and Lemma 5.2 shows that the $p_k$-curve represents $E_j(x)$ on the interval $[\tau(p_k), \chi(p_k)]$. Thus, $R_j^2$ together with the list $\langle \chi(p_k) \rangle$ constitutes a piece-wise representation of the $E_j$-curve. Method 2 uses the following procedure to update $R_j^2$. Method 1's exhaustive check for curves that cross between columns $j-1$ and $j$ is avoided, and the dominant time complexity is shifted to the binary search computing $(j-1) \times T[1]$.

"The *Iupdate* Procedure for Method 2"

```
procedure Iupdate²(T, j)
  {if j = 1 then
     {T ← ⟨0⟩
      χ(0) ← N
     }
   else
     {if j > χ(T[1]) then
        T ← T ~ T[1]
      if j − 1 <_j T[1] or j − 1 ≤_N T[1] then
```

```
        {while |T|>0 and (j-1<_{χ(T[1])}T[1] or j-1≤_N T[1]) do
            T←T~T[1]
         if |T|=0 then
            χ(j-1)←N
         else
            χ(j-1)←(j-1)×T[1]
         T←⟨j-1⟩·T
      }
   }
  return T
}                                                          (15)
```

THEOREM 2. *Let w be concave. Then* $R_1^2 = Iupdate^2(R, 1)$ *for any initial value of* $R$, *and* $R_j^2 = Iupdate^2(R_{j-1}^2, j)$ *for* $j \in [2, N]$.

*Proof.* $R_1^2 = \langle 0 \rangle = Iupdate^2(R, 1)$, so consider the call $Iupdate^2(R_{j-1}^2, j)$ for $j \in [2, N]$. By the definition of $R_{j-1}^2$, the initial value of $T$ satisfies

$$T = \langle p \in [0, j-2] : p@_{j-1}x \text{ for some } x \in [j-1, N] \rangle. \tag{16}$$

Define

$$Q = \langle p \in [0, j-2] : p@_{j-1}x \text{ for some } x \in [j, N] \rangle. \tag{17}$$

The initial values of $\chi$ are inherited from the previous call to $Iupdate^2$ (i.e. associated with $R_{j-1}^2$), so the definition of $\chi$ implies $\chi(T[1]) \in [j-1, N]$. It follows from Lemma 5 that if $\chi(T[1]) = j-1$ then $Q = T \sim T[1]$, otherwise $Q = T$. Thus, the lines

```
if j>χ(T[1]) then
   T←T~T[1]                                                (18)
```

result in $T = Q$.

We next show that the test

```
if j-1<_j T[1] or j-1≤_N T[1] then                         (19)
```

correctly decides whether $j-1 \in R_j^2$. First, suppose that $j-1$ satisfies the test (and hence is inserted in $T$ by $Iupdate^2$). $Q[1]$ represents the first $j-1$ curves at $j$, so $Q[1] \leqslant_j q$ for all $q \in [0, j-2]$. If $j-1 <_j Q[1]$, then for every $q \in [0, j-2]$ we have $j-1 <_j q$ and $q \not\to_j j-1$, so $j-1 @_j j$ and $j-1 \in R_j^2$. If $j-1 \leqslant_N Q[1]$, then $j-1 \leqslant_j Q[1]$ by Lemma 1.3 and $j-1 \to_j Q[1]$ by Lemma 2.1. Thus for every $q \in [0, j-2]$ we have $j-1 \leqslant_j q$ and $q \not\to_j j-1$ as $Q[1] \not\to_j j-1$ and $q \not\to_j Q[1]$ if $q \neq Q[1]$. Once again $j-1 @_j j$ and $j-1 \in R_j^2$. Conversely, suppose that the test

fails, i.e. $Q[1] \leqslant_j j-1$ and $Q[1] <_N j-1$. Then $Q[1] \rightarrow_j j-1$ by Lemma 2.1, so $j-1 \notin R_j^1$ and, hence, $j-1 \notin R_j^2$.

If the test fails then $j-1 \notin R_j^2$ implies that $j-1$ doesn't represent the $E_j$-curve anywhere. Thus the $E_{j-1}$- and $E_j$-curves are identical over $[j, N]$. So in this case, *Iupdate*$^2$ returns $T = Q = R_j^2$.

If the test succeeds then clearly $j-1 \in R_j^2$ and $R_j^2 - \{j-1\} \subseteq Q$. So what remains is to show that the **while** loop of *Iupdate*$^2$ correctly discards elements from $Q$. First, suppose that $p$ is discarded, i.e. $j-1 <_{\chi(p)} p$ or $j-1 \leqslant_N p$. If $j-1 <_{\chi(p)} p$, then $j-1 <_x p$ for all $x \in [\tau(p), \chi(p)]$ by Lemma 1.2. Thus $p$ *cannot* represent the first $j$ curves for any $x$ in the range where it represented the first $j-1$ curves. So $p$ cannot represent the first $j$ curves for all $x$, i.e. $p \notin R_j^2$. If $j-1 \leqslant_N p$, then $j-1 \rightarrow_j p$ and $p \notin R_j^1$, hence $p \notin R_j^2$. Conversely, suppose $p$ is not removed from $Q$ by the **while** loop. Then some $q$ preceding or equaling $p$ in $Q$ satisfies $q \leqslant_{\chi(q)} j-1$ and $q <_N j-1$. It follows by induction that $p \leqslant_{\chi(p)} j-1$ and $p <_N j-1$, so $j-1 \not\rightarrow_{\chi(p)} p$. Since the definition of $Q$ implies that $p @_{j-1} \chi(p)$, $p$ represents the first $j$ curves at $\chi(p)$, and hence $p \in R_j^2$. ■

Hirschberg and Larmore (1987) developed an algorithm related to Method 2 for solving what they call the 'least weight common subsequence problem'. Phrased in the terminology of this paper, their work differs from ours as follows. Least weight subsequence problems treat a family of $p$-curves that are each convex, i.e. concave upward. A $p$-curve is generally not a simple translation of a $q$-curve and a condition more complicated than the convexity of $w_k$ is needed to guarantee that two curves cross at most once (in the sense of Lemma 1). Hirschberg and Larmore's algorithm is not entirely symmetric with Method 2 because they must delete candidates from both ends of the list and because Method 2 is crafted so that its lists are sublists of Method 1's. Also, Hirschberg and Larmore prefer recomputation of crossing points to saving $\chi(p_k)$. Finally, a single application of their algorithm solves the least weight subsequence problem, whereas Method 2 is applied repeatedly when two sequences are compared.

7. *Performance.* Apart from the time spent in *Iupdate* and *Dupdate*, our candidate list algorithms spend $O(MN)$ time in the outline of Section 4. To formally account for the aggregate performance of the update calls, let $R_{i,j}$ and $S_{i,j}$ be the conservative lists for position $(i, j)$. With this notation the average candidate list size, $\bar{T}$, of a given method is $\Sigma_{i,j}(|R_{i,j}| + |S_{ij}|)/(2MN+M+N)$. Note that $\bar{T} \in [1, (M+N+2)/4]$ and $\bar{T}^1 \geqslant \bar{T}^2$ ($\bar{T}^m$ is the average candidate list size for method $m$). This parameter influences Method 1's time complexity and the space consumption of both methods. The efficiency of Method 2 depends on the number of intersection computations, which occur only when $|R_{i,j}| \geqslant 2$ and $j-1 \in R_{i,j}$. Suppose $B$ intersections are required for an application of the algorithm. Then $\bar{X} = B/(2MN+M+N)$ is the frequency with which an

intersection computation is needed in an update call. Note that $\bar{X} \in [0, 1]$, i.e. if an intersection is never needed then $\bar{X} = 0$ and at worst one is needed in every call, implying $\bar{X} = 1$.

LEMMA 6. *Method* 1 *takes* $O(\bar{T}^1 MN)$ *time and Method* 2 *takes* $O((1 + \bar{X} lg(M + N))MN)$ *time.*

*Proof.* An examination of *Iupdate*$^1$ shows that a call, *Iupdate*$^1(R_{i,j-1}, j)$, takes $O(|R_{i,j-1}|)$ time if $j > 1$ and $O(1)$ time if $j = 1$. Since $|R_{i,N}| = 1$, it follows that $O(\Sigma_{j=1}^{N} |R_{i,j}|)$ time is spent in *Iupdate*$^1$ for a given row $i$. A similar argument holds for *Dupdate*$^1$ and columns. Thus the total time spent updating lists in Method 1 is $O(\Sigma_{i,j} |R_{i,j}| + |S_{i,j}|) = O(\bar{T}^1 MN)$.

Computing $p \times q$ requires finding the smallest $x \in [p+1, N]$ such that $icost(i, x, p) - icost(i, x, q) \geqslant 0$. An $O(lg(N-p))$ binary search may be employed as $icost(i, x, p) - icost(i, x, q)$ monotonically increases in $x$. Thus a given call to *Iupdate*$^2$ spends $O(lgN)$ time if an intersection computation is needed, on the order of the number of elements deleted by the **while** loop, and a constant amount of time on the rest. Since an element can be deleted at most once, it follows that the amount of time spent in the **while** loop when amortized over a given row is $O(N)$. Thus, over the entire algorithm, $O(MN)$ time is spent in *Iupdate*$^2$ excluding the time needed for computing intersections. The same statement holds for the time spent in *Dupdate*$^2$. Letting $B$ be the total number of binary searches, Method 2 spends $O(MN + Blg(M+N)) = O((1 + \bar{X} lg(M+N))MN)$ time updating candidate lists. ■

Since $\bar{X} \leqslant 1$, Lemma 6 implies that Method 2 takes $O(MNlg(M+N))$ time in the worst case. Thus Method 2 is asymptotically superior to the $O(MN(M+N))$ WSB algorithm and only a log factor worse than Gotoh's algorithm for affine weighting functions (Gotoh, 1982). In terms of $M$ and $N$, Method 1's worst case performance is the same as the WSB algorithm. However, the empirical tests that follow, reveal that $\bar{T}^1$ grows very slowly as $M$ and $N$ increase. Thus Method 1's performance is slightly worse than $O(MN)$ in practice. On the other hand, $\bar{X}$ is rarely smaller than 0.1, implying that Method 2's worst-case and expected-case behavior are the same. So which method is superior in practice depends on the interplay between $\bar{X}$ and $\bar{T}^1$.

The empirical performance of Methods 1 and 2 will be compared with Waterman's method (Waterman, 1984). This method computes candidate lists $R_j^w = \langle p \in [0, j-1] : p \leqslant_{p+1} q \text{ for all } q < p \rangle$ for column position $j$ in a given row. It then follows that candidate lists increase in size across rows (i.e., $R_j^w \supseteq R_{j-1}^w$), and Method 1's lists are always a subset of Waterman's (i.e. $R_j^w \supseteq R_j^1$). Since this method takes $O(\bar{T}^w MN)$ time and $\bar{T}^w \geqslant \bar{T}^1$, Method 1 is asymptotically superior. However, candidate lists are simple stacks for Method 2 and Waterman's method, whereas stacks with arbitrary element deletion are

needed for Method 1. These implementation issues along with the statistical behavior of the $\bar{T}$ and $\bar{X}$ quantities, leave open the question of superiority in practice.

Figures 4A–F give a glimpse of the expected behavior of the $\bar{T}$ and $\bar{X}$ variables as a function of six parameters that control the nature of the input sequences, substitution costs, and weighting function. For each setting of the parameters, the expected value of each variable was estimated with 100 trials. Three parameters—$N>0$, $S\in[0, 1]$, and $\Sigma>1$—control the length, similarity, and underlying alphabet size of the input sequences. For each trial, two sequences of length $N$ were obtained in the following manner. First a sequence of length $SN$ was generated by randomly selecting symbols from a $\Sigma$-symbol alphabet with uniform probability. Then two sets of $(1-S)N$ random insertions (locations and symbols chosen uniformly) were performed on two copies of this common sequence to form the $N$-symbol inputs for the trial. Note that when $S=0$, the inputs are uncorrelated, and at the other extreme, they are identical when $S=1$.

A class of weighting functions was specified by two parameters: $C\in[0, 2]$ and $R>0$. For a given $C$ and $R$, the function $w^{C,R}$ is the parabola $x(2-x)$ linearly scaled so that domain $[0, C]$ is mapped to $[1, N]$ and its range over this domain is mapped to $[1, RN]$, i.e.

$$w_k^{C,R} = \begin{cases} 1+\dfrac{(RN-1)}{2-C}\dfrac{k-1}{N-1}\left(2-C\dfrac{k-1}{N-1}\right) & \text{if } C\leqslant 1 \\ 1+(RN-1)\,C\dfrac{k-1}{N-1}\left(2-C\dfrac{k-1}{N-1}\right) & \text{if } C>1. \end{cases} \tag{20}$$

All functions have $w_1=1$. $C$ controls the concavity of the function: for $C=0$ it is affine, for $C\leqslant 1$ it is still strictly increasing; for $C>1$ it 'peaks', reaching a maximum when $k=(N-1)/C+1$; and for $C=2$ it is so concave that $w_N=1$. The parameter $R$ controls the maximum gap penalty which is $RN$ regardless of $C$. Finally, substitution costs were zero for identical symbols (i.e. $\delta(a, a)=0$), and $\Delta>0$ for all other symbol pairs (i.e. $\delta(a, b)=\Delta$).

Years of computer time would be required to thoroughly explore the dependence of the expected values of $\bar{T}$ and $\bar{X}$ on the six parameters. In 28 user hours on a VAX 8600, we obtained the plots in Fig. 4A–F for which one parameter was varied and the others held constant. Every such 'slice' passes through the point $(N, S, \Sigma, C, R, \Delta)=(100, 0.85, 10, 1.3, 0.1, 1.0)$ shown as a solid circle in each figure. Combined with some additional experimentation, several observations and conjectures arose.

(i) Candidate list sizes are modest for both our methods over the range of experiments in Fig. 4: $\bar{T}^1$'s largest value was 5.73 and $\bar{T}^2$'s was 2.01. On the other hand, $\bar{X}$ is substantial (0.1 or more) except when $S$ or $C$ is small.

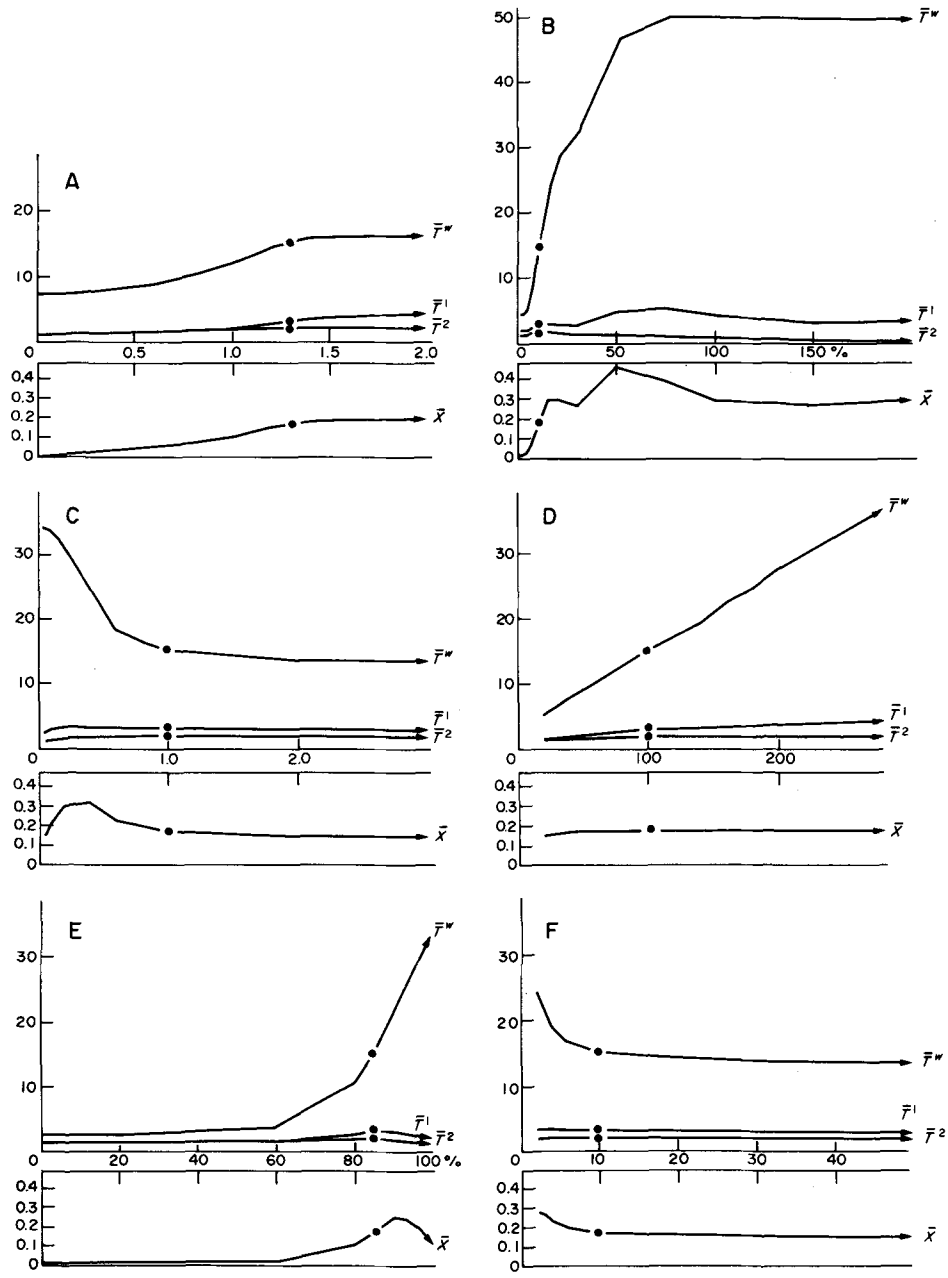


Figure 4. Slices through $(N, S, \Sigma, C, R, \Delta) = (100, 0.85, 10, 1.3, 0.1, 1.0)$. (A) Concavity, $C$; (B) range, $R$; (C) substitution cost, $\Delta$; (D) length, $N$; (E) similarity, $S$; (F) alphabet size, $\Sigma$.

(ii) The $\bar{T}$ variables for our methods appear to be bounded functions† of substitution cost, similarity and alphabet size. However, Waterman's candidate lists become $O(N)$ as $\Delta$ or $\Sigma$ decreases, and as $R$ or $S$ increases. In fact, we conjecture that his algorithm is $O(N^3)$ whenever $\Delta < \Delta w_1$ or $S$ is near 1.

(iii) Waterman's candidate lists appear to grow linearly with $N$ and not logarithmically as conjectured by Waterman (1984). Our lists grow at a decreasing rate; perhaps they are logarithmic or even bounded. Indeed, for experiments with $w_k = a + b \lg k$ and $w_k^{C,R/N}$ (i.e. the range $[1, R]$ of this $w$ is independent of $N$), $\bar{T}^1$ and $\bar{T}^2$ appear to be bounded functions of $N$, and $\bar{X}$ appears to vanish in the limit of $N$. We thus conjecture that our methods are $O(N^2)$ whenever the range of $w$ is not functionally dependent on $N$.

(iv) Methods 1 and 2 are provably $O(N^2)$ when $S = 1$ and $C \leqslant 1$ or when $C = 0$. The latter case, $w$ affine, implies that our methods subsume Gotoh's algorithm (Gotoh, 1982) as a special case.

Figure 5 plots the average running time of the algorithms for the 'slice' in Fig. 4D. The algorithms were written in *C* and run on a VAX 8600 running UNIX. For comparison, the time taken by the basic algorithm for single-symbol indels, Gotoh's algorithm for affine weighting functions, and the WSB algorithm are also shown. Waterman's algorithm exhibits $O(N^3)$ behavior as $\bar{T}^w$ grows linearly in $N$ in Fig. 4D. Both of our methods are much closer to the $O(N^2)$ curves for the basic and Gotoh algorithms. Method 2 is the clear winner in this case, but note that it crosses over with Method 1 for $N$ around 50 in the close-up at the lower left of Fig. 5. In other experiments, Method 2 was always faster than Method 1 for large enough $N$, but the crossover point varied between 30 and 200. Finally, observe that in Fig. 5 the generality of Method 2 costs a factor of less than 3 over Gotoh's algorithm although this factor is increasing logarithmically as $N$ increases.

To compute just the cost of an optimal conversion, only the current $N+2$ candidate lists and the current and previous rows of the *C*-matrix need be retained. Moreover, as shown in the next section, a divide-and-conquer technique permits one to deliver an optimal conversion in the same space and time efficiency as the cost-oriented problem. Since two rows of the *C*-matrix require $O(N)$ space, the main space constraint on our methods is $T^{peak}$ the maximum, over all positions, of the space consumed by the current $N+2$ candidate lists. For the experiments in Fig. 4, the largest consumption for Method 1 was $13.4(N+2)$ and $3.4(N+2)$ for Method 2. While $T^{peak}$ does not exactly correlate with $\bar{T}$, $T^{peak} \leqslant 3\bar{T}^1(N+2)$ and $T^{peak} \leqslant 2\bar{T}^2(N+2)$ for this range of experiments. Waterman's method requires $O(N^2)$ space for those

† We mean precisely that for a given setting of the other five parameters, there is a *constant* bounding $\bar{T}$ for all possible values of the parameter in question.

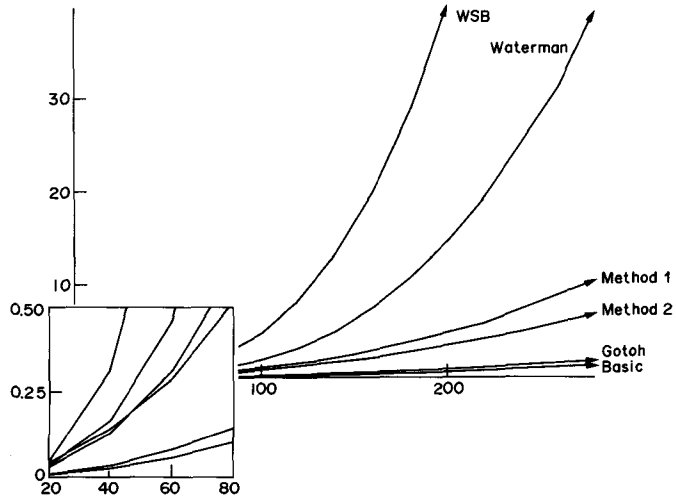


Figure 5. Running times as $N$ varies.

problems on which it performs poorly and always consumes more space than our methods.

In summary, Waterman's algorithm is competitive for small problems and some particular choices of input parameters. However, it is frequently cubic whereas both our methods exhibit near quadratic performance for all parameter choices. Methods 1 and 2 are competitive for mid-sized problems but Method 2 is superior for large $N$. Moreover, Method 2's space consumption is always the most parsimonious, a characteristic of importance when optimal conversions must be delivered. We conclude that Method 2 is the preferred algorithm, both theoretically and practically.

*8. Variations.* This section begins by sketching how the algorithms given in this paper can be applied to compute optimal sequence alignments. Then, the efficiency of Method 2 is improved for two classes of concave weighting functions, and Methods 1 and 2 are extended to a somewhat wider class of weights. The bulk of the section is devoted to showing how the space requirements of Method 2 can be reduced to $O(N)$ in expectation, when an optimal conversion (not merely its cost) is desired. Similar results hold for Waterman's method and Method 1. Finally, some open problems are mentioned.

*8.1. Alignments.* In the biological literature, an optimal alignment between sequences $A$ and $B$ is often desired (Needleman and Wunsch, 1970). The WSB algorithm can be used to compute optimal alignments using the rule that the gap penalty $w'_k$ for $k$-symbol gaps in the alignment corresponds to the indel weight

$$w_k = w'_k + \tfrac{1}{2}\mu k$$

where $\mu$ is the maximum of $\delta(a, b)$ over all symbols $a$ and $b$. For details, see Smith *et al.* (1981). The relation between $w'_k$ and $w_k$ is important for this paper because it preserves concavity. Thus, the algorithms given in this paper can be used to compute optimal alignments with 'concave' gap penalties.

*8.2. Analytically expressible crossing points.* The dominant cost of Method 2 is the $O(lg\ N)$ computation of $p \times q$. This cost can be reduced for two practical types of concave weighting functions. First, consider $w$ for which an analytic expression $k^*(x, y)$ can be formulated for the minimum solution to the equation $w_k = w_{k-x} + y$ where $x > 0$. For example, if $w_k = a + b lg\ k$ then $k^*(x, y) = x/(1 - 2^{-y/b})$. Given an expression for $k^*$, $p \times q = q + [k^*(p-q, C(i, p) - C(i, q))] - 1$ can be computed in $O(1)$ time. Thus, for 'analytically solvable' concave weighting functions, Method 2 can be modified to take $O(MN)$ worst-case time.

*8.3. Piece-wise affine curves.* A complexity reduction is also possible for concave weighting functions that are $P$-piece affine curves, i.e. the domain of $w$ can be partitioned into $P$ intervals such that an affine function delivers the values of $w$ in each interval. For example, the curve of Fig. 1 is a 3-piece affine curve. Formally, let a $P$-piece affine curve be specified by a $P$-element list of 'segments', $\langle(\tau_1, a_1, b_1), \ldots (\tau_P, a_P, b_P)\rangle$, where $\tau_1 = 1$ and $\tau_f < \tau_{f+1}$. For convenience, introduce $\chi_f$ equal to $\tau_{f+1} - 1$ if $f < P$, and $\infty$ if $f = P$. For $k$ in segment $f$'s interval $[\tau_f, \chi_f]$, $w_k$ is given by the affine function $\omega_f(k) = a_f + b_f k$. We show that for concave piece-wise affine curves, $k^*(x, y)$ can be found in $O(lg\ P)$ time.

Since $p \times q$ is computed only for properly intersecting curves, we can assume the equation $w_k = w_{k-x} + y$ has a minimum solution, $k^*$, in the interval $[1, N-1]$. Let $g$ be the minimum segment for which the equation $\omega_g(k) = w_{k-x} + y$ has a solution in the interval $[\tau_g, \chi_g]$ and let $k^*_g$ be the minimum such solution. Similarly, let $h$ be the minimum segment such that $w_k = \omega_h(k-x) + y$ has a minimum solution $k^*_h$ in the interval $[\tau_h + x, \chi_h + x]$. Because $\omega_g(k) = w_k$ for $k \in [\tau_g, \chi_g]$ and $\omega_h(k-x) = w_{k-x} + y$ for $k \in [\tau_h + x, \chi_h + x]$, it follows that $k^* = k^*_g = k^*_h$. But then $k^*$ is the minimum solution not less than $\max(\tau_g, \tau_h + x)$ to the equation $\omega_g(k) = \omega_h(k-x) + y$. Thus, $k^*$ can be computed analytically in $O(1)$ time once $g$ and $h$ have been found. Since $w$ is concave, a segment $f$ for which $w_{\tau_f} \geqslant w_{\tau_f - x} + y$ and $w_{\chi_f} \leqslant w_{\chi_f - x} + y$ must have a solution to the equation $\omega_f(k) = w_{k-x} + y$ in the interval $[\tau_f, \chi_f]$. Thus $g$ is the smallest segment satisfying this condition and it may be found with an $O(lg\ P)$ binary search over the segment list of $w$ since $w_k - w_{k-x}$ is strictly decreasing in $k$ by the concavity

of $w$. Similarly, $h$ is the smallest segment satisfying the condition, $w_{\tau_h} \geqslant w_{\tau_h + x} - y$ and $w_{\chi_h} \leqslant w_{\chi_h + x} - y$, and so can be found in $O(lg\ P)$ time. Thus when $w$ is $P$-piece affine, $p \times q$ can be computed in $O(lg\ P)$ time improving Method 2 to only $O(MNlg\ P)$ worst-case time.

*8.4. Ultimately concave weighting functions.* In the direction of greater generality, both methods can be extended to handle weighting functions that are *concave after* $K$, i.e. $\Delta w_k \geqslant \Delta w_{k+1}$ for all $k \geqslant K$. This class of curves permits the first $K$ values of $w$ to be arbitrary. The methods correctly handle the case where $K=1$, i.e. $w$ is concave. To treat larger $K$ it suffices to treat indels of length less than $K$ as in the WSB algorithm, and those of greater length with the candidate list paradigm. Thus a row candidate list $R$ for position $(i, j)$ will contain candidates in the range $[0, j-K]$. The impact on the formalisms is that $<_j, \rightarrow_j$ and $@_j$ are all relations with domain $[0, j-K]$. Algorithmically, the expression $j-1$ is replaced with $j-K$ and the predicate $j=1$ is replaced with $j=K$ in the *Iupdate* procedures. Then for $i, j \geqslant K$:

$$\begin{aligned} C(i,j) = \min\{ & \min\{icost(i,j,R[1]),\ dcost(i,j,S(j)[1],\ C(i-1,j-1)+\delta(a_i,b_j)\}, \\ & \min\{icost(i,j,p) : p \in [j-K+1, j-1]\}, \\ & \min\{dcost(i,j,p) : p \in [i-K+1, i-1]\} \\ & \}. \end{aligned} \tag{21}$$

A simple exercise yields similar expressions for the $i<K$ and $j<K$ boundary cases. The additional min-terms in the expression for $C(i, j)$ require $O(K)$ time implying that $O(KMN)$ time is spent in the outline of the algorithms. The time spent in *Update* calls remains unchanged. Thus, for weighting functions concave after $K$, Method 1 yields an $O((K+\bar{T}^1)MN)$ algorithm, and Method 2 an $O((K+\bar{X}lg(M+N))MN)$ algorithm.

*8.5. Optimal conversions in linear space.* In Section 7 it was noted that the cost of an optimal conversion can be computed in $O(T^{peak})$ space, but delivering such a conversion naively requires $O(MN)$ space. The application of the divide-and-conquer technique of Hirschberg (1975) gives candidate list algorithms that deliver an optimal conversion with only $O(T^{peak} + lg\ M)$ space and no change in their asymptotic time complexities. This reduction is especially desirable since in practice it is space consumption that limits the maximum problem solvable on a given machine.

All the algorithms discussed in this paper compute optimal conversions for progressively longer prefixes of the input sequences. One could equally well formulate algorithms that proceed by considering progressively longer suffixes. Specifically, let $C^R(i, j)$ be the cost of an optimal conversion of $a_{i+1}a_{i+2} \ldots a_M$ to $b_{j+1}b_{j+2} \ldots b_N$. The 'reverse' analog of a 'forward' algorithm computes $C^R$ in decreasing order of $i$ and $j$ in a row major fashion. Designing the reverse

algorithms is left as a simple exercise. The discussion now focuses on the modification of Method 2, but note that Method 1 and Waterman's method can be similarly treated.

Hirschberg's central idea is to compute the 'mid-point' of an optimal conversion using the forward and reverse cost-only algorithms in $O(T^{peak})$ space. The optimal conversion can then be delivered by recursively determining the optimal conversions on both sides of its mid-point. Observe that any conversion of $A$ to $B$ must either (i) convert $a_1 \ldots a_p$ to $b_1 \ldots b_j$ and $a_x \ldots a_M$ to $b_{j+1} \ldots b_N$ for $p = M/2 = x$ and some $j$, or (ii) convert $a_1 \ldots a_p$ to $b_1 \ldots b_j$, delete $x-p$ symbols, and convert $a_{x+1} \ldots a_M$ to $b_{j+1} \ldots b_N$ for some $j$, $p < M/2$, and $x > M/2$. Term $(p, j, x)$ is a mid-point of type 1 in case (i), and of type 2 in case (ii). Let the cost of a mid-point be:

$$Cost(p, j, x) = \begin{cases} C(p, j) + C^R(x, j) & \text{if } p = x \text{ (type 1)} \\ dcost(p, j, x) + C^R(x, j) & \text{if } p < x \text{ (type 2)} \end{cases}. \tag{22}$$

Note that if the minimum cost of converting $A$ to $B$ is $C$ then $C = Cost(p, j, x)$ for the mid-point of an optimal conversion. Also, for such an optimal mid-point, $p$ must be a column candidate at position $(x, j)$ and without loss of generality must represent the first $x$ curves at $x$. But since $p < M/2$, it follows that $p$ is a column candidate for position $(M/2, j)$ and represents the first $M/2$ curves at $x$. Thus given $j$ and $x$ of an optimal mid-point, there is only one choice $\rho(j, x)$ for $p$. Specifically, if $x = M/2$ then $\rho(j, x) = M/2$, otherwise $\rho(j, x)$ is the unique candidate in $S_{M/2,j}$ for which $x \in [\tau(p), \chi(p)]$ (recall $S_{M/2,j}$ is the column candidate list for position $(M/2, j)$). Thus to find the mid-point of an optimal conversion it suffices to find a triple $(\rho(j, x), j, x)$ of minimum cost, i.e.

$$C = \min\{Cost(\rho(j, x), j, x) : j \in [0, N] \text{ and } x \in [M/2, M]\}. \tag{23}$$

This requires $(M/2+1)(N+1)$ comparisons given that $Cost$ is available for the relevant mid-points.

To find an optimal mid-point, first compute $C(M/2, j)$ and $S_{M/2,j}$ for all $j$ with the forward algorithm. This takes $O(T^{peak})$ space using two vectors for the current and previous $C$ rows and associating the value $C(p, j)$ needed for evaluating $dcost(p, j, x)$ with the record for candidate $p$. Then compute $C^R(M/2, j)$ using a similar version of the reverse algorithm and simultaneously record the minimum mid-point of type 2. This can be done, as at the time $C^R(x, j)$ is computed, $p = \rho(j, x)$ can be found in $O(1)$ time by 'walking' $S_{M/2,j}$ in the reverse direction as $x$ decreases and $dcost(p, j, x)$ is available in the record for candidate $p$. Finally, the type 1 mid-points are evaluated as the relevant $C$ and $C^R$ values are available. Only $O(MN)$ time is spent recording the optimal mid-point, so the dominant complexity is for the forward and reverse algorithms which take $O(MN lg(M+N))$ time and $O(T^{peak})$ space.

Given an optimal mid-point $(p, j, x)$, an optimal conversion of $A$ to $B$ is an optimal conversion of $a_1 \ldots a_p$ to $b_1 \ldots b_j$, followed by a deletion of $x-p$ symbols if $p \neq x$, followed by an optimal conversion of $a_{x+1} \ldots a_M$ to $b_{j+1} \ldots b_N$. Thus for $N$, $M > 1$, an optimal $M$-by-$N$ conversion can be obtained by finding an optimal mid-point, and then recursively finding an optimal $p$-by-$j$ conversion and an optimal $(M-x)$-by-$(N-j)$ conversion. For the boundary case $M \leqslant 1$, $O(N)$ time suffices to evaluate all possible conversions of $A$ to $B$. Similarly, $O(M)$ time suffices when $N \leqslant 1$. Thus for large enough $c$ the time, $T(M, N)$, to compute an optimal $M$-by-$N$ conversion satisfies the recurrence inequality:

$$T(M, N) \leqslant \begin{cases} cMN\lg(M+N) + T(p, j) & \text{if } N, M > 1 \text{ and } 0 \leqslant p, x \leqslant M/2 \\ \quad + T(x, N-j) & \text{and } 0 \leqslant j \leqslant N \\ c(M+N) & \text{if } N \leqslant 1 \text{ or } M \leqslant 1. \end{cases} \tag{24}$$

It follows that $T(M, N) \leqslant 2cMN\lg(M+N) + c(M+N)$, i.e. an optimal conversion is delivered in $O(MN\lg(M+N))$ worst-case time. Each recursive invocation needs to locally record only its mid-point as the $O(T^{peak})$ candidates and $O(N)$ row vectors needed for a mid-point computation may be discarded once the mid-point is known. Moreover, the recursion depth is at most $O(\lg M)$. Thus the algorithm requires $O(T^{peak} + \lg M)$ space: $O(\lg M)$ for a recursion stack and $O(T^{peak})$ for a globally-shared mid-point computation structure. Since in practice $T^{peak}$ is linear in $N$ for Method 2, this variation delivers an optimal script in $O(N)$ space in expectation.

A software package implementing this linear space variation of Method 2 is available from the authors upon request. It uses $o(40N + 15T^{peak})$ bytes of memory and over a range of experiments was never more than 1.98 times slower than the cost-only version timed in Section 7. Estimating $T^{peak}$ at $5N$ seems sufficient for all problems and implies a space consumption of $o(120N)$ bytes in terms of $N$ alone. A problem with $M = N = 4{,}000$ required 370 Kbytes and 28 minutes on a VAX 8600. Configured with one megabyte of memory the package will handle problems as large as $M = N = 8{,}500$.

In our experience, Hirschberg's technique appears applicable to most sequence comparison algorithms giving rise to linear-space, conversion-delivering variations that are never more than twice as slow as their cost-only counterparts. Also, the space consumption of these variations is asymptotically and practically much superior to methods based on the 'trace back' technique. For these reasons, we highly recommend this approach to software implementors. For example, Gotoh's affine gap penalty algorithm when so refined, uses only $o(16N)$ bytes of memory to deliver a conversion. Configured with a megabyte of memory, problems as large as $M = N = 64{,}000$ can be solved.

*8.6. Open problems.* For weighting functions where $\Delta w_k$ approaches 0 rapidly as $k$ increases, an optimal alignment consists of a sequence of disjoint locally optimal alignments separated by large gaps. Intuitively, this is because once a sufficiently large gap is allowed, increasing its size does not substantially increase the score. This suggests that concave weighting functions may be useful in detecting local homologies within a global alignment. However, the use of concave gap penalties in conjunction with other local homology techniques (Smith and Waterman, 1981; Sellers, 1984), though technically feasible, has not yet been shown to have practical value.

In the RNA secondary structure problem (Zuker and Sankoff, 1984), thermodynamic experiments reveal the destabilizing free energy of loops to be a concave function of their length. Waterman and Smith (1986) have achieved practical efficiency gains by applying Waterman's (1984) method for concave weights. Application of our methods is expected to produce further efficiency improvements. However, unlike the sequence comparison context, it is an open problem whether our approach attains a worst-case asymptotic improvement for the RNA secondary structure problem.

Several additional issues remain to be investigated. First, it appears that an $O(CN)$ algorithm is possible where $C$ is the cost of an optimal conversion. The algorithms of Fickett (1984) and Ukkonen (1985) appear to generalize with our methods for handling concave weights. The technical details need to be addressed and the utility of the complex algorithms that arise needs to be assessed. In another direction, a point of concern is the effect of numerical instability on our algorithms: can the use of finite precision arithmetic cause the computed optimum conversion to drift away from the true optimum? Is it possible to efficiently compute the $K$ best conversions? Can arbitrary weighting functions be handled in less than $O(N^3)$ times? Are more than the class of concave functions needed in practical applications such as those in biology?

The authors are indebted to the reviewers for their helpful suggestions. Stephen Ntschul's 1987 Ph.D. dissertation at MIT, entitled *Aspects of Biological Sequence Comparison*, devotes several pages to a discussion of the $O(N^3)$ behaviour of Waterman's (1984) algorithm.

## LITERATURE

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. natn. 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.

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

Hirschberg, D. S. 1975. "A linear space algorithm for computing maximal common subsequences." *Communications of ACM* **18**, 341–343.

——— and L. L. Larmore. 1987. "The least weight subsequence problem." *SIAM J. on Computing* **16**, 628–638.

Miller, W. and E. W. Myers. 1986. "A simple row-replacement method." TR 86-28, Dept. of Computer Science, University of Arizona, Tucson, AZ 85721 (submitted to *Software—Practice and Experience*).

Myers, E. W. and W. Miller. 1988. "Row replacement algorithms for screen editors." *Trans. on Prog. Lang. and Systems* (in press).

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.

Sankoff, D. and J. B. Kruskal. 1983. *Time Warps, Strings Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Reading, MA: Addison-Wesley.

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

———, ——— and W. M. Fitch. 1981. "Comparative biosequence metrics." *J. molec. Evol.* **18**, 38–46.

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

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

——— and T. F. Smith. "Rapid dynamic programming algorithms for RNA secondary structure." *Adv. Appl. Math.* **7**, 455–464.

———, ——— and W. A. Beyer. 1976. "Some biological sequence metrics." *Adv. Math.* **20**, 367–387.

Zuker, M. and D. Sankoff. 1984. "RNA structures and their prediction." *Bull. math. Biol.* **44**, 591–621.

Received 17 July 1987
Revised 29 September 1987