Bulletin of Mathematical Biology Vol. 46, No. 4, pp. 473–500, 1984
Printed in Great Britain

0092-8240/84 $3.00 + 0.00
Pergamon Press Ltd
1984 Society for Mathematical Biology

GENERAL METHODS OF SEQUENCE COMPARISON

■ MICHAEL S. WATERMAN*
Departments of Mathematics and of Molecular Biology,
University of Southern California,
Los Angeles, CA 90089-1113, U.S.A.

Mathematical methods for comparison of nucleic acid sequences are reviewed. There are two major methods of sequence comparison: dynamic programming and a method referred to here as the regions method. The problem types discussed are comparison of two sequences, location of long matching segments, efficient database searches and comparison of several sequences.

1. Introduction. Nucleic acid data are increasing at a rapid rate and several international databases are being created to organize the information. The number of possible relationships between the various sequences is significantly larger than the number of sequences. To complicate further the problem of finding information within and relationships between sequences is the fact that, as molecular biology matures, the number of features of interest also increases. These trends do not show signs of abating.

Molecular biology celebrated its thirtieth birthday in 1983. As with any young science, new mathematical and analytical questions have been posed and sometimes solved. The solutions have not always been rigorous or mathematically sophisticated but are generated out of a need for answers. The present paper is an attempt to survey methods of comparing macromolecular sequences which is perhaps the most mathematically developed aspect of macromolecular sequence analysis.

Recently two issues of *Nucleic Acids Research*, Volume 10, No. 1 (1982) and Volume 12, No. 1 (1984), have been devoted entirely to computer methods for nucleic acid sequences. Analysis packages and methods of analysis are described in many of the papers and several are referred to in the present paper. Sequence comparison by the method of dynamic programming has recently been treated in a book edited by Sankoff and Kruskal (1983). The present treatment is focused on macromolecular sequences while the Sankoff and Kruskal book has a broad and fascinating range from biology to error correcting codes, spelling correction, geological sequences, speech recognition and birdsong studies. Techniques which are not dynamic

* This work was supported by a grant from the System Development Foundation.

programming are briefly mentioned or omitted in their book and very recent, important developments could not be included.

In this paper I take the point of view that sequence comparison is useful at various levels of refinements. For example, both rapid database searches and the most refined dynamic programming methods are important tools. To keep the discussion to a reasonable length I consider the comparison of two (or more) sequences to obtain an alignment or long similar regions. Palindromes, repeats, inverted repeats and helical regions will not be explicitly discussed. In addition, DNA sequences will be the example sequences, although any other linear macromolecular sequences could be used.

The paper is organized by problem type and methods of solution usually appear as sections. The outline is

The introduction concludes with a general and widely used method of analysis used in the problem of Sections 2–4.

1.1. Visual methods There is an important method of sequence analysis which is best described as "just look at it". An analyst might observe a long stretch of GCGCGC . . . and have found a feature of interest without any

sophisticated mathematical or computer methods. It is possible to formulate an analytical problem from such an observation: "How likely is the observation of such an event in random sequences?" A biologist is unlikely to be surprised, informed or impressed by being told that the GCGC. . . run was unusual. Instead the question of function of the GC region is the focus of the biologists attention. Just looking at sequences is useful.

The dot matrix method is a widely used visual method that generally utilizes computers. Here a matrix $M = (m_{ij})$ is formed where $m_{ij} = 0$ if the ith element of sequence a is unequal to the jth element of sequence b and $m_{ij} = 1$ otherwise. Runs of exact matches show up as diagonals of 1s. The dot matrix has been discovered independently several times and I do not know the history. See, for example, Maizel and Lenk (1981), Novotny (1982), Harr *et al.* (1982), Jagadeeswaran and McGuire (1982), and Gibbs and McIntyre (1970).

Many of these implementations filter the matches to display only runs of r or more, for example. It is possible to combine more sophisticated methods, described next, to filter matches.

The dot matrix method is useful in locating regions of high match between two DNA sequences. It is natural to ask for the probability distribution of the longest match between two sequences. This distribution will allow the analyst to locate the significant matches. In Section 3.1, a formula is given for the expectation and variance of the longest exact match and in Section 3.2 empirical and theoretical extensions are given for inexact matches. These methods allow the statistical significance to be estimated so that the analysis results are much less *ad hoc*.

2. Comparison of Two Sequences. The problem of this section can be described as that of finding the "smallest number" of steps which changes one sequence into another. The sequences are nucleic acid sequences: $\mathrm{a} = a_1 a_2 \ldots a_n$ and $\mathrm{b} = b_1 b_2 \ldots b_m$ where a_i and b_j are one of the nucelotides, adenine (A), thymine (T), guanine (G) or cytosine (C). A, T, G, C will also be referred to as bases. Therefore these sequences are finite words over a four-letter alphabet. The steps mentioned above correspond to evolution events which can alter the sequences. The simplest set of evolution events consists of mutations, where one letter is substituted for another, and insertions (or deletions) where one letter is inserted (or deleted) from a sequence.

If a = AATAG, then a substitution of T for a_2 = A transforms a into b:

$$\mathrm{a} = \mathrm{AATAG} \rightarrow \mathrm{b} = \mathrm{ATTAG}.$$

This correspondence is usually shown by representation in an alignment:

a: AATAG

b: ATTAG.

If a_4 = A is deleted, the transformation into c = AATG is represented by

a: AATAG

c: AATΔG,

where the deletion of a_4 = A is indicated by a "Δ" inserted into c. Correspondingly, the insertion of C between a_3 and a_4 is represented by

a: AATΔAG

e: AATCAG.

Kruskal and Sankoff (1983) place these problems in a general setting. Another general formulation is in Waterman *et al.* (1976), which is followed here. Let S be the set of finite words over a finite alphabet, including the empty word. Let $\tau = \{T \mid T : S \to S\}$ be a set of transformations which includes the identity transformations. Our interest is in sequences of transformations $T_1, T_2, \ldots, T_k$ from τ such that

$$T_1 o T_2 o \ldots o T_k(\mathbf{a}) = \mathbf{b}.$$

Various problems can be formulated based on this set-up. For the smallest number of steps problem, we are to search for minimum k where τ = {single-letter mutations, insertions and deletions}. The objective function can be changed and the set of transformations enlarged. There is a balance between biological reality and computable algorithms.

Several issues are treated in the subsections below. First I discuss relationships between distance and similarity (2.1). Then I take up dynamic programming algorithms for the simplest problem (2.3), some natural extensions (2.4), some recent important results for efficient calculation (2.5), a technique for solutions near the optimum (2.6) and finally some other techniques for solving these problems (2.7). I follow Kruskal (1983) and refer to insertion/deletions as indels.

One of the earliest string comparison algorithms was due to Levenshtein (1966) although this work did not influence the developments reported in this paper. The work of Fitch and Margoliash (1967) and Fitch (1969) brought the problems of sequence comparison to the attention of a large number of people, including Beyer *et al.* (1974).

2.1 Distance and similarity. When non-negative weights are assigned to the transformations, then the minimum sum of weights of $T_1 o T_2 o \ldots o T_k(\mathbf{a}) = \mathbf{b}$ can be viewed as the distance from **a** to **b**. This is made explicit in

Waterman *et al.* (1976) and, with obvious symmetrizing, a metric space is obtained. Interesting cases arise when τ is restricted to specific sets of transformations and specific weights. For example, it is important whether or not efficient algorithms exist.

For mutations, insertions and deletions, the alignment corresponding to the minimum sum of weights should be displayed as long as each sequence element is in no more than one evolutionary event. Letting $d(x, y)$ be the weight of substitution y for x, even having $d(x, x) = 0$ for all x does not make the minimum alignment sum of weights into a metric. If $d(\cdot\ ,\ \cdot)$ is a metric on the set of letters, then Sellers (1974b) shows that a metric on S results. In this case, the alignment can be displayed.

An earlier approach was taken by Needleman and Wunsch (1970) who present an algorithm for maximizing the number of matches minus the number of insertions and deletions. This is referred to as a maximum similarity criteria while the one above is a minimum distance criteria. Relating similarity and distance is important in psychology (Shepard, 1980) and the relationship is also interesting here. Sometimes $d(x, y) < 0$ is used and the resulting minimum referred to as a distance. Here I reserve distance to be the result of minimizing non-negative weights, with $d(x, x) = 0$. Situations with $d(x, y) < 0$ might be called negative similarity.

Each substitution or pair (a_i, b_j) in an alignment corresponds to one of the $k = 1, 2, \ldots, 16$ pairs (A, A) (A, T) . . . (G, G) and has a similarity $\alpha_k \geqslant 0$ or a distance $\beta_k \geqslant 0$. Indels are each given weight w for similarity or x for distance. The maximum similarity alignment coincides with the minimum distance alignment if and only if

$$\beta_i = \max_{1 \leqslant j \leqslant 16} (\alpha_j) - \alpha_i \quad \text{for } i = 1, 2, \ldots 16$$

and

$$x = \left(\max_{1 \leqslant j \leqslant 16} \alpha_j \right) / 2 + w.$$

A more general result than this with a complete proof appears in Smith *et al.* (1981).

Beyer *et al.* (1983) recently raise the question of relating similarity to distance for these algorithms. Beyond the relationships given above they ask for a general criteria for similarity which would be "complementary" to a metric distance. By careful examination of the above equations, I have (unpublished observations) accomplished that, for these specific algorithms. The general characterization of similarity and its relationship to metric distance remains a problem of interest.

2.2 Alignment combinatorics. As above let $\mathrm{a} = a_1 a_2 \ldots a_n$ and $\mathrm{b} = b_1 b_2 \ldots b_m$. An alignment can be produced by increasing the length of each

sequence with the insertion of Δs. If the length of such an alignment is L, then it can be written

$$a_1^* a_2^* \ldots a_L^*$$
$$b_1^* b_2^* \ldots b_L^*$$

where the subsequence of $\mathbf{a}^*(\mathbf{b}^*)$ of elements not equal to Δ is $\mathbf{a}(\mathbf{b})$.

It is of interest to count the number of alignments. If this number is not large, then a direct search is feasible for finding optimal alignments. As we will see, the number grows very rapidly.

An alignment of $a_1 \ldots a_n$ with $b_1 \ldots b_m$ can end in one of three ways

$$\begin{matrix} \ldots a_n & \ldots a_n & \ldots \Delta \\ \ldots \Delta & \ldots b_m & \ldots b_m \end{matrix}$$

(aligning Δ over Δ is eliminated, as it contributes no information). If $f(n, m)$ counts configurations generated by recursively ending alignments as above, then

$$f(n, m) = f(n-1, m) + f(n-1, m-1) + f(n, m-1).$$

The numbers generated by this recursion equation are known as the Stanton-Cowan (1970) numbers, where they arose from calculating the volume of a sphere of radius m in n dimensions using the Lee metric. Asymptotics is done for a generalization of these numbers by H. T. Laquer (1981). He shows that

$$f(n, n) \sim (1+\sqrt{2})^{2n+1}\sqrt{n}.$$

After close examination, however, $f(n, m)$ is seen to overcount the alignments or at least a reasonable definition of alignments. For example, the two alignments

$$\begin{matrix} A\ \Delta \\ \Delta\ T \end{matrix} \quad \text{and} \quad \begin{matrix} \Delta\ A \\ T\ \Delta \end{matrix}$$

might not be distinct in a biological sense. To design a recursion that does not double count these "tandem" deletions, let $g(n, m)$ be the number of such alignments. If an alignment ends in $\overset{a_n}{\Delta}$ there are three possibilities

$$\begin{matrix} \ldots a_{n-1}\ a_n & \ldots a_{n-1}\ a_n & \ldots \Delta\ \ a_n \\ \ldots b_m\ \ \Delta & \ldots \Delta\ \ \Delta & \ldots b_m\ \Delta \end{matrix}$$

and if an alignment ends in $\overset{\Delta}{b_m}$, there are three possibilities

$$\begin{matrix} \ldots a_n\ \ \Delta & \ldots \Delta\ \ \Delta & \ldots a_n\ \Delta \\ \ldots b_{m-1} b_m & \ldots b_{m-1} b_m & \ldots \Delta\ b_m \end{matrix}$$

Therefore

$$g(n, m) = g(n-1, m) + g(n, m-1) + g(n-1, m-1) - g(n-1, m-1)$$

or

$$g(n, m) = g(n-1, m) + g(n, m-1),$$

a recursion of simpler character than the Stanton-Cowan recursion.

The boundary conditions for our recursion must now be considered. Clearly

$$g(0, 0) = g(1, 0) = g(0, 1) = 1,$$

so that the explicit solution can be written

$$g(n, m) = \binom{n+m}{n}.$$

By Stirling's formula

$$g(n, n) \sim 2^{2n}/(4\sqrt{n\pi}).$$

For $n = 1000$, $g(n, n) > 10^{600}$ and direct examination of all alignments is impossible!. This is the reason for the development of efficient algorithms.

2.3 Basic dynamic programming methods. In the last section, the sequences $\mathbf{a} = a_1 a_2 \ldots a_n$ and $\mathbf{b} = b_1 b_2 \ldots b_m$ were used to generate alignments,

$$a_1^* a_2^* \ldots a_L^*$$

$$b_1^* b_2^* \ldots b_L^*$$

where the subsequences of $\mathbf{a}^*$ and $\mathbf{b}^*$ of elements not equal to Δ are the original sequences. For the cases of similarity and distance, the recursion equation for $f(n, m)$ from Section 2.3 is modified to provide an efficient way of calculating similarity and distance.

Historically, these methods began in biology with Needleman and Wunsch (1970) who present algorithm N-W stated below. Then Sankoff (1972) and Sankoff and Sellers (1973) find dynamic programming methods for optimal alignments with a given number of indels. Next Sellers (1974a, b) gives algorithms to compute the distance $D(\mathbf{a}, \mathbf{b})$, a problem posed by Ulam (1972). Gordon (1973) and Delcoigne and Hansen (1975) make useful contributions to sequence comparison which involves "slotting" the bases together in an optimal way. Probably their work goes unnoticed because the slotting alignments are not used by biologists. All of these methods are included in a class of techniques known as dynamic programming which was introduced by Richard Bellman. See Byers and Waterman (1984) for a general discussion and further references.

Consider the alphabet {A, C, G, T, Δ}, enlarged to include Δ. Let a weighting function $s(a, b)$ be defined on pairs of letters from the alphabet. A function of the type used by Needleman-Wunsch is

$$s(a, b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & \text{if } a \neq b,\ a \neq \Delta,\ b \neq \Delta, \\ -1.5 & \text{if } a \neq b \text{ and one of } a, b = \Delta. \end{cases}$$

Here matches receive positive weight and indels receive negative weight. The following statements appear in Smith and Waterman (1981b) where the proofs proceed along the line of the equation for $f(n, m)$ in Section 2.3. First define, where the maximum is taken over all alignments,

$$S(\mathbf{a}, \mathbf{b}) = \max \sum_{k=1}^{L} s(a_k^*, b_k^*).$$

Algorithm N-W Let

$$S_{0j} = \sum_{k=1}^{j} s(\Delta, b_k), \quad S_{00} = 0, \quad \text{and } S_{i0} = \sum_{k=0}^{i} s(a_k, \Delta).$$

If $S_{ij} = (a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_j)$, then

$$S_{ij} = \max\{S_{i-1,j} + s(a_i, \Delta), S_{i-1,j-1} + s(a_i, b_j), S_{i,j-1} + s(\Delta, b_j)\}.$$

If an initial distance function d is specified on {A, C, G, T, Δ} then a similar result is obtained. One such function is

$$d(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b. \end{cases}$$

Define, as above,

$$D(\mathbf{a}, \mathbf{b}) = \min \sum_{k=1}^{L} d(a_k^*, b_k^*).$$

Algorithm S Let

$$D_{0j} = \sum_{k=1}^{j} d(\Delta, b_k), \quad D_{00} = 0, \quad \text{and } D_{i0} = \sum_{k=0}^{i} d(a_k, \Delta).$$

If $D_{ij} = S(a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_j)$, then

$$D_{ij} = \min\{D_{i-1,j} + d(a_i, \Delta), D_{i-1,j-1} + d(a_i, b_j), D_{i,j-1} + d(\Delta, b_j)\}.$$

Both of these algorithms have computation time proportional to

$$\sum_{i=1}^{n} \sum_{j=1}^{m} 1 = nm.$$

The storage required to calculate $D(\mathbf{a}, \mathbf{b})$ is $\min\{n, m\}$ but $D(\mathbf{a}, \mathbf{b})$ is seldom of value without the set of associated optimal alignments.

There are two basic techniques to produce alignments. The first is to save pointers at each (i, j) to show which of $D_{i-1,j}$: $D_{i-1,j-1}$: $D_{i,j-1}$ are used in calculation of the optimal D_{ij}. The pointers are saved during the "forward" calculation so that during tracebacks the pointers can be followed to produce an optimal alignment. Where there are multiple optima, the pointers not followed can be stacked and in this manner (breadth-first search with stacking) all optimal alignments can be produced. If pointers are not saved, recomputing which of $D_{i-1,j}$: $D_{i-1,j-1}$: $D_{i,j-1}$ results in $D_{i,j}$ is easy to do if the D matrix is saved. In either case, required storage is $O(nm)$.

2.4 Extensions of the basic methods. The most important transformations of evolution treated in sequence comparison are single base mutations and indels. Above only single indels are treated. This section gives extensions to longer indels and describes a problem of long inversion of segments of a sequence.

While indels of many bases, 100 say, could be the sum of 100 single base indels, the likely explanation is that there was a single event. In a study of alignment parameters, Fitch and Smith (1983) show that, for certain chicken hemoglobin mRNA sequences, longer indels are necessary to obtain the correct alignment. The longer indels should not be weighted as the sum of single indels.

Let x_k be the weight chosen for an indel of k letters, $k \geqslant 1$. The following results appear in Waterman *et al.* (1976). If $x_1 \leqslant x_2 \leqslant \ldots$ and d is a metric on the set of sequence elements, then D is a metric on the set of sequences.

Algorithm W-S-B. Let $D_{i0} = x_i$, $D_{0j} = x_j$, $D_{00} = 0$, and $D_{ij} = D(a_1a_2 \ldots a_i, b_1b_2 \ldots b_j)$. Then

$$D_{ij} = \min\left\{D_{i-1,j-1} + d(a_i, b_j), \min_{k\geqslant 1}\{D_{i,j-k} + x_k\}, \min_{k\geqslant 1}\{D_{i-k,j} + x_k\}\right\}.$$

A corresponding algorithm will compute a generalized S.

It is important to note that computation time is increased to

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(ij) = O(nm^2 + n^2m).$$

An $O(n^3)$ algorithm, for two sequences of length $n = 1000$ say, is a significant price to pay for multiple indels. One approach to avoiding this is to assume x_k linear. That is, set

$$x_k = a + bk.$$

For secondary structure calculations this assumption is exploited by

Waterman (1976; p. 203) and Kanehisa and Goad (1982). In an elegant paper, Gotoh (1982) derives a related algorithm for linear x_k with running time $O(nm)$. Taylor (1984) gives more results along the lines of Gotoh. Of course, after x_1 and x_2, such x_k behave much like single indels. It is therefore desirable to extend the algorithm, especially to concave indel functions such as

$$x_k = a + b\log(k).$$

This has been accomplished in Waterman (1984) and has application to secondary structure problems as well where indels are analogous to bulges, interior loops and multibranch loops. The algorithm for concave x_k has running time $O(nm)$.

The problem of including inversions is very interesting. Interchanging two adjacent letters is a transformation considered by computer scientists. Recently Wagner (1983) has shown that this transformation can be included with computational time $O(\alpha nm4^{\alpha})$ where

$$\alpha \leqslant \min\{4 \cdot \max d(a, b), 2x_1\}/y + 1$$

where y is the cost of transposition. Including long inversions would seem to be a very difficult task. Certainly these long inversions occur in DNA sequences and should be included. A problem relevant to biology is to include inversions in an algorithm where there is a single cost of inversion plus the distance between the segments. It should be possible to allow inexact inversions with their own indels.

Reichart *et al.* (1973), Wong *et al.* (1974) and Cohen *et al.* (1975) produce a series of dynamic programming algorithms motivated by Turing machines and information theory. Their algorithms can be viewed as special cases of the general dynamic programming algorithms. See Waterman *et al.* (1976).

2.5 Ukkonen's dynamic programming algorithm. While the $O(n^2)$ dynamic programming algorithm is much faster than the $O(2^{2n})$ brute force algorithm, it becomes prohibitive for very large n. Recently, E. Ukkonen (1983, 1984) gives significantly faster algorithms for computing the distance between two sequences. The algorithms, which are outlined below, compute the distance s between sequences of lengths n and m, along with the alignment, in time and storage $O(s \cdot \min\{n, m\})$ (with storage $O(s^2)$ in some cases). If no alignment is desired, the storage is $O(s)$. The worst case behavior is equivalent to the standard algorithm, while for small s the improvement is dramatic. If no distance larger than a threshold t is desired, then the time is no larger than $O(t \cdot \min\{n, m\})$.

An algorithm similar in spirit to Ukkonen's is proposed by J. W. Fickett (1984). Fickett's algorithm is not as time efficient as Ukkonen's. (Each of these author's work treats the case of single indels.)

Ukkonen (1983) presents his algorithm for mismatches, single indels, and two letter transpositions. Below are the same results for mismatches and multiple indels. Let, as above,

$$D_{ij} = \min\left\{D_{i-1,j-1} + d(a_i, b_j), \min_{k\geq 1}\{D_{i,j-k} + x_k\}, \min_{k\geq 1}\{D_{i-k,j} + x_k\}\right\},$$

and assume $d(a, b) = 1$ unless $a = b$. Ukkonen proves a key lemma which is shown to be true for the general case of multiple indels:

LEMMA U. For all (i, j), $D_{ij} - 1 \leq D_{i-1,j-1} \leq D_{ij}$.

PROOF. The proof is by induction on $i + j$. The left-hand side is immediate from the recursion. If $D_{ij} = D_{i-1,j-1} + d(a_i, b_j)$ then $D_{ij} \geq D_{i-1,j-1}$ follows. Otherwise, without loss of generality, assume $D_{ij} = D_{i-k,j} + x_k$. The induction hypothesis implies $D_{i-k,j} \geq D_{i-(k+1),j-1}$ so that $D_{ij} \geq D_{i-1-k,j-1} + x_k$. The recursion equation now implies $D_{ij} \geq D_{i-1-k,j-1} + x_k \geq D_{i-1,j-1}$.

Lemma U, which is elementary, is the key to Ukkonen's elegant method. It states that $D_{i,i+c}$ is a non-decreasing function of i. This implies a structure for the matrix D_{ij}: it is shaped like a valley with increasing elevations along lines of constant $j - i$. The lowest elevation is $D_{00} = 0$. The focus below is on the boundaries of elevation changes when $D_{i,i+c} = k$ changes to $D_{i+1,i+1+c} = k + 1$.

Suppose all indels have cost 1, i.e. $x_k = k$. The basic idea of Ukkonen's algorithm (1983) is to start at $D_{00} = 0$ and extend along $j - i = 0$ until $D_{ii} = 1$. In general, there will be $2k + 1$ boundaries of the region $D_{ij} \leq k$. Each boundary $j - i = c$ is extended until $D_{ij} = k + 1$ (for $j - i = c$). The extension of the boundary to $k + 1$ can be determined from the boundaries for k, $k - 1$, . . . and tests of $a_i = b_j$. This procedure is followed until D_{nm} is reached. If $D_{nm} = s$, it is clear that no more than $(2s + 1)\min\{n, m\}$ entries have been computed. It is sufficient to only store these boundaries so that required storage is $O(s^2)$.

2.6 Near optimal alignments. Although the algorithms locate optimal alignments, the weights are determined by the user. Calibration of weights by alignments already known to be correct can be done but no set of weights can be assumed to be absolutely correct. Even if the weights are properly chosen, unknown biological constraints might cause the true alignment to be different from the computer generated optimal alignment.

A natural problem, then, is to find all alignments within a specified distance of the optimal. The motivation is that the correct alignment should be near the optimal one and that a biologist or sequence analyst might recognize it. This problem was solved in Waterman (1983) and received a more general treatment in Byers and Waterman (1984).

Formally stated, the problem is to find all alignments with alignment distance or score within ϵ of the distance $D_{n,m}$ between the two sequences a and b. Assume all $D_{i,j}$ are computed and stored.

At position (i, j) assume a traceback from (n, m) to $(0, 0)$ is being performed that can result in an alignment with score less than or equal to $D_{n,m} + \epsilon$. The score from (n, m) to but not including (i, j) is T_{ij}. T_{ij} is the sum of the possibly non-optimal alignment to reach (i, j). From (i, j), as usual, three steps are possible: $(i - 1, j)$, $(i - 1, j - 1)$, and $(i, j - 1)$. Each step is in a desired alignment if and only if

$$T_{ij} + d(a_i, \Delta) + D_{i-1,j} \leqslant D_{n,m} + \epsilon$$
$$T_{ij} + d(a_i, b_j) + D_{i-1,j-1} \leqslant D_{n,m} + \epsilon$$
$$T_{ij} + d(\Delta, b_j) + D_{i,j-1} \leqslant D_{n,m} + \epsilon,$$

respectively. Multiple near-optimal alignments can be produced by stacking unexplored directions.

A study of sequence alignment sensitivity to weights and multiple indels has been carried out by Fitch and Smith (1983). The sequences displayed below are chicken hemoglobin mRNA sequences, nucleotides 115–171 from the β chain (upper sequence) and 118–156 from the α chain (lower sequence).

```
UUUGCGUCCUUUGGGAACCUCUCCAGCCCCACUGCCAUCC
UUUCCCCACUUCG      AUCU

          UUUUGUCACACGGCAACCCCAUGGUC
                     GGCUCCGCUCAAAUC
```

This alignment is presumed correct from the analysis of the many known amino acid sequences for which such RNA sequences code.

By using a mismatch weight of 1 and a multiple indel function $x_k = 2.5 + k$, where k is the length of the indel, the correct alignment is found among the 14 optimal alignments. (This is region Q of the Fitch–Smith paper.) To indicate the size of neighborhoods in this example, there are 14 alignments within 0% of the optimum, 14 within 1%, 35 within 2%, 157 within 3%, 579 within 4% and 1317 within 5%.

A mismatch weight of 1 and a multiple indel function $2.5 + 0.5\,k$ is in region P of Fitch and Smith; accordingly, the correct alignment is not in the list of two optimal alignments. It is necessary to go to the list of 31 alignments within 4% of the optimal alignment to find the correct alignment. This example illustrates the sensitivity of alignment to weighting functions.

2.7 The regions method. An examination of dot matrices (Section 1.1) might suggest constructing a list of matching regions. Then, since an alignment

is just a special ordered subset of such a list, algorithms might be devised to find optimal alignments. This approach must have been pursued by several independent groups, although I do not know the early history. As described in Section 3.1, it is basic to find long matching regions. For secondary structure prediction, Studnicka *et al.* (1978) follow such a course. Martinez (1980, 1983) gives an algorithm which is much more mathematical. In fact, Martinez (1983) adapts his secondary structure algorithm into sequence alignment. I attempt to describe this procedure, following the outline of his secondary structure algorithm.

First we need a list L of matching regions. A region R is defined to be a triple $(w; i, j)$ which means a match of word w begins at $a_i = b_j$. In other words, if $l = |w|$ is the word length,

$$a_i = b_j, a_{i+1} = b_{j+1}, \ldots, a_{i+l-1} = b_{j+l-1}.$$

To obtain such a list, Martinez (1983) first concantenates the sequences into a single sequence S. He uses the technique of repeatedly sorting S. Using a lexicographic ordering, the first sorting groups all equal elements of S together. The second sorting operates on each of these equal element groups and groups together elements which are succeeded by equal elements in the original S. At the end of the kth sort, two elements of the permuted S will belong to the same group if and only if their locations i and j in the original S are such that the elements at locations $i + l$ and $j + l$ are equal for $l = 0, 1, \ldots, k - 1$.

As shown in Martinez (1983), the speed of this sorting procedure for generating regions is, in the expected sense, of order $N \log N$ where N is the total length of the concantenated sequences. The procedure is therefore comparable in speed to the standard computer science method of constructing "position trees" for identifying common substrings of two or more sequences, as described by Aho, Hopcroft and Ullman (1974), but has the advantage of ease of implementation.

To illustrate the concept of position trees let a = AATAATGCS, where S signals the end of the sequence. For each i, i = 1 to 8, let the substring S be the shortest substring beginning at i which does not occur elsewhere in a. This substring is said to identify i. For example, position i = 4 is identified by AATG. These identifying substrings are organized into a position tree which represents the information:

Position	Identifying substring
1	AATA
2	ATA
3	TA
4	AATG
5	ATG
6	TG
7	G
8	CS
9	S

The n terminal nodes of the position tree for $\mathbf{a} = a_1 a_2 \ldots a_n$ consist of $1, 2, \ldots, n$. The sequence of labels on the edges from the root to terminal node i is the identifying substring for position i. The position tree for the length 8 sequence from above is given in Fig. 1. Two sequences (or more) can be processed simultaneously to give a position tree where the longest matching regions can be easily found.

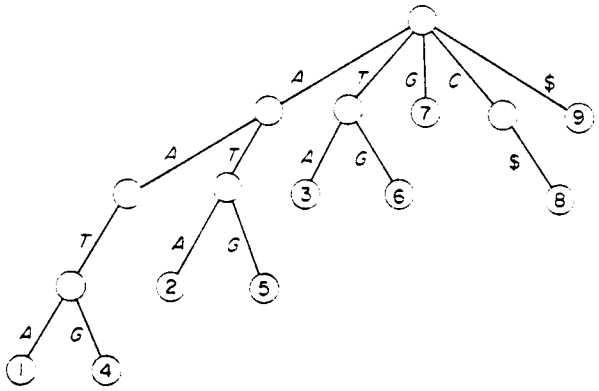

Figure 1. Position tree for a = AATAATGC.

Still another method for rapidly finding regions can be based on the concept of "hashing" as used in classical lexocographic search problems. The earliest reference is Dumey (1956). Described more fully by Dumas and Ninio (1982), the basic idea of this concept is to associate with each position of a sequence the numerical equivalent of the k − mer starting at that position. The numerical equivalent is obtained by regarding the sequence alphabet as defining the basis of a number system. Thus, a four-letter alphabet would give a number system to the base four, and for fixed k there are 4^k possible numbers. These numbers can then be used to identify positions of an array of size 4^k of lists of the positions of locations in the sequence at which the corresponding k − mer occurs. This method is used by Wilbur

and Lipman (1983) in making rapid similarity searches of data bases, and apparently also by Karlin *et al.* (1983) for finding exact repeats. The array can be constructed in time of order N. Longest repeats, and hence regions, are found by simply piecing together the repeats of size k, and the speed seems to be of order $N \log N$ (or order N?).

Given the list of regions found by any of these methods, we now utilize them to find optimal alignments. Two regions $R_1 = (w_1; i_1, j_1)$ and $R_2 = (w_2; i_2, j_2)$ are said to satisfy $R_1 < R_2$ if $i_1 + |w_1| - 1 < i_2$ and $j_1 + |w_1| - 1 < j_2$. $R_1 \leqslant R_2$ means that there are $i_2 - i_1 - |w_1|$ bases of a and $j_2 - j_1 - |w_1|$ bases of b between the regions, and that R_1 is left of R_2. A formula must be given to weight these unmatched bases. If there are x bases from a and y bases from b, let $z(x, y)$ give this weight. If mismatches cost 1 and indels cost δ with $1 < 2\delta$, then a reasonable choice for $z(x, y)$ is

$$z(x, y) = |x - y|\delta + \min\{x, y\}.$$

Formulae can be devised for other situations and $z(x, y) = x + y$ can be used.

Each region R can be considered to be at the left end of an optimal alignment $A(R)$, beginning with R and proceeding to a_n and b_m. Let $D(A(R))$ be the score of such an alignment. The optimization follows from

$$D(A(R)) = \min\{w(k - i - |w|, l - j - |w|) + D(A(R^*)): R = (w; i, j) < R^* = (w^*; k, l)\}.$$

General algorithms are known to run in time $O(|L|^2)$.

An implementation of this algorithm has been made by Martinez and Sobel and described in Martinez (1983). They also produce near optimal alignments by an adaptation of the ideas in Section 2.6.

3. Location of Long Matching Segments. In the past few years, the concept of a genome as a stable, slowly evolving collection of nucleic acids has been dramatically altered. While there is a general constancy of genome organization, recent discoveries of genetic elements such as transposons suggest that drastic reorganization can take place. Scientists have found unexpected relationships between viral DNA and host DNA (Doolittle *et al.*, 1983; Naharro *et al.*, 1984; Weiss, 1983). In these cases, it is not entire DNA sequences with high similarity but in fact contiguous subsequences (segments) with high similarity which are found. The modular organization of DNA into functional domains also suggests a search for highly similar segments is more appropriate than attempting to match long sequences of DNA. Of course, if both strings of DNA are known to serve the same purpose and are thought to have a direct common ancestor, then the methods of Section 2 should be used.

The problems of this section center on the search for segments of two DNA sequences which have unexpectedly high similarity. There are two basic approaches: search for the long exact matches or for long inexact matches. New developments in probability theory assist these searches.

3.1 Long exact matches. The first efficient method for locating exact matches was given by Korn *et al.* (1977). Their approach utilizes the position tree concept discussed in Section 2.7. That same method can be used to find long repeats in a fixed sequence. Repeats will share branches in the position tree. For example in Fig. 1 position 1 is identified by AATA while position 4 is identified by AATG, which implies the "long" AAT repeat. Korn *et al.* arrange the identifying substrings in a lexicographical order to find these repeats, among other things. This is a straightforward and useful application of modern computer science to DNA sequence analysis.

The algorithm given in Aho *et al.* (1977) for constructing position trees has a worst case running time of $O(n^2)$ for a sequence of length n. The running time is proportional to the number of vertices of the tree. However, if the letters of the word are independent and identically distributed, then the expected running time is $O(n)$. They also point out the existence of an algorithm which runs in $O(n)$ for all inputs.

In recent work Karlin *et al.* (1983) and Karlin *et al.* (1984) perform sequence analysis by locating long direct repeats. Their very interesting techniques use a hashing technique and locate all direct exact repeats.

Recently a theoretical determination of the statistical distribution of similarity has been given. If $M(n, m)$ is the length of the longest exact matching region between two sequences, then

$$E(M(n, m)) = \log((1-p)nm + \gamma/\lambda - 1/2 + r_1(n, m) + o(1)$$

and

$$\sigma^2(n, m) = \pi^2/6\lambda^2 + 1/12 + r_2(n, m) + o(1)$$

where $p = P$ (two random nucleotides match), $\log = \log_{1/p}$, $\gamma = 0.577\ldots$ is the Euler-Mascheroni constant, $\lambda = ln(1/p)$ and $r_1(n, m)$ and $r_2(n, m)$ are small. Notice that $\sigma(n, m)$ is essentially independent of n and m. This result is from Arratia *et al.* (1984). Karlin *et al.* (1984) also state a result differing by constants from this one. Also see Arratia and Waterman (1984) for related laws of large numbers. Matches between sequences are considered significant if they exceed $E(M(n, m)) + 2\sigma$. In the next section these results are generalized to include mismatches.

Collins and Coulson (1984) give a parallel processing algorithm to build the dot matrix of all matches of length greater than or equal to a set threshold. Their implementation accepts sequences of up to 49,152 bases and is an indication of the results of applying new technology to

these problems. The probability results above could be used to set the threshold.

3.2 Long inexact matches by dynamic programming. Consider the problem of locating similar segments between two sequences without requiring the segments to be identical. Sellers (1979, 1980) first consider this problem. He defines an interval I of **a** to most resemble **b** globally if $D(I, \mathrm{b}) \leqslant D(J, \mathrm{b})$ for all segments J of **a**. Both "forward" and "backward" distance matrices are required. One of the problems is finding the desirable matching segments from the many produced. (Sankoff and Kruskal (1983) estimate that n^4 matches result from sequences of length n.) Goad and Kanehisa (1982) modify Sellers' technique. Erickson and Sellers (1983) fully discuss this method and give two non-trivial applications of their techniques. In this issue Sellers refines his analysis and gives another algorithm for finding "best segments". He uses the Goad and Kanehisa (1982) concept of match density to find longest segments of a prescribed match density. Repeated passes through the matrix are required.

In another approach, Smith and Waterman (1981a, b) use similarity rather than distances. Similarity counts

AAAA A

vs

AAAA A

as four matches vs one, while distance is zero in each case. In order to filter out portions of the sequence with negative match, define H_{ij} to be the maximum similarity of two segments that end in a_i and b_j, or zero, whichever is larger. Define

$$H_{ij} = \max\{0,\ S(a_x a_{x+1} \ldots a_i,\ b_y b_{y+1} \ldots b_j):\quad 1 \leqslant x \leqslant i \quad \text{and } 1 \leqslant y \leqslant j\}.$$

Sellers' paper in this issue suggests that best segments are essentially maximum similarity segments which (1) have non-negative similarity, (2) have scores at least as large as any other segments with intersecting paths and (3) which have scores at least as large as some cut-off value. Smith and Waterman (1981a, b) recommend sequentially processing the H matrix, finding alignments with the largest, next largest, etc. similarity values with non-intersecting paths. After stating the algorithm for H, I give a new and more complete algorithm for finding alignments satisfying (1), (2) and (3) of Sellers' recommendations.

Algorithm S-W. Set $H_{i0} = H_{0j} = 0$ for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Then

$$H_{ij} = \max\left\{H_{i-1,j-1} + s(a_i, b_j),\ \max_{1 \leqslant k \leqslant i}\{H_{i-k,j} - x_k\},\ \max_{1 \leqslant k \leqslant j}\{H_{i,j-k} - x_k\},\ 0\right\}.$$

Values which might be used for weights are

$$s(x, y) = \begin{cases} 1 & \text{if } x = y \\ -1/3 & \text{if } x \neq y \end{cases}$$

and

$$x_k = 1 + k/3.$$

A reduction in computing time from $O(n^3)$ to $O(n^2)$ for linear or concave deletion functions can be achieved as in Section 2.4.

When constructing the matrix H, stack all (i, j, Y) with $Y = H_{ij}$ and $H_{ij} \geqslant C$ = cut-off value. The stack is ordered by $>$ where

$$(i, j) > (k, l) \quad \text{if} \quad (1)\ H_{ij} > H_{kl} \text{ or}$$

$$(2)\ H_{ij} = H_{kl} \text{ and } i + j < k + l \text{ or}$$

$$(3)\ H_{ij} = H_{kl},\ i + j = k + l, \text{ and } i < k.$$

During tracebacks for some stack entry, multiple alignments are resolved in the following manner: if two multiple alignments end at (i, j) and (k, l) output the one ending at (i, j) if (1) $i + j < k + l$, or if (2) $i + j = k + l$ and $i > k$. Once a trackback is successfully completed, the alignment entries in the matrix are multiplied by -1, the alignment output, and the corresponding stack entry is removed from the stack. Negative elements of H are not used in any future alignments.

If a stack entry has a negative corresponding matrix element, remove it and continue. If a traceback encounters a negative matrix element, it cannot continue. If the best alignment generated that far has score $= Y \geqslant C$, then the (i, j, Y) must be replaced into the ordered stack.

Boswell and McLachlan (1984) also suggest using similarity values for locating similar segments. Their forward matrix is calculated by

$$F(i, j) = s(a_i, b_j) + \lambda \max\{F_{i-2,j-1} - w_1, F_{i-1,j-1}, F_{i-1,j-2} - w_1\}.$$

The reverse matrix R is found by reversing the sequences. Then

$$M(i, j) = F(i, j) + R(i, j) - s(a_i, b_j).$$

The idea is that $M(i, j)$ is the sum of $s(a_i, b_j)$ plus the weighted best paths extending in either direction. The parameter $\lambda \in (0, 1)$ is a geometric damping factor.

The problem of distinguishing statistically significant values of

$$H^* = \max_{i,j} H_{ij}$$

is clearly important.

Arratia *et al.* (1984) have shown that the length $M(n, m)$ of the longest match interrupted by k mismatches satisfies

$$E(M(n, m)) = \log(nm) + k \log\log(nm) + (k + 1)\log(1 - p) - \log(k!)$$
$$+ k + \gamma/\lambda - \tfrac{1}{2} + r_1(n, m) + o(1)$$

and

$$\sigma^2(n, m) = \pi^2/6\lambda^2 + 1/12 + r_2(n, m) + o(1)$$

where $p = P$ (two random nucleotides match), $\log = \log_{1/p}$, $\gamma = 0.577$ is the Euler-Mascheroni constant, $\lambda = ln(1/p)$ and r_1, r_2 are small. It is possible to use, for example, $H_{ij} \geqslant E(M(n, m)) + 2\sigma(n, m)$ to decide which H_{ij} are of interest to output. Notice that $\sigma(n, m)$ is once again essentially independent of n and m.

In an empirical study, Smith *et al.* (1984) show that, for

$$s(x, y) = \begin{cases} 1 & \text{if } x = y \\ -0.9 & \text{if } x \neq y \end{cases}$$

and

$$x_k = \begin{cases} 2 & \text{if } k = 1 \\ \infty & \text{if } k > 1 \end{cases}$$

the values of $E(H^*)$ and σ are

$$E(H^*) = 2.5\log(nm) - 9$$

and

$$\sigma = 1.78,$$

where $\log = \log_{1/p}$ as above.

An asymptotic result that the $\log(n)$ law holds with indels as well as mismatches is given in Arratia and Waterman (1984). The empirical study reported above is evidence of the robustness of this distribution.

3.3 Long inexact matches by regions. An algorithm to find long inexact matches which does not use dynamic programming is proposed by Korn *et al.* (1977). It has some serious drawbacks, which are pointed out below, but it is a useful method and has received wide distribution to sequence analysts. Also see Queen *et al.* (1982).

The algorithm begins at positions (i, j) where $a_i = b_j$ and $a_{i+1} = b_{j+1}$. This match of length two is extended in a recursive manner, where the rules for extension are

(1) the next bases match (i.e. $a_{i+2} = b_{j+2}$),
(2) by deleting 1, 2 or 3 bases from sequence a there is a run of 3 matches,
(3) by deleting 1, 2 or 3 bases from sequence b there is a run of 3 matches.

or

(4) by mismatching a_{i+2} and b_{j+2}, two of the next 3 pairs match. Their program does not search over pairs (i, j) where (a_i, b_j) are already in an identified region.

Sankoff and Kruskal (1983) point out that this method will, when comparing AACAAA and AAAAA, find

AACAAA
AAΔAAA

but, with the sequences reversed, will not find this region. This could be an undesirable property. Sankoff and Kruskal also point out that AACCGT and AACGT will produce

AAC CGT
and
AAC CGT,

which have a base in common, instead of

AACCGT AACCGT
and
AACΔGT AAΔCGT.

The running time of this algorithm for two length n sequences is proportional to n^2. The constant is larger than the expectation of a geometric random variable. For equally likely bases, this means the constant exceeds 4. Since the dynamic programming algorithms have a constant of 3, this strongly suggests using the more mathematically rigorous algorithms.

The Wilbur-Lipman method, given in the next section, is also a regions method that can be applied to this problem.

4. Efficient Data Base Searches. The size of the nucleic acid sequence data is increasing rapidly with almost 3×10^6 bases in GenBank at this time. It is of interest to compare new sequences with those already known. While all comparisons might not be of interest, several important and unexpected discoveries have been made from these large searches. A recent computer finding, for example, indicates an oncogene product appears to have arisen as a result of recombination of two unrelated cellular genes (Naharro *et al.*, 1984). These searches will become increasingly important.

The mathematical problem of interest is how to rapidly search a large data base. For example to search 2000 sequences 500 base pairs (bp) each with a new 500 bp sequence will take, with dynamic programming techniques, time proportional to $2000(500)^2 = 7.5 \times 10^8$. This is an unacceptably large number, and below I present some approaches to the problem of rapid

searches. I do not include the new method of Ukkonen since I cannot see how to utilize Lemma U to find long matching segments (Section 3).

An already well-known method for these searches is that of Wilbur and Lipman (1983, 1984). Indeed their method has accounted for some of the biological discoveries mentioned above. In addition the dynamic programming maximum segments method is also used for these purposes. Finally, I suggest connections between the methods of Wilbur–Lipman and of Martinez. A generalized regions method suggested in Section 4.3 could utilize the output of the hashing algorithms for a list L of regions.

4.1 The Wilbur–Lipman method. In two important papers Wilbur and Lipman (1983, 1984) develop what they call context dependent sequence comparison. The outline of their method is that they (1) produce a list L of matching regions, all of a fixed length, (2) order the regions as in Section 2.7, (3) obtain an optimal alignment by processing the list L. They in fact develop a theory for much more general context dependence and in addition present conditions for their similarity measures to have an associated distance which is metric. I will limit my discussion to the case of regions.

In Section 2.7, regions methods are discussed. Here the list L of exact matching regions can be restricted, for example to those regions of length exactly 4. Wilbur and Lipman (1983) describe a linear time hashing algorithm for producing L with fixed length regions.

Denote a region r by $(w; i, j)$ where w is a word of length $|w|$ which begins at position i in sequence a and position j in sequence b. As in Section 2.7, $r_1 < r_2$ if $i_1 + |w_1| - 1 < i_2$ and $j_1 + |w_2| - 1 < j_2$. Also let region $r_0 = (\phi; 0, 0)$ be a minimal element and $r_* = (\phi; n, m)$ be a maximal element. $\Gamma = (r_1 r_2, \ldots, r_l)$ is a path if $p < q$ implies $r_p < r_q$. The score of a path Γ is given by

$$\text{score } (\Gamma) = \sum_{k=1}^{l} s(r_k) - \sum_{k=1}^{l-1} g(i_{k+1} - |w_k| - i_k - 1, j_{k+1} - |w_k| - j_k - 1)$$

where $s(\cdot)$ is a similarity score for region r_k, like $|w_k|$, and $g(\cdot, \cdot)$ is a gap penalty. Then

$$S(\text{a, b}) = \max\{\text{score } (\Gamma): \quad \Gamma \text{ is a path from } r_0 \text{ to } r_*\}.$$

Algorithm W-L. The algorithm is as follows: make two lists of regions L^-, ordered by $<$, and L^+, ordered by the usual order $\ll$ of best scores from r_0 to the region listed in L^+.

(0) Set $L^- = L$ and $L^+ = \phi$

(1) $r_q = \{\min r : r \in L^-\}$
score $(r_q) = S(r_q)$

(2) Begin at largest element of L^+.

(A) Move down L^+ until $r_u < r_q$ such that

$$\gamma = \text{score}(r_u) - g(i_q - |w_{r_u}| - i_u - 1, j_q - |w_{r_u}| - j_u - 1) + s(r_q) > \text{score}(r_q).$$

If there is no region r_u in L^+ below r_q under $<$ with a score greater than score $(r_q) - s(r_q)$, or if the inequality cannot be satisfied, go to (C)

(B) Set score $r_q = \gamma$ and go to (A)

(C) Remove r_q from L^- and insert it in L^+ under $\ll$.
If $L^- \neq \phi$, go to (A).

It is possible to modify this algorithm, along the lines of Section 3.2, to obtain maximum similarity segments instead of maximum similarity alignments. Wilbur and Lipman describe several useful modifications of their algorithm.

4.2 A vectorized maximum segments algorithm. Another approach to rapid computation was taken by T. F. Smith when at Los Alamos National Laboratory. Several CRAY-1 computers exist there and these vector machines are very fast. Smith modified the algorithm of Smith-Waterman to run on one of these computers. By utilizing the vector architecture of the CRAY-1, it is possible to perform comparisons among very large numbers of nucleic acid sequences in reasonable time. For example, a study by Smith *et al.* (1984) reports all pairwise comparisons among 204 vertebrate sequences (including the complement strands) were carried out in approx. 170 min, at the rate of over 240 sequences per min with an average sequence length of 800 nucleotides.

Vectorizing algorithms is a relatively new topic. The essential idea is that the machine performs a number of operations simultaneously and gains this factor over the usual linear sequence operations. If attention is focused on calculating H, no H_{ij} calculation being performed can depend on the results of another calculation being simultaneously performed. This eliminates row by row or column by column building up of the matrix. What remains is calculating blocks of H_{ij} on negative diagonals, i.e. with $i + j =$ constant.

Collins and Coulson (1984) discuss parallel processing algorithms for the Sellers' method. Smith's approach should allow much more efficient implementation for parallel processing.

4.3 The regions method. The regions method of Martinez (Section 2.7) and the Wilbur-Lipman method are clearly closely related, although Wilbur-Lipman devise a different optimization technique. Of course these were independently developed but the theme is now clear. For rapid calculation,

judiciously limit the list of regions and alignments will be produced more rapidly.

For example, Karlin *et al.* (1984) do not state an algorithm for aligning their long matching regions, although alignments result from these matchings. It is clear that using their regions as inputs to a regions method algorithm would produce meaningful alignments. In addition, if a regions list is not totally ordered, such an alignment method would be very useful in resolving the possible alignments.

5. Comparison of Several Sequences. In biological sequence analysis, problems frequently require the identification of relationships among R sequences, where $R > 2$. For example, deducing an evolutionary tree for 5S RNA molecules might involve $r = 100$ sequences of length 120. Another example might involve a set of coding regions which are assumed to have evolved from a common ancestor. One result of such studies is an alignment of the R sequences. Issues of *Science* and *Nature* frequently have articles containing such alignments.

As explained below, these problems appear to require a large amount of computation and storage for their rigorous solution. Often the biologist solves these problems by eye. The first mathematical results were obtained by Sankoff with others and are described below, and utilize dynamic programming. A second promising approach has been presented by Martinez who advocates the regions method. In this issue, Waterman and Perlwitz give geometry and algorithms for another dynamic programming approach to such problems, while Waterman *et al.* give a method which does not use dynamic programming or regions.

5.1 Dynamic programming algorithms. In a clearly written article, Sankoff and Cedergren (1983) review methods which give alignments of R sequences given a tree that relates the R sequences. Each interior node of the given tree T has a degree at least three, and the R sequences are attached to the R terminal nodes. The algorithm constructs sequences for each interior node and gives an alignment relating the original and the reconstructed sequences. If the tree has N interior nodes, this is an alignment of $R + N$ sequences.

While the Sankoff and Cedergren article is well written and need not be reproduced here, I will try to give the dynamic programming aspects of the problem. Suppose the R sequences are a, b, . . . , r. The cost of an overall alignment of the R original plus N constructed sequences is the sum of the pairwise alignment cost in T. The idea is to think of aligning $a_1a_2 \ldots a_i$, $b_1b_2 \ldots b_j, \ldots, r_1r_2 \ldots r_s$. The last column of the alignment can be broken off from the initial columns. For a, b, . . . , r the last column will appear as

$$\begin{matrix} \epsilon_1 a_i \\ \epsilon_2 b_j \\ \cdots \\ \epsilon_R r_s \end{matrix}$$

where $\epsilon_i \in \{0, 1\}$, $0 \cdot a_i = \Delta$, and $\epsilon = (\epsilon_1, \ldots, \epsilon_R) \neq 0$. This last restriction just keeps the last column from being all Δs. The dynamic programming step is

$$D_{i,j,\ldots,s} = \min_{\epsilon \neq 0} \left\{ D_{i-\epsilon_1, j-\epsilon_2, \ldots, s-\epsilon_R} + \min_{\epsilon \neq 0} \text{length} \begin{pmatrix} \epsilon_1 a_i \\ \epsilon_2 b_j \\ \vdots \\ \epsilon_R r_s \\ x_1 \\ \vdots \\ x_N \end{pmatrix} \right\}.$$

The last term is to indicate that the letters $x_1 x_2 \ldots x_N$ for the N interior nodes have been determined in an optimal way. Fitch's (1971) parsimony method, suitably generalized, is used for this purpose. The computation time for this algorithm for R sequences of length n is $O(2^R n^R N)$ where 2^R comes from $\epsilon_1, \ldots, \epsilon_R$ at each step, n^R comes from $i, j, \ldots, s$, and N comes from Fitch's parsimony method. For sequences of length 100, this is approx. $O(10^{2.3R} N)$ so that $R = 3$ is nearly as large as possible.

Unaware of Sankoff's work (1975), Waterman *et al.* (1976) present a similar algorithm which does not explicitly assume a tree. In Sankoff's framework, this corresponds to the tree, one interior node, and R terminal nodes. A function $d(x, y, \ldots, z)$ of R variables was used to generalize the metric on the letters so it is possible to have a different weighting, but the essential idea is in Sankoff's work.

Although no one seems to have suggested it, it is possible to generalize the Smith–Waterman algorithm for maximum similarity segments to the case of R sequences. Exactly the same computational problems limit the utility of the idea.

5.2 A regions algorithm. Martinez (1983) has suggested approaching these problems via regions. Locating all repeats of R sequences n long can be done in time proportional to nR. Repeat in this context refers to a word w which occurs in all R sequences. Such a repeat is called a region and is placed in a list L. A partial order "$<$" in R dimensions (instead of the two dimensions) is placed on L.

Exactly the same algorithm from Sections 2.7 and 4.3 can be used to produce a multiple sequence alignment. The limitation here is that a requirement of exact matches in all R sequences is quite stringent. Still, the aligned portions would be very convincing and such an alignment could be produced in reasonable time.

6. Conclusions. Computer analysis of macromolecular sequences will become more important and I foresee two major directions to algorithm changes. First, as more detailed biological information becomes available, it will be important to perform more finely tuned comparisons. For example as insertion/deletion mechanisms are studied it might become clear that the sequences at the insertion/deletion boundaries greatly influence the likelihood of the event. Such information should certainly be incorporated into algorithms. Second, the increasing nucleic acid data will require that rapid data base searches become even more rapid. Imagine what size the data base will grow to in even ten years. Computer science methods must be developed and applied to these important problems.

As mentioned in the introduction, this review does not exhaust the problem types of interest. Existing algorithms can be modified to handle a surprising variety of problems. For example, the best fit of a sequence into another or the best alignment of two sequences with either or both end gaps unweighted are possible with alterations of dynamic programming algorithms presented here (Sellers, 1980 and Smith *et al.*, 1981).

There is no lack of problems of theoretical and applied interest.

Over the last few years, several people have helped and encouraged me in studies of sequence comparison in molecular biology. They include S. Ulam, W. Beyer, T. Smith, W. Fitch, H. Martinez, P. Sellers, C. Smith and G. C. Rota. I am also grateful to M. Perlwitz for assistance in preparing this manuscript.

LITERATURE

Aho, V. A., J. E. Hopcroft and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Menlo Park, California.

Arratia, R. A. and M. S. Waterman. 1984. "An Erdos–Renyi Law with Shifts." *Adv. appl. Math.* (in press).

——, L. Gordon and M. S. Waterman. 1984. "An Extreme Value Distribution for Sequence Matching." Manuscript.

Beyer, W. A., C. Burks and W. B. Goad. 1983. "Quantitative Comparison of DNA Sequences." *Los Alamos Sci.* 9, 62–63.

——, M. L. Stein, T. F. Smith and S. M. Ulam. 1974. "A Molecular-sequence Metric and Evolutionary Trees." *Math Biosci.* 19, 9–25.

Boswell, D. R. and A. D. MacLachlan. 1984. "Sequence Comparison by Exponentially-damped Alignment." *Nucleic Acids Res.* 12, 457–464.

Byers, T. H. and M. S. Waterman. 1984. "Determining All Optimal and Near-optimal Solutions When Solving Shortest Path Problems by Dynamic Programming." *Operat. Res.* (in press).

Cohen, D. N., T. A. Reichert and A. K. C. Wong. 1975. "Matching Code Sequences Utilizing Context Free Quality Measures." *Math. Biosci.* 24, 25–30.

Collins, J. F. and A. F. W. Coulson. 1984. "Applications of Parallel Processing Algorithms for DNA Sequence Analysis." *Nucleic Acids Res.* 12, 181–192.

Delcoigne, A. and P. Hansen. 1975. "Sequence Comparison by Dynamic Programming." *Biometrika* 62, 661–664.

Doolittle, R. F., M. W. Hunkapiller, L. E. Hood, S. G. Devare, K. C. Robbins, S. A. Aaronson and H. M. Antoniades. 1983. "Simian Sarcoma Viruses Onc Gene v-sis is Derived from the Gene (or Genes) Encoding a Platelet-derived Growth Factor." *Science* 221, 275–276.

Dumas, J. P. and J. Ninio. 1982. "Efficient Algorithms for Folding and Comparing Nucleic Acid Sequences." *Nucleic Acids Res.* 80, 197–206.

Dumey, A. I. 1956. "Indexing for Rapid Random-access Memory." *Comput. Automat.* 5, 6–8.

Erickson, B. W. and P. H. Sellers. 1983. In *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Ed. D. Sankoff and J. B. Kruskal, pp. 55–90, Addison-Wesley, London.

Fickett, J. W. 1984. "Fast Optimal Alignment." *Nucleic Acids Res.* 12, 175–180.

Fitch, W. M. 1969. "Locating Gaps in Amino Acid Sequences to Optimize the Homology Between Two Proteins." *Biochem. Genet.* 3, 99.

——. 1971. "Towards Defining the Course of Evolution: Minimum Change for a Specific Tree Topology." *Syst. Zool.* 20, 406–416.

—— and E. Margoliash. 1967. "Construction of Polygenetic Trees." *Science* 155, 279–284.

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

Gibbs, A. J. and G. A. McIntyre. 1970. "The Diagram, a Method for Comparing Sequences." *Euro. J. Biochem.* 16, 1–11.

Goad, W. B., M. I. Kanehisa. 1982. "Pattern Recognition in Nucleic Acid Sequences. I. A General Method for Finding Local Homologies and Symmetries." *Nucleic Acids Res.* 10, 247–263.

Gordon, A. D. 1973. "A Sequence-comparison Statistic and Algorithm." *Biometrika* 60, 197–200.

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

Harr, R., P. Hagblom and P. Gustafsson. 1982. "Two-dimensional Graphic Analysis of DNA Sequence Homologies." *Nucleic Acids Res.* 10, 365–374.

Jagadeeswaran, P. and P. M. McGuire. 1982. "Interactive Computer Programs in Sequence Data Analysis." *Nucleic Acids Res.* 10, 433–447.

Kanehisa, M. I. and W. B. Goad. 1982. "Pattern Recognition in Nucleic Acid Sequences II. An Efficient Method for Finding Locally Stable Secondary Structures." *Nucleic Acids Res.* 10, 265–277.

Karlin, S., G., Ghandour and D. E. Foulser. 1984. "Comparative Analysis of Human and Bovine Papallimaviruses." *Mol. Biol. Evol.* 1, 357–370.

——, ——, F. Ost, S. Tavare and L. J. Korn. 1983. "New Approaches for Computer Analysis of Nucleic Acid Sequences." *Proc. natn. Acad. Sci U.S.A.* 80, 5660–5664.

Korn, L. J., C. L. Queen and M. N. Wegman. 1977. "Computer Analysis of Nucleic Acid Regulatory Sequences." *Proc. natn. Acad. Sci. U.S.A.* 74, 4401–4405.

Kruskal, J. B. 1983. "An Overview of Sequence Comparison." In *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Eds D. Sankoff and J. B. Kruskal, pp. 1–40. Addison-Wesley, London.

—— and D. Sankoff. 1983. In *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Eds D. Sankoff and J. B. Kruskal, pp. 265–310. Addison-Wesley, London.

Laquer, T. H. 1981. "Asymptotic Limits for a Two-dimensional Recursion." *Stud. appl. Math.* 64, 271–277.

Levenshtein, V. I. 1966. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Cybernet. Control Theor.* 10, 707–710: *Dolklady Akademii nauk SSSR* 163, 845–848.

Maizel, J. and R. Lenk. 1981. "Enhanced Graphic Matrix Analysis of Nucleic Acid and Protein Sequences." *Proc. natn. Acad. Sci. U.S.A.* 78, 7665–7669.

Martinez, H. M. 1980. "A New Algorithm for Calculating RNA Secondary Structure." Manuscript.

——. 1983. "An Efficient Method for Finding Repeats in Molecular Sequences." *Nucleic Acids Res.* 11, 4629–4634.

Naharro, G., K. C. Robbins and E. P. Reddy. 1984. "Gene Product of v-fgr Onc: Hybrid Protein Containing a Portion of Actin and Tyrosin-Specific Protein Kinase." *Science* 223, 63–66.

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. Mol. Biol.* 48, 444–453.

Novotny, J. 1982. "Matrix Program to Analyze Primary Structure Homology." *Nucleic Acids Res.* 10, 127–131.

Queen, C. M., N. Wegman and L. J. Korn. 1982. "Improvements to a Program for DNA Analysis: a Procedure to find Homologies Among Many Sequences." *Nucleic Acids Res.* 10, 449–456.

Reichert, T. A., D. N. Cohen and A. K. C. Wong. 1973. "An Application of Information Theory to Genetic Mutations and Matching of Polypeptide Sequences." *J. Theor. Biol.* 42, 245–261.

Sankoff, D. 1972. "Matching Sequences Under Deletion–Insertion Constraints." *Proc. natn. Acad. Sci. U.S.A.* 68, 4–6.

——. 1975. "Minimal Mutation Trees of Sequences." *SIAM J. appl. Math.* 78, 35–42.

—— and R. J. Cedergren. 1983. In *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Eds D. Sankoff and J. Kruskal, pp. 253–263. Addison-Wesley, London.

—— and J. B. Kruskal (eds). 1983. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Addison-Wesley, London.

—— and P. H. Sellers. 1973. "Shortcuts, Diversions, and Maximal Chains in Partially Ordered Sets." *Discrete Math.* 4, 287–293.

Sellers, P. 1974a. "An Algorithm for the Distance Between Two Finite Sequences." *Comb. Theory* 16, 253–258.

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

——. 1979. "Pattern Recognition in Genetic Sequences." *Proc. natn. Acad. Sci. U.S.A.* 76, 3041.

——. 1980. "The Theory and Computation of Evolutionary Distances: Pattern Recognition." *J. Algorithms* 1, 359–373.

Shepard, R. N. 1980. "Multidimentional Scaling, Tree-Fitting, and Clustering." *Science* 210, 390–398.

Smith, T. F. and M. S. Waterman. 1981a. "Identification of Common Molecular Subsequences." *J. Mol. Biol.* 147, 195–197.

—— and ——. 1981b. "Comparison of Biosequences." *Adv. appl. Math.* 2, 482–489.

——, —— and C. Burks. 1984. "The Statistical Distribution of Nucleic Acids Similarities." In prep.

——, —— and W. M. Fitch. 1981. "Comparative Biosequence Metrics." *J. Mol. Evol.* 18, 38–46.

Söll, D. and R. J. Roberts (Eds). 1982. *The Application of Computers to Research on Nucleic Acids I.* IRL Press, Oxford and Washington, D.C.

—— and ——. 1984. *The Application of Computers to Research on Nucleic Acids II.* IRL Press, Oxford and Washington, D.C.

Stanton, R. G. and D. D. Cowan. 1970. "Note on a 'Square Functional' Equation." *SIAM Rev.* 12, 277–279.

Studnicka, G., G. Rahn, I. Cummings and W. Salser. 1978. "Computer Method for

Predicting the Secondary Structure of Single-Stranded RNA." *Nucleic Acids Res.* 5, 3365–3387.

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 in Comp. Sci.* **158**, 487–496.

——. 1984. "Algorithms for Approximate String Matching." *Informat. Control* (in press).

Ulam, S. M. 1972. In *Applications of Number Theory to Numerical Analysis*, Ed. S. K. Zaremba, pp. 1–3. Academic Press, New York.

Wagner, R. H. 1983. In *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Eds. D. Sankoff and J. B. Kruskal, pp. 215–235. Addison-Wesley, London.

Waterman, M. S. 1976. "Secondary Structure of Single-stranded Nucleic Acids." *Adv. Math*, Suppl. Stud. 1, 167–212.

——. 1983. "Sequence Alignment in the Neighborhood of the Optimum with General Applications to Dynamic Programming." *Proc. natn. Acad. Sci. U.S.A.* **80**, 3123–3124.

——. 1984. "Efficient Sequence Alignment Algorithms." *J. Theor. Biol.* (in press).

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

Weiss, R. 1983. "Oncogenes and Growth Factors." *Nature* **304**, 12.

Wilbur, W. J. and D. J. Lipman. 1983. "Rapid Similarity Searches of Nucleic Acid and Protein Data Banks." *Proc. natn. Acad. Sci. U.S.A.* **80**, 726–730.

—— and ——. 1984. "The Context Dependent Comparison of Biological Sequences." *SIAM J. appl. Math.* (in press).

Wong, A. K. C., T. A. Reichert, D. N. Cohen and B. O. Aygun. 1974. "A Generalized Method for Matching Informational Macromolecular Code Sequences." *Comput. Biol. Med.* 4, 43–57.