ASPECTS OF BIOLOGICAL SEQUENCE COMPARISON

by

Stephen Frank Altschul
B.A., Harvard College (1979)

Submitted to the Department of Mathematics in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

June, 1987

Signature of Author Signature Redacted
Department of Mathematics
February 12, 1987

Certified by Signature Redacted
Prof. Daniel J. Kleitman
Thesis Supervisor

Accepted by Signature Redacted
Prof. Willem V. R. Malkus
Chairman, Applied Math Committee

Accepted by Signature Redacted
Prof. Sigurdur Helgason
Chairman, Departmental Graduate Committee
Department of Mathematics

TABLE OF CONTENTS

For
Stephan R. Wagner
1895-1975

ASPECTS OF BIOLOGICAL SEQUENCE COMPARISON

by

Stephen Frank Altschul

Submitted to the Department of Mathematics
on February 12, 1987 in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Mathematics.

ABSTRACT

This thesis investigates methods for finding optimal alignments and subalignments of biological sequences and assessing their statistical significance. Specifically, nonlinear similarity functions are proposed as the most appropriate for comparing subalignments, in contrast to the linear similarity functions in wide use. Algorithms are developed for finding locally optimal subalignments, using any reasonable similarity function as a selection criterion. The statistical significance of nucleic acid and protein sequence subalignments is investigated using Monte Carlo methods. The methods developed are used to find interesting alignments from several real biological sequences. In addition, the thesis identifies and corrects mistakes in the literature concerning the optimal alignment of two sequences using affine gap costs, the time complexity of finding optimal alignments using concave gap costs, and the random permutation of a sequence preserving its doublet frequency.

Thesis Supervisor: Professor Daniel J. Kleitman

ACKNOWLEDGEMENTS

First, I wish to thank Professor Bruce Erickson for his encouragement, time, ideas and advice over two and a half years. Without his help this thesis could not have been written.

I am indebted to Professors Peter Sellers and Joel Cohen for many useful suggestions and for providing me with space and resources with which to complete this work.

I am grateful also to Professor Bruce Merrifield for allowing me to join his laboratory at the Rockefeller University.

I am appreciative of financial support from various institutions, namely the National Science Foundation, which supported me with a Graduate Fellowship; the Mathematics Department, which supported me with teaching assistantships; and the Office of Naval Research and the Army Research Office which supported me through a research contract with Professor Bruce Erickson.

Finally, I would like to thank Professor Peter Sellers and the members of my thesis committee, Professors Daniel Kleitman, Bruce Erickson and Mike Sipser, for their criticism of the manuscript of this thesis.

I. INTRODUCTION

A sequence is a finite string of letters drawn from a finite alphabet. While a sequence may be treated as a purely abstract entity, many sequences have their origin in the real world. Examples of real-world sequences are pieces of English text and sequences representing the primary structure of biological macromolecules, such as proteins or nucleic acids. The set of questions it is fruitful to ask about real-world sequences depends upon the origin of the sequences. The questions investigated in this thesis are some of those that seem appropriate for protein or nucleic acid sequences. Since the discussion will be abstract, the results will apply to sequences of any kind.

Biological sequences. Deoxyribonucleic acids (DNA), ribonucleic acids (RNA) and protein molecules are each constructed as a linear sequence of a small set of chemical building blocks. (In some cases the DNA sequence is circular.) DNA molecules generally are double stranded with each strand composed of four types of deoxyribonucleotide in a specific order. The two strands of a DNA molecule are complementary so that the sequence of nucleotides in one strand determine the sequence of nucleotides in the other. RNA molecules generally are single stranded, with a strand constructed from four types of ribonucleotide. Protein molecules consist of one or more polypeptide chains constructed from twenty types of amino acid. The building blocks of DNA, RNA and protein molecules frequently are represented by single letters from the Latin alphabet, as shown in Table 1-1.

Table 1-1. One letter codes for DNA, RNA and protein sequences

DNA Sequences		RNA Sequences	
A	Adenine	A	Adenine
C	Cytosine	C	Cytosine
G	Guanine	G	Guanine
T	Thymine	U	Uracil

Protein Sequences

A	Alanine	M	Methionine
C	Cysteine	N	Asparagine
D	Aspartic acid	P	Proline
E	Glutamic acid	Q	Glutamine
F	Phenylalanine	R	Arginine
G	Glycine	S	Serine
H	Histidine	T	Threonine
I	Isoleucine	V	Valine
K	Lysine	W	Tryptophan
L	Leucine	Y	Tyrosine

The primary biological function of a DNA molecule is carrying genetic information encoded in the sequence of its nucleotides. It is reasonable that much of the relevant information about a DNA molecule can be captured in an abstract sequence from an alphabet of four letters representing the four deoxyribonucleotides. In contrast, the biological function of many RNA and all protein molecules is dependent upon their conformation in space. It is thought that under physiological conditions this conformation is strongly dependent upon the sequence of building blocks from which the molecules are constructed. Predicting the three-dimensional structure of these molecules from their sequences alone remains a distant goal. Nevertheless, much information can be gleaned from protein sequence data. For example, two proteins with similar sequences may be evolutionarily related or may share a similar three-dimensional structure. We shall be content to represent RNA and protein as well as DNA molecules by abstract sequences of letters (Table 1-1).

Sequence transformation. Consider the three sequences "SHEHERAZADE", "SHEVARDNADZE" and "IGLOO" drawn from the space of sequences of letters of the Latin alphabet. Intuitively, the first two sequences are closer or more similar to one another than either is to the third. One approach to formalizing this concept is to define a set of operators on the space of sequences. For example, one operator might change the letter "A" to the letter "B"; a second might delete the letter "X"; a third might reverse the order of any two adjacent letters. A sufficiently rich set of operators will allow any sequence in the space to be transformed into any other by repeated use of the operators. If a positive cost is assigned to each

operator, then the cost of a given transformation of one sequence into another can be defined as the sum of the costs of the operators employed. The distance between two sequences can be defined as the minimum cost for transforming one into the other. Generally operators are defined so that if a given operator will transform sequence X to sequence Y, the inverse operator exists and has the same cost. When this is the case, distance as defined above behaves as a metric on the space of sequences.

Real-world considerations may affect the choice of operators and the costs assigned to them. For example, one may hope to correct spelling errors in typed text by finding the nearest neighbor(s) of an unrecognized word. A reasonable operator to use in such a context might be one that allowed two adjacent letters to be transposed. The cost of replacing one letter by another might be related to the physical distance of two letters on a keyboard. For errors made in electronic text transfer a totally different set of operators and costs would probably be desirable.

Sequence alignment. The theory of evolution provides the main justification for comparing biological sequences. Two proteins with the same function in different species may have evolved from the same ancestral protein; two sections of DNA in the same gene may have descended from a duplicated ancestral fragment. There are mechanisms of mutation that permit the substitution of one nucleotide for another and the deletion or insertion of one or more nucleotides. These mutations at the DNA level can appear at the protein level as substitutions, deletions or insertions. While more complicated mutations are possible, these three types are the most common.

When the only operators permitted are those that substitute one letter for another and those that delete or insert one or more letters, a natural object of study that is central to this thesis is a sequence alignment. An alignment of two sequences is a one-to-one ordered correspondence of their elements, where an arbitrary number of nulls (missing letters) may be inserted into each sequence. Two nulls may not be placed into correspondence. One alignment of "SHEHERAZADE" and "SHEVARDNADZE" is

SHE-HER--A-ZADE

SHEV-ARDNADZ--E

There is no simple correspondence between sequence transformations, as described above, and sequence alignments. For example, a transformation of "SHEHERAZADE" to "SHEVARDNADZE" may pass through the sequence "THEHERAZADE". There is no way of indicating this in an alignment.

Both sequence transformations and sequence alignments can be thought of as explaining relationships between sequences. Alignments promote parsimony of explanation. A letter of one sequence is either unaltered, changed into a different letter, or deleted. More roundabout explanations of relationship are excluded.

In this thesis, a subsequence of a sequence is a string of adjacent letters from the sequence. A subalignment of two sequences is an alignment of a subsequence from each sequence. A diagonal (sub)alignment is a (sub)alignment containing no nulls.

The cost of an alignment. As with sequence transformations, the intuition that one sequence alignment is better (more convincing as an

explanation of relationship) than another is formalized by defining a real-valued cost function on alignments. While an infinite variety of cost functions can be imagined, we use the operator notion to narrow the definition.

DEFINITION. A substitution cost function is a real-valued function on unordered pairs of letters.

DEFINITION. A gap cost function is a real-valued function on strings of letters.

DEFINITION. Given substitution and gap costs, the cost of an alignment or subalignment is the sum of the substitution cost for each pair of aligned letters and the gap cost for each maximal string of letters aligned with nulls.

In this thesis, we shall assume that alignments with lower cost are better and that all costs are rational and non-negative. Further, we shall assume that the gap cost of the concatenated string XY is never greater than the sum of the gap costs of the individual strings X and Y. For purposes of illustration we shall frequently use the substitution cost function $c_{id}(x,y)$ defined to be 0 when x = y and 1 otherwise.

In the literature of biological sequence comparison, alignment cost as defined above has sometimes been called distance or similarity (Needleman and Wunsch, 1970). We shall use the word distance to mean the minimum cost of aligning two sequences and shall reserve the word similarity for a concept to be elaborated in Chapter IV. Occasionally additional

constraints to those given above are either implied or explicitly imposed on the substitution and gap cost functions. For instance, Sellers (1974a,b) places conditions on substitution and gap cost functions that force the minimum cost of aligning two sequences to act as a metric on the space of sequences.

A brief history of biological sequence comparison. The method of dynamic programming (described below) was introduced to biological sequence comparison by Needleman and Wunsch (1970) to find optimal alignments of two protein sequences. Their algorithm had time complexity O(MN), where M is the length of the shorter and N the length of the longer sequence. Though they speculated on the possibility of using various substitution and gap costs, they confined their study to the case in which all gap costs are identical. Sankoff (1972) described how the dynamic programming approach could be modified so that only alignments with fewer than a fixed number of gaps were studied. Sellers (1974a), treating the null element as a member of the alphabet, proved that if the substitution costs provide a metric for the alphabet, then this metric can be extended to the space of sequences. He also showed (Sellers, 1974b) that a dynamic programming algorithm can be used to find the distance between any two sequences. Waterman et al. (1976) described an algorithm of time complexity $O(MN^2)$ for finding the optimal alignments of two sequences when arbitrary gap costs are allowed. This contrasted with Sellers' algorithm, which required the gap cost of a string to be the sum of the gap costs for each element of the string.

Dayhoff et al. (1978) derived a substitution cost function for protein sequence comparison from studies of related proteins with known sequences.

Schwartz and Dayhoff (1978) studied the relative effectiveness of this and other substitution cost functions for detecting distant relationships among proteins. Smith and Waterman (1981a,b) proposed that Needleman-Wunsch similarity can be used to define optimal subalignments for two sequences, and described an algorithm for finding optimal subalignments. Smith et al. (1981) showed that the Sellers minimum-cost and the Needleman-Wunsch maximum-similarity definitions of optimal alignments are equivalent.

Gotoh (1982) described an O(MN) time algorithm for finding an optimal alignment of two sequences when affine gap costs are used. His paper was only partly correct (Chapter III; Altschul and Erickson, 1986a) since his algorithm correctly finds the minimum alignment cost but in general does not produce an alignment that has this cost. Sankoff and Kruskal (1983) edited a book reviewing the entire field of sequence comparison, in which Erickson and Sellers (1983) described an algorithm for finding optimal alignments of a specified sequence with unspecified subsequences of a second sequence. Fitch and Smith (1983) showed that affine gap costs may be necessary to find the biologically correct alignment of two sequences.

Fitch (1983a) described a procedure for permuting a sequence while preserving its doublet frequency. His conjecture that this procedure generates with equal probability all sequences that have identical doublet composition is incorrect (Appendix; Altschul and Erickson, 1985). Ukkonen (1983) described an algorithm for finding the optimal alignment of two sequences which in most cases runs much faster than the traditional dynamic programming algorithm. Waterman (1984a) reviewed the field of sequence comparison.

Waterman (1984b) described two algorithms for finding the optimal alignment cost of two sequences when arbitrary concave gap costs are allowed. He conjectured that these algorithms have time complexity O(MN). This conjecture is incorrect (Chapter III; Altschul, 1987). Sellers (1984) defined local optimality for subalignments and described an algorithm that will find all and only the locally optimal subalignments of two sequences when a linear similarity function is used. He also showed that the algorithm of Smith and Waterman (1981a,b) can produce different "optimal" subalignments when the two input sequences are reversed.

Altschul and Erickson (1985) described an algorithm for randomly permuting a nucleotide sequence while preserving its dinucleotide, its dinucleotide and trinucleotide, or its dinucleotide and codon usage. Altschul and Erickson (1986a) described a modification of Gotoh's algorithm that correctly finds all and only the optimal alignments of two sequences in O(MN) time when affine gap costs are used. Altschul and Erickson (1986b) proposed the use of a nonlinear similarity function for comparing subalignments and studied the significance levels of this function when used for nucleic acid comparison. Altschul and Erickson (1986c) described algorithms for finding locally optimal subalignments of two sequences when nonlinear similarity functions are used. While this list of papers is not intended to be exhaustive, it does mention most of the papers with major bearing upon the concerns of this thesis.

Outline of the thesis. Chapter II describes the basic terminology and methods used through much of the thesis. In Chapter III a modification of Gotoh's algorithm is described that correctly finds in O(MN) time all and

only the optimal alignments of two sequences when affine gap costs are employed. A more detailed form of path graph than that traditionally used is needed to represent accurately all and only the optimal alignments. Also, the time complexity of Waterman's 1984 algorithms for concave gap costs is analyzed.

The problem of finding locally optimal subalignments can be divided into three separate problems: first, defining a similarity function with which to compare subalignments; second, devising algorithms to find subalignments which maximize the chosen similarity function; third, assessing the statistical significance of a subalignment in the context of the search performed. These three problems are addressed in Chapters IV, V and VI. In Chapter IV, functions nonlinear in length and cost are proposed as more appropriate measures of subalignment similarity than the linear functions previously used. In Chapter V, various algorithms are described for finding locally optimal subalignments using the criterion of any reasonable similarity function. In Chapter VI, the distribution of optimal similarity scores from the comparison of two nucleic acid or protein sequences is studied.

Chapter VII presents applications of the methods developed in earlier chapters. The Appendix presents a method for randomly permuting nucleic acid sequences while preserving their dinucleotide, dinucleotide and trinucleotide, or dinucleotide and codon usage. This method can be useful in Monte Carlo simulations for preserving statistical correlations often found in real biological sequences.

II. BASIC TERMINOLOGY AND METHODS

Path graphs. Almost all of the algorithms described in this thesis are based on the underlying concept of a path graph. The two sequences being compared are $X = x_1x_2...x_M$ and $Y = y_1y_2...y_N$. Without loss of generality, X is the shorter of the two sequences, so that $M \leq N$. The path graph for these two sequences consists of $(M+1)(N+1)$ nodes and $3MN + M + N$ edges, as shown in Figure 2-1a.

Each node $N_{i,j}$ has as many as six adjacent edges (Figure 2-1b). If each is present, the upper adjacent edges of $N_{i,j}$ are $V_{i,j}$, $H_{i,j}$ and $D_{i,j}$ and the lower adjacent edges are $V_{i+1,j}$, $H_{i,j+1}$ and $D_{i+1,j+1}$. Similarly, each node $N_{i,j}$ has as many as six adjacent nodes. The upper adjacent nodes of $N_{i,j}$ are $N_{i-1,j}$, $N_{i,j-1}$ and $N_{i-1,j-1}$ and the lower adjacent nodes are $N_{i+1,j}$, $N_{i,j+1}$ and $N_{i+1,j+1}$.

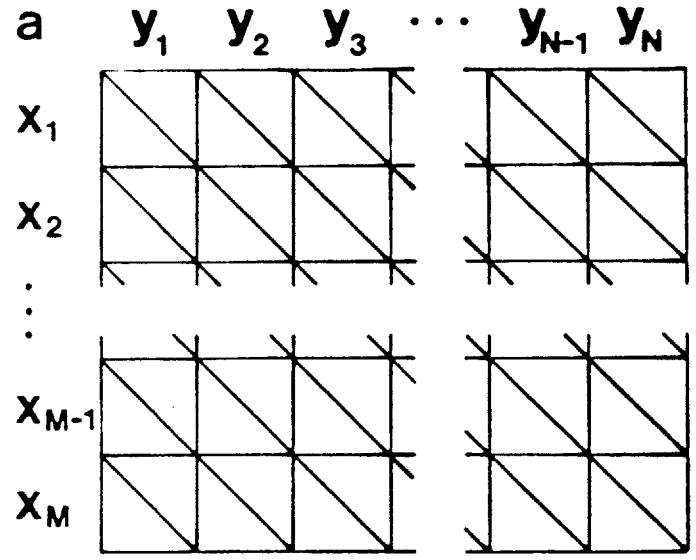
DEFINITION. A path in a path graph is a sequence of nodes and edges $n_0e_1n_1...e_zn_z$ where for all $i\varepsilon\{1,2,...,z\}$ n_{i-1} is an upper adjacent node of n_i and e_i is the edge connecting them.

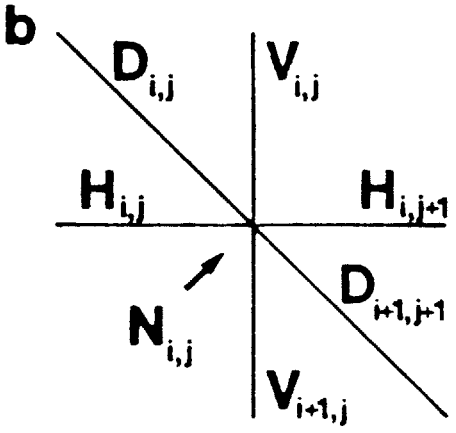
Each subalignment of X and Y corresponds to a unique path in the path graph, and conversely. Each alignment of X and Y corresponds to a unique path from $N_{0,0}$ to $N_{M,N}$, and conversely. In general, vertical edge $V_{i,j}$ aligns x_i with a null after y_j, horizontal edge $H_{i,j}$ aligns y_j with a null after x_i, and diagonal edge $D_{i,j}$ aligns x_i with y_j (Figure 2-1). Because of this isomorphism, we shall frequently refer to paths in path graphs as subalignments or alignments.

Figure 2-1. Nomenclature for a path graph.

(a) The complete graph for sequences $X = x_1x_2...x_M$ and $Y = y_1y_2...y_N$.

(b) The node $N_{i,j}$ and its six adjacent edges.





The locus L_k is the set of index pairs (i,j) such that i + j = k. For example, the set $\{N_{3,0}, N_{2,1}, N_{1,2}, N_{0,3}\}$ contains the nodes of locus L_3. Pair (i,j) is below L_k if i + j > k and above L_k if i + j < k. The notation $Z_{i,j}$ (note comma) refers to a specific element of the array Z_{ij} (no comma).

The algorithms described in this thesis associate various arrays with the nodes and edges of a path graph. They frequently use six rectangular number arrays (P_{ij}, Q_{ij}, R_{ij}, A_{ij}, B_{ij}, C_{ij}). The first three arrays store numbers associated with the graph nodes N_{ij}; the last three store numbers associated respectively with the graph edges V_{ij}, H_{ij} and D_{ij}. The algorithms also frequently use seven rectangular bit arrays (a_{ij}, b_{ij}, ..., g_{ij}) to store data associated with the graph edges. Arrays a_{ij}, d_{ij} and e_{ij} are associated with the graph edges V_{ij}; arrays b_{ij}, f_{ij} and g_{ij} are associated with the graph edges H_{ij}; array c_{ij} is associated with the graph edges D_{ij}. For all arrays, index i ranges from 0 to M+1 and index j ranges from 0 to N+1. It is possible to limit the first index to the range 0 to M and the second index to the range 0 to N, but then dealing with the borders of the arrays sometimes becomes more complicated.

Dynamic programming. A good description of the dynamic programming method can be found in Kruskal (1983). The main idea is that the optimal alignments of two sequences can be found by an inductive procedure that discovers first the optimal alignments of initial subsequences. The simplest example of dynamic programming arises when we treat a null (represented by the character '-') as if it were another letter of the alphabet, so that only substitution costs c(x,y) need be specified

(Sellers, 1974a). Using the notation developed above, the basic dynamic programming algorithm for finding all and only the optimal alignments of two sequences can be written as follows. All number variables with a negative index are assumed to have value $+\infty$. All bit variables with first index greater than M or second index greater than N are assumed to have value 0.

{1} For i from 0 to M and j from 0 to N:

If $i = j = 0$ set $R_{i,j}$ to 0; otherwise set $R_{i,j}$ to
$\min [R_{i-1,j} + c(x_i,-),\ R_{i,j-1} + c(-,y_j),\ R_{i-1,j-1} + c(x_i,y_j)]$.
If $R_{i,j} = R_{i-1,j} + c(x_i,-)$ set $a_{i,j}$ to 1, otherwise to 0.
If $R_{i,j} = R_{i,j-1} + c(-,y_j)$ set $b_{i,j}$ to 1, otherwise to 0.
If $R_{i,j} = R_{i-1,j-1} + c(x_i,y_j)$ set $c_{i,j}$ to 1, otherwise to 0.

{2} For i from M to 0 and j from N to 0:

Except when $i = M$ and $j = N$, if $a_{i+1,j} = b_{i,j+1} = c_{i+1,j+1} = 0$, set $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ to 0.

After step {1}, $R_{M,N}$ records the minimum cost of aligning sequences X and Y. After step {2}, all and only the optimal alignments of X and Y are represented by paths from $N_{0,0}$ to $N_{M,N}$ that use only edges whose associated bits are set to 1 in arrays a_{ij}, b_{ij} and c_{ij}. Both the time and space complexity of the basic dynamic programming algorithm are O(MN).

III. OPTIMAL SEQUENCE ALIGNMENT USING AFFINE GAP COSTS

While it is possible to define gap costs that depend explicitly upon the string deleted or inserted, it is generally difficult to find a justification for doing so. The usual practice and, for ease of presentation, the one we shall adopt is to choose gap costs dependent only on the length of the string. Such gap costs can be represented by a function $w(x)$, where x is the length of the gap.

Sellers' algorithm for finding all optimal alignments (1974a,b) has time complexity $O(MN)$. The algorithm requires the cost of a gap to be the sum of costs for each letter in the gap. Such gap costs will be called linear because if the cost for each letter is U then $w(x) = Ux$.

Since a single mutation event can insert or delete an entire segment of a genetic sequence, a long gap should arguably cost only slightly more than a shorter one. Waterman et al. (1976) have generalized the Sellers algorithm so that any gap cost function $w(x)$ can be used. Fitch and Smith (1983) have discussed a case in which such gap costs are necessary in order to produce the correct alignment. The major disadvantage of the algorithm of Waterman et al. (1976) is that its time complexity is $O(MN^2)$. Recently, Waterman (1984b) has described two algorithms for concave gap costs that he conjectures to have time complexity $O(MN)$. It is shown below that this conjecture is incorrect.

The general approach of Waterman et al. (1976) allows each null in a gap to have a different cost. Gotoh (1982) considered the more restricted case of affine gap costs. Specifically, the cost of a gap is V plus the

sum of costs for each letter in the gap. If the cost for each letter is U, then $w(x) = V + Ux$, where $V, U \geq 0$. The major advantage of Gotoh's algorithm is that it finds the minimum cost of aligning two sequences in O(MN) steps.

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

We describe in this chapter a modification of Gotoh's algorithm, called the SS-2 algorithm, that correctly finds all and only the optimal alignments of two sequences in O(MN) steps. We also present two sequences and a set of gap costs for which Gotoh's algorithm fails to find the single optimal alignment, and two sequences for which Taylor's algorithm can not find all and only the optimal alignments. First, we introduce a more precise form of path graph than previously used, which is needed to represent accurately all optimal alignments for affine gap costs. In this chapter the gap costs c_{id} are used for purposes of illustration.

Affine path graphs. If linear gap costs are employed, all optimal alignments of two sequences can simultaneously be represented in a standard linear path graph. For example, if $w(x) = x$, the five optimal alignments

of sequences AGCCT and AGGTCC are represented by the five overlapping paths of the linear path graph in Figure 3-1. For affine gap costs, however, a linear path graph can be ambiguous in indicating precisely which paths are optimal. For example, if $w(x) = 1 + x$, sequences AGT and TGAGTT have a minimum alignment cost of 5. Panels a-c of Figure 3-2 show all three optimal alignments with this cost and the corresponding simple path graphs. In Figure 3-3a these three optimal paths are combined to give a composite graph. This graph contains a fourth path (Figure 3-2d) but fails to indicate that this fourth path is not optimal.

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

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

Figure 3-1. Paths representing the five optimal alignments of AGCCT and AGGTCC.

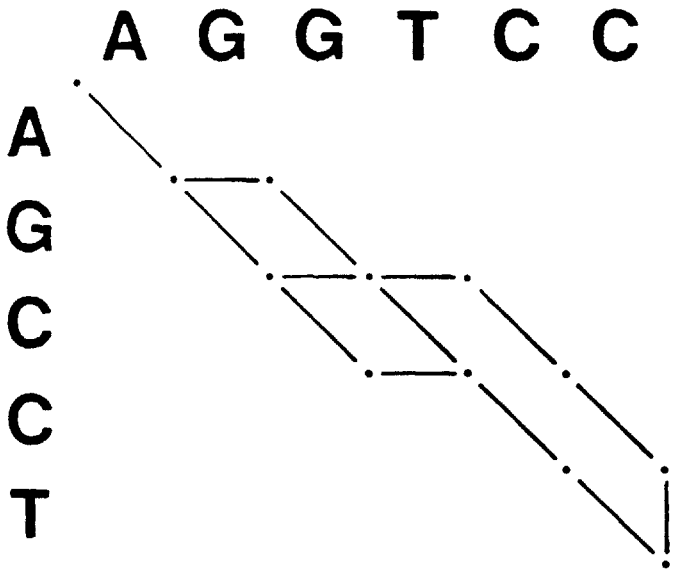


Figure 3-2. Four alignments of AGT and TGAGTT, and their path graphs.

(a-c) Optimal alignments for $w(x) = 1 + x$.

(d) A non-optimal alignment.

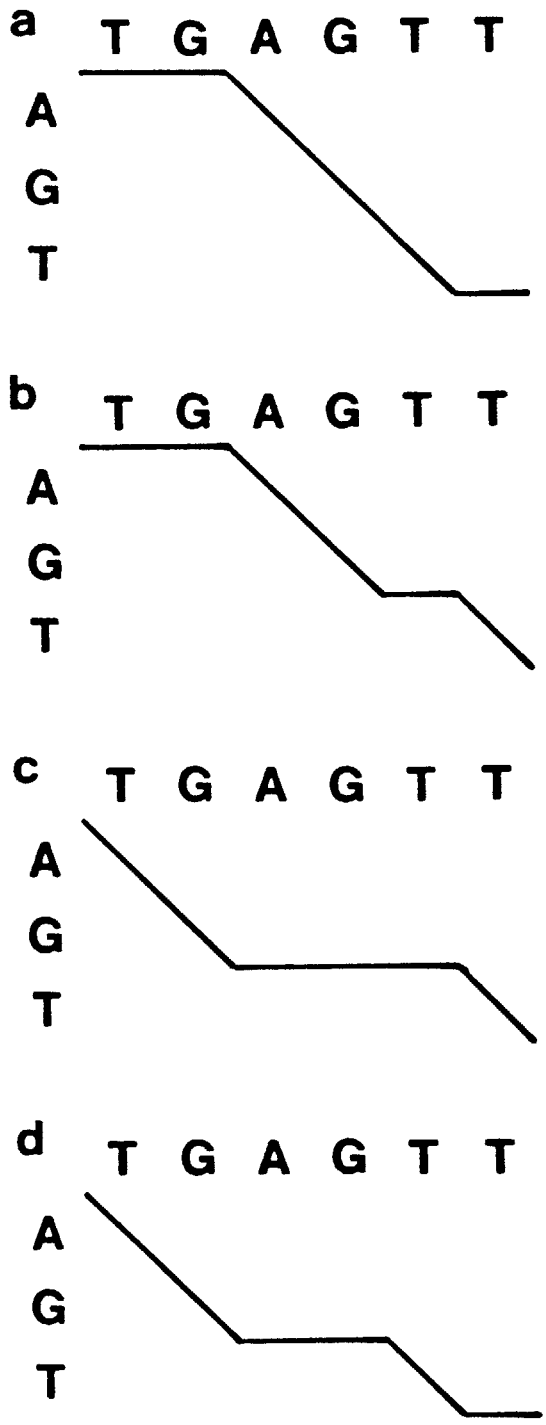


Figure 3-3. Composite path graphs representing the optimal alignments of AGT and TGAGTT for $w(x) = 1 + x$.

(a) The linear path graph.

(b) An affine path graph representing only the optimal alignments.

(c) The affine path graph produced by the SS-2 algorithm.

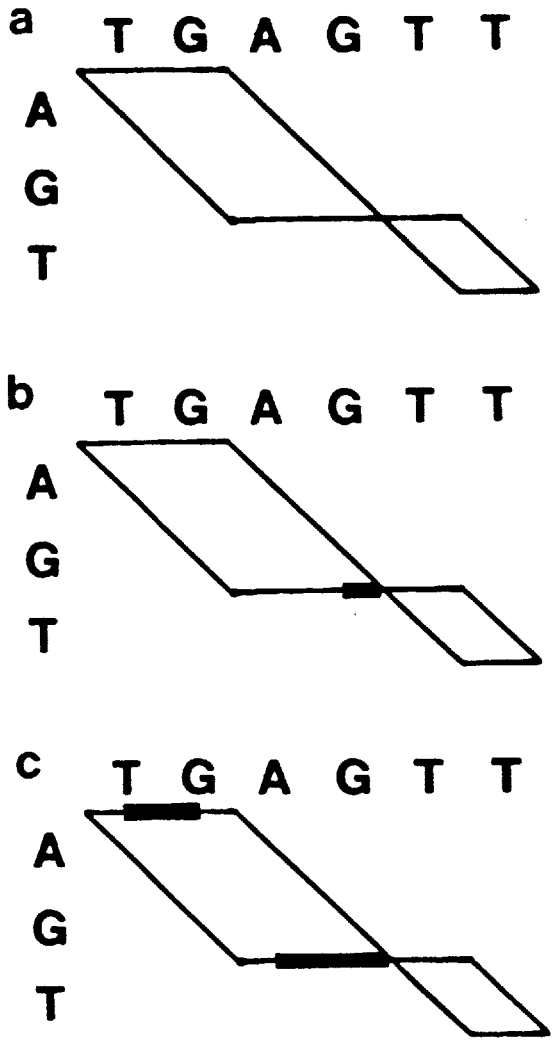
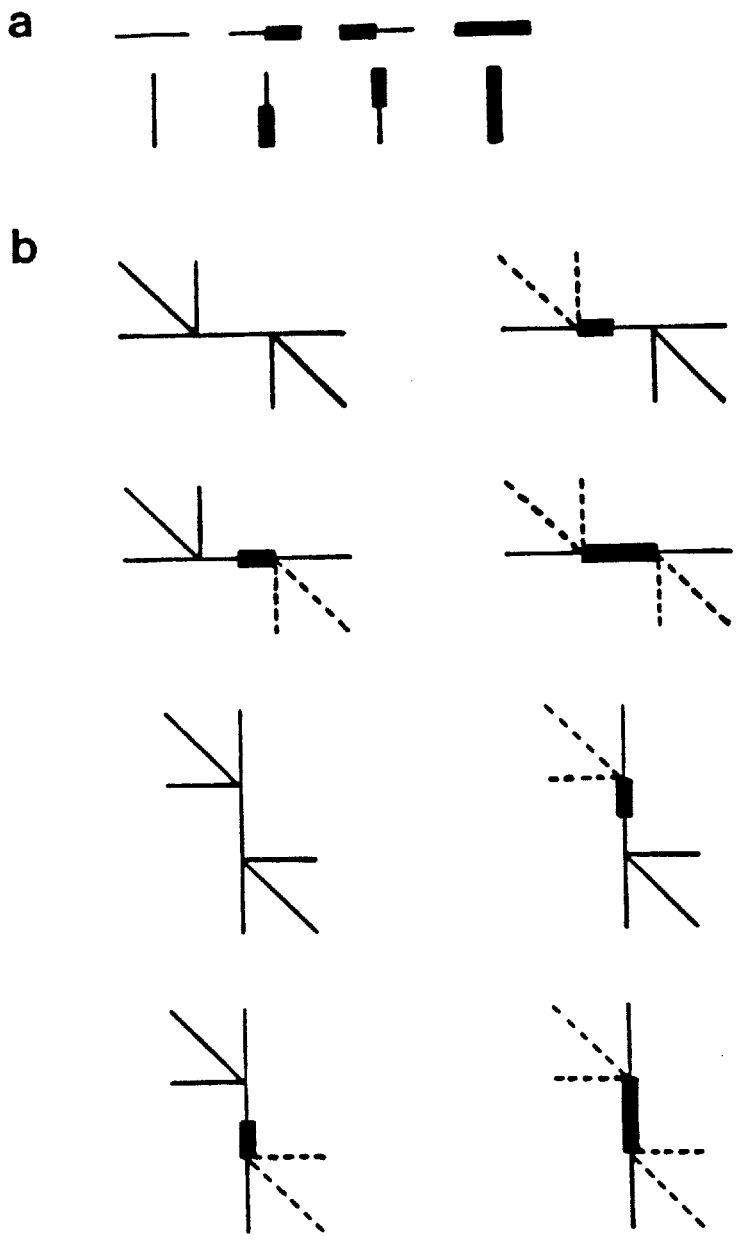


Figure 3-4. New edge symbols and their meaning.

(a) Eight symbols for horizontal and vertical edges.

(b) No path using the central edge may use the dotted edges.



When affine gap costs are used, the minimum cost of continuing a path from a given node to the lower right node of the graph may depend upon whether the given node was entered using a vertical, horizontal or diagonal edge. Paths that enter a node through different edges may have different optimal continuations. Thus new edge symbols, such as those in Figure 3-4a, are necessary if all and only the optimal paths are to be represented in a single path graph. Using affine gap costs $w(x) = V + Ux$ is equivalent to charging $V + U$ for the first null in a gap and U for each subsequent null. Since any path that uses a vertical or horizontal edge has already opened a gap, each subsequent null in the gap will have identical cost U. Thus all paths that enter a node through a given edge will have the same optimal continuations. Therefore the edge symbols of Figure 3-4a provide sufficient modification of the linear path graph to indicate precisely all and only the optimal alignments for affine gap costs.

When non-affine gap costs are employed, however, even an affine path graph will not suffice in the general case to represent accurately all and only the optimal alignments of two sequences. For example, if $w(1) = 1.2$ and $w(x) = 0.7 + 0.7(x)$ for $x > 1$, there are two optimal alignments of the sequences AGTCGA and GTTACCG (Figure 3-5a). A linear path graph containing the paths that represent each of these alignments appears in Figure 3-5c. It is not possible to use the horizontal edge symbols of Figure 3-4a in this graph in a way that includes both optimal alignments but excludes the two non-optimal alignments shown in Figure 3-5b. Further generalization of the vertical and horizontal edge symbols, however, would allow precise

Figure 3-5. Non-affine gap costs.

(a) The two optimal alignments of AGTCGA and GTTACCG for $w(1) = 1.2$ and $w(x) = 0.7 + 0.7(x)$ when $x > 1$.

(b) Two non-optimal alignments.

(c) A composite path graph representing the optimal alignments.

a

	A		G		T		-		C		G		A	
Cost:	1	+	1	+	0	+	1.2	+	0	+	1	+	1	= 5.2
	G		T		T		A		C		C		G	

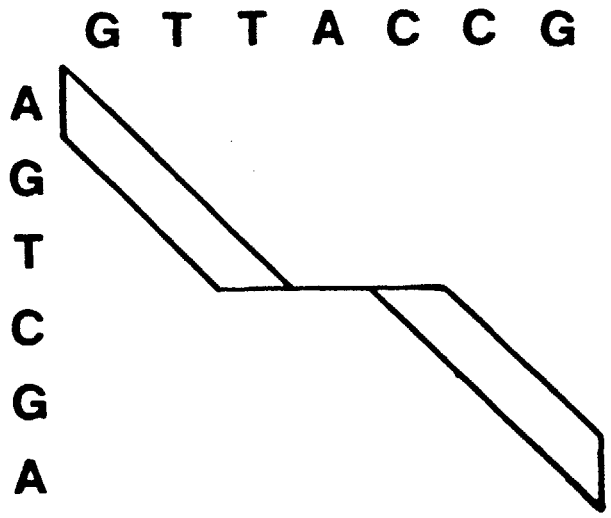
	A		G		T		-		-		-		C		G		A	
Cost:	1.2	+	0	+	0	+	1.2	+	0.9	+	0.7	+	0	+	0	+	1.2	= 5.2
	-		G		T		T		A		C		C		G		-	

b

	A		G		T		-		-		C		G		A	
Cost:	1	+	1	+	0	+	1.2	+	0.9	+	0	+	0	+	1.2	= 5.3
	G		T		T		A		C		C		G		-	

	A		G		T		-		-		C		G		A	
Cost:	1.2	+	0	+	0	+	1.2	+	0.9	+	0	+	1	+	1	= 5.3
	-		G		T		T		A		C		C		G	

c



representation of the optimal alignments implied by these gap costs. For general gap costs, a set of pointers can be used at each node to indicate from which nodes above or to the left optimal alignments can come.

An alignment algorithm for affine gap costs. An extension of Gotoh's algorithm is presented here that finds all and only the optimal alignments of two sequences for affine gap costs and represents them by an affine path graph in the manner described above. Let the affine gap costs be $w(x) = V + Ux$. In place of the arrays P and R two one-dimensional number arrays, and in place of array Q a variable, provide sufficient storage (Gotoh, 1982). The three rectangular arrays are used here for ease of exposition.

The SS-2 algorithm consists of the following 11 steps. It calculates the minimum alignment cost, $R_{M,N}$, and an affine path graph containing all and only those paths that represent optimal alignments of sequences X and Y. All statements involving a negative index should be omitted.

INITIALIZATION. Execute step {1}.

{1} Initialize the number and bit arrays:

For j from 0 to N, set $P_{0,j}$ to $+\infty$ and $R_{0,j}$ to $V + Uj$.

For i from 0 to M, set $Q_{i,0}$ to $+\infty$ and $R_{i,0}$ to $V + Ui$.

Set $R_{0,0}$ to 0.

Set bit arrays a-g uniformly to 0.

Set $c_{M+1,N+1}$ to 1.

COST ASSIGNMENT. For i from 0 to M and j from 0 to N, execute steps {2} to {7}.

{2} Find the minimum cost of a path ending at node $N_{i,j}$ and using edge $V_{i,j}$:

Set $P_{i,j}$ to U + min ($P_{i-1,j}$, $R_{i-1,j}$ + V).

{3} Determine if cost $P_{i,j}$ can be achieved using edge $V_{i-1,j}$ and if it can be achieved without using edge $V_{i-1,j}$:

If $P_{i,j} = P_{i-1,j} + U$, set $d_{i-1,j}$ to 1.

If $P_{i,j} = R_{i-1,j} + V + U$, set $e_{i-1,j}$ to 1.

{4} Find the minimum cost of a path ending at node $N_{i,j}$ and using edge $H_{i,j}$:

Set $Q_{i,j}$ to U + min ($Q_{i,j-1}$, $R_{i,j-1}$ + V).

{5} Determine if cost $Q_{i,j}$ can be achieved using edge $H_{i,j-1}$ and if it can be achieved without using edge $H_{i,j-1}$:

If $Q_{i,j} = Q_{i,j-1} + U$, set $f_{i,j-1}$ to 1.

If $Q_{i,j} = R_{i,j-1} + V + U$, set $g_{i,j-1}$ to 1.

{6} Find the minimum cost of a path ending at node $N_{i,j}$:

Set $R_{i,j}$ to min ($P_{i,j}$, $Q_{i,j}$, $R_{i-1,j-1} + c(x_i,y_j)$).

{7} Determine if cost $R_{i,j}$ can be achieved by using edge $V_{i,j}$, $H_{i,j}$ or $D_{i,j}$:

If $R_{i,j} = P_{i,j}$, set $a_{i,j}$ to 1.

If $R_{i,j} = Q_{i,j}$, set $b_{i,j}$ to 1.

If $R_{i,j} = R_{i-1,j-1} + c(x_i,y_j)$, set $c_{i,j}$ to 1.

EDGE ASSIGNMENT. For i from M to 0 and j from N to 0, execute steps {8} to {11}.

{8} If there is no optimal path passing through node $N_{i,j}$ which has cost $R_{i,j}$ at node $N_{i,j}$, remove edges $V_{i,j}$, $H_{i,j}$ and $D_{i,j}$:

If ($a_{i+1,j} = 0$ or $e_{i,j} = 0$) and ($b_{i,j+1} = 0$ or $g_{i,j} = 0$) and ($c_{i+1,j+1} = 0$),

set $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ to 0.

{9} If no optimal path passes through $N_{i,j}$, proceed to the next node:

If $a_{i+1,j} = b_{i,j+1} = c_{i+1,j+1} = 0$,

skip steps {10} and {11}.

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

If $a_{i+1,j} = 1$ and $d_{i,j} = 1$,

[set $d_{i+1,j}$ to $1 - e_{i,j}$, set $e_{i,j}$ to $1 - a_{i,j}$ and]

set $a_{i,j}$ to 1.

[Otherwise, set $d_{i+1,j}$ and $e_{i,j}$ to 0.]

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

If $b_{i,j+1} = 1$ and $f_{i,j} = 1$,

[set $f_{i,j+1}$ to $1 - g_{i,j}$, set $g_{i,j}$ to $1 - b_{i,j}$ and]

set $b_{i,j}$ to 1.

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

Comments on the SS-2 algorithm. The meaning of the bit arrays changes during the execution of the algorithm. Let an (i,j) path be a path from $N_{0,0}$ to $N_{i,j}$. After the cost assignment section is complete, the seven bit arrays store the following information:

$a_{i,j}$ = 1 iff an optimal (i,j) path uses $V_{i,j}$.

$b_{i,j}$ = 1 iff an optimal (i,j) path uses $H_{i,j}$.

$c_{i,j}$ = 1 iff an optimal (i,j) path uses $D_{i,j}$.

$d_{i,j}$ = 1 iff among (i+1,j) paths through $N_{i,j}$, an optimal one uses $V_{i,j}$.

$e_{i,j}$ = 1 iff among (i+1,j) paths through $N_{i,j}$, an optimal one does not use $V_{i,j}$.

$f_{i,j}$ = 1 iff among (i,j+1) paths through $N_{i,j}$, an optimal one uses $H_{i,j}$.

$g_{i,j}$ = 1 iff among (i,j+1) paths through $N_{i,j}$, an optimal one does not use $H_{i,j}$.

After the edge assignment section is complete, the seven bit arrays store the affine path graph, as illustrated in Figure 3-6 and described below:

$a_{i,j}$ = 1 iff an optimal (M,N) path uses $V_{i,j}$.

$b_{i,j}$ = 1 iff an optimal (M,N) path uses $H_{i,j}$.

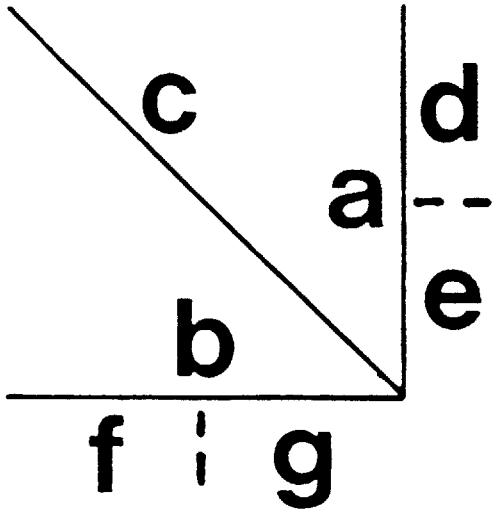
$c_{i,j}$ = 1 iff an optimal (M,N) path uses $D_{i,j}$.

$d_{i,j}$ = 1 iff every optimal (M,N) path that uses $V_{i,j}$ also uses $V_{i-1,j}$. (The top half of $V_{i,j}$ is bold.)

$e_{i,j}$ = 1 iff every optimal (M,N) path that uses $V_{i,j}$ also uses $V_{i+1,j}$. (The bottom half of $V_{i,j}$ is bold.)

$f_{i,j}$ = 1 iff every optimal (M,N) path that uses $H_{i,j}$ also uses

Figure 3-6. Bit array assignments. Arrays a-c correspond to full edges and d-g to half edges.



$H_{i,j-1}$. (The left half of $H_{i,j}$ is bold.)

$g_{i,j}$ = 1 iff every optimal (M,N) path that uses $H_{i,j}$ also uses $H_{i,j+1}$. (The right half of $H_{i,j}$ is bold.)

Note that if $a_{i,j} = 0$, bits $d_{i,j}$ and $e_{i,j}$ are meaningless. Similarly, if $b_{i,j} = 0$, $f_{i,j}$ and $g_{i,j}$ are meaningless. Table 3-1 shows the values of the number and bit arrays after the cost and edge assignment for sequences X = AGT and Y = TGAGTT where w(x) = 1 + x. The affine path graph for this example is presented in Figure 3-3c.

Since the algorithm involves a fixed number of steps for each node, its execution requires O(MN) steps. In the initialization step, ∞ need only be a number larger than any number with which it will be compared during execution of the algorithm; the number 2V + U max(M,N) + 1 will suffice. The bit $c_{M+1,N+1}$ is initially set to 1 so that the conditions of steps {8} and {9} are false for node $N_{M,N}$. If the four expressions in brackets in steps {10} and {11} are omitted, the linear path graph represented by bit arrays a-c contains all and only those edges that are part of <u>some</u> optimal path; such a graph often contains non-optimal paths. All seven bit arrays are still required to find these edges. While a specific set of substitution costs has been used in all examples in this paper, the SS-2 algorithm allows any set of substitution costs to be employed. Also, slight modification will allow it to employ different sets of affine gap costs for terminal and interior gaps. When the SS-2 algorithm is implemented on a computer lacking bit storage, the seven bits associated with a given node may conveniently be packed into a single byte.

Table 3-1. Number and bit arrays during execution of the SS-2 algorithm

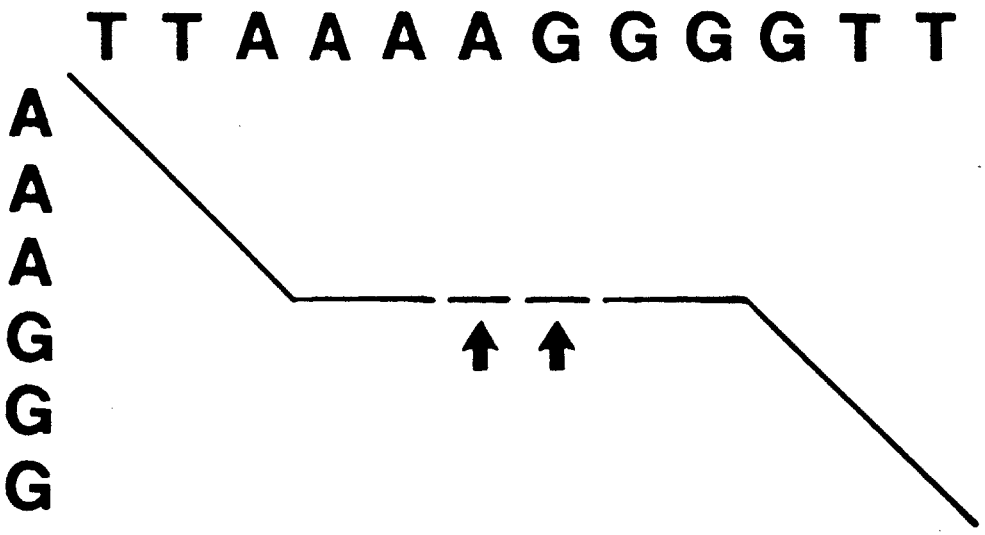
Node Index		After Cost Assignment								After Edge Assignment						
i	j	P	Q	R	a	b	c	d	e	f	g	a	b	c	d	e	f	g
0	0	∞	∞	0	0	0	0	0	1	0	1	0	0	0	*	*	*	*
0	1	∞	2	2	0	1	0	0	1	1	0	0	1	0	*	*	0	1
0	2	∞	3	3	0	1	0	0	1	1	0	0	1	0	*	*	1	0
0	3	∞	4	4	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	4	∞	5	5	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	5	∞	6	6	0	1	0	0	1	1	0	0	0	0	*	*	*	*
0	6	∞	7	7	0	1	0	0	1	0	0	0	0	0	*	*	*	*
1	0	2	∞	2	1	0	0	1	0	0	1	0	0	0	*	*	*	*
1	1	4	4	1	0	0	1	0	1	0	1	0	0	1	*	*	*	*
1	2	5	3	3	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	3	6	4	3	0	0	1	0	1	1	1	0	0	1	*	*	*	*
1	4	7	5	5	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	5	8	6	6	0	1	1	0	1	1	0	0	0	0	*	*	*	*
1	6	9	7	7	0	1	1	0	1	0	0	0	0	0	*	*	*	*
2	0	3	∞	3	1	0	0	1	0	0	1	0	0	0	*	*	*	*
2	1	3	5	3	1	0	1	1	0	0	1	0	0	0	*	*	*	*
2	2	5	5	1	0	0	1	0	1	0	1	0	0	1	*	*	*	*
2	3	5	3	3	0	1	0	0	1	1	0	0	1	0	*	*	0	1
2	4	7	4	3	0	0	1	0	1	1	1	0	1	1	*	*	1	1
2	5	8	5	5	0	1	0	0	1	1	0	0	1	0	*	*	0	0
2	6	9	6	6	0	1	0	0	1	0	0	0	0	0	*	*	*	*
3	0	4	∞	4	1	0	0	0	0	0	1	0	0	0	*	*	*	*
3	1	4	6	3	0	0	1	0	0	0	1	0	0	0	*	*	*	*
3	2	3	5	3	1	0	0	0	0	0	1	0	0	0	*	*	*	*
3	3	5	5	2	0	0	1	0	0	0	1	0	0	0	*	*	*	*
3	4	5	4	4	0	1	1	0	0	1	0	0	0	0	*	*	*	*
3	5	7	5	3	0	0	1	0	0	0	1	0	0	1	*	*	*	*
3	6	8	5	5	0	1	1	0	0	0	0	0	1	1	*	*	0	0

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

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

If $w(x) = 1 + x$, sequences AGT and TGAGTT have only the three optimal alignments shown in panels a-c of Figure 3-2. Taylor's algorithm for finding all optimal alignments (Taylor, 1984) saves at each node one vertical and one horizontal pointer for path traceback. Such a pointer system can find any pair of optimal alignments or all four alignments of Figure 3-2, but it can not find precisely the three optimal alignments. In order to identify all and only the optimal alignments in the general case, a pointer system similar to that described by Smith et al. (1981) or Taylor (1984) must allow for an arbitrary number of pointers at each node, where many bits may be required to represent each pointer. In contrast, the system described above requires the storage of only seven bits per node even when gaps of arbitrary length are present. The optimal alignments can easily be represented by the corresponding affine path graph.

Figure 3-7. The optimal alignment of AAAGGG and TTAAAAGGGGTT for $w(x) = 5 + x$. Arrows indicate two edges of the path graph not saved by Gotoh's algorithm.



	A	A	A	-	-	-	-	-	-	G	G	G		
Cost:	1	1	0	6	1	1	1	1	1	0	1	1	=	15
	T	T	A	A	A	A	G	G	G	G	T	T		

Alternatively, all optimal alignments may be extracted from the bit arrays a-g and output linearly, as shown on the right of Figure 3-2. When non-affine gap costs are employed, a generalization of this system may be inferior to a system of pointers.

A biological example. The advantage of using affine gap costs when comparing biological sequences is illustrated by two DNA sequences for interleukin 2 (IL-2), an important regulator of T-cell clonal expansion. The DNA sequences code for human IL-2 (Taniguchi et al., 1983) and murine IL-2 (Yokata et al., 1985). The DD algorithm (see Chapter V) was used to search for interesting gap-free subalignments of the two IL-2 sequences using similiarity function s_1 (see Chapter IV). Two of the four best subalignments found were human segment 65-107 with mouse segment 77-119 (43 nucleotides) and human segment 91-299 with mouse segment 133-341 (209 nucleotides). The ends of these subalignments overlap, as shown by the two paths in Figure 3-8a for that part of the DD graph involving human segment 76-107 (H) and murine segment 88-149 (M). Joining these two subalignments requires a net deletion of 30 nucleotides from M, which can be achieved by inserting one or more gaps into segment H.

Using the SS algorithm (Sellers, 1974a,b) and gap costs $w(x) = 2x$ (Erickson and Sellers, 1983) to align segments H and M produces a large number of optimal subalignments (Figure 3-8b). Because it costs nothing to open a gap, every optimal alignment contains at least 12 separate gaps in segment H, as illustrated by line H1 of Figure 3-9. These alignments imply that at least 12 insertions or deletions are needed to explain the evolutionary divergence of segments H and M from a common ancestral gene

Figure 3-8. Five path graphs for two DNA sequences from interleukin 2. Vertical sequence M is a 62-nucleotide murine segment and horizontal sequence H is a 32-nucleotide human segment.

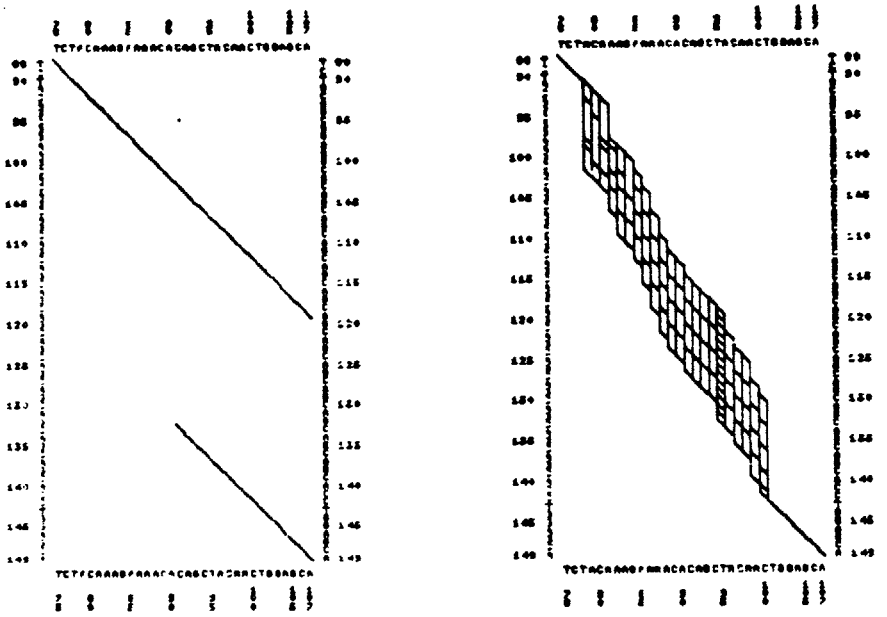
(a) Part of a larger DD path graph.

(b) The linear path graph for $w(x) = 2x$; paths contain 12-18 gaps.

(c) The affine path graph for $w(x) = 0.5 + Ux$ and $U \geq 1$; paths contain 3-11 gaps.

(d) The affine path graph for $w(x) = 1 + Ux$ and $U \geq 1$; paths contain 1-3 gaps.

(e) The affine path graph for $w(x) = V + Ux$, $V > 1$ and $U \geq 0.5$; paths contain only 1 gap.



a b

Figure 3-8. (Continued). Five path graphs for two DNA sequences from interleukin 2. Vertical sequence M is a 62-nucleotide murine segment and horizontal sequence H is a 32-nucleotide human segment.

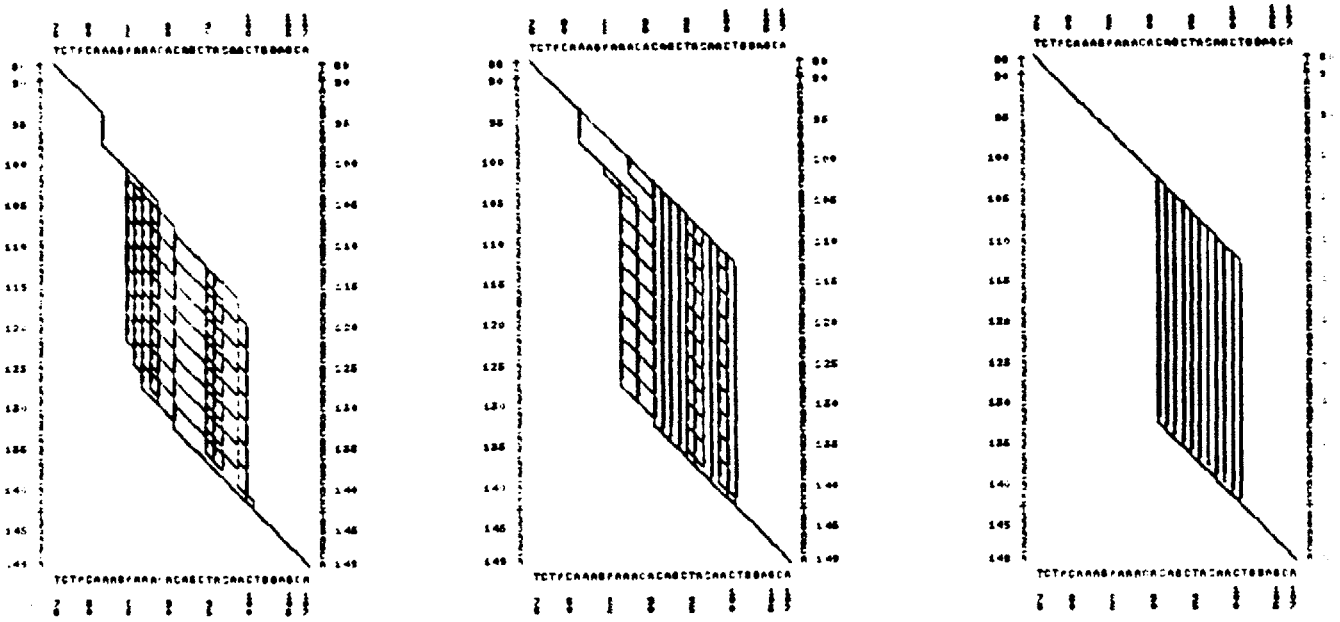
(a) Part of a larger DD path graph.

(b) The linear path graph for $w(x) = 2x$; paths contain 12-18 gaps.

(c) The affine path graph for $w(x) = 0.5 + Ux$ and $U \geq 1$; paths contain 3-11 gaps.

(d) The affine path graph for $w(x) = 1 + Ux$ and $U \geq 1$; paths contain 1-3 gaps.

(e) The affine path graph for $w(x) = V + Ux$, $V > 1$ and $U \geq 0.5$; paths contain only 1 gap.



c d e

Figure 3-9. Three representative optimal alignments of murine segment M and human segment H from interleukin 2. Each of the lines H1-H3 is aligned with line M. Smaller letters are different from the corresponding letters in M.

```
  88         100        110        120        130        140        149
M  TCTACAGCGGAAGCACAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCACCTGGAGCA
H1 TCTACA----AAG-A-A--A--A-CA-CA-CAGCT--A-CA--A--A-C------TGGAGCA
H2 TCTACAAAGAAAACA----------------------------CAGCTACAACTGGAGCA
H3 TCTACAAAGAAAACACAGCTACAAC----------------------------TGGAGCA
  76  80         90        100                             101   107
```

segment. Such a large number of events is considered to be unlikely.

The number of insertions or deletions needed becomes smaller when a positive cost V is imposed for opening a gap (Figure 3-8c,d,e). For $V > 1$ and $U \geq 0.5$, all 30 nulls must be joined into a single gap, which can be inserted into any one of 11 different places in segment H (Figure 3-8e). Specifically, it can be placed after nucleotide 90 (line H2 of Figure 3-9), after nucleotide 100 (line H3 of Figure 3-9), or after any nucleotide in between. Since murine segment 103-132 encodes 10 glutamine residues, the alignment of lines M and H2 in Figure 3-9 seems the most plausible. Our experience in comparing nucleotide sequences using affine gap costs suggests that the costs $w(x) = 2.5 + 0.5(x)$ are useful.

Concave gap costs. The algorithm of Waterman et al. (1976) allows arbitrary gap costs $w(x)$. It finds the optimal alignment cost by setting

$$R_{i,j} = \min [(R_{i-k,j} + w(k))_{k \leq i}, (R_{i,j-1} + w(l))_{l \leq j}, R_{i-1,j-1} + c(x_i, y_j)]$$

The time complexity of this algorithm is $O(MN^2)$.

Waterman (1984b) has recently described two new algorithms for concave gap costs that he conjectures have time complexity essentially $O(MN)$. It is shown below that the first of these algorithms has worst-case time complexity at least $O(M^2N)$. For reasonable concave gap costs its average time complexity is also $O(M^2N)$. This algorithm therefore is asymptotically better than that of Waterman et al. (1976) for sequences with greatly different lengths, but for the usual case of sequences with approximately equal length it is better by at most a constant factor. The average time complexity of the second algorithm of Waterman (1984b) is difficult to

analyze. Its worst-case time complexity appears to be $O[\max(M^3,MN)]$.

DEFINITION. Gap costs w(x) are concave (Waterman, 1984b) if for all $i,j,k > 0$, $w(i+j+k) - w(i+k) \leq w(j+k) - w(k)$.

Waterman's first algorithm (1984b) for finding the minimum cost, $R_{M,N}$, of aligning sequences X and Y using concave gap costs can be written thus:

{1} For i from 0 to M, set $R_{i,0}$ to w(i) and the set S_i to the empty set.
For j from 1 to N, set $R_{0,j}$ to w(j) and the set T_j to the empty set.

{2} For i from 1 to M and j from 1 to N:

Set P to min $[R_{i-1,j} + w(1), (R_{k,j} + w(i-k))_{k\epsilon T_j}]$.
If $P = R_{i-1,j} + w(1)$, add i-1 to the set T_j.
Set Q to min $[R_{i,j-1} + w(1), (R_{i,k} + w(j-k))_{k\epsilon S_i}]$.
If $Q = R_{i,j-1} + w(1)$, add j-1 to the set S_i.
Set $R_{i,j}$ to min $[P, Q, R_{i-1,j-1} + c(x_i,y_j)]$.

Waterman conjectures that $|S_i|$ grows no faster than log(j) and $|T_j|$ no faster than log(i). He concludes that the time complexity of the algorithm is effectively O(MN). An example is presented below in which $|S_i|$ grows linearly with j for $j \leq i$ and $|T_j|$ grows linearly with i for $i \leq j$. This alone renders the time complexity of the algorithm $O(M^2N)$; any assumption may be made about the behavior of $|S_i|$ as $j \to \infty$.

Let c(x,y) = 0 if x = y and 1 otherwise, and assume that $w(i+1) > w(i) > 0$ for all $i > 0$. Let X = Y = AAA...AA. During iteration (i,j) of step {2}, $P = R_{i-1,j} + w(1)$ if and only if $i \leq j+1$. Similarly,

$Q = R_{i,j-1} + w(1)$ if and only if $j \leq i+1$. After completion of iteration (i,j), $|T_j| = \min(i,j+1)$ and $|S_i| = \min(i+1,j)$. The minimization during iteration (i,j) thus requires examining more than $2\min(i,j)$ values. Therefore the algorithm has worst-case time complexity at least $O(M^2N)$.

Gap costs for which the marginal cost of a null is less than half the mean cost of aligning two elements do not seem appropriate. With such gap costs, the optimal alignment of two long sequences frequently consists of one long deletion and one long insertion. This observation motivates the following definition.

DEFINITION. Gap costs $w(x)$ are <u>reasonable</u> if $\lim_{x\to\infty} [w(x+1)-w(x)] \geq \bar{c}/2$ where $\bar{c}$ is the expected value of $c(x,y)$ for letters x and y chosen randomly with an appropriate distribution.

When two arbitrary sequences are aligned using reasonable concave gap costs it is usually the case that $Q = R_{i,j-1} + w(1)$ when $j \leq i+1$: on the average two nulls are replaced by a pair of aligned elements. Thus for reasonable gap costs the algorithm's average as well as its worst-case time complexity is at least $O(M^2N)$.

Waterman (1984b) suggests that his first algorithm can be improved as follows. Whenever $Q = R_{i,j-1} + w(1)$ in step {2}, for each $k \in S_i$ solve

$$R_{i,j-1} + w(1+h) = R_{i,k} + w(j-k+h) \qquad [3.1]$$

for h. If there is no solution or if $j+h > N$ then remove k from S_i. If $j+h \leq N$ then $R_{i,k}$ need not be considered when calculating $R_{i,x}$ until $x \geq j+h$, at which point $j-1$ may be removed from S_i. The analagous

calculations are done for T_j.

The worst-case time complexity of this algorithm depends strongly upon the function w(x) and is difficult to analyze. Yet consider sequence X consisting of M C's and sequence Y consisting of M A's followed by an arbitrary number of C's. For reasonable concave gap costs $|S_i| = 1$ while $j \leq M+1$. For $j > M+1$, however, it appears that $|S_i|$ can grow linearly with j-M at least until j reaches M+1+i. If $N \geq 2M$ this requires a total of $O(M^3)$ solutions of equation 3.1. The worst-case time complexity of the algorithm therefore appears to be $O[\max(M^3,MN)]$ for at least some concave gap costs.

For average sequences and reasonable gap costs, while $j \leq i$ it is expected not only that $Q = R_{i,j-1} + w(1)$ but also that equation 3.1 has no solution. Therefore $|S_i|$ should stay near 1 for $j \leq i$. It is not clear how $|S_i|$ grows on the average for $j > i$. The average time complexity of Waterman's second algorithm (1984b) thus remains an open question.

Conclusion. In this chapter it has been shown how to find in O(MN) time all and only the optimal alignments of two sequences using affine gap costs. Previously, Waterman et al. (1976) had shown how to find the optimal alignments in $O(MN^2)$ time, Gotoh (1982) how to find the optimal alignment cost in O(MN) time, and Taylor (1984) how to find at least one of the optimal alignments in O(MN) time.

Further, it has been shown that Waterman's algorithms (1984b) for concave gap costs have worst-case time complexity at least $O(M^3)$. It remains a challenge to find an O(MN) time algorithm for concave gap costs.

IV. SUBALIGNMENT SIMILARITY FUNCTIONS

Finding good subalignments of two sequences involves more than simply looking for subalignments with minimum cost. Although short perfect subalignments always exist, a long subalignment with greater cost may be much more interesting (less likely to occur by chance). To compare subalignments of different lengths a function of subalignment length and cost is required (Sellers, 1984). The length of a subalignment is defined as the average of the lengths of the two aligned segments (Sellers, 1984); the cost of a subalignment was defined in Chapter I.

DEFINITION. A similarity function $s(l,c)$ is a real-valued function that increases with length l and decreases with cost c.

A similarity function may be thought of as acting upon subalignments. Given a similarity function $s(l,c)$, an optimal subalignment of two sequences is a subalignment that maximizes $s(l,c)$. Since a similarity function is used to compare subalignments, its relative rather than its absolute value on (l,c) pairs is important.

DEFINITION. Two similarity functions s and s' are equivalent if for all (l_1,c_1) and (l_2,c_2) in their domains, $s(l_1,c_1) > s(l_2,c_2)$ iff $s'(l_1,c_1) > s'(l_2,c_2)$.

DEFINITION. A similarity function s is normalized if $s(l,0) = l$ for all positive lengths l.

Similarity functions s and s' are equivalent if and only if there exists a monotonically increasing function f such that $s'(l,c) = f(s(l,c))$.

Given a similarity function defined for all (1,0), it is always possible to find an equivalent similarity function s' that is normalized.

Smith and Waterman (1981a,b) suggested that maximizing the similarity measure of Needleman and Wunsch (1970) can be used to find interesting subalignments. Needleman-Wunsch similarity can be defined as a function on subalignments by the formula

$$s_{NW}(A) = i - Yj - Zk \qquad [4.1]$$

where subalignment A has i matches, j mismatches and k nulls. The arbitrary parameters Y and Z can be set to values that prove useful in distinguishing interesting subalignments from ones that seem insignificant. Smith et al. (1985) have suggested that useful values are Y = 0.9 and Z = 2.0. Sellers (1984) has proposed a similarity function for biological sequence comparison that is linear in l and c. The normalized Sellers similarity function is given by the formula

$$s_S(l,c) = l - c/T \qquad [4.2]$$

where the arbitrary parameter T determines the relative importance of length and cost in assessing subalignments. Hidden in the term c is another parameter, the null cost U. With appropriate choice of parameters, the Needleman-Wunsch and Sellers similarities are identical. Equations for transforming between the parameters of Needleman-Wunsch and Sellers similarity are:

s_{NW} to s_S: $T = 1/(Y + 1)$ $U = (Z + 0.5)/(Y + 1)$

s_S to s_{NW}: $Y = 1/T - 1$ $Z = U/T - 0.5$.

Due to this equivalence, statements about Sellers similarity apply equally to Needleman-Wunsch similarity and vice versa.

One undesirable property of similarity function s_S is that, for any choice of parameter T, subalignments with the same similarity value often have very different probabilities of occurring by chance. As a result, when optimal subalignments are found by maximizing s_S, a subalignment is often rejected in favor of another that is much more likely to occur at random. This problem can be avoided by choosing a similarity function that in general is nonlinear in the sense that $s(l,c) \neq k_1 l + k_2 c$. Nonlinear functions frequently have been used to evaluate subalignments once they have been found by algorithms employing linear similarity functions (Goad and Kanehisa, 1982; Reich et al., 1984).

The major advantage of s_S is that it is linear. This property is vital to fast algorithms for finding locally optimal subalignments, such as Sellers' TT algorithm (1984). Algorithmic considerations, however, should not determine the definition of a good subalignment.

Probabilistic similarity function s_1. Initially, we shall consider only diagonal subalignments. The key question is how to compare subalignments of different lengths. Given only the length and cost of two different subalignments, the most straightforward approach is to determine which is less likely to occur by chance.

Suppose that every letter has an a priori probability of appearing at any position in the first sequence X and another probability of appearing at any position in the second sequence Y. Specifically, these probabilities may vary from one letter to another and for the two sequences. For randomly chosen letters x from the distribution for sequence X and y from the distribution for sequence Y, let p_k be the probability that $c(x,y) = k$, where $c(x,y)$ is the cost of substituting x and y. Given letter probability distributions for sequences X and Y, and substitution cost function c, the p_k are easily computed. We shall assume that $p_0 \neq 0$, for otherwise the substitution cost function can be redefined so that this is the case.

Let $P(l,c)$ be the probability that a random diagonal subalignment of length l has cost c. $P(l,c)$ can be calculated recursively.

$$P(1,k) = p_k \qquad [4.3]$$

$$P(j,k) = \sum_{i=0}^{k} P(j-1,i)p_{k-i} \text{ for } j > 1 \qquad [4.4]$$

A normalized nonlinear similarity function s_1 can be defined as

$$s_1(l,c) = \log_{p_0}[\sum_{i=0}^{c} P(l,i)] \qquad [4.5]$$

In other words, $s_1(l,c)$ is the logarithm to the base p_0 of the probability that a random diagonal subalignment of length l has cost c or less.

For substitution cost function c_{id}, s_1 reduces to

$$s_1(l,c) = \log_p[\sum_{i=0}^{c} \binom{l}{i}(1-p)^i p^{l-i}] \qquad [4.6]$$

where p is the probability of a match. Notice that in this case $s_1(1,c)$ has the convenient property that for all lengths 1 the similarity of a run of 1 mismatches is 0 and the similarity of a run of 1 matches is 1. Therefore a nonintegeral similarity value can be immediately appreciated in relation to its integral bounds. For example, if nucleic acid sequences are being compared and p = 0.25, $s_1(50,18) = 13.6$ and $s_1(25,5) = 13.1$: a subalignment of length 50 with 18 mismatches (64% matches) is more interesting (less likely to occur by chance) than a subalignment of length 25 with 5 mismatches (80% matches). Both are more interesting than a run of 13 matches and less interesting than a run of 14 matches. Tables 4-1 and 4-2 list values of $s_1(1,c)$ from formula 4.6, with p = 0.25.

Interpolation of function s_1. Similarity function $s_1(1,c)$ is defined by formula 4.5 only for integral values of 1 and c. For use with many of the algorithms of this thesis, it must be extended to half-integral values of 1. If non-integral gap costs are used, it must also be extended to non-integral values of c. $s_1(1,c)$ can be extended to all non-negative values of 1 and c by linear interpolation in two dimensions.

$$s_1(l+x,c+y) = s_1(l,c)(1-x)(1-y) + s_1(l+1,c)x(1-y) + s_1(l,c+1)(1-x)y + s_1(l+1,c+1)xy \qquad [4.7]$$

for $0 < x,y < 1$ and integral l and c. Interpolation using formula 4.7 requires values of s_1 at four points.

Alternatively, a nonlinear interpolation of length for fixed cost also requires four values of s_1. Let $a = s_1(l-1,c)$, $b = s_1(l,c)$, $d = s_1(l+1,c)$ and $e = s_1(l+2,c)$. Then

Table 4-1. Values of $s_1(1,c) \geq 1$ from formula 4.6, with p = 0.25.

Length										
1	1.00									
2	2.00									
3	3.00	1.34								
4	4.00	2.15								
5	5.00	3.00	1.64							
6	6.00	3.88	2.37	1.28						
7	7.00	4.77	3.14	2.91	1.02					
8	8.00	5.68	3.94	2.60	1.57					
9	9.00	6.60	4.77	3.32	2.18	1.30				
10	10.00	7.52	5.62	4.08	2.83	1.84	1.08			
11	11.00	8.46	6.48	4.86	3.52	2.43	1.56			
12	12.00	9.40	7.35	5.66	4.24	3.07	2.10	1.33		
13	13.00	10.34	8.23	6.48	4.99	3.73	2.68	1.82	1.14	
14	14.00	11.29	9.12	7.31	5.76	4.43	3.30	2.35	1.58	
15	15.00	12.24	10.02	8.15	6.54	5.15	3.95	2.93	2.07	1.38
Cost:	0	1	2	3	4	5	6	7	8	9

Table 4-2. Selected values of $s_1(1,c) \geq 1$ from formula 4.6, with p = 0.25.

Length								
10	10.00							
20	20.00	3.09						
30	30.00	9.53	1.17					
40	40.00	17.16	5.39					
50	50.00	25.39	11.27	3.08				
60	60.00	33.96	18.08	7.60	1.72			
70	70.00	42.77	25.47	13.24	5.12			
80	80.00	51.75	33.26	19.62	9.77	3.40		
90	90.00	60.86	41.35	26.52	15.27	7.20	2.20	
100	100.00	70.06	49.66	33.82	21.39	11.92	5.25	1.37
Cost:	0	10	20	30	40	50	60	70

$$s_1(l+x,c) = b + [(a-3b+3d-e)(2x^2-5x+3)x^3 + (a-2b+d)x^2 + (d-a)x]/2 \quad [4.8]$$

for $0 < x < 1$ and integral l and c. This interpolation provides $s_1(l,c)$ with continuous first and second partial derivatives with respect to l. Usually, only specific values of x are needed. For example, if $x = 0.5$, formula 4.8 becomes

$$s_1(l+0.5,c) = (-a+9b+9d-e)/16 = (b+d)/2 + (b-a+d-e)/16 \quad [4.9]$$

in contrast to the linear interpolation (b+d)/2.

The analogous interpolation of cost can be done subsequently without the restriction to integral values of l. For $0 < x,y < 1$, and integral l and c, $s_1(l+x,c+y)$ is calculated by letting $a = s_1(l+x,c-1)$, $b = s_1(l+x,c)$, $d = s_1(l+x,c+1)$ and $e = s_1(l+x,c+2)$ in formula 4.8. Reversing the order of the two interpolations does not alter the result. The double interpolation requires values of s_1 at sixteen points.

The values of s_1, defined by formulas 4.3 to 4.5, cannot be calculated quickly for specific values of l and c. Therefore, when using most of the algorithms of this thesis, a table of s_1 values must be precomputed. The size of this table is approximately $L^2C/2$, where L is the length of the longest possible subalignment and C is the largest value of c(x,y). If storing this table requires more main computer memory than is available, s_1 values above a certain number can be kept in peripheral memory and retrieved only when needed.

Analytic function s_2. Another approach is use of a function that is nearly equivalent to s_1 but whose values are more easily calculated. Instead of being stored in a table, these values can be calculated as needed. One such function is

$$s_2(l,c) = (Tl - c - 0.5)/\sqrt{l} \qquad [4.10]$$

where T is the expected value of c(x,y) for randomly chosen x and y. The distribution of costs for random diagonal subalignments of length l is approximated by a normal distribution with a mean of Tl and a standard deviation proportional to the square root of l. For a subalignment having length l, cost c and no gaps, its s_2 value is proportional to the number of standard deviations its cost falls beneath its expected cost. The term 0.5, a correction for the discreteness of the cost distribution, is based on the assumption that for subalignments of length l all integral costs near Tl are possible. If only costs separated by some number C are attainable, the term 0.5 should be replaced by C/2.

Similarity function s_2', the equivalent normalized form of s_2, is obtained by algebraic manipulation.

$$s_2' = [(s_2)^2 + T + s_2\sqrt{(s_2)^2 + 2T}]/(2T^2) \qquad [4.11]$$

Substituting formula 4.10 into formula 4.11 yields

$$s_2'(l,c) = (x + \sqrt{x^2 - 1})/2T \text{ where } x = Tl-2c+(2c+1)^2/4Tl \qquad [4.12]$$

A computer program can use the quickly calculated values of s_2 internally but report the values of the equivalent normalized similarity function s_2'.

Other similarity functions. When gaps are allowed, and particularly when low gap costs are used, the justification for employing s_1 and s_2 is somewhat vitiated. The most fruitful approach may be to use a similarity function, similar to s_2, of the form

$$s_3(l,c) = [\mu(l) - c - 0.5]/\sigma(l) \qquad [4.13]$$

Functions $\mu(l)$ and $\sigma(l)$ are the mean and standard deviation of the distribution of minimum costs for aligning the first k with the first 21-k elements of two random sequences, where k ranges from 0 to 21. Reich et al. (1984) have experimentally determined $\mu(l)$ and $\sigma(l)$ for certain gap costs, but much work remains to be done in this area.

V. ALGORITHMS FOR FINDING LOCALLY OPTIMAL SUBALIGNMENTS

At present, an algorithm that searches a data bank for interesting subalignments must employ a linear similarity function in order to attain reasonable speed. But an algorithm that searches two specific sequences for interesting subalignments can use a nonlinear similarity function as the selection criterion. Previous algorithms that search for good subalignments have assumed that the similarity function is linear.

Gap costs. It is difficult to specify a priori what gap costs should be used for the comparison of biological sequences. In practice, gap costs have been chosen experimentally, by determining those that will yield alignments that are believed to be accurate from a biological perspective (Fitch and Smith, 1983). Therefore, during a preliminary analysis, it may be best to look only for interesting diagonal subalignments rather than to prejudice the results by choosing particular gap costs. The DD algorithm described below finds all weakly locally optimal diagonal subalignments using the criterion of any reasonable similarity function. During a subsequent analysis, however, it is desirable to be able to find the locally optimal subalignments of two sequences when using specific gap costs. This problem for nonlinear similarity functions is also addressed below. Table 5-1 summarizes the function of all the subroutines and algorithms described in this chapter.

Although all algorithms in this chapter can use a cost function $u(x)$ for aligning element type x with a null, for simplicity we assume that $u(x)$ has the constant value U for all x. Charging a cost V for the presence of

Table 5-1. Subroutines and Algorithms

Subroutine or Algorithm	Time Complexity	Subroutines Called	Function
SIM	$O(MN)$	---	Given edge similarities, finds edges belonging to locally optimal subalignments
LES	$O(MN)$	---	Calculates all edge similarities for linear similarity functions
TT-2	$O(MN)$	LES, SIM	Finds all and only those edges that are part of some locally optimal subalignment for linear similarity functions
NES-1	$O(M^3N)$	---	Calculates all edge similarities for nonlinear similarity functions
VV-1	$O(M^3N)$	NES-1, SIM	Finds all and only those edges that are part of some locally optimal subalignment for nonlinear similarity functions
NES-2	$O(M^2N)$	---	Calculates edge similarities of locally optimal subalignments for nonlinear similarity functions
VV-2	$O(M^2N)$	NES-2, SIM	Finds all edges that are part of some locally optimal subalignment for nonlinear similarity functions
RPF	$O(MN)$	---	Removes some edges that can not be part of any weakly locally optimal subalignment
APF	$O(MN)$	LES	Removes some edges that can not be part of any alignment with similarity $\geq s^*$
DD	$O(M^2N)$	---	Finds all and only those edges that are part of some weakly locally optimal subalignment lacking gaps for nonlinear similarity functions
CC-1	$O(M^3N^2)$[a]	APF, RPF, DD, NES-2	Finds all and only those edges that are part of some weakly locally optimal subalignment for nonlinear similarity functions
CC-2	$O(M^2N)$[a]	APF, RPF, DD, VV-2	Finds all edges that are part of some locally optimal subalignmnet for nonlinear similarity functions

a. For large s^*, the average time complexity is $O(MN)$.

each gap generates affine gap costs; certain algorithms have a more complicated form for affine gap costs than for linear gap costs.

When affine gap costs are used, in place of the linear path graph used in this chapter the more detailed affine path graph described in Chapter III is needed to represent accurately all and only the locally optimal subalignments of two sequences. Though they can be generalized to affine path graphs, the algorithms of this chapter employ only linear path graphs even when affine gap costs are used. Any ambiguity that results can be resolved by the SS-2 algorithm.

Locally optimal subalignments. Two subalignments intersect if their paths have a node in common. Given a similarity function, Sellers (1984) proposed the following definition of local optimality based on the concept of a path graph.

DEFINITION. A subalignment A_1 is locally optimal if no set $\{A_1, A_2, \ldots, A_k\}$ of subalignments with equal similarity exists such that A_i intersects A_{i+1} for all $i < k$ and A_k intersects a subalignment with greater similarity.

For linear similarity functions, the TT algorithm (Sellers, 1984) finds all and only the locally optimal subalignments of two sequences. It can not be used with nonlinear similarity functions.

DEFINITION. The similarity of an edge (node) of a path graph is the maximum similarity of all subalignments whose paths contain that edge (node).

The problem of finding locally optimal subalignments can be divided into two parts. The first part is finding the similarity of each edge of a path graph. The second part is finding each edge that is part of a locally optimal subalignment. The SIM subroutine addresses the second part, where the edge similarities may be defined by any similarity function, linear or nonlinear.

The SIM subroutine. Given the edge similarities of a path graph, all and only those edges that are part of a locally optimal subalignment with similarity $\geq s^*$ are found by performing steps {1} through {3}. Edge similarities are initially stored in arrays A_{ij}, B_{ij} and C_{ij}. Each expression involving an index outside the array bounds is ignored.

{1} NODE SIMILARITIES. Set the value of each node to the largest of the values of its adjacent edges. In other words, for all index pairs (i,j):

Set $R_{i,j}$ to max $(A_{i,j}, B_{i,j}, C_{i,j}, A_{i+1,j}, B_{i,j+1}, C_{i+1,j+1})$.

For all (i,j), if $R_{i,j} \neq \infty$ execute step {2}.

{2} RECURSIVE COMPARISON. If the value of an edge adjacent to the current node is less than the value of the node, but equal to the value of the other node it touches, set the value of the other node to and execute step {2} on the other node. In other words:

If $A_{i+1,j} < R_{i,j}$ and $A_{i+1,j} = R_{i+1,j}$, set $R_{i+1,j}$ to ∞ and execute step {2} on (i+1,j).

If $B_{i,j+1} < R_{i,j}$ and $B_{i,j+1} = R_{i,j+1}$, set $R_{i,j+1}$ to ∞ and execute step {2} on (i,j+1).

If $C_{i+1,j+1} < R_{i,j}$ and $C_{i+1,j+1} = R_{i+1,j+1}$, set $R_{i+1,j+1}$ to ∞ and

execute step {2} on (i+1,j+1).

If $A_{i,j} < R_{i,j}$ and $A_{i,j} = R_{i-1,j}$, set $R_{i-1,j}$ to ∞ and execute step {2} on (i-1,j).

If $B_{i,j} < R_{i,j}$ and $B_{i,j} = R_{i,j-1}$, set $R_{i,j-1}$ to ∞ and execute step {2} on (i,j-1).

If $C_{i,j} < R_{i,j}$ and $C_{i,j} = R_{i-1,j-1}$, set $R_{i-1,j-1}$ to ∞ and execute step {2} on (i-1,j-1).

{3} EDGE STATES. Set to 1 the state of all and only those edges that are part of a locally optimal subalignment with similarity $\geq s^*$. In other words, for all (i,j):

If $A_{i,j} = R_{i,j} \geq s^*$ set $a_{i,j}$ to 1, otherwise to 0.

If $B_{i,j} = R_{i,j} \geq s^*$ set $b_{i,j}$ to 1, otherwise to 0.

If $C_{i,j} = R_{i,j} \geq s^*$ set $c_{i,j}$ to 1, otherwise to 0.

Comments on the SIM subroutine. When step {2} is executed on a given node as a result of a call from within step {2}, the value of that node is set to ∞. As a result, step {2} can be executed at most twice on any given node and the time complexity of the SIM subroutine is O(MN). Implicit in the TT algorithm (Sellers, 1984) is a similar subroutine that requires an undetermined number of visits to each node. Its worst-case time complexity may therefore be greater than O(MN).

Each node that is not part of a locally optimal subalignment has its value set to ∞ before the SIM subroutine is complete. Therefore all and only those edges that are part of a locally optimal subalignment with similarity $\geq s^*$ have their states set to 1 in step {3}.

Sellers (1986) has noted that a locally optimal subalignment may be defined on any graph for which edge similarities are defined. He further observed that to find all and only the edges that belong to locally optimal subalignments it is sufficient to know whether or not the similarity of each edge is equal to the similarity of each of its two adjacent nodes. This information requires two bits of storage per edge. It is easy to modify the SIM subroutine to run on arbitrary finite graphs using two bits of storage per edge.

The LES subroutine. The LES subroutine calculates all edge similarities of a path graph using the linear similarity function $s_S(1,c)$ and affine gap costs $w(x) = V + Ux$. The linear edge similarities of a path graph are calculated and stored in arrays A_{ij}, B_{ij} and C_{ij} by performing steps {1} and {2}. Each expression involving an index outside the array bounds has the value $-\infty$. Let $u = 1/2 - U/T$.

{1} FORWARD SIMILARITIES. For all (i,j), calculate the greatest similarity of subalignments that end at node $N_{i,j}$ and use edge $V_{i,j}$, that end at $N_{i,j}$ and use edge $H_{i,j}$, and that end at $N_{i,j}$ without restriction. In other words, for i from 0 to M and j from 0 to N:

Set $P_{i,j}$ to $u + \max (R_{i-1,j} - V/T, P_{i-1,j})$.

Set $Q_{i,j}$ to $u + \max (R_{i,j-1} - V/T, Q_{i,j-1})$.

Set $R_{i,j}$ to $\max (0, P_{i,j}, Q_{i,j}, R_{i-1,j-1} + 1 - c(x_i,y_j)/T)$.

{2} REVERSE SIMILARITIES. For all (i,j), calculate the greatest similarity of subalignments that begin at node $N_{i,j}$ and use edge $V_{i+1,j}$, that begin at $N_{i,j}$ and use edge $H_{i,j+1}$, and that begin at $N_{i,j}$ without restriction. Also

calculate the similarities of edges $V_{i,j}$, $H_{i,j}$ and $D_{i,j}$. In other words, for i from M to 0 and j from N to 0:

Set $P_{i,j}$ to $u + \max(R_{i+1,j} - V/T, P_{i+1,j})$.

Set $Q_{i,j}$ to $u + \max(R_{i,j+1} - V/T, Q_{i,j+1})$.

Set $R_{i,j}$ to $\max(0, P_{i,j}, Q_{i,j}, R_{i+1,j+1} + 1 - c(x_{i+1},y_{j+1})/T)$.

Set $A_{i,j}$ to $\max(R_{i-1,j} + P_{i,j}, P_{i-1,j} + R_{i,j}, R_{i-1,j} + R_{i,j} - V/T, P_{i-1,j} + P_{i,j} + V/T) + u$.

Set $B_{i,j}$ to $\max(R_{i,j-1} + Q_{i,j}, Q_{i,j-1} + R_{i,j}, R_{i,j-1} + R_{i,j} - V/T, Q_{i,j-1} + Q_{i,j} + V/T) + u$.

Set $C_{i,j}$ to $R_{i-1,j-1} + R_{i,j} + 1 - c(x_i,y_j)/T$.

Comments on the LES subroutine. Since the LES subroutine performs a fixed number of operations for each node, its time complexity is O(MN). This subroutine is used in the APF subroutine described below. For linear gap costs, the LES subroutine is simpler:

{1} For i from 0 to M and j from 0 to N:

Set $R_{i,j}$ to $\max(0, R_{i-1,j} + u, R_{i,j-1} + u, R_{i-1,j-1} + 1 - c(x_i,y_j)/T)$.

{2} For i from M to 0 and j from N to 0:

Set $R_{i,j}$ to $\max(0, R_{i+1,j} + u, R_{i,j+1} + u, R_{i+1,j+1} + 1 - c(x_{i+1},y_{j+1})/T)$.

Set $A_{i,j}$ to $R_{i-1,j} + R_{i,j} + u$.

Set $B_{i,j}$ to $R_{i,j-1} + R_{i,j} + u$.

Set $C_{i,j}$ to $R_{i,j-1} + R_{i,j} + 1 - c(x_i,y_j)/T$.

The TT-2 algorithm. All and only those edges that are part of a locally optimal subalignment using similarity function s_S are found by performing the LES subroutine followed by the SIM subroutine.

Like its subroutines, this algorithm has time complexity O(MN). The TT-2 algorithm is equivalent to the TT algorithm (Sellers, 1984) but has the advantage that it can use affine gap costs.

The NES-1 subroutine. The NES-1 subroutine calculates all edge similarities of a path graph using any similarity function. This subroutine is presented here for linear gap costs but is easily generalized for affine gap costs. Although of theoretical interest, the NES-1 subroutine is impractical because it has time complexity $O(M^3N)$. Practical algorithms that employ nonlinear similarity functions are described in later sections.

The NES-1 subroutine uses a three-dimensional array R_{ijk}, where the locus index k ranges from 0 to M+N, to store M+N+1 values for each node of the path graph. All edge similarities of a path graph are calculated and stored in arrays A_{ij}, B_{ij} and C_{ij} by performing steps {1} and {2}. Each expression involving an index outside the array bounds has the value $-\infty$.

{1} MINIMUM COSTS TO LOCUS L_k. For each node $N_{i,j}$ find the minimum cost of a subalignment from that node to a node of locus L_k. In other words, for locus index k from 0 to M+N:

For all (i,j) of L_k, set $R_{i,j,k}$ to 0.

For all (i,j) of L_{k-1} and L_{k+1}, set $R_{i,j,k}$ to U.

For all (i,j) of L_{k-2} back to L_0, set $R_{i,j,k}$ to

$\min (R_{i+1,j,k} + U,\ R_{i,j+1,k} + U,\ R_{i+1,j+1,k} + c(x_{i+1},y_{j+1}))$.

For all (i,j) of L_{k+2} forward to L_{M+N}, set $R_{i,j,k}$ to

$\min (R_{i-1,j,k} + U,\ R_{i,j-1,k} + U,\ R_{i-1,j-1,k} + c(x_i,y_j))$.

{2} EDGE SIMILARITIES. Determine the similarities of all edges. In other words, for all (i,j) and locus indices k from $i+j$ to $M+N$, k' from 0 to $i+j-1$, and k'' from 0 to $i+j-2$:

Set $A_{i,j}$ to max $s[(k-k')/2,\ R_{i,j,k} + R_{i-1,j,k'} + U]$.

Set $B_{i,j}$ to max $s[(k-k')/2,\ R_{i,j,k} + R_{i,j-1,k'} + U]$.

Set $C_{i,j}$ to max $s[(k-k'')/2,\ R_{i,j,k} + R_{i-1,j-1,k''} + c(x_i,y_j)]$.

Comments on the NES-1 subroutine. As presented, the time complexity of the subroutine is $O(MN^3)$ because in step {2} $O(N^2)$ operations are performed for each of $O(MN)$ edges. The subroutine can be modified to reduce its time complexity to $O(M^3N)$. The key observation is that the maximum similarity subalignment that passes through a given edge can not begin or end at a node more than $O(M)$ loci away from the locus of the edge.

As presented, the space complexity of the NES-1 subroutine is $O(MN^2)$ due to the array R_{ijk}. The subroutine can be modified to reduce its space complexity to $O(MN)$. The key observation is that if the values of $A_{i,j}$, $B_{i,j}$ and $C_{i,j}$ are calculated one locus at a time, the relevant $R_{i,j,k}$ may be calculated using only $O(MN)$ space. This requires $O(N)$ repetitions of step {1}, which does not alter the time complexity of the subroutine.

The VV-1 algorithm. For any similarity function s, all and only those edges that are part of a locally optimal subalignment with similarity $\geq s^*$ are found by performing the NES-1 subroutine followed by the SIM subroutine.

Like the NES-1 subroutine, the VV-1 algorithm has time complexity $O(M^3N)$.

The NES-2 subroutine. For any similarity function, the similarities of all edges that are part of a locally optimal subalignment can be found in $O(MN^2)$ time by the NES-2 subroutine. The similarities of other edges may be underestimated. For linear gap costs (V = 0), the steps in brackets can be omitted and the algorithm can be rewritten to dispense with arrays P_{ij} and Q_{ij}. Edge similarities are output as arrays A_{ij}, B_{ij} and C_{ij}, which are initialized to 0. Expressions involving an index outside the array bounds are ignored.

For locus index k from 0 to M+N execute steps {1} to {4}.

{1} MINIMUM COST FROM LOCUS L_k. For each node on or below L_k, find the minimum cost of a subalignment ending at that node and starting at a node of locus L_k:

For all (i,j) of L_k, set $R_{i,j}$ to 0.

[For all (i,j) of L_k, set $P_{i,j}$ and $Q_{i,j}$ to $-\infty$.]

For all (i,j) of L_{k+1}, set $P_{i,j}$, $Q_{i,j}$ and $R_{i,j}$ to V + U.

For all (i,j) of L_{k+2} to L_{M+N}:

Set $P_{i,j}$ to $R_{i-1,j}$ + U + V.

[Set $P_{i,j}$ to min ($P_{i,j}$, $P_{i-1,j} + U$).]

Set $Q_{i,j}$ to $R_{i,j-1} + U + V$.

[Set $Q_{i,j}$ to min ($Q_{i,j}$, $Q_{i,j-1} + U$).]

Set $R_{i,j}$ to min ($P_{i,j}$, $Q_{i,j}$, $R_{i-1,j-1} + c(x_i,y_j)$).

{2} PATH GRAPH BELOW LOCUS L_k. For all (i,j) below L_{k+1}:

If $R_{i,j} = P_{i,j}$, set $a_{i,j}$ to 1, otherwise to 0.

If $R_{i,j} = Q_{i,j}$, set $b_{i,j}$ to 1, otherwise to 0.

If $R_{i,j} = R_{i-1,j-1} + c(x_i,y_j)$, set $c_{i,j}$ to 1, otherwise to 0.

[If $P_{i,j} = P_{i-1,j} + U$, set $d_{i,j}$ to 1, otherwise to 0.]

[If $P_{i,j} = R_{i-1,j} + V + U$, set $e_{i,j}$ to 1, otherwise to 0.]

[If $Q_{i,j} = Q_{i,j-1} + U$, set $f_{i,j}$ to 1, otherwise to 0.]

[If $Q_{i,j} = R_{i,j-1} + V + U$, set $g_{i,j}$ to 1, otherwise to 0.]

{3} MAXIMUM SIMILARITY FROM LOCUS L_k. For each node below L_k, find the maximum similarity of all subalignments starting at a node on locus L_k and passing through that node. In other words, for all (i,j) of L_{M+N} to L_{k+1}:

Set $R_{i,j}$ to $s[(i+j-k)/2, R_{i,j}]$.

If $c_{i+1,j+1} = 1$, set $R_{i,j}$ to max ($R_{i,j}$, $R_{i+1,j+1}$).

[Set K to $R_{i,j}$.]

If $a_{i+1,j} = 1$, set $R_{i,j}$ to max ($R_{i,j}$, $R_{i+1,j}$).

If $b_{i,j+1} = 1$, set $R_{i,j}$ to max ($R_{i,j}$, $R_{i,j+1}$).

Adjust the path graph into node $N_{i,j}$ (Chapter III):

[If ($R_{i,j} \neq K$) and ($R_{i,j} \neq R_{i+1,j}$ or $a_{i+1,j} = 0$ or $e_{i+1,j} = 0$) and ($R_{i,j} \neq R_{i,j+1}$ or $b_{i,j+1} = 0$ or $g_{i,j+1} = 0$), set $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ to 0.]

[If $R_{i,j} = R_{i+1,j}$ and $a_{i+1,j} = 1$ and $d_{i+1,j} = 1$, set $a_{i,j}$ to 1.]

[If $R_{i,j} = R_{i,j+1}$ and $b_{i,j+1} = 1$ and $f_{i,j+1} = 1$, set $b_{i,j}$ to 1.]

{4} EDGE SIMILARITIES BELOW LOCUS L_k. Store the similarity of all locally optimal subalignments beginning at a node of L_k. In other words, for (i,j) below L_k:

If $a_{i,j} = 1$, set $A_{i,j}$ to max $(A_{i,j}, R_{i,j})$.

If $b_{i,j} = 1$, set $B_{i,j}$ to max $(B_{i,j}, R_{i,j})$.

If $c_{i,j} = 1$, set $C_{i,j}$ to max $(C_{i,j}, R_{i,j})$.

Any locally optimal subalignment beginning on a node of loci L_0 through L_k now has its similarity stored in the value of each of its edges.

Comments on the NES-2 subroutine. As presented, the NES-2 subroutine has time complexity $O(MN^2)$ because steps {1} through {4} involve O(MN) operations and are executed for each of M+N loci. Because the subalignment with maximum similarity that passes through a given edge can not begin at a node more than O(M) loci above the locus of the edge, the time complexity of the subroutine can be reduced to $O(M^2N)$.

When the subroutine is finished, arrays A_{ij}, B_{ij} and C_{ij} store the similarity of each edge that is part of a locally optimal subalignment. Because the similarities of subalignments that are not locally optimal are not necessarily stored during step {4}, the values of other edges may underestimate their similarities.

The VV-2 algorithm. All edges that are part of a locally optimal subalignment with similarity $\geq s^*$ are found by performing the NES-2 subroutine followed by the SIM subroutine.

The edges of some subalignments that are not locally optimal may also be found. Except for the optimal subalignment, it is not possible to say without further investigation whether or not any found subalignment is locally optimal. Figure 5-1 illustrates the operation of the VV-2 algorithm.

Weak local optimality. A problem with finding only locally optimal subalignments is that good subalignments are sometimes suppressed even though they do not intersect a better subalignment that appears in the path graph. A very good subalignment A appears in the graph flanked by a large silent region containing no subalignments. Subalignment A suppresses the appearance of poorer intersecting subalignments as well as, indirectly, some flanking (but nonintersecting) subalignments. A subalignment C flanking the very good subalignment A can often be extended (at the cost of several mismatches or nulls) to be a better subalignment B that contains a part of A. Since B intersects and is better than C, C does not appear. Likewise, since A intersects and is better than B, B does not appear either. In essence, A suppresses the appearance of C through the intermediate subalignment B, even though C does not intersect A. This phenomenon suggests that a less stringent definition of local optimality than that proposed by Sellers (1984) may be useful.

DEFINITION. A subalignment is weakly locally optimal if it intersects no weakly locally optimal subalignment with greater similarity.

This definition is not circular but recursive. Any subalignment that is locally optimal is also weakly locally optimal. Finding all weakly

Figure 5-1. Stages in the execution of the VV-2 algorithm on the DNA sequences TGATG and CGACTGA using gap costs $w(x) = x$, similarity function s_1 with $p = 0.25$, and $s^* = 1$.

(a) Costs $R_{i,j}$ after initialization of L_2 and L_3 in step {1} of the NES-2 subroutine (locus counter $k = 2$).

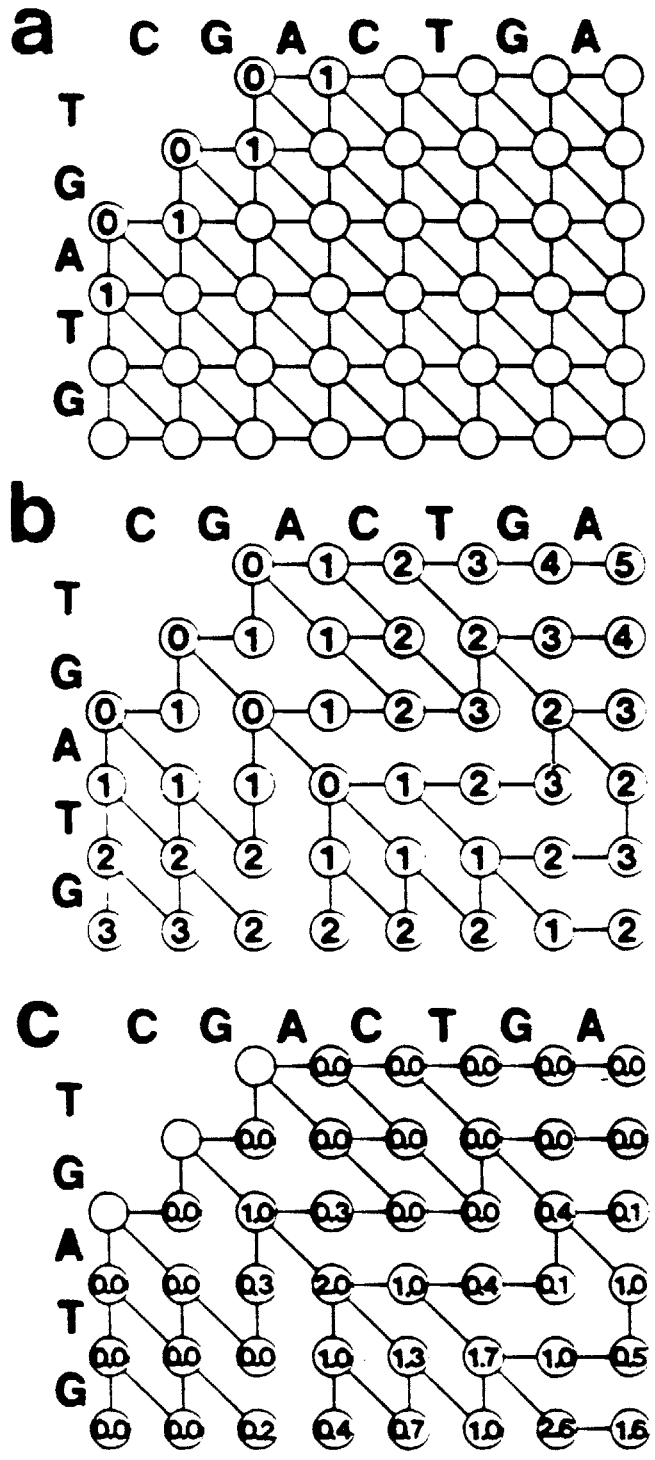
(b) Costs after step {2} of the NES-2 subroutine ($k = 2$); lines represent edge states of 1.

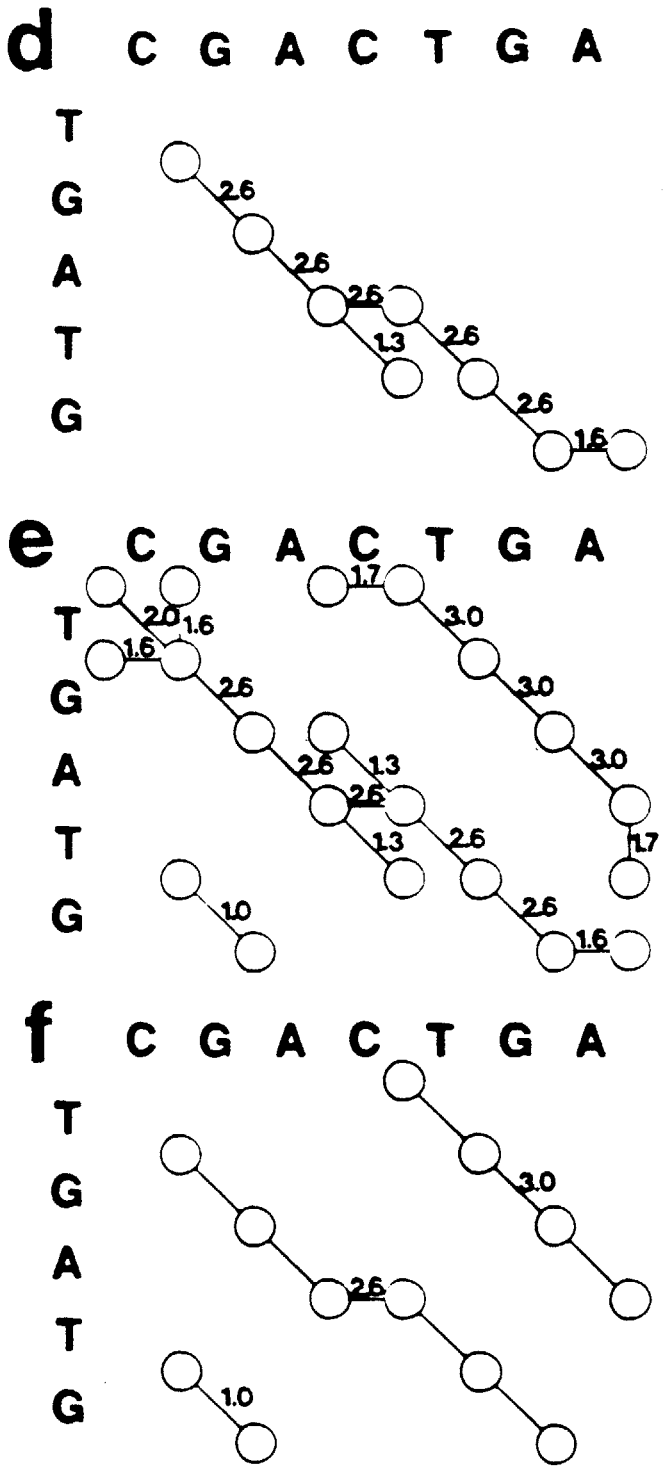
(c) Costs converted to similarities ($k = 2$).

(d) All edges whose value is set ≥ 1.0 during step {3} of the NES-2 subroutine ($k = 2$).

(e) All edges with value ≥ 1.0 after the NES-2 subroutine is complete.

(f) All edges with state 1 after the SIM subroutine.


a
C G A C T G A
T G A T G
b
C G A C T G A
T G A T G
C
C G A C T G A
T G A T G


d
C G A C T G A
T G A T G
2.6
2.6
2.6
1.3
2.6
2.6
1.6
e
C G A C T G A
T G A T G
2.0
1.6
1.6
2.6
2.6
1.3
2.6
1.3
2.6
1.7
3.0
3.0
3.0
1.7
2.6
1.6
1.0
f
C G A C T G A
T G A T G
3.0
2.6
1.0

locally optimal subalignments is usually more desirable than finding all locally optimal subalignments because interesting regions of similarity may appear that otherwise would have been suppressed.

The CC-1 algorithm described below finds all and only those edges that are part of some weakly locally optimal subalignment with similarity $\geq s^*$. The algorithm employs two filters that remove in O(MN) time most edges that can not be part of any such subalignment.

Isosimilarity curves and the feasible region. Occasionally it is convenient to represent subalignments by points in the l,c plane, which is the quarter plane whose positive x-axis represents length and positive y-axis represents cost. All subalignments that have the same length and cost are represented in the l,c plane by the same point. A similarity function that is defined for all non-negative l and c can be represented in the l,c plane by its isosimilarity curves. Equivalent similarity functions have the same isosimilarity curves; only the value each associates with a given curve is different.

When two specific sequences are compared, the sequence lengths impose an upper limit on the length of subalignments at any given cost. For example, with sequence lengths 40 and 60 and gap costs $w(x) = 4 + x$, the the maximum length for a subalignment with cost 10 is 43, corresponding to aligning subsequences of lengths 40 and 46 using a single gap of length 6. In this example, the maximum length of a subalignment independent of its cost is 50, corresponding to aligning the whole of both sequences. This constraint can be represented in the l,c plane by a right boundary

(Figure 5-2). All points representing subalignments must lie to the left of this boundary.

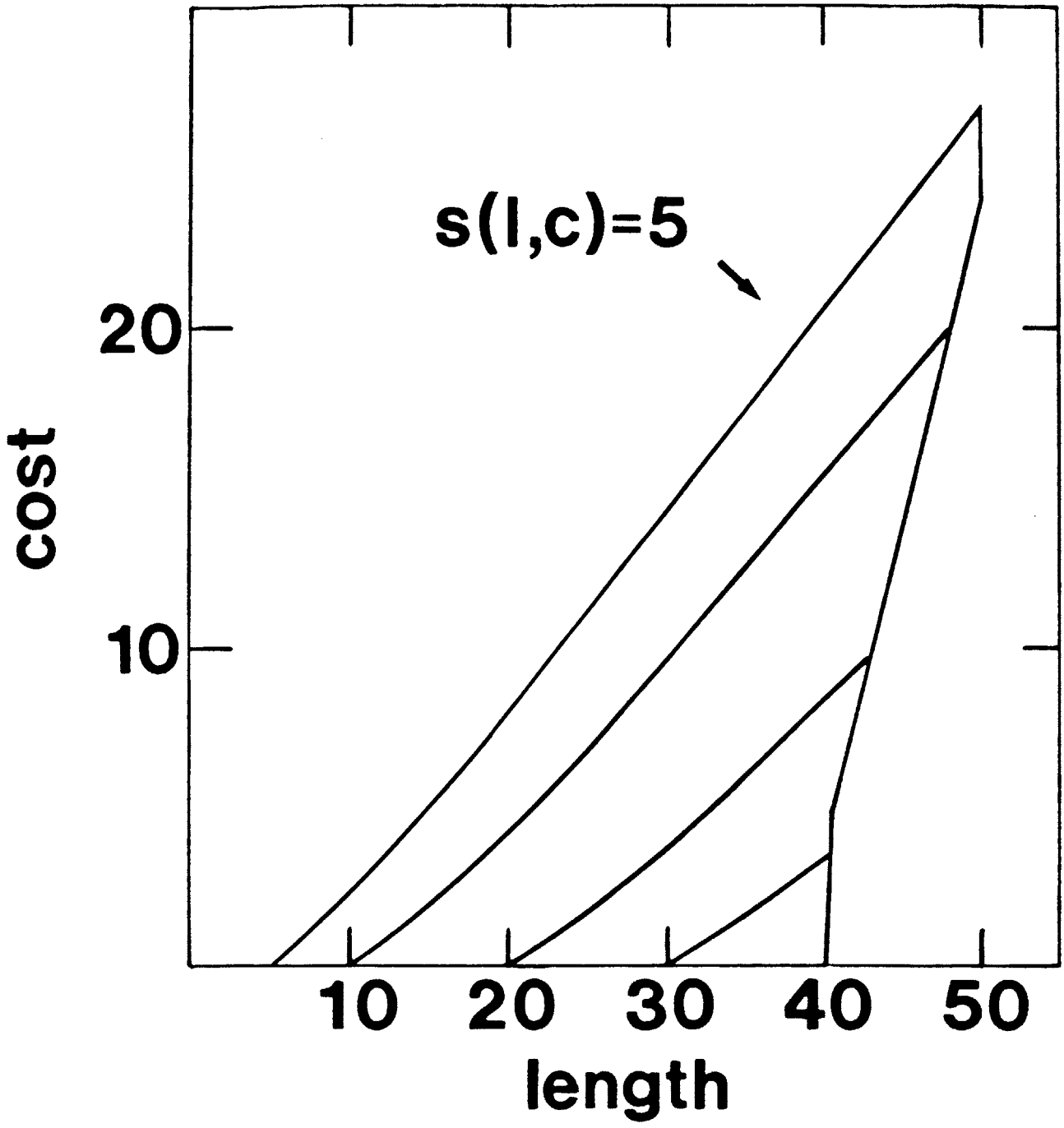
Typically one is interested only in subalignments whose similarity exceeds a cutoff value s^*. Points representing such subalignments must lie below the isosimilarity curve $s(l,c) = s^*$. This curve, along with the right boundary, confines subalignments of interest to a feasible region of the l,c plane (Figure 5-2). The apex of the feasible region is the point where the right boundary and the curve $s(l,c) = s^*$ intersect. Certain of the algorithms that follow require two assumptions about the slopes of isosimilarity curves in the feasible region. Both assumptions are true of all reasonable similarity functions and cutoff values s^*.

First, we assume that in the feasible region no isosimilarity curves are concave down. For $s_2(l,c)$ this is true when $s^* \geq 0$ because from formula 4.10 $d^2c/dl^2 = s/(4l^{3/2})$, which is non-negative for $s \geq s^*$. Since we are interested only in subalignments with similarity greater than that expected of a random subalignment, s^* will always be chosen greater than 0. Thus in the feasible region

$$\Delta c > T^* \Delta l \quad \text{implies} \quad s(l,c) > s(l+\Delta l, c+\Delta c) \qquad [5.1]$$

where T^* is the maximum slope of an isosimilarity curve in the feasible region. For reasonable $s(l,c)$ and s^*, the slope T^* always occurs at the apex. Thus it is easily estimated by determining the largest l in the feasible region, finding c_l and c_{l+1} such that $s(l,c_l) = s^*$ and $s(l+1,c_{l+1}) = s^*$, and taking T^* as $c_{l+1} - c_l$.

Figure 5-2. The feasible region for a comparison of sequences of lengths 40 and 60, with $w(x) = 4 + x$, using similarity function s_1 with $p = 0.25$ and $s^* = 5$. The function is defined for non-integral l and c by interpolation. Subalignments with lengths greater than 40 must contain nulls and can not have 0 cost. The right boundary of the feasible region reflects this fact. The upper boundary of the feasible region is the isosimilarity curve $s_1(l,c) = 5$. Isosimilarity curves are shown also for values of 10, 20 and 30. The maximum slope (T^*) of the isosimilarity curves in the feasible region is 0.66 and occurs at the apex. An alignment that begins with a match followed by two mismatches can not be self-optimal with $T^* = 0.66$.


s(l,c)=5
20
10
cost
10
20
30
40
50
length

Second, we assume that T* is less than 2U. This assumption merely corresponds to the intuition that adding a mismatch to a subalignment should decrease its similarity.

Any locally optimal subalignment A found using a nonlinear similarity function s whose isosimilarity curves are not concave down in the feasible region can also be found using the linear similarity function s_S (formula 4.2) if the parameter T is the slope of the isosimilarity curve of s at (l(A),c(A)). The reasons for not using a linear similarity function are that in general the appropriate T is different for different subalignments and that even for the optimal subalignment the appropriate T is not known a priori.

The DD algorithm. When affine gap costs are used, two parameters must be chosen to define the cost of a gap. It is difficult to determine what the values of these parameters should be and the choice made is generally somewhat arbitrary. The DD algorithm described below uses no parameters to describe the cost of a gap. Since it considers only diagonal subalignments, all gaps are assigned by visual inspection of the resulting DD graph. In practice, the strategy of not assigning gaps during the initial pattern search is not a serious limitation because the DD graph usually draws attention to a few obvious places where gaps should be assigned. Gaps can be assigned visually or the more costly algorithms for finding locally optimal subalignments containing gaps can be run on these regions of the path graph using a variety of gap costs. Thus, arbitrary decisions concerning gap costs do not prejudice the DD output.

A subalignment found by one of the Sellers algorithms (1974b; 1984) may contain any edge in the complete path graph (Figure 5-3a). A subalignment found by the DD algorithm, however, may contain only the diagonal edges of the diagonal graph (Figure 5-3b), which consists of diagonals that begin and end at the borders of the graph. Thus the strategy of allowing no insertions and deletions reduces the search for locally optimal subalignments in the complete path graph to a search in diagonals of the diagonal graph.

A diagonal subalignment can be represented by a string of numbers corresponding to the appropriate substitution costs. In the feasible region, the addition of the paired elements (x,y) to a subalignment can increase the subalignment similarity only if $c(x,y) < T^*$. A T-match run is a string of adjacent substitutions with identical costs $\leq T$, and an T-mismatch run is a string of substitutions each with cost $> T$. The cost of a T-match or T-mismatch run is the sum of the costs of its substitutions.

All weakly locally optimal diagonal subalignments whose similarity is at least s^* are found by performing steps {1} through {6}.

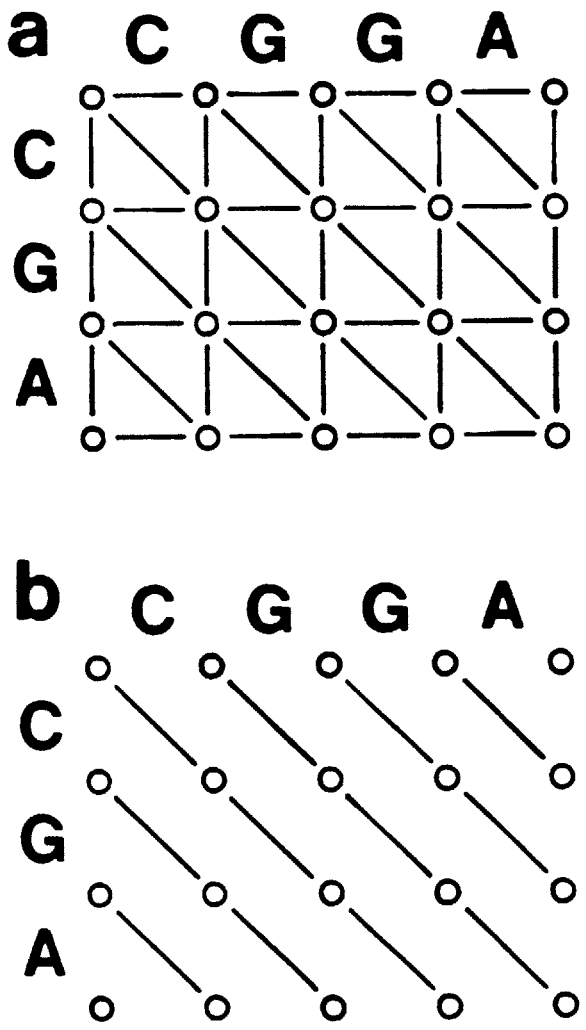
{1} Calculate the similarity s(1,c) for all values of $1 \leq M$ and all relevant values of c. Start at the first (lower left) diagonal of the graph.

{2} Find the beginning, length, and cost of each of the T^*-match runs and T^*-mismatch runs of the current diagonal.

Figure 5-3. Two path graphs for the sequences CGGA and CGA.

(a) The complete graph.

(b) The diagonal graph.



{3} Record the similarity of all diagonal subalignments that start at the beginning of a T*-match run and stop at the end of one.

{4} Find the diagonal subalignment with the highest similarity. If its similarity is not below s^*, mark this subalignment in the graph and execute step {5}. If its similarity is less than s^* and the current diagonal is not the last (upper right) diagonal move to the next diagonal and execute step {2}. Otherwise, execute step {6}.

{5} Repeat step {4} for the set of subalignments of the current diagonal that do not overlap any of the subalignments already marked in the graph.

{6} Print the DD graph containing just the marked subalignments.

Comments on the DD algorithm. Not all subalignments in the DD graph need be locally optimal. For example, consider the diagonal of Table 5-2. If s_1 is used with $p = 0.25$, and $s^* = 3$, both subalignments A and C would appear in the graph. Subalignment C is not locally optimal because it overlaps B, which has a higher similarity. Since A appears, B does not appear because it overlaps A. Thus C appears by step {5}. In practice, finding all the weakly locally optimal subalignments is more useful than finding only the locally optimal subalignments.

The length of the longest diagonal is M, the length of the shorter sequence. A diagonal of length M contains O(M) match runs and $O(M^2)$ subalignments to be examined. In practice, s^* is chosen so that on the average less than one subalignment is marked in the graph per diagonal. Since the number of diagonals is O(N), where N is the length of the longer

Table 5-2. Three substrings of the diagonal 10000111100011

Name	Substring	Similarity
A	0000	4.00
B	00001111000	3.52
C	000	3.00

sequence, the time complexity of the DD algorithm is approximately $O(M^2N)$.

This algorithm was implemented in the C programming language. The program calculated a DD graph of dimensions 500 x 800 in 5.5 processor minutes on a Digital Equipment Corp. VAX 11-780 minicomputer running under the Berkeley 4.2 version of the UNIX operating system.

Self-optimality. Generalizing an argument due to Sellers (1984), it can be shown that an alignment is not optimal in the context of its own subalignments if it can be separated into two subalignments such that one has $c/l > T^*$. This leads to a fast subroutine for eliminating edges that cannot be part of any weakly locally optimal subalignment.

DEFINITION. An alignment A is self-optimal if for every subalignment B of A $s[l(A), c(A)] \geq s[l(B), c(B)]$.

THEOREM. Suppose that the cost of increasing the length of a gap by 1 is always at least U. Then given a similarity function that satisfies [5.1] for some $T^* < 2U$, an alignment A with similarity $\geq s^*$ is not self-optimal if there exists a pair of subalignments B and C, produced by dividing A at some position, such that $c(B)/l(B) > T^*$ or $c(C)/l(C) > T^*$.

PROOF. Without loss of generality, assume that $c(C)/l(C) > T^*$, so that $c(C) > T^*l(C)$. If A is divided in the middle of a gap to make B and C, produce a different division of A into B' and C' where the whole of the gap is in C'. C' differs from C by the addition of k nulls. Thus

$$c(C')/l(C') \geq [c(C) + kU]/[l(C) + k/2] > [c(C) + T^*k/2]/[l(C) + k/2] > T^*$$

Therefore we may assume without loss of generality that A is not divided in the middle of a gap. Thus

$$c(A) = c(B) + c(C) > c(B) + T^*1(C) \qquad [5.2]$$

By formula 5.1,

$$s[1(B), c(B)] > s[1(B) + 1(C), c'] = s[1(A), c']$$

for all $c' > c(B) + T^*1(C)$. By formula 5.2, one such c' is $c(A)$, so that

$$s[1(B), c(B)] > s[1(A), c(A)]$$

which by definition means that A is not self-optimal.

A test for self-optimality. This theorem indicates that an alignment is not self-optimal if it contains a terminal subalignment for which $s_S = 1 - c/T^* < 0$.

DEFINITION. The terminal subalignment $\underline{S_i}$ consists of the first i positions of an alignment.

One can calculate $s_S(S_i)$ recursively from $s_S(S_{i-1})$ by adding $1 - c(x,y)$ if position i aligns elements x and y, adding $0.5 - V/T^* - U/T^*$ if it contains the first null of a gap, and adding $0.5 - U/T^*$ if it contains a subsequent null. For example, consider the case of substitution costs c_{id}, gap costs $w_k = 2k$ and $T^* = 2/3$. Shown below are the values of $s_S(S_i)$ for one alignment of X and Y.

i =	1	2	3	4	5	6	7	8
X =	A	T	C	C	T	-	C	T
Y =	A	T	G	G	T	G	C	T
$s_S(S_i)$ =	1.0	2.0	1.5	1.0	2.0	-0.5	0.5	1.5

Since the sixth score is negative, this alignment is not self-optimal. Note that if all scores from each end of an alignment are non-negative, the theorem is silent as to whether the alignment is self-optimal.

A subalignment that is not self-optimal contains a section with greater similarity. The existence of this section insures that the subalignment can not be weakly locally optimal. Therefore, if no subalignment that uses a given edge can be self-optimal, that edge can be removed from the path graph without destroying any weakly locally optimal subalignments. This strategy can also be used to speed up the DD algorithm by splitting long diagonals into several sections that can be considered separately.

A relative prefilter: the RPF subroutine. The states of some edges that can not be part of a weakly locally optimal subalignment are set to 0 by performing steps {1} to {3}. A path graph is initially stored in arrays a_{ij}, b_{ij} and c_{ij} by having the states of those edges present in the path graph set to 1. Any expression involving an index outside the array bounds has the value $-\infty$. For linear gap costs (V = 0), the steps in brackets can be omitted and the algorithm can be rewritten to dispense with arrays P_{ij} and Q_{ij}.

{1} INITIALIZATION. Calculate $T^* < 2U$, an upper bound on the maximum slope of isosimilarity curves in the feasible region.

Set X to $0.5 - V/T^* - U/T^*$.

[Set Y to $0.5 - U/T^*$.]

{2} FORWARD SELF-OPTIMALITY TEST. For i from 0 to M and j from 0 to N:

When $a_{i,j} = 1$,

set $P_{i,j}$ to $R_{i-1,j} + X$;

[set $P_{i,j}$ to max $(P_{i,j}, P_{i-1,j} + Y)$;]

if $P_{i,j} < 0$, set $a_{i,j}$ to 0.

When $b_{i,j} = 1$,

set $Q_{i,j}$ to $R_{i,j-1} + X$;

[set $Q_{i,j}$ to max $(Q_{i,j}, Q_{i,j-1} + Y)$;]

if $Q_{i,j} < 0$, set $b_{i,j}$ to 0.

When $c_{i,j} = 1$,

set $R_{i,j}$ to $R_{i-1,j-1} + 1 - c(x_i,y_j)/T^*$;

if $R_{i,j} < 0$, set $c_{i,j}$ to 0.

Set $R_{i,j}$ to max $(0, P_{i,j}, Q_{i,j}, R_{i,j})$.

If no edge state was changed during this step, and step {3} has been executed at least once, stop.

{3} REVERSE SELF-OPTIMALITY TEST. For i from M to 0 and j from N to 0:

When $a_{i+1,j} = 1$,

set $P_{i,j}$ to $R_{i+1,j} + X$;

[set $P_{i,j}$ to max $(P_{i,j}, P_{i+1,j} + Y)$;]

if $P_{i,j} < 0$, set $a_{i+1,j}$ to 0.

When $b_{i,j+1} = 1$,

set $Q_{i,j}$ to $R_{i,j+1} + X$;

[set $Q_{i,j}$ to max $(Q_{i,j}, Q_{i,j+1} + Y)$;]

if $Q_{i,j} < 0$, set $b_{i,j+1}$ to 0.

When $c_{i+1,j+1} = 1$,

set $R_{i,j}$ to $R_{i+1,j+1} + 1 - c(x_{i+1},y_{j+1})/T^*$;

if $R_{i,j} < 0$, set $c_{i+1,j+1}$ to 0.

Set $R_{i,j}$ to max $(0, P_{i,j}, Q_{i,j}, R_{i,j})$.

If some edge state was changed during this step, return to step {2}.

Comments on the RPF subroutine. Steps {2} and {3} each require O(MN) time. The RPF subroutine removes edges that can not be part of any weakly locally optimal subalignment, but in general it can not remove all such edges. Therefore, steps {2} and {3} may be executed until no more edges can be removed (as presented above), or they may be executed a small fixed number of times. Figure 5-4 illustrates the edges of the path graph that remain after the RPF subroutine has been executed on a specific pair of sequences.

An absolute prefilter: the APF subroutine. Since we seek weakly locally optimal subalignments with similarity $\geq s^*$, any edge whose similarity is $< s^*$ may be eliminated. Let s(1,c) be a normalized similarity function. Then $(s^*,0)$ is the point where the isosimilarity curve $s(1,c) = s^*$ intersects the x-axis (Figure 5-5a). Let A be apex of the feasible region. Let $s_S(1,c) = 1 - c/T$, where T is the slope of the line L connecting $(s^*,0)$ and A. The s_S-similarity of any point on L is s^*.

Figure 5-4. Operation of the RPF subroutine on the complete path graph for comparison of segments 188-263 and 488-568 of the human DNA sequence for beta-1 interferon, using gap costs $w(x) = 0.5 + 1.5(x)$ and similarity function s_1 with $p = 0.25$.

(a) The edges of the path graph that remain after filtering with $s^* = 5$.

(b) The edges of the path graph that remain after filtering with $s^* = 7$.

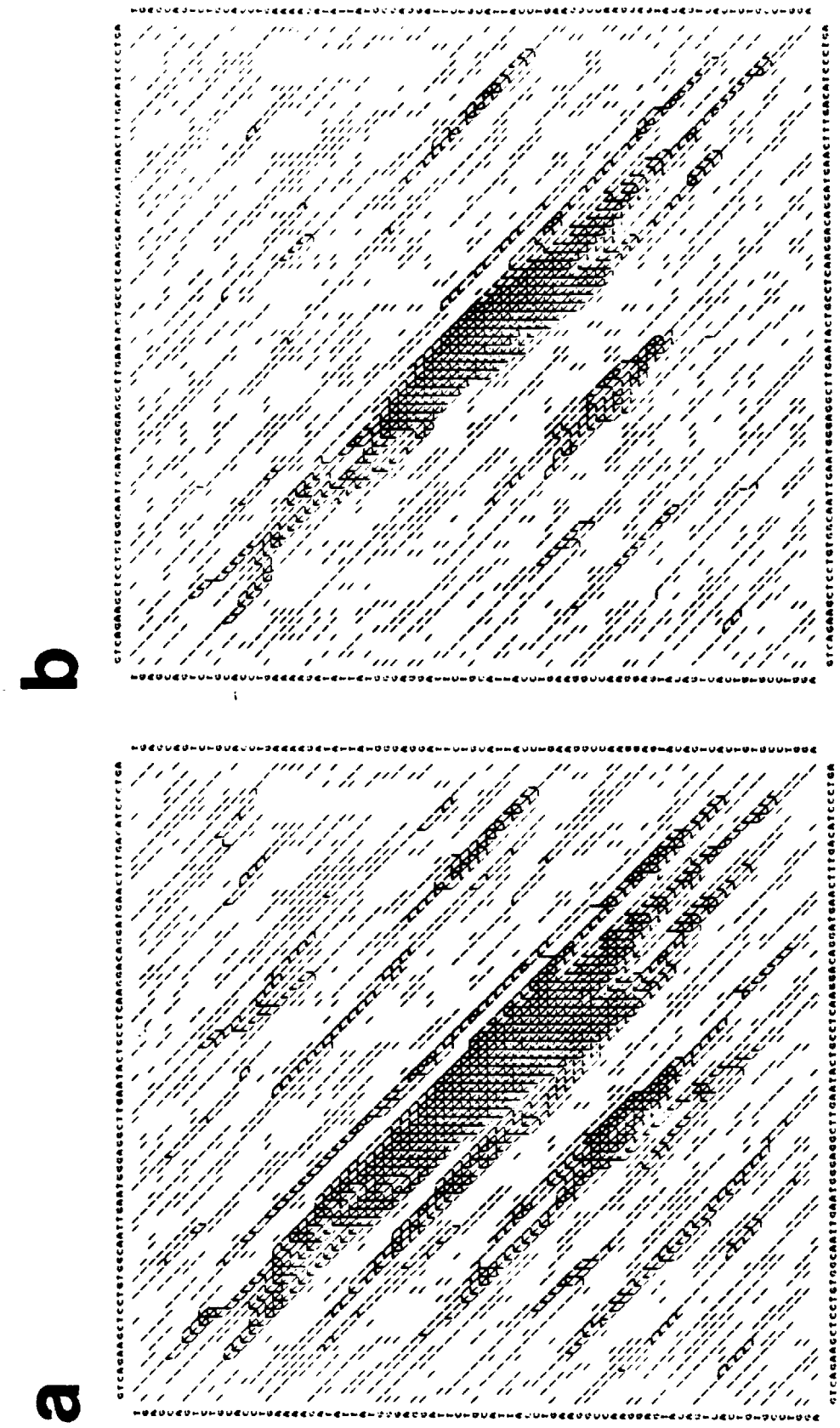
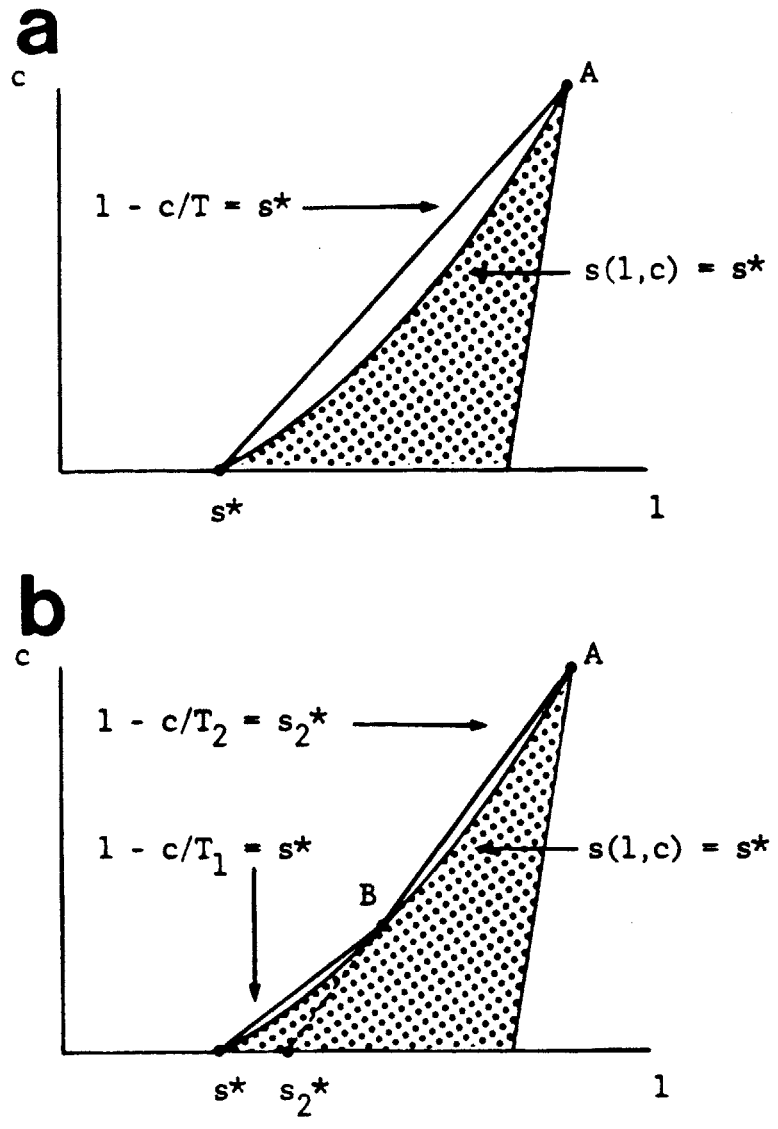

a
b

Figure 5-5. Isosimilarity lines for linear similarity functions that approximate the upper boundary of the shaded feasible region.

(a) Points (1,c) above the line $1 - c/T = s^*$ can not be in the feasible region.

(b) Points (1,c) above both lines $1 - c/T_1 = s^*$ and $1 - c/T_2 = s_2^*$ can not be in the feasible region.



Since the s-isosimilarity curves are never concave down in the feasible region, this region lies below L in the 1,c plane. Any subalignment in the feasible region therefore has s_S-similarity $\geq s^*$.

The states of some edges whose s-similarity must be $< s^*$ are set to 0 by performing steps {1} and {2}:

{1} Determine the apex A of the feasible region. Let $s_S(1,c) = 1 - c/T$, where T is the slope of the line through $(s^*,0)$ and A.

{2} Execute the LES subroutine using s_S and remove all edges whose s_S-similarity is less than s^*.

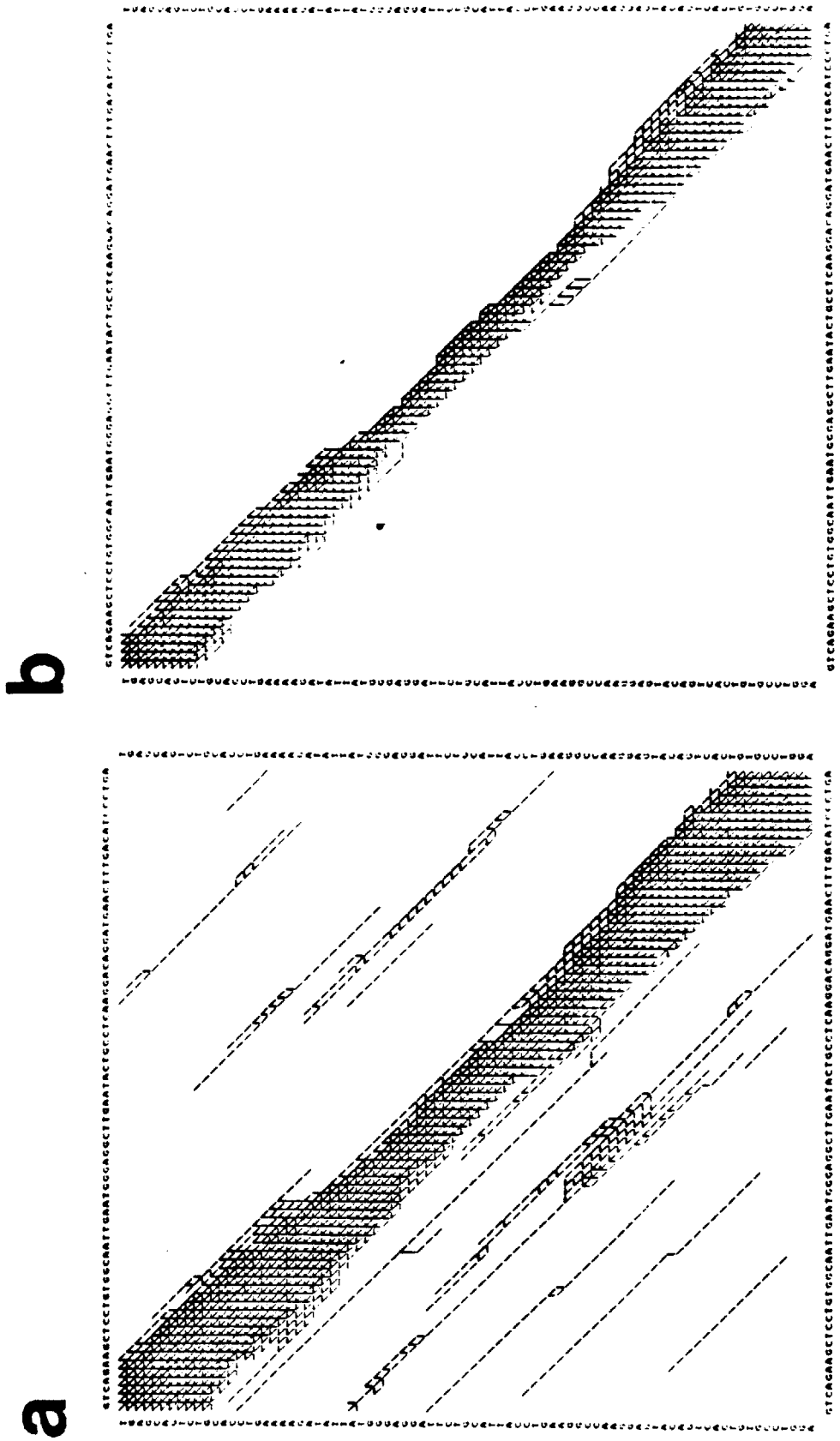
Comments on the APF subroutine. Like the LES subroutine, the time complexity of the APF subroutine is O(MN). Figure 5-6 illustrates the edges of the path graph that remain after the APF subroutine has been executed on a specific pair of sequences. The APF subroutine may be executed on partial or complete path graphs.

It is possible the extend the APF subroutine so that it will remove more edges. Let B be some point between $(s^*,0)$ and A on the curve $s(1,c) = s^*$ (Figure 5-5b). Let T_1 be the slope of the line through $(s^*,0)$ and B, let T_2 is the slope of the line through A and B, and let $(s_2^*,0)$ be the 1-intercept of this line. Then if $1 - c/T_1 < s^*$ and $1 - c/T_2 < s_2^*$, (1,c) can not be in the feasible region. This fact suggests an obvious extension of the APF subroutine. The usefulness of this extension depends on the degree to which the isosimilarity curves of s(1,c) are concave up.

Figure 5-6. Operation of the APF subroutine on the complete path graph for comparison of segments 188-263 and 488-568 of the human DNA sequence for beta-1 interferon, using gap costs $w(x) = 0.5 + 1.5(x)$ and similarity function s_1 with $p = 0.25$.

(a) The edges of the path graph that remain after filtering with $s^* = 5$.

(b) The edges of the path graph that remain after filtering with $s^* = 7$.


b
a

The CC pattern recognition algorithms. The VV-2 algorithm runs too slowly to be practical for even relatively small sequence comparisons, such as 100 x 300. The APF and RPF subroutines described above can remove from the complete path graph many edges that can not be part of any weakly locally optimal subalignment. While these prefilters may be executed in either order, we have found that a program that executes the APF subroutine first runs slightly faster. The path graph that remains consists of disjoint connected components (Figure 5-7). An algorithm that executes the DD algorithm and the NES-2 subroutine on these connected components can run much faster than the VV-2 algorithm, which executes the NES-2 subroutine on the complete path graph.

The CC-1 algorithm. All and only those edges that are part of a weakly locally optimal subalignment with similarity $\geq s^*$ are found by performing steps {1} through {4}.

{1} Execute the APF subroutine on the complete path graph.

{2} Execute the RPF subroutine on the resulting path graph.

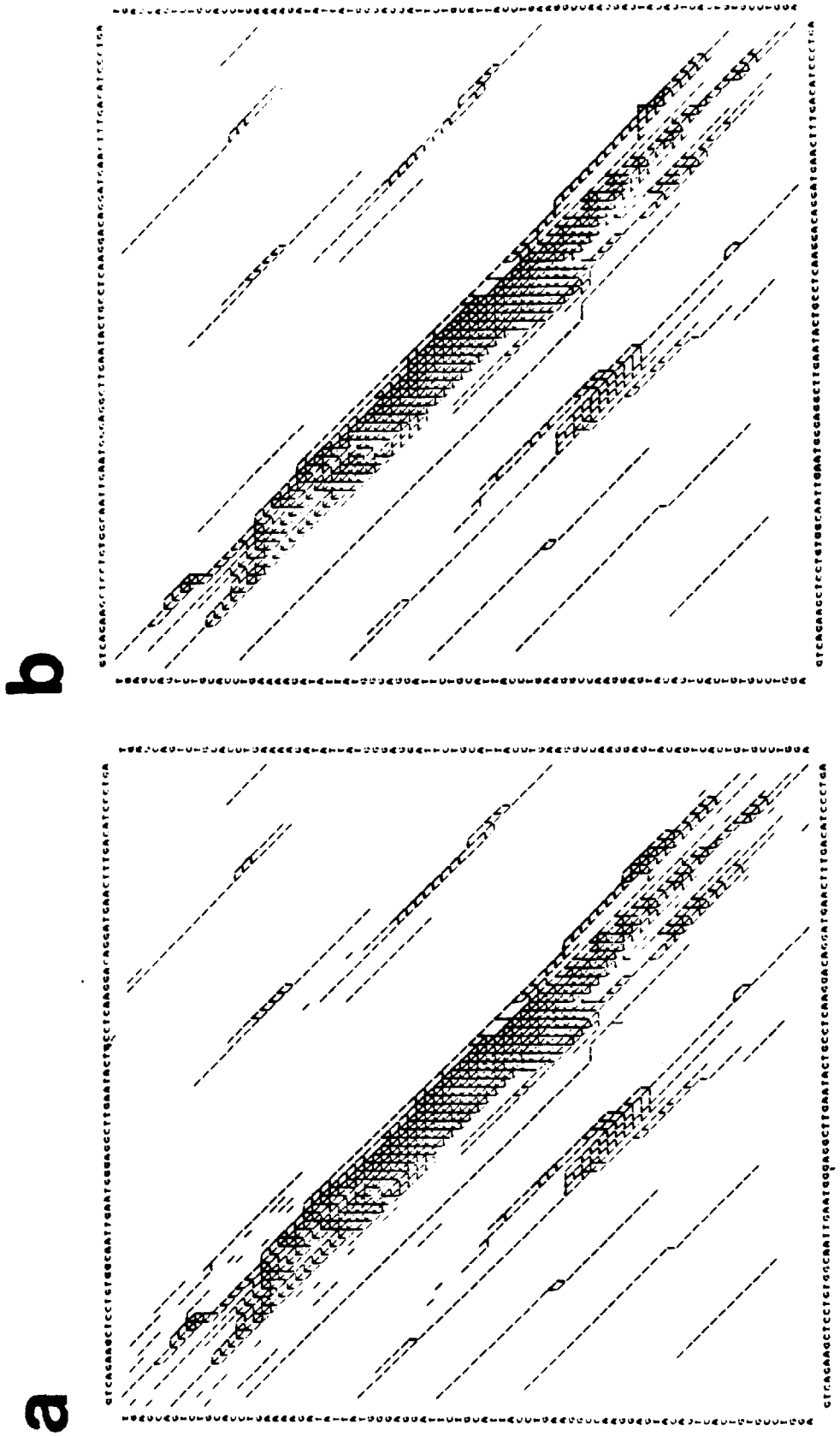
{3} Execute the DD algorithm on each connected component whose nodes all lie on the same diagonal. Save all edges found by the DD algorithm.

{4} Execute the NES-2 subroutine on each connected component whose nodes do not all lie on the same diagonal. If the greatest edge similarity found is less than s^*, save no edges of the connected component. Otherwise, save only the edges with the greatest similarity and execute the CC-1 algorithm on the connected component minus these saved edges.

Figure 5-7. Connected components of the path graph for comparison of segments 188-263 and 488-568 of the human DNA sequence for beta-1 interferon, using gap costs $w(x) = 0.5 + 1.5(x)$ and similarity function s_1 with $p = 0.25$, after filtering with the RPF and APF subroutines.

(a) APF followed by RPF with $s^* = 5$.

(b) RPF followed by APF with $s^* = 5$.


b
a

Comments on the CC-1 algorithm. The recursion of step {4} makes the worst-case time complexity of the CC-1 algorithm $O(M^3N^2)$. In practice, for reasonably large s*, each connected component contains only a small number of weakly locally optimal subalignments. For very large s*, steps {1} and {2} remove almost all edges of the path graph and the average time complexity approaches O(MN). Since the CC-1 algorithm can be very slow for small s*, it is recommended only as a secondary tool for sequence comparison. A faster algorithm, such as DD, can be used as a primary tool to find subalignments with high similarity. The CC-1 algorithm can then be executed on a region of the path graph containing these subalignments, using a large s*.

The CC-2 algorithm. All edges of locally optimal subalignments with similarity $\geq$ s* plus the edges of some subalignments with similarity $\geq$ s* that are not locally optimal are found by performing steps {1} to {3} of the CC-1 algorithm and the following non-recursive step:

{4a} Execute the VV-2 algorithm on each connected component whose nodes do not all lie on the same diagonal. Save all edges of the path graph found by the VV-2 algorithm.

The non-recursive CC-2 algorithm runs faster than CC-1. Like the VV-2 algorithm, its worst-case time complexity $O(M^2N)$. For reasonably large s*, it runs much faster than VV-2.

Implementation of the CC-1 algorithm. The CC-1 algorithm has been implemented in the C programming language. The use of this program is illustrated in the context of comparing a 636-nucleotide DNA segment

encoding all of human beta-1 interferon with itself, a problem previously studied with the TT algorithm (Erickson et al., 1984). Using similarity s_1 of formulas 4.6 and 4.7 as the selection criterion, with $w(x) = 0.5 + 1.5(x)$, a 76 and an 81-nucleotide segment from the human gene for beta-1 interferon were compared using the CC-1 program. Running under version 4.2BSD of the UNIX operating system on a Digital Equipment Corp. VAX 11/780 minicomputer, the the CC program required 23 seconds with s^* set to 5 and only 10 seconds with s^* set to 10. Figure 5-8 shows the final path graph for $s^* = 5$. The two long, nearly identical central paths (solid arrows) are equivalent to the DNA subalignments of Figure 5-9. These globally optimal subalignments have greater s_1-similarity than the best subalignment noted in a previous analysis (Erickson et al., 1984).

Figure 5-8. The path graph produced by the CC-1 algorithm for comparison of segments 188-263 and 488-568 of the human DNA sequence for beta-1 interferon, using gap costs $w(x) = 0.5 + 1.5(x)$, similarity function s_1 with $p = 0.25$, and $s^* = 5$.

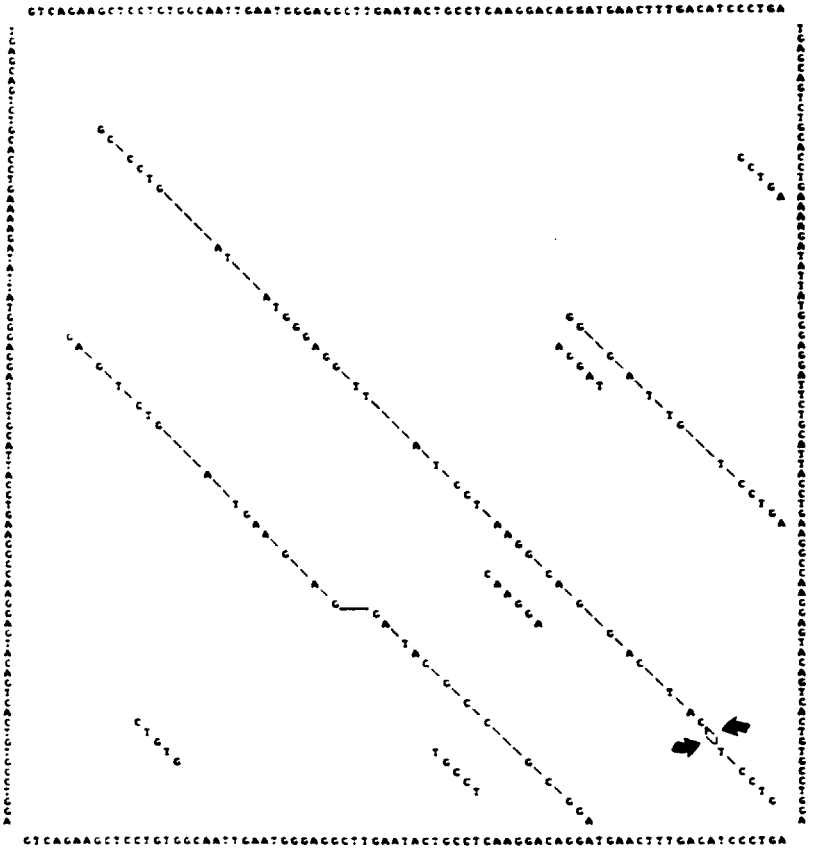


Figure 5-9. The two globally optimal subalignments found by the CC-1 algorithm for comparison of a 636-nucleotide DNA sequence (encoding human beta-1 interferon) with itself. One subalignment is shown explicitly and the other differs only in having A-256 aligned with T-559 instead of G-560. These two best alignments of DNA segments 195-262 and 498-566 correspond to the nearly identical pair of locally optimal paths (solid arrows) present in the path graph of Fig. 5-8. The DNA subalignment shown is annotated (Erickson and Sellers, 1983; Erickson et al., 1984) to display the amino acid residues of the beta-1 interferon protein encoded by these DNA segments and the five common amino acids. Since both alignments contain a null near the end of the first DNA segment, the amino acids encoded by the last several alignment positions are out of phase and thus can not be common amino acids. Both alignments contain 29 matches, 27 mismatches and 1 null (V+U = 2), so the cost of each is 29. By formula 4.6 with p = 0.25, the similarity of this pair of subalignments is $s_1(68.5,29) = 13.3$. Deleting the last eight alignment positions from the alignment shown here gives a subalignment with lower similarity ($s_1(61,25) = 12.9$), which is the best subalignment noted previously (Erickson et al., 1984). In the context of comparing a 636-nucleotide segment with itself, a value of $s_1 \geq 12.9$ for a subalignment lacking gaps is significant at the 99% confidence level (Chapter VI).

```
                          |    20              25              30              35              40 41
beta-1 interferon (20-41) |  sLeuLeuTrpGlnLeuAsnGlyArgLeuGluTyrCysLeuLysAspArgMetAsnPheAsp IleProG
different local alignments:  ----------------------------------------------------------2:------
                          | 195  200        210        220        230        240        250        262
beta-1 DNA segment (195-262)|  GCTCCTGTGGCAATTGAATGGGAGGCTTGAATACTGCCTCAAGGACAGGATGAACTTTGAC-ATCCCTG
          common nucleotides:  GC CCTG      AT    ATGGGAGG TT     A T CCT AAGG CA G   G AC   T AC   T CCTG
beta-1 DNA segment (498-566)|  GCACCTGAAAAGATATTATGGGAGGATTCTGCATTACCTGAAGGCCAAGGAGTACAGTCACTGTGCCTG
                          | 498         510        520        530        540        550        560   566
                 codon phase:  <--- in phase ------------------------------------------------> < out >
beta-1 interferon (121-142)|  uHisLeuLysArgTyrTyrGlyArgIleLeuHisTyrLeuLysAlaLysGluTyrSerHisCysAlaTr
                          |   121         125             130             135             140   142
          common amino acids:     Leu             GlyArg          LeuLys
```

VI. SIMILARITY SIGNIFICANCE LEVELS

It is important to know how great the similarity of a subalignment must be before the subalignment can be considered surprising. Specifically, given a number $\varepsilon \lesssim 0.1$ and two random sequences of lengths M and N, we wish to estimate the value s_ε for which the probability is less than ε that the greatest similarity of a subalignment of these sequences will be greater than s_ε. For various similarity functions, the distribution of maximal scores from comparison of random sequences has been studied recently by both theoretical derivation (Karlin and Ghandour, 1985; Arratia and Waterman, 1985; Gordon et al., 1986; Arratia et al., 1986) and experimental simulation (Smith et al., 1985).

For a wide range of similarity functions, the distribution of maximal scores is well modeled by the extreme value distibution (Gumbel, 1962), whose cumulative distribution function is

$$\exp(-\exp(-\lambda(x-u))) \qquad [6.1]$$

Parameter u, called the characteristic value, measures the center of the distribution. Parameter λ measures the exponential rate of decay of the right-hand tail of the distribution.

The extreme value distribution is the limiting ditribution for the maximum of a large number of i.i.d. (independent identically distributed) continuous unbounded random variables with at least exponential decay (Gumbel, 1962). The similarity of the optimal subalignment of two random sequences does not satisfy these conditions, but there is sufficient

analogy that one may expect an extreme value distribution to apply. A subalignment can begin with any pair of elements from the two sequences. The greatest similarity of a subalignment beginning at a given position is dependent on that of subalignments beginning at other positions. Furthermore, since the two sequences are finite, the distribution of such greatest similarities is position dependent. Nevertheless, the dependence between the maximum similarity of subalignments starting at different positions is in general very weak, and the position dependence is important only for those subalignments that begin near the end of one sequence.

Gordon et al. (1986) have shown that for a sequence of n independent coin tosses, the longest head run interrupted by k tails is closely approximated by an extreme value distribution. Parameter λ remains constant with increasing n and parameter u grows approximately as a linear function of ln(n). When two random sequences are compared using a normalized similarity function such as s_1 or s_2', the similarity of an optimal subalignment is analogous to the longest head run studied by Gordon et al., because a subalignment of length 1 and cost 0 has similarity 1. The product of the lengths of the two sequences (MN) is analagous to the number of tosses, because a subalignment may begin at any of MN different positions. While these are no more than analogies, they lead us to expect that for pairs of sufficiently long sequences the distribution of optimal similarities can be well approximated by an extreme value distribution. For normalized similarity functions, the parameter λ should not change with MN and the parameter u should be well approximated by the formula

$$u = \ln(k_1 MN)/k_2 \qquad [6.2]$$

For $\varepsilon \leq 0.1$, the significance levels of the extreme value distribution of formula 6.1 are then closely approximated by the formula

$$s_\varepsilon = u - \ln(\varepsilon)/\lambda \qquad [6.3]$$

Range of the analysis. While for certain similarity functions an extreme value distribution has been shown to apply, and formulas for u and λ have been derived theoretically (Arratia and Waterman, 1985; Gordon et al., 1986; Arratia et al., 1986), theory for similarity functions such as s_1 remains undeveloped. Assuming an extreme value distribution, parameters u and λ must be estimated experimentally. In this chapter we investigate the distibution of optimal similarities for the function s_1 of formula 4.5 for both nucleic acid and protein sequence comparison. As mentioned in Chapter IV, the use of similarity function s_1 is justified for only diagonal subalignments, and we accordingly investigate only such subalignments.

For nucleic acid sequences, we allow a variety of probability distributions for nucleotide usage, but investigate only the substitution cost function c_{id} and its associated s_1-similarity function. For protein sequences, we use only one distribution for amino acid usage, but investigate a variety of substitution cost functions and their associated s_1-similarity functions. We also investigate the distribution of optimal similarities for three-sequence comparison.

While our results are confined to a specific similarity function and to diagonal subalignments, it is evident how they can be extended to more

general cases. Since the methods described require substantial computer time, it is hoped that analytic methods can be developed to render Monte Carlo simulation unnecessary.

Estimation of parameters. In this section we describe in detail how parameters u and λ are estimated for the simple case of nucleic acid sequences with uniform nucleotide usage, substitution costs c_{id}, and the associated s_1-similarity of formula 4.6 with p = 0.25. Later this method is applied to a variety of other cases.

For nine sequence lengths M = N between 70 and 518, a series of 1000 pairs of random sequences were generated. Nucleic acid sequences were simulated by sequences having four types of elements selected randomly with equal probabilities. For each pair of sequences, the DD algorithm was used to find the maximum s_1-similarity of all diagonal subalignments. From these 1000 similarity values, the parameters λ and u were estimated by the method of moments using the formulas $\lambda = \pi/\sqrt{6V}$ and $u = \mu - \gamma/\lambda$, where $\gamma = 0.577...$ is Euler's constant, μ is the sample mean, and V is the sample variance (Gumbel, 1962).

The results are summarized in Table 6-1. In order to test the claim that the distribution of optimal similarity scores can reasonably be modelled by an extreme value distribution, we did a χ^2 goodness-of-fit test on the data. A typical result, that for M = N = 245, is shown in Table 6-2. In order to compensate for the discreteness of the data, a similarity score of the form i + x, where i is an integer and $0 \leq x < 1$, counts as the fraction (1 - x) of an observation at i and the fraction x of

Table 6-1. Estimation of the parameters u and λ for M = N

M	N	ln(MN)	u	λ
70	70	8.50	5.98 ± 0.03	1.36 ± 0.04
90	90	9.00	6.42 ± 0.03	1.38 ± 0.04
116	116	9.51	6.89 ± 0.03	1.34 ± 0.04
148	148	9.99	7.33 ± 0.04	1.29 ± 0.05
191	191	10.50	7.79 ± 0.04	1.27 ± 0.04
245	245	11.00	8.23 ± 0.04	1.31 ± 0.05
314	314	11.50	8.64 ± 0.04	1.27 ± 0.04
403	403	12.00	9.13 ± 0.04	1.28 ± 0.05
518	518	12.50	9.44 ± 0.03	1.32 ± 0.05

Table 6-2. χ^2 goodness-of-fit test for 1000 optimal s_1-scores for M = N = 245.
Estimated u: 8.23 Estimated λ: 1.31

i	X_i	O_i	$(X_i-O_i)^2/X_i$
7-	74.12	90.00	3.40
8	421.43	405.48	0.60
9	331.87	321.74	0.31
10	122.75	125.84	0.08
11	36.13	42.28	1.05
12+	13.70	14.66	0.07
		Total:	5.51

an obsevation at (i + 1). Let X_i be the number of scores expected at i, and O_i the number observed. Then since two parameters for the extreme value distribution of formula 6.1 are estimated from the data, the statistic

$$\sum_{i=7}^{12} [(X_i - O_i)^2/X_i] \qquad [6.4]$$

is distributed as χ^2 with 3 degrees of freedom (Larsen and Marx, 1981, pp 365-366). 90% of the distribution for χ^2 with 3 degrees of freedom falls below 6.25. Since the observed value for the statistic of formula 6.4 is 5.51 (Table 6.2), the data fit the extreme value model within statistical uncertainty.

Above sequence lengths M = N = 130, the estimate for parameter λ shows no tendency to increase or decrease with increasing size of the product MN. Its average value for the six cases examined is

$$\lambda = 1.29 \qquad [6.5]$$

Linear regression analysis of u versus ln(MN) yielded a correlation coefficient of 0.9995. The best straight line through these points is given by the formula

$$u = 0.879 \ln(MN) - 1.471 \qquad [6.6]$$

The standard error in the coefficient of ln(MN) is about 0.010 (Mood et al., 1974, pp 482-502). The estimated value of the constant term is dependent upon the estimated coefficient of ln(MN), and is not as important for the estimate of u as MN grows. Formula 6.6 can be written as

$$u = \ln(0.19\ MN)/1.14 \qquad [6.7]$$

From formulas 6.3, 6.5 and 6.7, the significance levels for nucleic acid sequence comparison using similarity function s_1 of formula 4.6 can be estimated as

$$s_\varepsilon = \ln(0.19\ MN)/1.14 - \ln(\varepsilon)/1.29 \qquad [6.8]$$

for $\varepsilon \leq 0.1$.

Formulas 6.5 and 6.6 were derived by Monte Carlo simulation using sequence lengths of equal size. By analogy with results for the longest perfect subalignment of two sequences (Arratia and Waterman, 1985), we expect formula 6.6 to overestimate u when the ratio of ln(M) to ln(MN) is far from 0.5. The parameters u and λ were estimated as above for six pairs of sequence lengths with the same product. The results appear in Table 6-3. Although the estimated u decreases with decreasing ln(M)/ln(MN), formula 6.7 overestimates u by less than 0.3 even for the case of M = 60 and N = 2160. When ln(M)/ln(MN) is far from 0.5, formula 6.8 will tend to underestimate the significance of subalignments.

Nucleic acid sequences with non-uniform nucleotide usage. Frequently nucleic acid sequences have nucleotide usage that is far from uniform. If similarity function s_1 of formula 4.6 is used to compare nucleic acid sequences, it is important that p accurately reflect the probability that random nucleotides from each sequence match. If too great a p is used, then short subalignments will be emphasized over long ones. Conversely, if too small a p is used, then long subalignments will be emphasized over short ones. Even large deviations from non-uniform nucleotide usage can leave p close to 0.25. In practice it is rare for p to be outside the

Table 6-3. Estimation of the parameters u and λ for ln(MN) = 11.77

M	N	ln(M)/ln(MN)	u	λ
($MN = 360^2$)			8.87[a]	1.29[b]
360	360	0.500	8.84 ± 0.04	1.27 ± 0.05
240	540	0.466	8.84 ± 0.04	1.32 ± 0.04
180	720	0.441	8.83 ± 0.04	1.27 ± 0.04
120	1080	0.407	8.70 ± 0.04	1.37 ± 0.04
90	1440	0.382	8.68 ± 0.03	1.33 ± 0.04
60	2160	0.348	8.59 ± 0.03	1.30 ± 0.04

[a] From formula 6.6. [b] From formula 6.5.

range 0.2 to 0.3.

We have investigated the distribution of maximal subalignment similarities for several sets of nucleotide frequencies. Specifically, two distributions of nucleotide frequency were specified. These distributions determined p, the probability of a match, from which s_1-similarity values were calculated using formula 4.6. Then 400 pairs of random sequences were generated; random letters for each sequence were chosen with reference to the appropriate distribution. The DD algorithm was used to find the optimal subalignment similarity for each pair of sequences. Finally, the method described in the previous section was used to estimate parameters k_1, k_2 and λ for the formula

$$s_\varepsilon = \ln(k_1 MN)/k_2 - \ln(\varepsilon)/\lambda \qquad [6.9]$$

Since it is difficult to determine the value that λ approaches, the mean of the estimated estimated values of λ for each set of lengths was used as an estimate for λ. It is not known how to attach a standard error to this estimate, but it appears to be good to about 5%.

The results appear in Tables 6-4 through 6-21, where L.R. means linear regression, and in Table 6-26. In every case the correlation coefficient of the estimated characteristic value u with ln(MN) is greater than 0.9988. While k_1 stays approximately constant at 0.15, k_2 and λ decrease with increasing p. However, as seen in Table 6-26, the ratio of k_2 and λ to ln(p) is essentially constant. This is expected because s_1 is defined to be the logarithm to the base p of a certain probability. If s_1 were defined instead as simply the natural logarithm of this probability then k_2

Table 6-4. Estimation of u & λ
Nucleic acid frequencies:
(0.4, 0.3, 0.2, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.20

M,N	u	λ
70	5.00 ± 0.05	1.34 ± 0.07
90	5.39 ± 0.05	1.29 ± 0.06
116	5.75 ± 0.05	1.39 ± 0.08
148	6.19 ± 0.06	1.29 ± 0.08
191	6.55 ± 0.05	1.48 ± 0.07
245	6.91 ± 0.05	1.44 ± 0.17
314	7.28 ± 0.05	1.31 ± 0.07
403	7.63 ± 0.05	1.34 ± 0.07
518	8.02 ± 0.05	1.51 ± 0.10

Mean : 1.38
L.R.: u = 0.753 ln(MN) - 1.381
Slope standard error: 0.014

Table 6-5. Estimation of u & λ
Nucleic acid frequencies:
(0.3, 0.4, 0.2, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.21

M,N	u	λ
70	5.10 ± 0.05	1.37 ± 0.06
90	5.60 ± 0.05	1.35 ± 0.06
116	5.94 ± 0.05	1.30 ± 0.06
148	6.28 ± 0.06	1.26 ± 0.06
191	6.75 ± 0.05	1.36 ± 0.05
245	7.17 ± 0.05	1.53 ± 0.07
314	7.54 ± 0.05	1.43 ± 0.07
403	8.02 ± 0.05	1.39 ± 0.08
518	8.35 ± 0.05	1.39 ± 0.07

Mean : 1.38
L.R.: u = 0.812 ln(MN) - 1.776
Slope standard error: 0.014

Table 6-6. Estimation of u & λ
Nucleic acid frequencies:
(0.4, 0.2, 0.3, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.21

M,N	u	λ
70	5.21 ± 0.05	1.39 ± 0.08
90	5.54 ± 0.06	1.32 ± 0.11
116	5.96 ± 0.05	1.48 ± 0.06
148	6.38 ± 0.05	1.37 ± 0.06
191	6.81 ± 0.05	1.41 ± 0.07
245	7.21 ± 0.05	1.33 ± 0.07
314	7.59 ± 0.05	1.42 ± 0.07
403	7.95 ± 0.05	1.36 ± 0.07
518	8.26 ± 0.05	1.34 ± 0.06

Mean : 1.38
L.R.: $u = 0.784 \ln(MN) - 1.467$
Slope standard error: 0.014

Table 6-7. Estimation of u & λ
Nucleic acid frequencies:
(0.3, 0.4, 0.1, 0.2) and
(0.1, 0.2, 0.3, 0.4). p = 0.22

M,N	u	λ
70	5.27 ± 0.05	1.28 ± 0.05
90	5.75 ± 0.05	1.36 ± 0.07
116	6.23 ± 0.05	1.34 ± 0.06
148	6.62 ± 0.06	1.24 ± 0.06
191	6.93 ± 0.05	1.33 ± 0.07
245	7.39 ± 0.06	1.26 ± 0.10
314	7.80 ± 0.05	1.32 ± 0.07
403	8.20 ± 0.05	1.39 ± 0.05
518	8.51 ± 0.05	1.39 ± 0.07

Mean : 1.32
L.R.: $u = 0.808 \ln(MN) - 1.513$
Slope standard error: 0.014

Table 6-8. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.4, 0.3, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.23

M,N	u	λ
70	5.44 ± 0.06	1.29 ± 0.08
90	5.92 ± 0.06	1.20 ± 0.04
116	6.36 ± 0.06	1.28 ± 0.07
148	6.74 ± 0.05	1.31 ± 0.04
191	7.12 ± 0.05	1.31 ± 0.07
245	7.61 ± 0.05	1.24 ± 0.05
314	8.04 ± 0.06	1.28 ± 0.07
403	8.40 ± 0.05	1.31 ± 0.07
518	8.85 ± 0.06	1.25 ± 0.07

Mean : 1.27
L.R.: u = 0.844 ln(MN) - 1.697
Slope standard error: 0.016

Table 6-9. Estimation of u & λ
Nucleic acid frequencies:
(0.3, 0.2, 0.4, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.23

M,N	u	λ
70	5.44 ± 0.06	1.16 ± 0.06
90	5.86 ± 0.05	1.26 ± 0.05
116	6.25 ± 0.06	1.21 ± 0.05
148	6.71 ± 0.06	1.29 ± 0.08
191	7.21 ± 0.05	1.30 ± 0.06
245	7.58 ± 0.05	1.35 ± 0.06
314	8.02 ± 0.05	1.31 ± 0.06
403	8.43 ± 0.05	1.27 ± 0.06
518	8.80 ± 0.05	1.26 ± 0.05

Mean : 1.27
L.R.: u = 0.852 ln(MN) - 1.804
Slope standard error: 0.015

Table 6-10. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.3, 0.4, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.24

M,N	u	λ
70	5.59 ± 0.05	1.30 ± 0.05
90	6.08 ± 0.05	1.29 ± 0.04
116	6.47 ± 0.05	1.25 ± 0.05
148	7.00 ± 0.06	1.08 ± 0.05
191	7.37 ± 0.05	1.26 ± 0.06
245	7.79 ± 0.06	1.22 ± 0.07
314	8.20 ± 0.06	1.26 ± 0.08
403	8.70 ± 0.06	1.20 ± 0.07
518	9.11 ± 0.05	1.26 ± 0.04

Mean : 1.24
L.R.: u = 0.873 ln(MN) - 1.801
Slope standard error: 0.016

Table 6-11. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.2, 0.3, 0.3) and
(0.3, 0.3, 0.2, 0.2). p = 0.24

M,N	u	λ
70	5.76 ± 0.04	1.52 ± 0.05
90	6.20 ± 0.05	1.33 ± 0.08
116	6.67 ± 0.06	1.26 ± 0.06
148	7.10 ± 0.05	1.32 ± 0.06
191	7.58 ± 0.06	1.24 ± 0.07
245	7.95 ± 0.05	1.39 ± 0.04
314	8.44 ± 0.06	1.23 ± 0.07
403	8.74 ± 0.05	1.34 ± 0.07
518	9.16 ± 0.05	1.30 ± 0.07

Mean : 1.32
L.R.: u = 0.808 ln(MN) - 1.513
Slope standard error: 0.014

Table 6-12. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.4, 0.1, 0.3) and
(0.1, 0.2, 0.3, 0.4). p = 0.25

M,N	u	λ
70	5.77 ± 0.07	1.14 ± 0.08
90	6.28 ± 0.06	1.22 ± 0.06
116	6.71 ± 0.05	1.31 ± 0.05
148	7.13 ± 0.06	1.20 ± 0.05
191	7.69 ± 0.06	1.23 ± 0.07
245	8.07 ± 0.06	1.26 ± 0.09
314	8.43 ± 0.05	1.28 ± 0.06
403	8.91 ± 0.06	1.13 ± 0.06
518	9.37 ± 0.06	1.22 ± 0.05

Mean : 1.22
L.R.: u = 0.889 ln(MN) - 1.742
Slope standard error: 0.017

Table 6-13. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.3, 0.2, 0.3) and
(0.2, 0.2, 0.3, 0.3). p = 0.25

M,N	u	λ
70	5.98 ± 0.05	1.39 ± 0.09
90	6.39 ± 0.06	1.25 ± 0.06
116	6.91 ± 0.07	1.17 ± 0.08
148	7.29 ± 0.05	1.38 ± 0.07
191	7.72 ± 0.06	1.34 ± 0.09
245	8.23 ± 0.05	1.27 ± 0.05
314	8.60 ± 0.05	1.28 ± 0.05
403	9.11 ± 0.06	1.29 ± 0.09
518	9.48 ± 0.05	1.36 ± 0.09

Mean : 1.30
L.R.: u = 0.883 ln(MN) - 1.526
Slope standard error: 0.016

Table 6-14. Estimation of u & λ
Nucleic acid frequencies:
(0.1, 0.4, 0.3, 0.2) and
(0.1, 0.2, 0.3, 0.4). p = 0.26

M,N	u	λ
70	6.04 ± 0.06	1.20 ± 0.07
90	6.49 ± 0.06	1.25 ± 0.08
116	7.02 ± 0.05	1.26 ± 0.04
148	7.37 ± 0.06	1.21 ± 0.06
191	7.94 ± 0.06	1.21 ± 0.06
245	8.39 ± 0.06	1.22 ± 0.06
314	8.79 ± 0.07	1.10 ± 0.07
403	9.18 ± 0.06	1.22 ± 0.07
518	9.65 ± 0.05	1.28 ± 0.06

Mean : 1.22
L.R.: u = 0.903 ln(MN) - 1.604
Slope standard error: 0.017

Table 6-15. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.2, 0.3, 0.3) and
(0.2, 0.2, 0.3, 0.3). p = 0.26

M,N	u	λ
70	6.17 ± 0.06	1.33 ± 0.08
90	6.56 ± 0.06	1.20 ± 0.06
116	7.05 ± 0.05	1.27 ± 0.05
148	7.51 ± 0.06	1.25 ± 0.07
191	7.99 ± 0.05	1.29 ± 0.05
245	8.39 ± 0.05	1.35 ± 0.05
314	8.81 ± 0.06	1.26 ± 0.06
403	9.31 ± 0.06	1.26 ± 0.10
518	9.75 ± 0.06	1.29 ± 0.07

Mean : 1.28
L.R.: u = 0.899 ln(MN) - 1.494
Slope standard error: 0.017

Table 6-16. Estimation of u & λ
Nucleic acid frequencies:
(0.1, 0.3, 0.4, 0.2) and
(0.1, 0.2, 0.3, 0.4). p = 0.27

M,N	u	λ
70	6.28 ± 0.06	1.24 ± 0.07
90	6.67 ± 0.06	1.21 ± 0.06
116	7.12 ± 0.07	1.08 ± 0.07
148	7.68 ± 0.06	1.13 ± 0.04
191	8.17 ± 0.06	1.11 ± 0.05
245	8.57 ± 0.05	1.27 ± 0.05
314	9.06 ± 0.06	1.22 ± 0.06
403	9.43 ± 0.06	1.19 ± 0.07
518	9.99 ± 0.06	1.15 ± 0.06

Mean : 1.18
L.R.: u = 0.930 ln(MN) - 1.656
Slope standard error: 0.017

Table 6-17. Estimation of u & λ
Nucleic acid frequencies:
(0.3, 0.2, 0.4, 0.1) and
(0.1, 0.2, 0.3, 0.4). p = 0.27

M,N	u	λ
70	6.07 ± 0.06	1.12 ± 0.04
90	6.66 ± 0.06	1.20 ± 0.05
116	7.09 ± 0.06	1.17 ± 0.06
148	7.53 ± 0.06	1.19 ± 0.09
191	7.90 ± 0.06	1.21 ± 0.05
245	8.43 ± 0.06	1.19 ± 0.06
314	8.89 ± 0.06	1.15 ± 0.06
403	9.39 ± 0.06	1.12 ± 0.05
518	9.88 ± 0.06	1.16 ± 0.04

Mean : 1.17
L.R.: u = 0.931 ln(MN) - 1.797
Slope standard error: 0.017

Table 6-18. Estimation of u & λ
Nucleic acid frequencies:
(0.2, 0.1, 0.4, 0.3) and
(0.1, 0.2, 0.3, 0.4). p = 0.28

M,N	u	λ
70	6.24 ± 0.06	1.17 ± 0.06
90	6.76 ± 0.06	1.13 ± 0.06
116	7.29 ± 0.07	1.10 ± 0.07
148	7.82 ± 0.06	1.17 ± 0.06
191	8.34 ± 0.06	1.17 ± 0.05
245	8.83 ± 0.06	1.15 ± 0.06
314	9.26 ± 0.06	1.18 ± 0.07
403	9.82 ± 0.06	1.16 ± 0.04
518	10.24 ± 0.07	1.16 ± 0.08

Mean : 1.15
L.R.: u = 1.005 ln(MN) - 2.260
Slope standard error: 0.017

Table 6-19. Estimation of u & λ
Nucleic acid frequencies:
(0.1, 0.2, 0.4, 0.3) and
(0.1, 0.2, 0.3, 0.4). p = 0.29

M,N	u	λ
70	6.49 ± 0.06	1.13 ± 0.06
90	7.03 ± 0.06	1.14 ± 0.04
116	7.57 ± 0.06	1.08 ± 0.04
148	8.00 ± 0.06	1.14 ± 0.07
191	8.70 ± 0.06	1.11 ± 0.06
245	9.04 ± 0.06	1.15 ± 0.06
314	9.58 ± 0.06	1.10 ± 0.06
403	10.09 ± 0.06	1.10 ± 0.05
518	10.54 ± 0.06	1.13 ± 0.05

Mean : 1.12
L.R.: u = 1.015 ln(MN) - 2.098
Slope standard error: 0.018

Table 6-20. Estimation of u & λ
Nucleic acid frequencies:
(0.1, 0.3, 0.2, 0.4) and
(0.1, 0.2, 0.3, 0.4). p = 0.29

M,N	u	λ
70	6.50 ± 0.06	1.27 ± 0.06
90	7.06 ± 0.07	1.07 ± 0.06
116	7.54 ± 0.06	1.11 ± 0.06
148	8.03 ± 0.06	1.12 ± 0.05
191	8.52 ± 0.07	1.06 ± 0.08
245	9.11 ± 0.06	1.09 ± 0.05
314	9.58 ± 0.07	1.08 ± 0.06
403	10.06 ± 0.06	1.12 ± 0.05
518	10.51 ± 0.06	1.10 ± 0.05

Mean : 1.10
L.R.: u = 1.007 ln(MN) - 2.028
Slope standard error: 0.019

Table 6-21. Estimation of u & λ
Nucleic acid frequencies:
(0.1, 0.2, 0.3, 0.4) and
(0.1, 0.2, 0.3, 0.4). p = 0.30

M,N	u	λ
70	6.69 ± 0.07	1.05 ± 0.05
90	7.17 ± 0.07	1.01 ± 0.07
116	7.75 ± 0.06	1.17 ± 0.07
148	8.24 ± 0.07	1.03 ± 0.06
191	8.91 ± 0.07	1.08 ± 0.07
245	9.36 ± 0.06	1.17 ± 0.04
314	9.86 ± 0.06	1.11 ± 0.06
403	10.31 ± 0.06	1.14 ± 0.06
518	10.81 ± 0.07	1.03 ± 0.06

Mean : 1.09
L.R.: u = 1.042 ln(MN) - 2.149
Slope standard error: 0.019

Table 6-22. Estimation of u & λ
Standard amino acid frequencies.
Costs: c_{id} $p_0 = 0.06$

M,N	u	λ
55	2.68 ± 0.03	2.38 ± 0.14
70	2.88 ± 0.03	2.57 ± 0.18
90	3.15 ± 0.03	2.43 ± 0.10
116	3.32 ± 0.03	2.57 ± 0.12
148	3.56 ± 0.03	2.47 ± 0.11
191	3.80 ± 0.02	2.71 ± 0.09
245	4.01 ± 0.02	2.87 ± 0.14
314	4.21 ± 0.03	2.65 ± 0.13
403	4.39 ± 0.03	2.66 ± 0.09
518	4.62 ± 0.03	2.62 ± 0.10

Mean : 2.59
L.R.: $u = 0.433 \ln(MN) - 0.772$
Slope standard error: 0.007

Table 6-23. Estimation of u & λ
Standard amino acid frequencies.
Costs: c_{d6} $p_0 = 10^{-4}$

M,N	u	λ
55	0.88 ± 0.01	8.5 ± 0.4
70	0.94 ± 0.01	7.9 ± 0.5
90	1.01 ± 0.01	9.4 ± 0.5
116	1.07 ± 0.01	7.4 ± 0.4
148	1.13 ± 0.01	8.0 ± 0.4
191	1.21 ± 0.01	8.2 ± 0.5
245	1.27 ± 0.01	8.5 ± 0.4
314	1.34 ± 0.01	8.3 ± 0.3

Mean : 8.3
L.R.: $u = 0.132 \ln(MN) - 0.180$
Slope standard error: 0.003

Table 6-24. Estimation of u & λ
Standard amino acid frequencies.
Costs: c_{gc} $p_0 = 0.06$

M,N	u	λ
55	2.83 ± 0.03	2.72 ± 0.18
70	3.02 ± 0.03	2.67 ± 0.11
90	3.24 ± 0.03	2.60 ± 0.11
116	3.49 ± 0.03	2.57 ± 0.13
148	3.72 ± 0.03	2.55 ± 0.20
191	3.91 ± 0.03	2.51 ± 0.14
245	4.10 ± 0.03	2.49 ± 0.12
314	4.29 ± 0.03	2.53 ± 0.13
403	4.55 ± 0.03	2.65 ± 0.13

Mean : 2.59
L.R.: $u = 0.428 \ln(MN) - 0.601$
Slope standard error: 0.009

Table 6-25. Estimation of u & λ
Standard amino acid frequencies.
Costs: c_{gm} $p_0 = 0.06$

M,N	u	λ
55	2.80 ± 0.03	2.57 ± 0.13
70	3.02 ± 0.03	2.70 ± 0.13
90	3.22 ± 0.03	2.65 ± 0.13
116	3.47 ± 0.03	2.27 ± 0.10
148	3.67 ± 0.03	2.43 ± 0.13
191	3.93 ± 0.03	2.70 ± 0.17
245	4.14 ± 0.03	2.48 ± 0.14
314	4.32 ± 0.03	2.71 ± 0.14
403	4.55 ± 0.03	2.57 ± 0.10

Mean : 2.59
L.R.: $u = 0.433 \ln(MN) - 0.772$
Slope standard error: 0.007

Table 6-26. Estimated parameters k_1, k_2 and λ for a variety of letter distributions, substitution costs, and the associated similarity functions s_1.

Table	p_0	k_1	k_2	λ	$-k_2/\ln(p_0)$	$-\lambda/\ln(p_0)$
6-4	0.20	0.16	1.328	1.38	0.83 ± 2%	0.86
6-5	0.21	0.11	1.232	1.38	0.79 ± 2%	0.88
6-6	0.21	0.15	1.276	1.38	0.82 ± 2%	0.88
6-7	0.22	0.15	1.238	1.32	0.82 ± 2%	0.87
6-8	0.23	0.13	1.185	1.27	0.80 ± 2%	0.86
6-9	0.23	0.12	1.174	1.27	0.80 ± 2%	0.86
6-10	0.24	0.13	1.145	1.24	0.81 ± 2%	0.87
6-11	0.24	0.18	1.171	1.33	0.82 ± 2%	0.93
6-12	0.25	0.14	1.125	1.22	0.81 ± 2%	0.88
6-13	0.25	0.18	1.133	1.30	0.82 ± 2%	0.94
6-14	0.26	0.17	1.107	1.22	0.82 ± 2%	0.91
6-15	0.26	0.19	1.112	1.28	0.82 ± 2%	0.95
6-16	0.27	0.17	1.075	1.18	0.82 ± 2%	0.90
6-17	0.27	0.15	1.074	1.17	0.82 ± 2%	0.89
6-18	0.28	0.11	0.995	1.15	0.79 ± 2%	0.90
6-19	0.29	0.13	0.985	1.12	0.80 ± 2%	0.90
6-20	0.29	0.13	0.993	1.10	0.80 ± 2%	0.89
6-21	0.30	0.13	0.960	1.09	0.80 ± 2%	0.91
6-22	0.06	0.17	2.31	2.59	0.82 ± 2%	0.92
6-23	10^{-4}	0.25	7.58	8.3	0.82 ± 2%	0.90
6-24	0.06	0.19	2.27	2.56	0.81 ± 2%	0.91
6-25	0.06	0.25	2.34	2.59	0.83 ± 2%	0.92

and λ would remain independent of p. As a result, we can present the following general formula for the significance levels of s_1 when used for nucleic acid sequence comparison:

$$s_\varepsilon = [\ln(0.15\ MN)/0.81 - \ln(\varepsilon)/0.90] \ / \ [-\ln(p)] \qquad [6.10]$$

While the ratio of k_2 to ln(p) is constant within statistical error, the ratio of λ to ln(p) shows substantial variation. In particular, this ratio seems to depend upon the distributions of nucleotide frequencies. Since significance levels are desired generally for only a relatively small range of ε, say from 0.1 to 0.001, the value of λ is not as important as that of k_2 for estimating s_ε. Nevertheless, better prediction of the value of λ is desirable.

Protein sequences. Several sets of substitution costs have been used for protein sequence comparison. We have investigated four of these. The first set, c_{id} (identity), was discussed above. The second set is c_{d6} (Dayhoff-6), which is shown in Table 6-27. It is a scaled version of the PAM-250 matrix of Dayhoff et al. (1978). The third set is c_{gc} (genetic code), which is shown in Table 6-28. For c_{gc} the cost of aligning two amino acids is the minimum number of point mutations needed to change a codon for one amino acid into a codon for the other. The fourth set of substitution costs, c_{gm} (genetic metric), is a metric version of c_{gc}, in which all 3's are replaced with 2's (Erickson and Sellers, 1983). Using the amino acid frequencies shown in Table 6-29 (Dayhoff et al., 1978), s_1 values were calculated for each of these substitution cost functions.

Table 6-27. Amino acid substitution costs c_{d6}.

	A	S	G	L	K	V	T	P	E	D	N	I	Q	R	F	Y	C	H	M	W
A	3	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	4	4	4	5
S	4	3	4	5	4	4	4	4	4	4	4	4	4	4	5	5	4	4	4	4
G	4	4	3	5	4	4	4	4	4	4	4	5	4	5	5	5	5	4	5	6
L	4	5	5	2	5	3	4	5	5	5	5	3	4	5	3	4	5	4	3	4
K	4	4	4	5	3	4	4	4	4	4	4	4	4	3	5	5	5	4	4	5
V	4	4	4	3	4	3	4	4	4	4	4	3	4	4	4	4	4	4	3	5
T	4	4	4	4	4	4	3	4	4	4	4	4	4	4	5	5	4	4	4	5
P	4	4	4	5	4	4	4	2	4	4	4	4	4	4	5	5	5	4	4	5
E	4	4	4	5	4	4	4	4	3	3	4	4	3	4	5	5	5	4	4	6
D	4	4	4	5	4	4	4	4	3	3	3	4	3	4	5	5	5	4	5	6
N	4	4	4	5	4	4	4	4	4	3	3	4	4	4	5	4	5	3	4	5
I	4	4	5	3	4	3	4	4	4	4	4	3	4	4	4	4	4	4	3	5
Q	4	4	4	4	4	4	4	4	3	3	4	4	3	4	5	5	5	3	4	5
R	4	4	5	5	3	4	4	4	4	4	4	4	4	2	5	5	5	3	4	3
F	5	5	5	3	5	4	5	5	5	5	5	4	5	5	2	2	5	4	4	4
Y	5	5	5	4	5	4	5	5	5	5	4	4	5	5	2	1	4	4	4	4
C	4	4	5	5	5	4	4	5	5	5	5	4	5	5	5	4	1	5	5	6
H	4	4	4	4	4	4	4	4	4	4	4	3	4	3	3	4	4	5	2	4	5
M	4	4	5	3	4	3	4	4	4	5	4	3	4	4	4	4	5	4	2	5
W	5	4	6	4	5	5	5	5	6	6	5	5	5	3	4	4	6	5	5	0

Table 6-28. Amino acid substitution costs c_{gc}.

	A	S	G	L	K	V	T	P	E	D	N	I	Q	R	F	Y	C	H	M	W
A	0	1	1	2	2	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2
S	1	0	1	1	2	2	1	1	2	2	1	1	2	1	1	1	1	2	2	1
G	1	1	0	2	2	1	2	2	1	1	2	2	2	1	2	2	1	2	2	1
L	2	1	2	0	2	1	2	1	2	2	2	1	1	1	1	2	2	1	1	1
K	2	2	2	2	0	2	1	2	1	2	1	1	1	1	3	2	3	2	1	2
V	1	2	1	1	2	0	2	2	1	1	2	1	2	2	1	2	2	2	1	2
T	1	1	2	2	1	2	0	1	2	2	1	1	2	1	2	2	2	2	1	2
P	1	1	2	1	2	2	1	0	2	2	2	2	1	1	2	2	2	1	2	2
E	1	2	1	2	1	1	2	2	0	1	2	2	1	2	3	2	3	2	2	2
D	1	2	1	2	2	1	2	2	1	0	1	2	2	2	2	1	2	1	3	3
N	2	1	2	2	1	2	1	2	2	1	0	1	2	2	2	1	2	1	2	3
I	2	1	2	1	1	1	1	2	2	2	1	0	2	1	1	2	2	2	1	3
Q	2	2	2	1	1	2	2	1	1	2	2	2	0	1	3	2	3	1	2	2
R	2	1	1	1	1	2	1	1	2	2	2	1	1	0	2	2	1	1	1	1
F	2	1	2	1	3	1	2	2	3	2	2	1	3	2	0	1	1	2	2	2
Y	2	1	2	2	2	2	2	2	2	1	1	2	2	2	1	0	1	1	3	2
C	2	1	1	2	3	2	2	2	3	2	2	2	3	1	1	1	0	2	3	1
H	2	2	2	1	2	2	2	1	2	1	1	2	1	1	2	1	2	0	3	3
M	2	2	2	1	1	1	1	2	2	3	2	1	2	1	2	3	3	3	0	2
W	2	1	1	1	2	2	2	2	2	3	3	3	2	1	2	2	1	3	2	0

Table 6-29. Amino acid frequencies (Dayhoff et al., 1978)

Gly	0.089	Val	0.065	Arg	0.041	His	0.034
Ala	0.087	Thr	0.058	Asn	0.040	Cys	0.033
Leu	0.085	Pro	0.050	Phe	0.040	Tyr	0.030
Lys	0.081	Glu	0.050	Gln	0.038	Met	0.015
Ser	0.070	Asp	0.047	Ile	0.037	Trp	0.010

Table 6-30. Estimation of the parameters u and λ.

L	M	N	ln(LMN)	u_N	λ_N	u_P	λ_P
40	40	40	11.07	3.62 ± 0.05	2.8 ± 0.3	1.97 ± 0.03	5.1 ± 0.4
60	60	60	12.28	4.27 ± 0.05	2.6 ± 0.2	2.20 ± 0.03	4.5 ± 0.6
80	80	80	13.15	4.68 ± 0.06	2.3 ± 0.2	2.33 ± 0.03	4.1 ± 0.3
100	100	100	13.82	4.88 ± 0.05	2.8 ± 0.3	2.49 ± 0.03	4.6 ± 0.5
120	120	120	14.36	5.20 ± 0.06	2.3 ± 0.1	2.61 ± 0.03	5.1 ± 0.6
140	140	140	14.82	5.48 ± 0.06	2.4 ± 0.2	2.70 ± 0.03	5.0 ± 0.4
160	160	160	15.23	5.56 ± 0.05	2.9 ± 0.3	2.82 ± 0.03	5.5 ± 0.7
180	180	180	15.58	5.76 ± 0.05	2.8 ± 0.2	2.90 ± 0.03	5.2 ± 0.5
200	200	200	15.89	5.80 ± 0.05	2.6 ± 0.1	2.99 ± 0.03	4.7 ± 0.7

Using the frequencies in Table 6-29, 400 pairs of random protein sequences of various lengths were generated, and the parameters k_1, k_2 and λ were estimated as described above. As is seen in Tables 6-22 to 6-26, the significance levels for protein sequence comparison using any of the four substitution cost functions and their associated similarity functions are well approximated by formula 6.10, with p replaced by p_0. The main difference is that for the four protein sequence comparisons the estimated value of k_1 averages 0.21 rather than 0.15.

Hypothesis testing. The significance levels derived above are appropriate for any diagonal subalignment. A subalignment need not be found by using the DD algorithm in order to calculate its s_1-similarity and evaluate its significance using formula 6.10. Even the simple exercise of sliding one sequence past another and visually inspecting the resulting alignments can occasionally yield useful diagonal subalignments. What is required is both the subalignment and knowledge of the context in which it was found.

When testing experimentally the hypothesis that a segment of a given gene probe is significantly similar to a segment present somewhere in the entire genome of an organism, M must be set equal to the length of the gene probe and N to the effective genome length, which is somewhat less than the actual length if long stretches of repetitive DNA are present.

When testing the hypothesis that a particular gene segment has internal repeats, both M and N should be set equal to the length of the gene segment. In order to allow for the symmetry of the comparison, the

term MN of formula 6.10 should be replaced by MN/2. For example, two 61-nucleotide segments (195-255 and 498-558) of the human interferon beta-1 gene align with 25 mismatches (Erickson et al., 1984). Using s_1 of formula 4.6 with p = 0.25, the similarity of this subalignment is $s_1(61,25) = 12.9$. Since this result was found by comparing the first 636 nucleotides of this gene with itself (M = N = 636), formula 6.10 with the correction mentioned above gives $s_{0.01} = 12.9$. Thus the subalignment is significant at the 99% level. The dinucleotide and codon usage of these segments does not account for their similarity (Appendix; Altschul and Erickson, 1985).

The Needleman-Wunsch similarity (formula 4.1 with Y = 0.9) explored by Smith et al. (1985) scores this subalignment as $s_{NW}(61,25) = 13.5$. This is essentially equal to the mean optimal s_{NW} score (13.48) for comparisons of two unrelated sequences of length 636, corrected for symmetry (Smith et al., 1985). Thus the subalignment is not significant by their criterion. One reason for this discrepancy is that subalignments more likely to appear by chance are also given this s_{NW} score. For example, a subalignment of length 23 with 5 or fewer mismatches is seven times more likely to appear by chance than one of length 61 with 25 or fewer mismatches, but $s_{NW}(23,5) = 13.5$. Similarly, a subalignment of length 25 with 6 or fewer mismatches is also seven times more likely to appear by chance than one of length 61 with 25 or fewer mismatches, but $s_{NW}(25,6) = 13.6$. Because similar s_{NW} scores are assigned to all three forms of subalignment, the $s_{NW}(61,25)$ score is not statistically significant. Smith et al. (1985) used natural nucleotide sequences, rather than a random model of nucloetide sequences, to derive their heuristic formula. Although using a different

value for the probability of a match might decrease the estimated significance of this interferon beta-1 subalignment, the basic point remains.

Generalizations. As discussed in Chapter IV, parameters need to be found for similarity function s_3 defined by formula 4.13 so that it may be used with non-infinite gap costs. Once such parameters have been found, the CC-2 can be used to do a statistical analysis for s_3 similar to that provided for s_1 above.

For nucleic acid sequences, in addition to non-uniform nucleotide usage, dinucleotide usage can differ significantly from that predicted by base composition alone (Swartz et al., 1962; Fitch, 1983b; Lipman et al., 1984) and in coding regions codon usage can be markedly non-random (Smith et al., 1983). These factors can render the significance levels of formula 6.10 unreliable. Markov sequence generation or the dinucleotide and codon preserving permutation methods described in the Appendix can be used to investigate what effect such non-uniformity has upon significance levels.

Three-sequence comparison. Alignments, subalignments, costs and similarities can be defined for three-sequence and multi-sequence comparison just as they are defined for two-sequence comparison. All of the alignment algorithms presented in this thesis have obvious generalizations to three or more sequences. At present, time considerations render rigorous three-sequence alignment algorithms practical only for sequences containing a few hundred elements, Nevertheless, it is worthwhile investigating the significance levels of

three-sequence similarity functions because these significance levels can be applied to alignments from any source.

Given a substitution cost function c for aligning two letters, the cost c(x,y,z) of a three-way alignment of letters x, y and z is best defined as the minimum of c(x,w) + c(y,w) + c(z,w) where w may be any allowed letter (Kruskal, 1983). For example, $c_{id}(x,y)$ extends to the function $c_{id}(x,y,z)$, which has the value 0 when x = y = z, 1 when exactly two of the three arguments are equal, and 2 otherwise.

The function s_1 of formula 4.5 can be used for multiple-sequence comparison as well as two-sequence comparison. For nucleic acid sequences in which all types of nucleotide have the same probability of occurrence,

$$p_0 = 1/16,\ p_1 = 9/16,\ p_2 = 6/16 \qquad [6.11]$$

For protein sequence comparison using the amino acid frequencies given in Table 6-29,

$$p_0 = 0.0041,\ p_1 = 0.1684,\ p_2 = 0.8275 \qquad [6.12]$$

For ten sequence lengths L = M = N between 40 and 200, a series of 100 sets of three random sequences were generated. Nucleic acid sequences were simulated by sequences having four types of elements selected randomly with equal probabilities. Protein sequences were simulated by sequences whose elements were selected randomly using the probabilities shown in Table 6-29. For each set of three random sequences, the DDD algorithm (a generalization to three sequences of the DD algorithm) was used to find the

maximum s_1-similarity of all diagonal three-way subalignments. From these 100 similarity values, the extreme value distribution parameters λ and u were estimated as described above.

The results for nucleic acid sequences (subscript N) using the probabilities of formula 6.11 are summarized in Table 6-30. Parameter λ_N shows no tendency to increase or decrease with increasing size of the product LMN. Its average value for the ten cases examined is

$$\lambda_N = 2.6 \qquad [6.13]$$

Linear regression analysis of u_N versus ln(LMN) yielded a correlation coefficient of 0.997. The best straight line through these points is given by the formula $u_N = 0.457\ \ln(LMN) - 1.384$, which can be written as

$$u_N = \ln(0.05\ LMN)/2.19 \qquad [6.14]$$

From formulas 6.3, 6.13 and 6.14, the significance levels for three-way nucleic acid sequence comparison using s_1 with parameters p_i of formula 6.11 can be estimated as

$$s_{\varepsilon,N} = \ln(0.05\ LMN)/2.19 - \ln(\varepsilon)/2.6 \qquad [6.15]$$

for $\varepsilon \lesssim 0.1$.

The results for protein sequences (subscript P) using the probabilities of formula 6.12 are also summarized in Table 6-30. A similar analysis to that provided above yields

$$s_{\varepsilon,P} = \ln(0.15\ LMN)/4.74 - \ln(\varepsilon)/4.9 \qquad [6.16]$$

for $\varepsilon \lesssim 0.1$, where linear regression analysis of u_p versus ln(LMN) gave a correlation coefficient of 0.996.

When the similarity of a diagonal subalignment of three nucleic acid or protein sequences appears significant by formula 6.15 or 6.16, great care should be taken before the alignment is claimed to be significant. First, two of the three sequences may be very similar but the third may not be significantly similar to either. Therefore, the significance of each of the three two-sequence alignments should be compared to that of the three-sequence alignment. Including the third sequence may actually decrease the significance of the subalignment. The parameters p_k used with s_1 should correspond reasonably closely to those calculated from the compositions of the sequences being studied. If different s_1 parameters than those of formulas 6.11 and 6.12 are used, new values of λ and u need to be estimated.

VII. APPLICATIONS

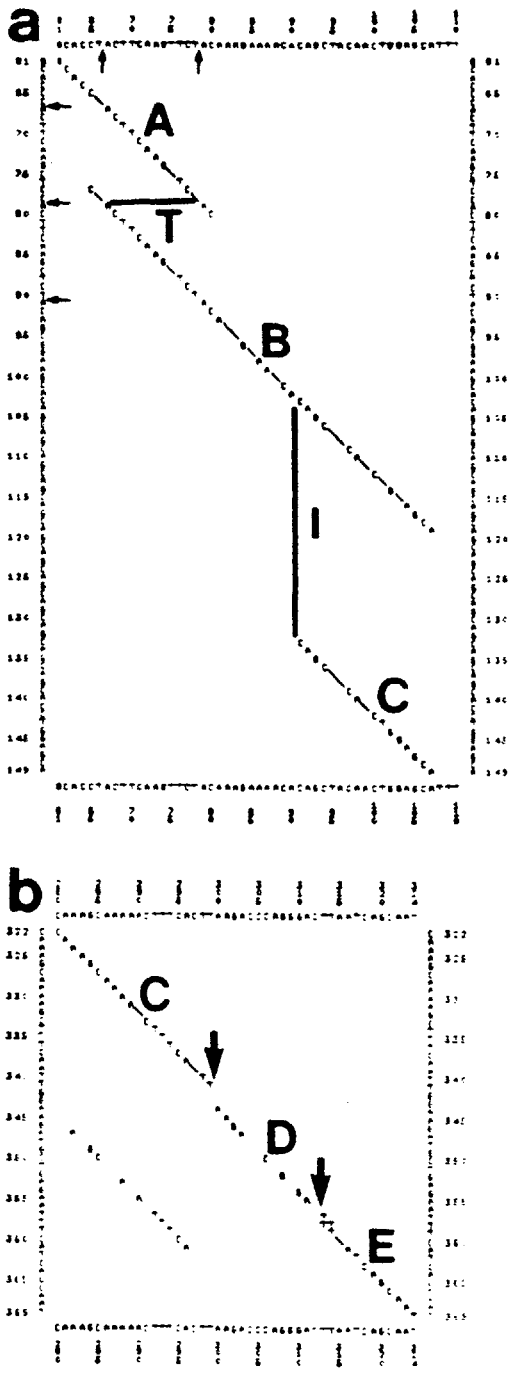
Comparison of interleukin 2 cDNA. The DD algorithm and similarity function s_1 are useful in recognizing similarities and differences between two nucleotide sequences, such as the cDNA sequences for interleukin 2 (IL-2) from man and mouse. This lymphokine plays a key role in the regulation of T-cell clonal expansion. The coding cDNA sequences for human IL-2 (h; 459 nucleotides) (Taniguchi et al., 1983) and murine IL-2 (m; 507 nucleotides) (Yokota et al., 1985) were compared using the DD algorithm. In the context of the comparison (by formula 6.10, with p = 0.25, $s_{0.1}$ = 11.2 and $s_{0.001}$ = 14.9), four significant diagonal subalignments were found: A (h1-80 vs. m1-80; s = 24.6), B (h65-107 vs. m77-119; s = 18.0), C (h91-299 vs. m133-341; s = 88.3) and E (h313-453 vs. m358-498; s = 64.6). Murine IL-2 cDNA (814 nucleotides) was identified by screening for IL-2 activity about 10^4 independent clones containing plasmids with cDNA inserts about 1-2 x 10^3 nucleotides in length derived from murine T-cell mRNA (Yokota et al., 1985). By formula 6.10, the probability is less than 0.002 that this effective experimental comparison of 814 nucleotides with 2 x 10^7 nucleotides would find a subalignment of similarity 24.2 or higher by chance. Thus subalignments A, C and E are each significant at the 99.8% level by this criterion.

Joining these four subalignments into a composite alignment required assignment of the insertions and deletions indicated in two small portions of the DD graph. Figure 7-1a shows the end of A, all of B, and the beginning of C, and Figure 7-1b shows the end of C and the beginning of E. Joining A and B required a horizontal jump from the end of A to the

Figure 7-1. Two portions of the DD graph for two interleukin 2 cDNA sequences. See text for details.

(a) Joining of subalignments A, B and C.

(b) Joining of C, D and E.



beginning of B or a vertical jump to the middle of B. Both jumps could correspond to insertion of a segment into the m (side) sequence, while the vertical jump could also represent a deletion from the h (top) sequence. Figure 7-1a shows that the 12-nucleotide human segment h67-78 is very similar to two adjacent murine segments, m67-78 and m79-90 (arrows). In addition, all three segments encode the same four residues, Thr-Ser-Ser-Ser. This situation is consistent with jump T of Figure 7-1a and probably resulted from tandem duplication of the original 12-nucleotide segment in murine IL-2.

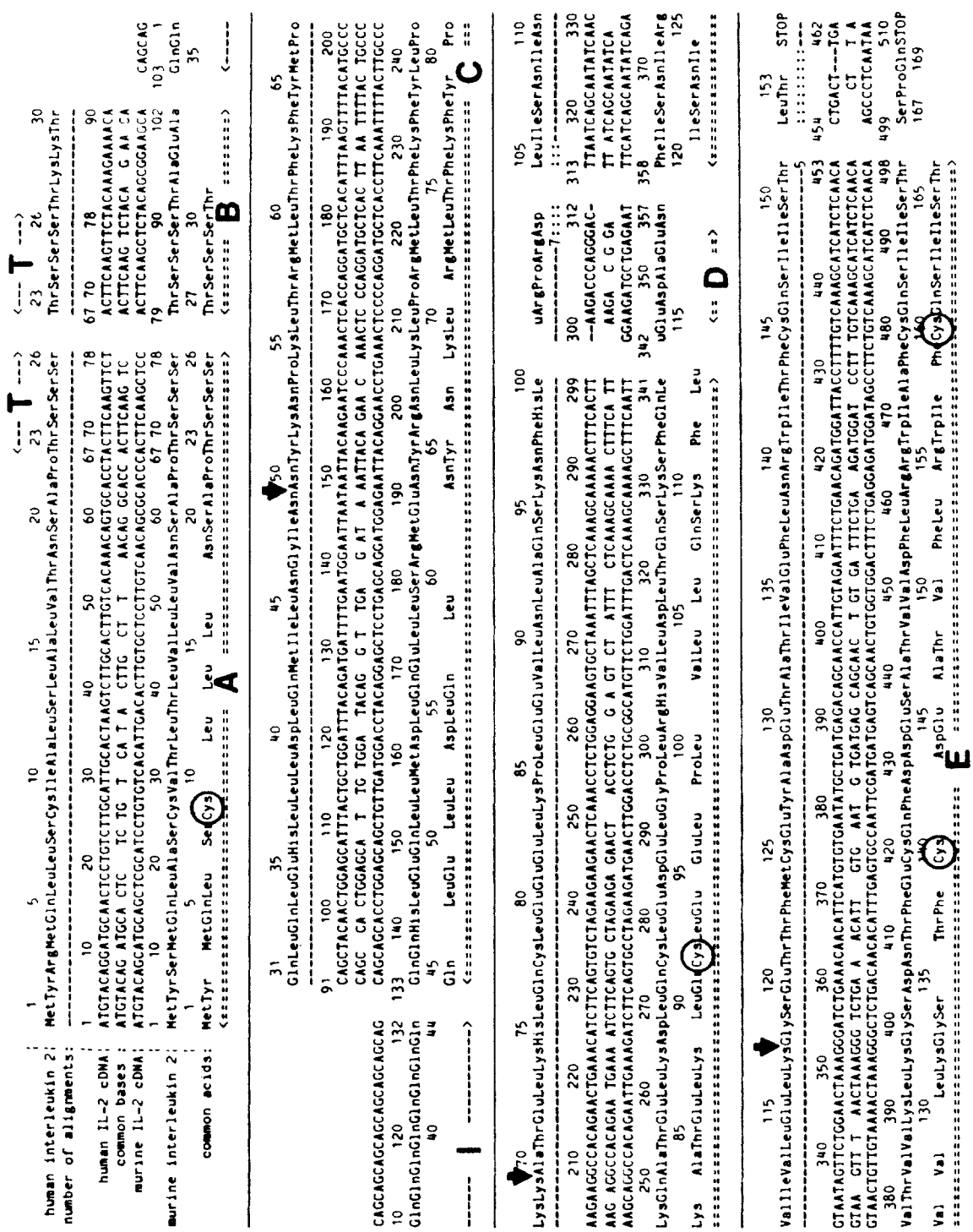
Vertical jump I from the middle of B to the beginning of C (Figure 7-1a) represents a net insertion of 30 nucleotides (m103-132) into the murine sequence. This insert consists of 10 adjacent CAG triplets encoding 10 glutamine residues. It could have arisen by duplication of segment m133-138 encoding Gln-Gln to give m127-138 encoding Gln_4 followed by two tandem duplications of the latter to give m103-138 encoding Gln_{12}. Joining of C and E is mediated by subalignment D (Figure 7-1b), which is significant not by its similarity of 5.1 but by its position between C and E in the DD graph. A vertical jump (arrow) from the end of C to the beginning of D corresponds to a two-nucleotide insertion in m or deletion in h. A second vertical jump (arrow) from D to the beginning of E corresponds to a similar one-nucleotide insertion/deletion event. The cDNA segments that start at the beginning of C and stop at the end of E (h91-453, m133-498) were aligned using the SS algorithm (Sellers, 1974a,b) with a null cost of 2 (Erickson and Sellers, 1983). Since the resulting 7 alignments have length 364.5, cost 93 and similarity 143.6 by linear

interpolation of formula 4.6 with p = 0.25, they are probably much more significant than either C or E alone. These optimal alignments differ only in the position of the jump from D to E.

A composite alignment of the human and murine cDNA sequences that encode IL-2 is shown in Figure 7-2, in which the bold letters A-E correspond to all or part of the subalignments. None of the jumps between subalignments occur near the three intron junction sites (arrows) present in the genomic DNA sequence for human IL-2 (Fujita et al., 1983). The amino acids encoded by the cDNA sequences are indirectly aligned in Figure 7-2. Of the 153 residues in human IL-2, 100 (65%) are preserved in murine IL-2 ("common acids"), including all four Cys residues (circles). This alignment of the IL-2 protein sequences differs from the visually derived alignment reported previously (Yokota et al., 1985) in three regions (human residues 13-18, 23-33 and 101-105). Thus the DD algorithm has revealed interesting DNA and protein patterns not previously described.

This example illustrates the following search procedure for recognition of patterns shared by two nucleotide sequences. The DD graph is examined visually to identify two (or more) subalignments that might be combined into a single subalignment. Each such subalignment is generated using the SS algorithm and its similarity is calculated. If its similarity is higher than $s_{0.1}$, its biological relevance is considered.

Figure 7-2. Composite alignment of the human and murine cDNA sequences that encode interleukin 2. Bold letters correspond to all or part of five diagonal subalignments (A-E) found by the DD algorithm or to assigned jumps (T, I) in the DD graph. See text for details.



Comparison of yeast and E. coli pyrophosphatases. The complete amino acid sequence (Figure 7-3a) for yeast pyrophosphatase has been determined (Cohen et al., 1978), and recently a 3.0 A three-dimensional structure has been proposed (Terzyan et al., 1984). Partial amino acid sequences (Figure 7-3b,c) for E. coli pyrophosphatase are known (Cohen, 1978). We used the DD algorithm to search for similarities between the yeast and E. coli sequences.

First, we employed the substitution costs c_{id} and the associated similarity function s_1 of formula 4.6 with p = 0.06. The best seven diagonal subalignments found when sequences y and e of Figure 7-3 were compared are shown in Table 7-1. Using the parameters k_1, k_2 and λ of Table 6-26, for sequences of lengths 285 and 86, u = 3.6 and $s_{0.1} = 4.5$. Thus none of the diagonal subalignments found are statistically significant. Nevertheless, it is interesting that four of the seven best subalignments (numbers 1, 2, 5 and 6) are very near each other in the path graph, suggesting they may be part of a longer and more interesting subalignment that contains gaps.

We compared the same two sequences using substitution costs c_{d6} (Table 6-27) and the associated similarity function s_1 generated from formulas 4.3 to 4.5 by the amino acid frequencies of Table 6-29. The best eight diagonal subalignments are shown in Table 7-2. Variations of the four close diagonal subalignments found with c_{id} are also found with c_{d6}. The subalignment of diagonal 27 extends from lenth 7 to length 34 while the subalignment of diagonal 49 is trimmed from length 20 to length 8.

Figure 7-3. Three protein sequences.

(a) Yeast pyrophosphatase.

(b) E. coli pyrophosphatase, peptide I.

(c) E. coli pyrophosphatase, peptide II.

a. Yeast pyrophosphatase (y1-285)

TYTTR QIGAK NTLEY KVYIE KDGKP VSAFH DIPLY ADKED NIFNM VVEIP
RWTNA KLEIT KEETL NPIIQ NTKGK LRFVR NCFPH HGYIH NYGAF PQTWE
DPNVS HPETK AVGDN NPIDV LQIGE TIAYT GQVKE VKALG IMALL DEGET
DWKVI AIDIN DPLAP KLNDI EDVEK YFPGL LRATD EWFRI YKIPD GKPEN
QFAFS GEAKN KKYAL DIIKE THNSW KQLIA GKSSD SKGID LTNVT LPDTP
TYSKA ASDAI PPASP KADAP IDKSI DKWFF ISGSV

b. E. coli pyrophosphatase peptide I (e1-86)

SLLNV PAGKD LPEDI YVVIE IPANA DPIKY EIDKE SGALF VDRFM STAMF
YPCNY GYINH TLSLD GDPVD VLVPT PYPLE KGQVI R

c. E. coli pyrophosphatase peptide II (E1-23)

RCHPV GVLKM TDEAG EDAKL VAV

Table 7-1. The seven best weakly locally optimal subalignments from comparison of sequence e with sequence y using substitution costs c_{id} and the associated similarity function s_1.

No.	Sim.	Length	Diagonal	Start e	Start y	End e	End y
1	4.06	20	49	65	114	84	133
2	3.95	7	27	29	56	35	62
3	3.94	33	136	3	139	35	171
4	3.62	8	243	27	270	34	277
5	3.56	30	37	32	69	61	98
6	3.45	5	28	18	46	22	50
7	3.45	5	-43	65	22	69	26

Table 7-2. The eight best weakly locally optimal subalignments from comparison of sequence e with sequence y using substitution costs c_{d6} and the associated similarity function s_1.

No.	Sim.	Length	Diagonal	Start e	Start y	End e	End y
1	1.24	34	27	2	29	35	62
2	1.20	8	49	65	114	72	121
3	1.11	36	13	12	25	47	60
4	1.09	7	31	54	85	60	91
5	1.08	3	126	50	176	52	178
6	1.02	21	-43	65	22	85	42
7	0.98	6	28	17	45	22	50
8	0.98	6	37	51	88	56	93

Table 7-3. Yeast pyrophosphatase residues identified as belonging to the active site (Kuranova et al., 1983).

* E-48	* Y-88	* D-146	Y-191
* E-58	* Y-92	* E-147	N-200
x Q-70	x N-116	x E-149	
* R-77	* D-119	* D-151	

Starred residues are aligned with an identical E. coli residue in Figure 7-4. Residues marked with an x are aligned with a different E. coli residue. Unmarked resides are not aligned.

We attempted to piece together the close subalignments found on diagonals 27, 28, 37 and 49 by running the SS-2 algorithm (Chapter III) on the segments e2-84 and y29-133, using substitution costs c_{d6} and a variety of gap costs. One set of gap costs that yielded an alignment containing sections from all the previously observed subalignments was $w(x) = 2.2 + 2.5(x)$. One of the optimal alignments of e2-84 and y29-133 using these costs is shown in Figure 7-4a.

Is the subalignment of Figure 7-4a statistically significant? We do not have the tools to answer this question, as we have studied statistical significance only for diagonal subalignments. However, there is an observation that supports the claim that this subalignment is significant. Of the 105 amino acids of y29-133, 27 (26%) are aligned with identical amino acids in e2-84. Kuranova et al. (1983) claim the active site of yeast pyrophosphatase consists of the residues listed in Table 7-3. By chemical modification studies, Bond et al. (1980) provide additional evidence that Arg_{77} is an active-site residue and Gonzalez and Cooperman (1986) that Glu_{149} is an active-site residue. Of the 8 putative active-site residues contained in segment y29-133, 6 (75%) are aligned with identical amino acids. One of the two remaining amino acids (Asn_{116}) is aligned with an amino acid that is conformationally almost identical (aspartic acid). Furthermore, the active site residues are generally found in regions that are very similar to regions of the E. coli sequence. This suggests that the active sites of the two enzymes may be very similar while the rest of the enzymes have little in common.

When sequence E (Figure 7-3c) is compared with sequence y

Figure 7-4. Two subalignments of yeast pyrophosphatase and E. coli pyrophosphatase. Identities are echoed on central line. Plus signs identify aligned elements with an substitution cost of 3 or less. Yeast pyrophosphatase residues identified as being in the active site are underlined.

(a) An optimal alignment of segments e2-84 and y29-133 using substitution costs c_{d6} and gap costs $w(x) = 2.2 + 2.5(x)$.

(b) The optimal subalignment of sequences E and y using substitution costs c_{d6} and the associated similarity function s_1.

a.

```
      e2      10         20         30         40
       LLNVPAGKDLPEDIY-VVIEIPANADPIKYEIDKES--GALFVD-----RFMS---
       + ++P   D  ++I+ +V+EIP   +  K EI KE     +  +     RF+
       FHDIPLYADKEDNIFNMVVEIPRWTNA-KLEITKEETLNPIIQNTKGKLRFVRNCF
     y29        40         50         60         70         80

        50              60                70        80  84
       TAMFYPCNYG-----YINHTLSL-------DGDPVDVLVPTPYPLEKGQV
           Y  NYG        +  +S         D +P+DVL          GQV
       PHHGYIHNYGAFPQTWEDPNVSHPETKAVGDNNPIDVLQIGETIAYTGQV
            90        100        110        120          133
```

b.

```
      E5   10        20 23
       VGVLKMTDEAGEDAKLVAV
       +G++ + DE   D K++A+
       LGIMALLDEGETDWKVIAI
     y139       150    157
```

(Figure 7-3a) using the substitution costs c_{d6} and the associated similarity function s_1, the optimal subalignment is that shown in Figure 7-4b. Again, while only 6 of 19 (32%) residues of segment y139-157 are aligned with identical amino acids, 3 of the 4 putative active-site residues are so aligned. The one remaining active-site residue, Glu_{149}, is aligned one position away from an identical residue, as is the adjacent residue Gly_{148}.

Work is in progress (Barry Cooperman, private communication) to use the alignments presented in this section to build a three-dimensional model of the sequenced portions of E. coli pyrophosphatase based upon the published structure of yeast pyrophosphatase (Terzyan et al.).

APPENDIX. RANDOM DINUCLEOTIDE AND CODON PRESERVING SEQUENCE PERMUTATION

As described in Chapter VI, the question of statistical significance for sequence alignments is often approached with Monte Carlo methods that require the generation of random sequences. The question arises from what set the random sequences should be chosen. Several methods have been used for choosing such a set. One is to use existing data banks data banks of real biological sequences. Care must be taken to separate related from unrelated sequences in such data banks. This approach has the disadvantage that the problem of deciding whether a given pair of sequences shows significant relationship expands into the problem of deciding which among many hundreds of sequences are related. A second approach is to generate random sequences having certain well defined statistical properties such as nucleotide or amino acid composition. This can be done either by using the original sequences to define probabilities of occurance for each letter or by shuffling the original sequences.

A simple permutation approach can be too optimistic in claiming sequence similarity. Natural nucleotide sequences are often statistically non-random, which can increase their similarity compared to that of artificial nucleotide sequences generated by random permutation. For instance, dinucleotide usage can differ significantly from that predicted from base composition alone (Swartz et al., 1962; Fitch, 1984b; Lipman et al., 1984). In coding regions, the codon usage can be markedly non-random (Smith et al., 1983). It is important to avoid claiming that sequence similarity is due to nucleotide order if it can be explained merely by non-random usage of dinucleotides and/or codons.

Fitch (1983a) has discussed two methods that can be used to simulate non-random dinucleotide and codon usage. The Markov method, produces sequences that preserve the chosen properties of the original sequence only on the average. The permutation method chooses sequences at random among those that exactly preserve the chosen properties. As noted by Fitch (1983a), the permutation method requires but the Markov method only expects a random sequence to preserve the chosen properties. This appendix describes and illustrates a method that generates with equal probability all permutations with a given dinucleotide usage or dinucleotide and codon usage.

Fitch (1983a) states that for two sequences of sufficient length (i.e. longer than about 300 residues for dinucleotide usage), the two methods should be equivalent. For two interferon DNA sequences of length 60 described later, we found that the Markov and permutation methods yield notably different distributions of alignment distances. A sample of 1000 pairs of sequences that preserve dinucleotide usage was generated. The Markov method gave a sample mean of 41.7 and a sample standard deviation of 3.9; the permutation method gave a sample mean of 39.2 and a sample standard deviation of 2.9. The Markov method also gave a broader range of distances (27-55) than the permutation method (30-46). Fitch (1983a) has discussed the appropriateness of using these methods in different contexts.

Terminology. Although motivated by permutation of nucleotide sequences, the following algorithms can be applied to sequences of any kind. They are based on a theorem first proved by van Aardenne-Ehrenfest and de Bruijn (1951) and later restated by Kasteleyn (1967),

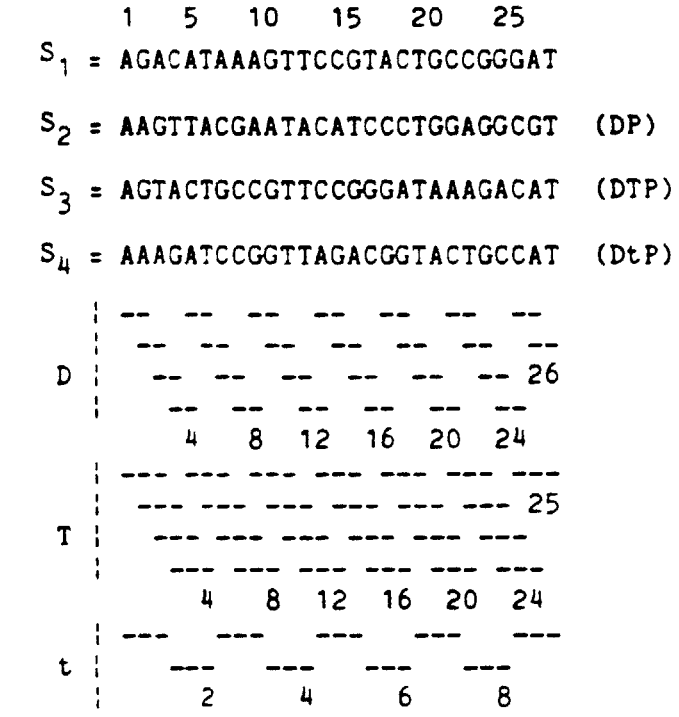
Knuth (1973, p 375) and Zaman (1984).

We shall use the term singlet rather than mononucleotide, doublet in place of dinucleotide, and triplet instead of trinucleotide. Rather than speak of codons, we define a triplon to be a member of a set of consecutive non-overlapping triplets. These terms are illustrated in Figure A-1.

By definition, sequence permutation preserves singlet usage and sequence length. Generation of a random triplon preserving (tP) permutation is easy, since it involves only random permutation of one set of non-overlapping triplons. Generation of a random doublet preserving (DP), doublet and triplet preserving (DTP), or doublet and triplon preserving (DtP) permutation is more difficult because the elements to be preserved overlap. Fitch (1983a) has described an algorithm for generating DP permutations, but it does not generate all permutations with equal probability. (For example, there are two DP permutations of the sequence AATAT, the original sequence and ATAAT. Fitch's algorithm generates the original sequence with probability 1/3 and the second sequence with probability 2/3. If one adds the extra edge TA to the doublet graph as Fitch suggests, his method still generates ATAAT twice as frequently as AATAT.) We describe a modification of Fitch's algorithm that does generate random DP permutations. Lipman et al. (1984) mention without details that they have used such an algorithm. We show how our algorithm can be extended to generation of random DTP or DtP permutations.

As Fitch (1983a) has noted, generating a random DP permutation is equivalent to finding a random Eulerian walk in a directed multigraph. We

Figure A-1. Sequence S_1 and three permutations of S_1. The set of doublets (D), triplets (T) and one set of triplons (t) for S_1 are illustrated. S_2 is a doublet preserving (DP) permutation, S_3 is a doublet and triplet preserving (DTP) permutation and S_4 is a doublet and triplon preserving (DtP) permutation.



use the following definitions. A subgraph of a graph G consists of the set of vertices in G and a subset of the edges in G. An ordered graph is a graph whose edges are ordered. An Eulerian walk is a walk that uses each edge in the graph exactly once. Vertex x_1 is connected to vertex x_2 in a graph G if there is a walk from x_1 to x_2 in G. Note that x_1 may be connected to x_2 even though x_2 is not connected to x_1. Every vertex is connected to itself.

Graphs and edge orderings. Given a sequence $S = s_1 s_2 \ldots s_f$, construct a doublet graph G which has a vertex for each singlet that appears somewhere in S and one edge from vertex s_i to vertex s_{i+1} for each occurence of the doublet $s_i s_{i+1}$ in S. The s edge list is an ordered list of all edges from vertex s. An edge ordering E of the graph is a complete set of edge lists for G. Note that the sequence S determines an edge ordering E(S) as well as the graph G, because the doublets beginning at $s_1 s_2$ occur in a specific order in S. For example, DNA sequence S_1 of Figure A-1 specifies the graph $G(S_1)$ of Figure A-2 and determines edge ordering $E(S_1)$ of Figure A-3. As in this example, the doublet graph of a sequence is a directed multigraph that may contain loops.

Just as S uniquely determines an edge ordering E(S) of its graph, so conversely this edge ordering uniquely determines S. Furthermore, E(S) uniquely determines an Eulerian walk in G from s_1 in the obvious way. For this reason, E(S) will be called an Eulerian edge ordering. On the left of Figure A-4, the edges of G are numbered according to the edge ordering $E(S_1)$. This numbering can be interpreted as an Eulerian walk in G.

Figure A-2. The doublet graph $G(S_1)$. Ten edges are used twice, as indicated.

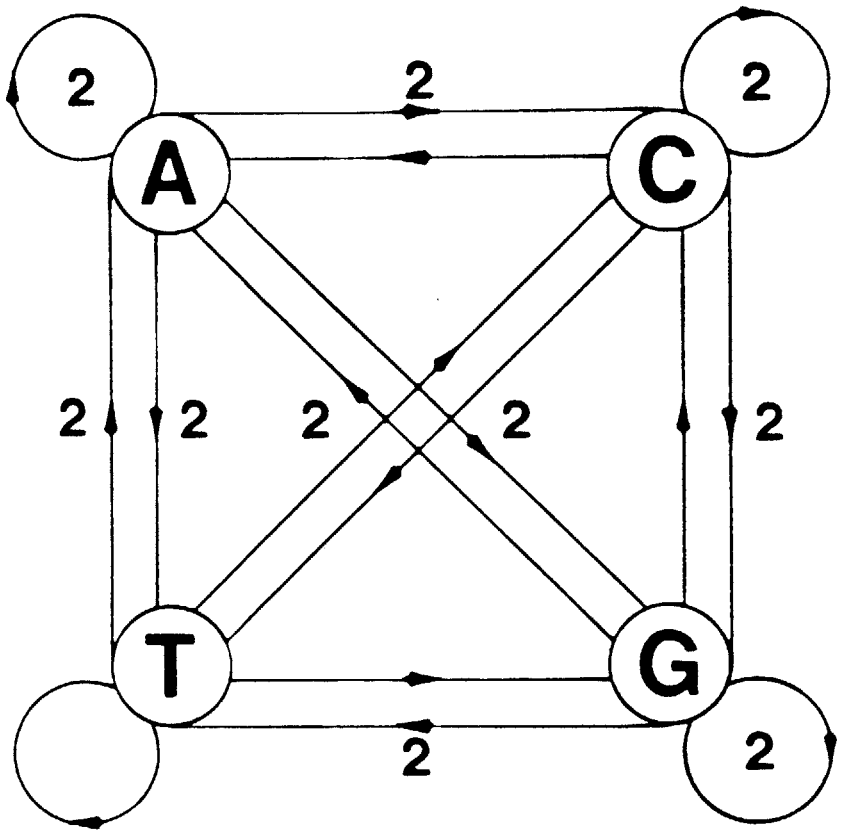
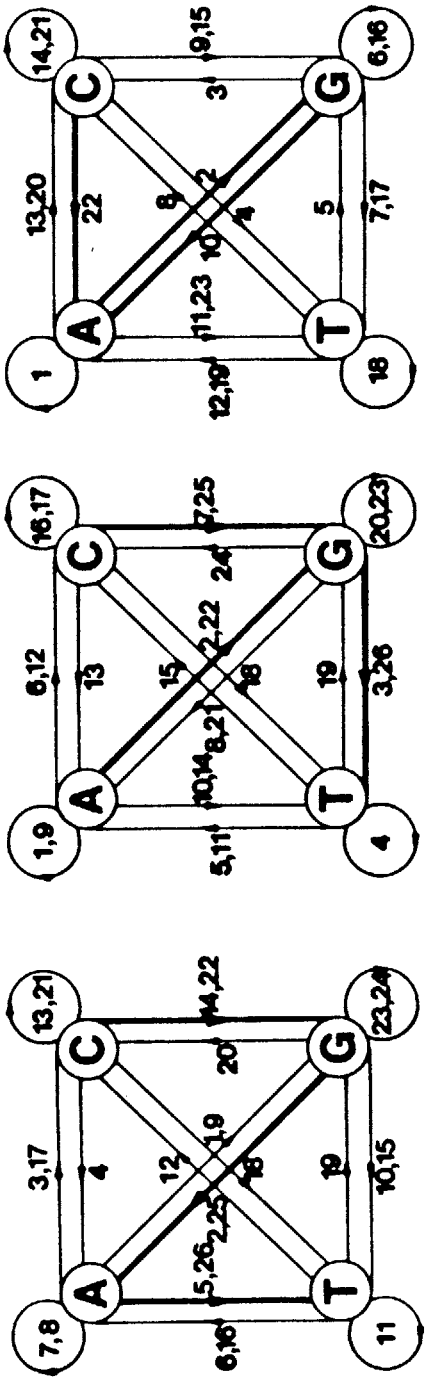


Figure A-3. The edge orderings $E(S_1)$, $E(S_2)$ and E^*. In each edge ordering, numbers indicate the order of each edge in the corresponding long walk. Edges in brackets are not used in the long walk. Underlined edges belong to the last edge graph determined by the ordering.

Edge list	$E(S_1)$	$E(S_2)$	E^*
A:	1 3 5 7 8 9 17 26 AG AC AT AA AA AG AC AT	1 2 6 9 10 12 14 22 AA AG AC AA AT AC AT AG	1 2 11 13 20 23 AA AG AT AC AC AT [AA AG]
C:	4 13 14 18 21 22 CA CC CG CT CC CG	7 13 16 17 18 25 CG CA CC CC CT CG	4 9 14 15 21 22 CT CG CC CG CC CA
G:	2 10 15 20 23 24 25 GA GT GT GC GG GG GA	3 8 20 21 23 24 26 GT GA GG GA GG GC GT	3 6 7 10 16 17 GC GG GT GA GG GT [GA]
T:	6 11 12 16 19 TA TT TC TA TG	4 5 11 15 19 TT TA TA TC TG	5 8 12 18 19 TG TC TA TT TA

Figure A-4. Three long walks. Left, the Eulerian long walk determined by $E(S_1)$. Center, the Eulerian long walk determined by $E(S_2)$. Right, the non-Eulerian long walk determined by E^*. The emphasized edges belong to the last edge graph determined by the corresponding edge ordering.



If sequence S' is a doublet preserving permutation of S, it determines the same doublet graph as S. The edge orderings E(S) and E(S') differ only in the internal order of their edge lists. For example, sequence S_2 of Figure A-1 is a DP permutation of S_1. Its edge ordering $E(S_2)$ is shown in Figure A-3 as a set of edge lists and in the center of Figure A-4 as the edge numbering of an ordered graph. In general, each DP permutation S' of S preserves the terminal singlets s_1 and s_f of S and specifies a unique Eulerian edge ordering of G. Conversely, each Eulerian walk in G from s_1 to s_f specifies a unique DP permutation of S.

A new edge ordering of G can be generated by separately and randomly permuting each edge list of E(S). This random edge ordering determines a long walk in G that starts at vertex s_1 and ends at a vertex whose edge list has been exhausted. This vertex must be s_f, the final vertex of S. Since such a long walk in G usually ends before every edge is used, it is seldom an Eulerian walk. For example, consider the edge ordering E* on the right of Figure A-3. Its long walk on the right of Figure A-4 is not Eulerian because three edges (AA, AG, GA) are still not used when final vertex T is reached for the last time. It is inefficient to trace a long walk most of the way through G before finding that it is not Eulerian. The following theorem provides a general criterion for quickly determining whether or not an edge ordering of G is Eulerian. Given an edge ordering E of G, let the last edge from vertex x be the final edge of the x edge list. The last edge graph Z is the subgraph of G consisting of all last edges except that of the final vertex s_f.

EULERIAN EDGE ORDERING THEOREM. An edge ordering E is Eulerian iff all vertices in the last edge graph Z are connected in Z to s_f.

PROOF. If E is Eulerian, then all vertices are connected in Z to s_f.

A vertex is exhausted during a walk in G when its last edge is used. Note that when a vertex other than s_f is exhausted, not only have all edges from it been used but also all edges to it. If E is Eulerian, the long walk determined by E exhausts all vertices in G. Except for s_f, number these vertices from 1 to N in the order in which they are exhausted. The last edge from N must point to s_f because all long walks must end at s_f and all other vertices have been exhausted. Thus vertex N is connected in Z to s_f. The last edge from N-1 must point to N or s_f because vertices 1, ..., N-1 have been exhausted; thus N-1 is also connected in Z to s_f. By similar reasoning, every vertex is connected in Z to s_f.

If E is not Eulerian, then not all vertices are connected in Z to s_f.

Let U be the set of all vertices not exhausted during the long walk determined by E. Since E is not Eulerian, U must contain at least one vertex because the long walk determined by E does not use all edges of G. Vertex s_f is not a member of U because s_f is exhausted in all long walks. Each edge not used in the walk points to a vertex in U because all other vertices are exhausted. In particular, the last edge of each vertex in U points to a vertex in U. Thus all walks in Z that begin at a vertex in U must end at a vertex in U. Therefore, no vertex in U can be connected in Z to s_f.

Random DP permutation algorithm. A random doublet preserving permutation S' of sequence S is generated by following steps {1} to {6}.

{1} Construct the doublet graph G and edge ordering E corresponding to S.

{2} For each vertex s in G except s_f, randomly select one edge from the s edge list of E(S) to be the last edge of the s list in a new edge ordering.

{3} From this set of last edges, construct the last edge graph Z and determine whether or not all of its vertices are connected to s_f.

{4} If any vertex is not connected in Z to s_f, the new edge ordering will not be Eulerian, so return to {2}. If all vertices are connected in Z to s_f, the new edge ordering will be Eulerian, so continue to {5}.

{5} For each vertex s in G, randomly permute the remaining edges of the s edge list of E(S) to generate the s edge list of the new edge ordering E(S').

{6} Construct sequence S', a random DP permutation of S, from E(S') as follows. Start at the s_1 edge list. At each s_i edge list, add s_i to S', delete the first edge $s_i s_j$ of the edge list, and move to the s_j edge list. Continue this process until all edge lists are exhausted.

E(S') is Eulerian because all vertices of its last edge graph are connected to s_f. Sequence S' is a DP permutation of S because by construction both S and S' specify Eulerian edge orderings of the same graph G. Finally, S' is a random DP permutation because edge ordering E(S') was randomly selected from the set of all Eulerian edge orderings.

DNA examples. This algorithm is efficient because only one edge must be chosen from each edge list except the s_f edge list in order to determine whether or not a DP permutation will result. In particular, the Eulerian edge ordering theorem guarantees that for any DNA sequence no more than three last edges (doublets) need to be selected before deciding whether or not a DP permutation will result. For example, generation of the random DP permutation S_2 from S_1 is illustrated in Figure A-3. AG was randomly selected from the eight doublets of the A edge list to be a new last edge. Similarly, CG was selected from the C edge list and GT from the G edge list. As shown in the middle graph of Figure A-4, all vertices of the emphasized last edge graph are connected to T, so $E(S_2)$ is Eulerian. The last edge of the T edge list is not present in Z because T is the final singlet of S_1. In contrast, as shown by the graph on the right in Figure 4, three vertices of this emphasized last edge graph are not connected to T, so this edge ordering of $G(S_1)$ is not Eulerian.

The Eulerian edge ordering theorem can be used to calculate not only the probability p that a random edge ordering of a graph G will be Eulerian but also the number of possible Eulerian walks starting at s_1 (van Aardenne-Ehrenfest and de Bruijn, 1951). Although p depends on graph G, an approximate value of p can be calculated by assuming that the last edge from each vertex is equally likely to point to any vertex. For a long DNA sequence S containing all four nucleotides, p is approximately 1/4. In other words, about 3/4 of the random edge orderings of G are not Eulerian. Each of these undesired edge orderings is efficiently rejected as soon as one vertex of its last edge graph is known to be not connected to s_f.

Random doublet and triplet preserving permutation. The DP permutation algorithm can be modified to generate a random permutation of sequence S that preserves not only singlet and doublet usage but also triplet usage. The triplet graph G' of S is the graph having a vertex for each doublet that appears somewhere in S and an edge from $s_i s_{i+1}$ to $s_{i+1} s_{i+2}$ for each occurence of the triplet $s_i s_{i+1} s_{i+2}$ in S. S uniquely defines an Eulerian walk in G' from $s_1 s_2$ to $s_{f-1} s_f$ and vice versa. A random Eulerian walk in G' starting at $s_1 s_2$ corresponds to a random DTP permutation of S.

The Eulerian edge ordering theorem can be applied to G' just as it was to the doublet graph G. The random DP permutation algorithm is readily extended to an algorithm for generating a random DTP permutation of sequence S. For instance, DNA sequence S_1 specifies an edge ordering $E'(S_1)$ that consists of 16 edge lists (Figure A-5). Edge ordering $E'(S_3)$ was generated by separately and randomly permuting these lists. The 15 edges of $Z(S_3)$ are underlined. All 16 doublet vertices in $Z(S_3)$ are connected to AT, so edge ordering $E'(S_3)$ is Eulerian and sequence S_3 of Figure A-1 is a random DTP permutation of S_1.

These algorithms can be extended to a set of algorithms for finding random permutations that preserve all doublets, triplets, ..., and n-tuplets of sequence S. As the length of the largest preserved n-tuplet increases, the number of possible permutations decreases, until at some point only S is possible.

Random doublet and triplon preserving permutation. Both dinucleotide usage and codon usage of coding DNA sequences are often non-random. The

Figure A-5. Edge orderings $E'(S_1)$ and $E'(S_3)$ for the triplet graph $G'(S_1)$.

Edge list	$E'(S_1)$	$E'(S_3)$	Edge list	$E'(S_1)$	$E'(S_3)$
AA:	7 8 AAA AAG	20 21 AAA AAG	GA:	2 25 GAC GAT	17 23 GAT GAC
AC:	3 17 ACA ACT	4 24 ACT ACA	GC:	20 GCC	7 GCC
AG:	1 9 AGA AGT	1 22 AGT AGA	GG:	23 24 GGG GGA	15 16 GGG GGA
AT:	5 ATA	18 ATA	GT:	10 15 GTT GTA	2 10 GTA GTT
CA:	4 CAT	25 CAT	TA:	6 16 TAA TAC	3 19 TAC TAA
CC:	13 21 CCG CCG	8 13 CCG CCG	TC:	12 TCC	12 TCC
CG:	14 22 CGT CGG	9 14 CGT CGG	TG:	19 TGC	6 TGC
CT:	18 CTG	5 CTG	TT:	11 TTC	11 TTC

appropriate permutation method for this situation should preserve both the doublet usage and usage of just one of the three sets of triplons. Consider a sequence S whose length is divisible by 3 and whose first triplon is $t_1 = s_1s_2s_3$, as illustrated in Figure A-1. Since triplon $s_is_{i+1}s_{i+2}$ contains doublets s_is_{i+1} and $s_{i+1}s_{i+2}$, preservation of a triplon also preserves both intratriplon doublets. Thus the problem of preserving the triplons and doublets is reduced to the problem of preserving the triplons and intertriplon doublets, which is solved by using the random DP permutation algorithm.

Random DtP permutation algorithm. A random doublet and triplon preserving permutation S' of sequence S is generated by following steps {1} to {6}.

{1} Represent sequence S as a sequence of upper case letters.

{2} Change the upper case letters at positions 3, 6, 9, ... into the corresponding lower case letters to generate sequence S*.

{3} Assign each triplon to an ordered triplon list according to its first and last letters and its order in S*. For example, AGa and ATa are both stored in the Aa triplon list.

{4} Delete the letters at positions 2, 5, 8, ... from S* to form a reduced sequence R, which contains alternating upper-lower case (UL) doublets and lower-upper case (LU) doublets.

{5} Treating the upper and lower case letters as distinct, generate from sequence R a random DP permutation R' using the DP permutation algorithm.

{6} Expand random DP permutation R' into the random DtP permutation S' as follows. Randomly permute each triplon list. Start at the first UL doublet of R'. Replace the current UL doublet in R' by the first triplon of the corresponding triplon list, delete this triplon from the list, and move to the next UL doublet in R'. Continue this process until the triplon lists are exhausted. Change each lower case letter back into the corresponding upper case letter to produce S'.

During step {5}, the UL doublets correspond to triplons in S and the LU doublets correspond to intertriplon doublets in S. The random DP permutation R' preserves the alternation of these distinct sets of doublets. The intratriplon doublets are stored during step {3}, removed in step {4}, and replaced during step {6}. Since each triplon list is randomly permuted before R' is expanded, sequence S' is a random DtP permutation of S. This algorithm is readily extended to a set of algorithms that preserve all doublets and one set of n-tuplons, where an n-tuplon is a member of a set of contiguous non-overlapping n-tuplets.

A DNA example. The random DtP permutation algorithm is illustrated by the generation from DNA sequence S_1 of the new sequence S_4, which preserves all 8 triplons (codons) and 23 doublets (dinucleotides) of S_1. First, S_1 is converted into equivalent sequence S_1* as shown in Figure A-6. Next, the 9 triplons are stored in the 5 triplon lists of S_1* shown in Figure A-7. Then reduced sequence R_1 is generated by deleting the middle singlet of each triplon.

Next, edge ordering $E(R_1)$ is constructed and 7 doublets are randomly

Figure A-6. A series of sequences constructed during generation of S_4, a DtP permutation of S_1.

```
          3     6     9    12    15    18    21    24    27
S1  = A G A C A T A A A G T T C C G T A C T G C C G G G A T

          2     5     8    11    14    17    20    23    26
S1* = A G a C A t A A a G T t C C g T A c T G c C G g A A t

          1     3     5     7     9    11    13    15    17  18
R1  = A   a C   t A   a G   t C   g T   c T   c C   g A   t

          1     3     5     7     9    11    13    15    17  18
R4  = A   a G   t C   g G   t A   a C   g T   c T   c C   t

          2     5     8    11    14    17    20    23    26
S4* = A A a G A t C C g G T t A G a C G g T A c T G c C A t

          3     6     9    12    15    18    21    24    27
S4  = A A A G A T C C G G T T A G A C G G T A C T G C C A T
```

selected, one from each edge list except the t list, as shown in Figure A-7. The 8 vertices of the last edge graph having these 7 doublets as its edge set are connected to final vertex t, as shown at the top of Figure A-8. Thus a new edge ordering having these last edges will be Eulerian. One such edge ordering, $E(R_4)$ of Figure A-7, is randomly selected by completing the random permutation of each list of $E(R_1)$. This edge ordering corresponds to the Eulerian long walk at the bottom of Figure A-8. Then random DP permutation R_4 is expanded into sequence S_4* of Figure A-6 by replacing each UL doublet of R_4 by the next unused triplon from the correponding permuted triplon list. Finally, the 8 lower case letters are changed back into upper case letters to generate DNA sequence S_4, which is a random DtP permutation of DNA sequence S_1.

Significance of an interferon alignment. A practical application of these permutation algorithms is the significance of the similarity of two sequences from the human gene for beta-1 interferon (Erickson et al., 1984). The DNA sequence of this gene was determined by Ohno and Taniguchi (1981) and by Fiers et al. (1982). Using a substitution cost of 1 and an insertion/deletion cost of 2, the distance between sequence A (nucleotides 196-255) and sequence B (nucleotides 499-558) is 25. The alignment having this distance is shown in Figure A-9. Can this distance be explained by base composition alone? To answer this question, each sequence was permuted 1,000 times and 1,000 permutation distances were calculated. A histogram of these distances is shown at the top of Figure A-10. The distance of 25 is 6.0 standard deviations less than the mean permutation distance. If the distribution of permutation distances is assumed to be

Figure A-7. Edge orderings $E(R_1)$ and $E(R_4)$ and triplon lists for S_1^* and S_4^*.

Edge list	$E(R_1)$	$E(R_4)$	Triplon list	S_1^*	S_4^*
	1 5	1 9		1 3	1 5
A:	Aa Aa	Aa Aa	AXa:	AGa AAa	AAa AGa
	3 9 15	5 11 17		5 8	3 6
C:	Ct Cg Cg	Cg Cg Ct	CXg:	CCg CGg	CCg CGg
	7 17	3 7		2	9
G:	Gt Gt	Gt Gt	CXt:	CAt	CAt
	11 13	13 15		4 9	2 4
T:	Tc Tc	Tc Tc	GXt:	GTt GAt	GAt GTt
	2 6	2 10		6 7	7 8
a:	aC aG	aG aC	TXc:	TAc TGc	TAc TGc
	12 14	14 16			
c:	cT cC	cT cC			
	10 16	6 12			
g:	gT gG	gG gT			
	4 8	4 8			
t:	tA tC	tC tA			

Figure A-8. Last edge graph $Z(R_4)$ and the Eulerian long walk in $G(R_4)$ determined by $E(R_4)$.

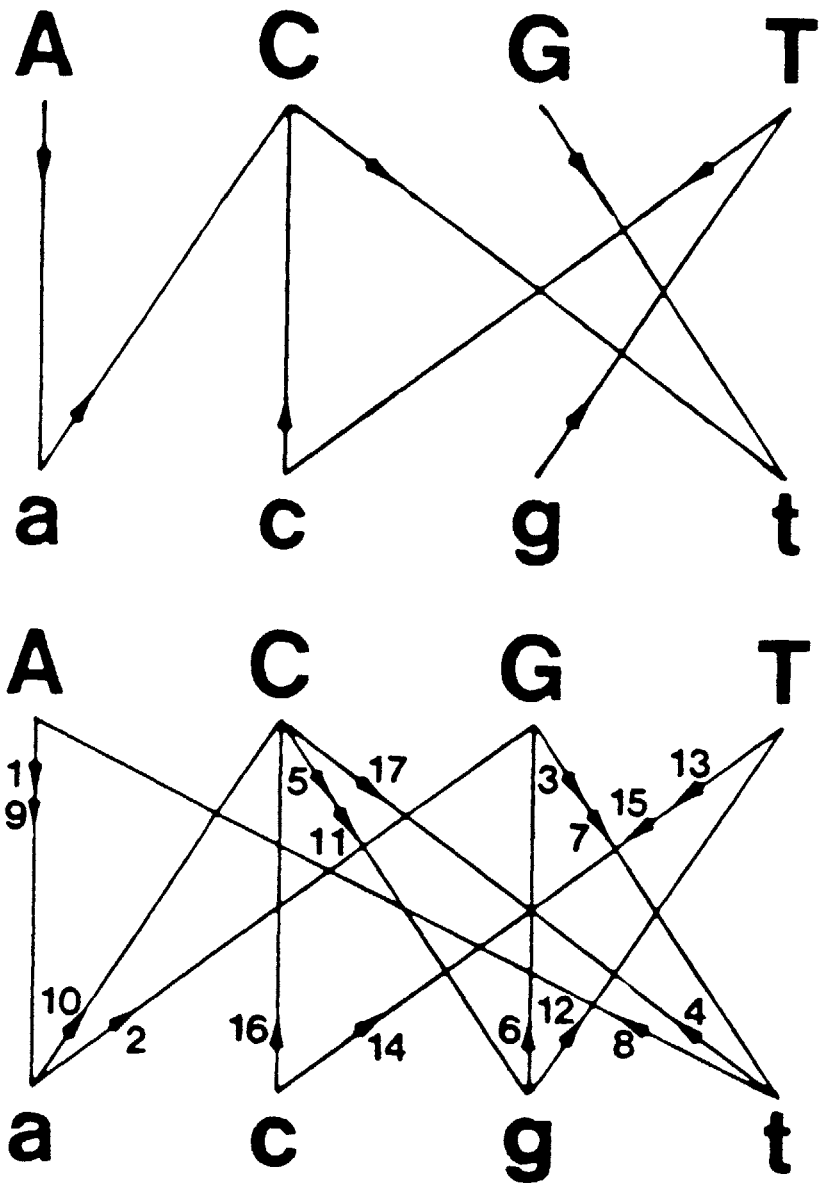


Figure A-9. Alignment of sequences A and B from the human beta-1 interferon gene. The center line echoes the 35 nucleotide identities.

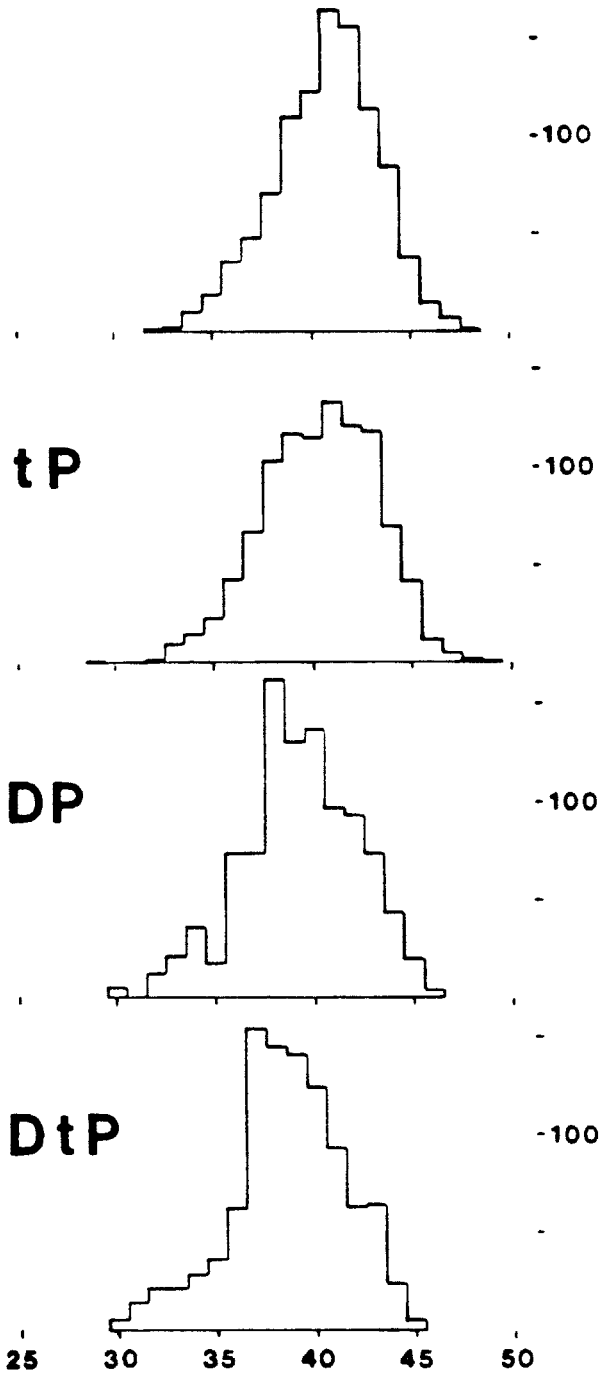
```
196 200       210       220       230       240       250  255
  CTCCTGTGGCAATTGAATGGGAGGCTTGAATACTGCCTCAAGGACAGGATGAACTTTGAC
  C CCTG     AT   ATGGGAGG TT    A T CCT AAGG CA G  G AC  T AC
  CACCTGAAAAGATATTATGGGAGGATTCTGCATTACCTGAAGGCCAAGGAGTACAGTCAC
499        510       520       530       540       550     558
```

Gaussian, this is a highly significant result. In fact, the distribution is somewhat non-Gaussian. If one wishes to make no assumptions about tail behavior, order statistics can be used to assess the significance of the result. If N permutation distances are calculated and M of the N are as small or smaller than the evolutionary distance in question, then the probability that the distance is due to chance is (M+1)/(N+1). In the present case, order statistics allow one to reject with 99.9% confidence the null hypothesis that the evolutionary distance between A and B is explainable by the similarity of their base compositions alone.

The algorithms described above were used to permute DNA sequences A and B while preserving their codon usage, dinucleotide usage or both simultaneously. Histograms representing 1,000 tP, DP and DtP permutation distances are shown in Figure A-10. The evolutionary distance of 25 is respectively 5.3, 4.9 and 4.7 standard deviations less than the mean permutation distance. The DP and DtP distributions are definitely non-Gaussian, so it is best to use order statistics when assessing the significance of the evolutionary distance. The hypothesis that the distance between A and B can be explained by the similarity of their dinucleotide and codon usage alone can be rejected with 99.9% confidence.

In this example, the small distance between the beta-1 interferon sequences A and B is better explained by the order of their nucleotides (Erickson et al., 1984). In other cases, the dinucleotide and codon usage may be sufficient to explain an evolutionary distance that would seem significant if only base composition were considered. The permutation algorithms described in this appendix are useful in recognizing such cases.

Figure A-10. Four histograms of 1,000 permutation distances for sequences A and B from the human beta-1 interferon gene. Top, simple permutations; tP, triplon preserving permutations; DP, doublet preserving permutations; DtP, doublet and triplon preserving permutations.



LITERATURE

Altschul, S. F. 1987. "Optimal Sequence Alignment Using Concave Gap Costs." J. Theor. Biol., submitted.

Altschul, S. F. and B. W. Erickson. 1985. "Significance of Nucleotide Sequence Alignments: A Method for Random Sequence Permutation That Preserves Dinucleotide and Codon Usage." Mol. Biol. Evol. 2, 526-538.

-----. 1986a. "Optimal Sequence Alignment Using Affine Gap Costs." Bull. Math. Biol., in press.

-----. 1986b. "A Nonlinear Measure of Subalignment Similarity and its Significance Levels." Bull. Math. Biol., in press.

-----. 1986c. "Locally Optimal Subalignments Using Nonlinear Similarity Functions." Bull. Math. Biol., in press.

Arratia, R., L. Gordon and M. S. Waterman. 1986. "An Extreme Value Theory for Sequence Matching." Ann. Stat. 14, 971-993.

Arratia, R. and M. S. Waterman. 1985. "Critical Phenomena in Sequence Matching." Ann. Prob. 13, 1236-1249.

Bond, M. W., N. Y. Chiu and B. S. Cooperman. 1980. "Identification of an Arginine Important for Enzymatic Activity within the Covalent Structure of Yeast Inorganic Pyrophosphatase." Biochemistry 19, 94-102.

Cohen, S. A. 1978. Ph.D. Thesis, University of Chicago.

Cohen, S. A., R. Sterner, P. S. Kein and R. L. Heinrikson. 1978. "Covalent Structural-analysis of Yeast Inorganic Pyrophosphatase." J. Biol. Chem. 253, 889-897.

Dayhoff, M. O., R. M. Schwartz and B. C. Orcutt. 1978. "A Model of Evolutionary Change in Proteins." In Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3, Ed. M. O. Dayhoff, pp. 345-352, Natl. Biomed. Res. Found., Washington.

Erickson, B. W., L. T. May and P. B. Sehgal. 1984. "Internal Duplication in Human Alpha-1 and Beta-1 Interferon." Proc. Natl. Acad. Sci. USA 81, 7171-7175.

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

Fiers, W., E. Remaut, R. Devos, H. Cheroutre, R. Contreras, D. Gheyson, W. Degrave, P. Stanssens, J. Tavernier, Y. Taya, and J. Content. 1982. "The Human Fibroblast and Human Immune Interferon Genes and Their Expression in Homologous and Heterologous Cells." Phil. Trans. R. Soc. Lond. B299, 29-38.

Fitch, W. M. 1983a. "Random Sequences." J. Mol. Biol. 163, 171-176.

-----. 1983b. "Calculating the Expected Frequencies of Potential Secondary Structure in Nucleic Acids as a Function of Stem Length, Loop Size, Base Composition and Nearest-neighbor Frequencies." Nucl. Acids Res. 11, 4655-4663.

Fitch, W. M. and T. F. Smith. 1983. "Optimal Sequence Alignments." Proc. Natl. Acad. Sci. USA 80, 1382-1386.

Fujita, T., C. Takaoka, H. Matsui and T. Taniguchi. 1983. "Structure of the Human Interleukin 2 Gene." Proc. Natl. Acad. Sci. U.S.A. 80, 7437-7441.

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

Gonzalez, M. A. and B. S. Cooperman. 1986. "Glutamic Acid-149 is Important for Enzymatic Activity of Yeast Inorganic Pyrophosphatase." Biochemistry 25, 7179-7185.

Gordon, L., M. F. Schilling and M. S. Waterman. 1986. "An Extreme Value Theory for Long Head Runs." Prob. Th. Rel. 72, 279-287.

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

Gumbel, E. J. 1962. "Statistical Theory of Extreme Values (Main Results)." In Contributions to Order Statistics, Ed. A. E. Sarhan and B. G. Greenberg, pp. 56-93, Wiley, New York, NY.

Karlin, S. and G. Ghandour. 1985. "Comparative Statistics for DNA and Protein Sequences: Single Sequence Analysis." Proc. Natl. Acad. Sci. USA 82, 5800-5804.

Kasteleyn, P. W. 1967. "Graph Theory and Crystal Physics." In Graph Theory and Theoretical Physics, Ed. F. Harary, pp. 43-110 Academic Press, London.

Knuth, D. E. 1973. Fundamental Algorithms. Addison-Wesley, Reading, MA.

Kruskal, J. B. 1983. "An Overview of Sequence Comparison." In Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, D. Sankoff and J. B. Kruskal, Ed., Reading, MA: Addison-Wesley, pp. 1-44.

Kuranova, I. P., S. S. Terzyan, A. A. Voronova, E. A. Smirnova, B. K. Vainstein, W. Hohne and G. Hansen. 1983. "Active-site Structure of the Inorganic Pyrophosphatase from Baker Yeast Based on the X-ray-investigation." Bioorg. Khim. 9, 1611-1619.

Larsen, R. J. and M. L. Marx. 1981. An Introduction to Mathematical Statistics and its Applications. Prentice-Hall, Englewood Cliffs, NJ.

Lipman, D. J., W. J. Wilbur, T. F. Smith and M. S. Waterman. 1984. "On the Statistical Significance of Nucleic-acid Similarities." Nucl. Acids Res. 12, 215-226.

Mood, A. M., F. A. Graybill and D. C. Boes. 1974. Introduction to the Theory of Statistics (Third Edition). McGraw-Hill, New York, NY.

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, 443-453.

Ohno, S. and T. Taniguchi. 1981. "Structure of a Chromosomal Gene for Human Interferon Beta." Proc. Natl. Acad. Sci. USA 78, 5305-5309.

Reich, J. G., H. Drabsch and A. Daumler. 1984. "On the Statistical Assessment of Similarities in DNA Sequences." Nucl. Acids Res. 12, 5529-5543.

Sankoff, D. 1972. "Matching Sequences under Deletion/Insertion Constraints." Proc. Natl. Acad. Sci. USA 69, 4-6.

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

Schwartz, R. M. and M. O. Dayhoff. 1978. "Matrices for Detecting Distant Relationships." In Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3, Ed. M. O. Dayhoff, pp. 353-358, Natl. Biomed. Res. Found., Washington.

Sellers, P. H. 1974a. "On the Theory and Computation of Evolutionary Distances." SIAM J. Appl. Math. 26, 787-793.

-----. 1974b. "An Algorithm for the Distance Between Two Finite Sequences." J. Comb. Th. A16, 253-258.

-----. 1984. "Pattern Recognition in Genetic Sequences by Mismatch Density." Bull. Math. Biol. 46, 501-514.

-----. 1986. Lecture presented at the 827th Meeting of the American Mathematical Society, Baltimore, May 1986.

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

-----. 1981b. "Comparison of Biosequences." Adv. Appl. Math. 2, 482-489.

Smith, T. F., M. S. Waterman and C. Burks. 1985. "The Statistical Distribution of Nucleic Acid Similarities." Nucl. Acids Res. 13, 645-656.

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

Smith, T. F., M. S. Waterman and J. R. Sadler. 1983. "Statistical Characterization of Nucleic Acid Sequence Functional Domains." Nucl. Acids Res. 11, 2205-2220.

Swartz, M. N., T. A. Trautner and A. Kornberg. 1962. "Enzymatic Synthesis of Deoxyribonucleic Acid. XI. Further Studies on Nearest Neighbor Base Sequences in Deoxyribonucleic Acids." J. Biol. Chem. 237, 1961-1967.

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

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

Terzyan, S. S., A. A. Voronova, E. A. Smirnova, I. P. Kuranova, Y. V. Nekrasov, E. G. Harutyunyan, B. K. Vainstein, W. Hohne and G. Hansen. 1984. "The Spatial Structure of Inorganic Pyrophosphatase from Yeast 3-A Resolution." Bioorg. Khim. 10, 1469-1482.

Ukkonen, E. 1983. "On Approximate String Matching." Proc. Int. Conf. Comp. Theor. Lecture Notes in Comp. Sci. 158, 487-496.

van Aardenne-Ehrenfest, T. and N. G. de Bruijn. 1951. "Circuits and Trees in Oriented Linear Graphs." Simon Stevin 28, 203-217.

Waterman, M. S. 1984a. "General Methods of Sequence Comparison." Bull. Math. Biol. 46, 473-500.

-----. 1984b. "Efficient Sequence Alignment Algorithms." J. Theor. Biol. 108, 333-337.

Waterman, M. S., T. F. Smith and W. A. Beyer. 1976. "Some Biological Sequence Metrics." Adv. Math. 20, 367-387.

Yokota, T., N. Arai, F. Lee, D. Rennick, T. Mosmann and K. Arai. 1985. "Use of a cDNA Expression Vector for Isolation of Mouse Interleukin 2 cDNA Clones: Expression of T-cell Growth-factor Activity After Transfection of Monkey Cells." Proc. Natl. Acad. Sci. USA 82, 68-72.

Zaman, A. 1984. "Urn Models for Markov Exchangeability." Ann. Prob. 11, 223-229.