SIAM J. APPL. MATH.
Vol. 49, No. 1, pp. 197-209, February 1989

TREES, STARS, AND MULTIPLE BIOLOGICAL SEQUENCE ALIGNMENT*

STEPHEN F. ALTSCHUL† AND DAVID J. LIPMAN†

Abstract. One important problem in biological sequence comparison is how to simultaneously align several nucleic acid or protein sequences. A multiple alignment avoids possible inconsistencies among several pairwise alignments and can elucidate relationships not evident from pairwise comparisons. The basic dynamic programming algorithm for optimal multiple sequence alignment requires too much time to be practical for more than three sequences, the length of an average protein. Recently, Carrillo and Lipman (*SIAM J. Appl. Math.*, 48 (1988), pp. 1073-1082) have rendered feasible the optimal simultaneous alignment of as many as six sequences by showing that a consideration of minimal pairwise alignment costs can vastly decrease the number of cells a dynamic programming algorithm need consider. Their argument, however, requires the cost of a multiple alignment to be a weighted sum of the costs of its projected pairwise alignments. This paper presents an extension of Carrillo and Lipman's algorithm to the definition of multiple alignment cost as the cost of an evolutionary tree. An interesting generalization of the linear programming problem arises from the analysis.

Key words. sequence alignment, biological sequences, dynamic programming, linear programming

AMS(MOS) subject classifications. 49, 68, 92

1. Introduction. The comparison of nucleic acid and protein sequences is an important tool in molecular biology. It has been used to study molecular evolution (Dayhoff [9]), RNA folding (Trifonov and Bolshoi [29]), gene regulation (Galas et al. [14]), and protein structure-function relationships (Wu and Kabat [34], Kabat and Wu [17]). The dynamic programming method was introduced to biological sequence comparison by Needleman and Wunsch [22]. Subsequent work on the two-sequence alignment problem has included: the proof that certain distance measures can act as a metric on the space of sequences (Sellers [25]); the construction of measures appropriate for protein sequence comparison (Dayhoff et al. [10]); improved algorithms for pairwise comparisons (Ukkonen [30], Fickett [12]); the description of algorithms that can treat gaps in a manner consistent with biological intuition (Waterman et al. [32], Gotoh [15], Fitch and Smith [13], Altschul and Erickson [3], Miller and Myers [20]); the development of methods for finding optimal subalignments (Smith and Waterman [27], Sellers [26], Altschul and Erickson [4], [5]); and the rapid search of existing data banks for sequences similar to a query sequence (Wilbur and Lipman [33]).

Many of the methods developed for two-sequence comparison can be extended to three or more sequences (Sankoff [23], Sankoff and Cedergren [24], Murata et al. [21]). Some novel issues arise for multiple-sequence comparison. One of these is how to extend the definition of cost from pairwise to multiple-sequence alignments. The simplest method is to define the cost of a multiple alignment to be the sum of the costs of all the implied pairwise alignments (Murata et al. [21], Bacon and Anderson [6]). A method that fits biological intuition more closely is treating the input sequences as the leaves of an evolutionary tree and reconstructing ancestral sequences that minimize the cost of the tree (Sankoff [23], Sankoff and Cedergren [24]). For more than three sequences these methods have been primarily of theoretical interest because of the excessive computation time they require for sequences of reasonable length. As a result

* Received by the editors November 2, 1987; accepted for publication (in revised form) March 23, 1988.

† Mathematical Research Branch, National Institute of Diabetes and Digestive and Kidney Diseases, National Institutes of Health, Building 31, Room 4B-54, Bethesda, Maryland 20892.

programs for multiple-sequence alignment generally use heuristics or minimize alignment costs that are not clearly tied to models of molecular evolution (Johnson and Doolittle [16], Sobel and Martinez [28], Waterman [31]).

Carrillo and Lipman [7] recently described an algorithm that reduces the computation time for optimal multiple-sequence alignment to such an extent that it becomes possible to align up to six sequences of reasonable length (approximately 200). Their method, however, makes explicit use of the definition of multiple alignment cost as the sum of the costs of the implied pairwise alignments. This paper extends Carrillo and Lipman's algorithm [7] to multiple alignments whose cost is defined as that of an evolutionary tree (Sankoff [23]).

One problem that arises in extending Carrillo and Lipman's algorithm [7] closely resembles the classic linear programming problem (Dantzig [8]). The main difference is that the objective function of linear programming implies hyperplanes of constant value arranged linearly in hyperspace. The objective function of the variant problem implies hyperplanes of constant value that rotate about a hyperline. The simplex algorithm for linear programming (Dantzig [8]) is easily modified to solve this new problem.

A special case of the Sankoff cost for multiple alignments arises when the evolutionary tree assumed is a star. In other words, all the observed sequences are assumed to have arisen by separate lineages from a single ancestral sequence. For this case, further analysis allows our extension of Carrillo and Lipman's algorithm [7] to avoid using the simplex algorithm.

2. Definitions.

A *null* is the symbol "-."

An *alphabet* Σ is a finite set of symbols containing null.

An *element* is a member of the alphabet Σ.

A *letter* is an element other than null.

A *sequence* is a finite string of letters.

A *padded sequence* is a finite string of elements.

A *pseudo-alignment* of the sequences $S_1, \cdots, S_n$ is an ordered set of padded sequences $(S'_1, \cdots, S'_n)$ of equal length such that removing the nulls from any S'_i leaves the sequence S_i.

The jth *column* of a pseudo-alignment α is the ordered set consisting of the jth element from each padded sequence of α.

An *alignment* is a pseudo-alignment all of whose columns contain a letter.

A *pairwise alignment* is an alignment of two sequences.

The *projection* of alignment $(S'_1, \cdots, S'_n)$ onto sequences $S_{x_1}, \cdots, S_{x_i}$ is the alignment obtained from pseudo-alignment $(S'_{x_1}, \cdots, S'_{x_i})$ by removing all columns that do not contain a letter.

3. The cost of an alignment. To formalize the concept of the quality of an alignment, we need a *cost function* defined on alignments. There are an infinite variety of possible functions, but in general two considerations restrict the reasonable choices. The first is biological: does the cost function correspond to ideas of biological relatedness? The second is algorithmic: can an *optimal* (minimal cost) alignment be found in reasonable time? Sometimes there is tension between these considerations, and an appropriate compromise must be found.

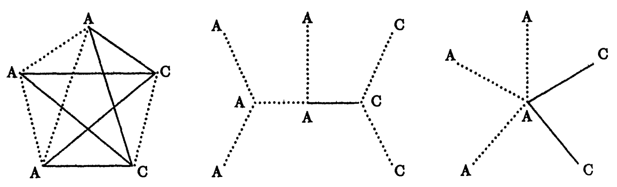
Because genetic mutations are binary events, changing one nucleic acid or protein sequence into a second, the cost of a multiple alignment is often defined in terms of the cost of pairwise alignments. Most approaches start with a *substitution cost function* that specifies the cost of aligning one element with another (Needleman and Wunsch

[22], Sellers [25], Dayhoff et al. [10]). Occasionally additional costs are assigned to *gaps*, which are maximal strings of adjacent letters in one sequence aligned with nulls in the other. The cost of a pairwise alignment is then defined to be the sum of the costs for each pair of aligned elements and the costs for any gaps. (Note that as defined here gap costs are assessed separately from, and in addition to, the costs of aligning individual letters with nulls.) The *distance* between two sequences will denote the mimimum cost for aligning the sequences. Only for some substitution and gap cost functions does the implied distance act as a metric on the space of sequences (Sellers [25]).

Cost may be extended to multiple alignments in several ways. Since a multiple alignment has a projection onto every pair of sequences, its cost may be taken to be the sum of the costs of these alignments. Carrillo and Lipman [7] define the cost of a multiple alignment to be the sum of the costs of all projected pairwise alignments, but allow the cost of each projection to be calculated differently. In particular, they allow each pairwise projection to be given a different weight. We will call an alignment whose cost is so defined an SP-*alignment*, where SP is short for "Sum-of-the-Pairs."

Alternately, and perhaps in closer agreement with biological intuition, an evolutionary tree may be assumed, with the *input sequences* assigned to the leaves. Additional *reconstructed sequences*, corresponding to the internal nodes of the tree, are added to the multiple alignment. The cost of this extended alignment is defined to be the sum of the projected pairwise alignment cost associated with each edge of the tree (Sankoff [23], Sankoff and Cedergren [24]). We will call an extended multiple alignment whose cost is so defined a *tree-alignment.* An algorithm for finding an optimal tree-alignment must assign sequences to the internal nodes of the tree, as well as align the given and reconstructed sequences (Sankoff [23], Sankoff and Cedergren [24]).

A *star-alignment* will denote the special case in which the tree has only one internal node. This is a reasonable model for the evolutionary history of some input sequences. For example, it is appropriate when comparing homologous proteins from a primate, a carnivore, an insectivore, and an ungulate, because all of these diverged at approximately the same time from a common ancestor. Furthermore, Sankoff's algorithm [23] is always simplified by assuming a star, especially when it is generalized to allow gap costs (Altschul [2]). Examples of SP-, tree-, and star-alignments of the same five one-letter sequences are given in Fig. 1.



	SP-alignment	tree-alignment	star-alignment
input sequences	A A A C C	A A A C C	A A A C C
reconstructed sequences		A A C	A
cost	6	1	2

FIG. 1. SP-, *tree-, and star-alignments for five one-letter input sequences. Pairwise alignments with cost one are indicated by solid lines, and pairwise alignments with cost zero by dotted lines.*

Even ignoring reconstructed sequences, different definitions of multiple alignment cost can yield different optimal alignments. For example, if aligning two different letters has cost 1 and aligning a letter with a null has cost 2, the optimal SP-alignment of the four sequences ACC, ACC, TCT, and ATCT is different than the optimal star-alignment, as shown below:

input sequences	–ACC	ACC–
	–ACC	ACC–
	–TCT	TCT–
	ATCT	ATCT
		. . .
reconstructed sequence		ACC–
	optimal SP-alignment	optimal star-alignment

Basically, optimal tree-alignments minimize the total number of mutations from postulated common ancestors, while optimal SP-alignments tend to maximize the number of positions at which all, or nearly all, of the aligned sequences agree. Because in proteins certain regions are generally more mutable than others, optimal SP-alignments may sometimes make more biological sense than optimal tree-alignments. Felsenstein [11] provides an excellent discussion of related issues in phylogenetic tree reconstruction. The extension of Carrillo and Lipman's algorithm [7] to tree- and star-alignments presented in this paper permits many sequences to be simultaneously aligned using whichever definition of alignment cost appears most appropriate.

4. Review of Carrillo and Lipman's algorithm. The central idea of Carrillo and Lipman's algorithm [7] is that the cost of the projection of an optimal multiple alignment onto two of its sequences must be at least as great as the distance between these sequences. While this is a trivial observation, it may nevertheless be exploited to great advantage.

Given an SP-alignment α, let α_{ij} be its projection onto sequences S_i and S_j. Let c be the cost function for SP-alignments, and c_{ij} for pairwise alignments of S_i and S_j, so that

$$c(\alpha)=\sum_{i<j} c_{ij}(\alpha_{ij}). \tag{1}$$

Let β be an optimal SP-alignment, and let $C'\geqq c(\beta)$. Let C_{ij} be the distance between S_i and S_j. Then we have

$$C'\geqq c(\beta)=\sum_{i<j} c_{ij}(\beta_{ij})\geqq c_{xy}(\beta_{xy})-C_{xy}+\sum_{i<j} C_{ij}, \tag{2}$$

which implies

$$c_{xy}(\beta_{xy})\leqq C_{xy}+\left(C'-\sum_{i<j} C_{ij}\right). \tag{3}$$

The C_{ij} may be calculated quickly by standard algorithms for finding optimal pairwise alignments. Using any number C' at least as great as the optimal multiple alignment cost $c(\beta)$, (3) gives bounds on the cost of the pairwise projections of an optimal alignment. An appropriate C' may either be taken as the cost of an actual SP-alignment (which must be $\geqq c(\beta)$), or simply guessed.

Treating an n-sequence alignment as a path through a lattice in n-space, the projection of an optimal alignment onto two sequences is simply a path in a standard two-dimensional path graph (Carrillo and Lipman [7]). A bound on the cost of such a projection limits the points in the path graph through which the projection can possibly pass. This in turn limits the points in the original lattice through which the optimal alignment can pass. Each projection thus defines a subset of the original lattice that contains the paths of all optimal alignments. The intersection of these subsets still contains all such paths. It is only this intersection, therefore, that must be considered by a dynamic programming algorithm seeking optimal alignments. In practice, this approach so limits the number of lattice points that it becomes possible to find optimal multiple alignments for as many as six sequences.

5. Tree-alignments.

Carrillo and Lipman's algorithm [7] for finding optimal SP-alignments divides into three parts: (i) finding an upper bound on the cost of each projection of an optimal alignment; (ii) using these bounds to reduce the size of the dynamic programming lattice; (iii) finding an optimal alignment within the reduced lattice. For tree-alignments, part (ii) of their algorithm is unchanged, and in part (iii) Sankoff's algorithm [23] can be applied to the reduced lattice. However, because (1) no longer applies, we need a more sophisticated approach to finding the upper bounds in part (i).

The cost of an optimal tree-alignment β is the sum of the pairwise alignment costs associated with all edges of the tree. The cost of β's projection onto a given pair of input sequences is bounded above by the cost of the edges connecting these sequences (Fig. 2), i.e., by the cost of β minus the cost of any unused edges. While each pairwise distance can provide us with a lower bound on the cost of a subset of the tree's edges, these subsets in general share edges. Therefore, the problem is to subtract various pairwise distances from the cost of β in such a way that we are left with a good bound on the cost of a specific projection of β. To make these ideas precise we introduce some new notation.

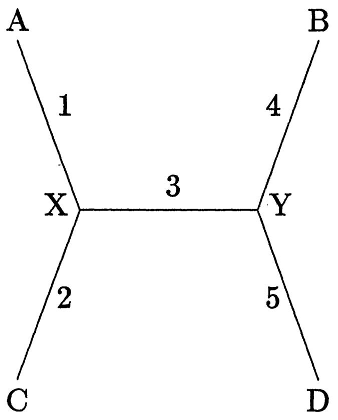


FIG. 2. *A tree-alignment of input sequences A, B, C, and D, including reconstructed sequences X and Y. Input sequences A and B are connected in the tree through edges* 1, 3, *and* 4. *Therefore* $\delta(AB, 1) = \delta(AB, 3) = \delta(AB, 4) = 1$, *but* $\delta(AB, 2) = \delta(AB, 5) = 0$.

Let P be the set of pairs of sequences chosen from the sequences to be aligned. In other words, if the input sequences are $S_1, \cdots, S_n$,

$$P = \{(S_i, S_j): 0 < i < j \leqq n\}. \tag{4}$$

Let E be the set of edges of the imposed evolutionary tree. Define the function $\delta: P \times E \to \{0, 1\}$ by

$$\delta(p, e) = \begin{cases} 1 & \text{if the path between the sequences of pair } p \\ & \text{passes through edge } e \text{ of the tree,} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Figure 2 illustrates the function $\delta(p, e)$.

As described above, a tree-alignment includes a padded sequence that corresponds to each of the input sequences, and also one that corresponds to each of the internal nodes. There exist projections of the tree-alignment onto any pair of these sequences. We will be interested only in certain of these projections. Specifically, given a tree-alignment α, and a pair $p \in P$, let α_p be the projection of α onto the sequences that constitute p. Also, for each edge $e \in E$, let α_e be the projection of α onto the sequences associated with the nodes at either end of edge e.

To simplify our discussion, we assume that all pairwise alignment costs are calculated identically, using the pairwise cost function c_2. Furthermore, we assume that c_2 consists of a nonnegative (possibly zero) cost for each gap and a symmetric substitution cost function f that satisfies the triangle inequality. In other words, if $f(x, y)$ is the cost of aligning elements x and y, then for all elements x, y, and z

$$f(x, y) + f(y, z) \geqq f(x, z). \tag{6}$$

Several of these conditions can be relaxed, as will be discussed in the section on generalizations.

For each $p \in P$, let C_p be the cost of an optimal alignment of the sequences of pair p, i.e., the distance between these sequences. We observe that our assumptions about c_2 imply that given any tree-alignment α, for each $p \in P$,

$$C_p \leqq c_2(\alpha_p) \leqq \sum_{e \in E} c_2(\alpha_e)\delta(p, e). \tag{7}$$

In other words, the cost of the projection of alignment α onto pair p is less than or equal to sum of the costs of the projections of α onto each of the edges of the path connecting the sequences of pair p in the tree. It is also, by definition, greater than or equal to the distance between the sequences of pair p.

Let c be the cost function for tree-alignments. By the definition of tree-alignments,

$$c(\alpha) = \sum_{e \in E} c_2(\alpha_e). \tag{8}$$

Let β be an optimal tree-alignment of sequences $S_1, \cdots, S_n$, and let $C' \geqq c(\beta)$. We then have

$$C' \geqq c(\beta) = \sum_{e \in E} c_2(\beta_e). \tag{9}$$

We seek an upper bound on $c_2(\beta_q)$, the cost of the projection of β onto a given pair q of input sequences. As before, we can find such a bound by subtracting pairwise distances C_p from C', but now we need to ensure that the cost associated with any edge of the imposed tree is implicitly counted no more than once. We thus proceed by giving a nonnegative weight x_p to each pair of input sequences, insisting that a positive weight be given to pair q. In addition we require that for each edge e of the imposed tree, the weights of all pairs of input sequences connected in the tree through

e sum to no more than 1. Formally, we choose any set of x_p for $p \in P$ satisfying the inequalities

$$x_q > 0, \tag{10}$$

$$\text{for all } p \in P \quad x_p \geqq 0, \tag{11}$$

$$\text{for all } e \in E \quad \sum_{p \in P} x_p \delta(p, e) \leqq 1. \tag{12}$$

Inequalities (9) and (12) then yield

$$C' \geqq \sum_{e \in E} c_2(\beta_e) \geqq \sum_{e \in E} \left[c_2(\beta_e) \sum_{p \in P} x_p \delta(p, e) \right] = \sum_{p \in P} \left[x_p \sum_{e \in E} c_2(\beta_e) \delta(p, e) \right]. \tag{13}$$

Combining (7) and (13), we obtain

$$C' \geqq c_2(\beta_q) x_q + \sum_{p \neq q} C_p x_p. \tag{14}$$

Basically, $c(\beta)$ in (9) has been replaced by a lower bound on the sum of the cost of its edges. This bound has been written in terms of the cost of projection β_q and of the distances C_p. Rewriting inequality (14), we get

$$c_2(\beta_q) \leqq \left(C' - \sum_{p \neq q} C_p x_p \right) \Big/ x_q, \tag{15}$$

which is the bound we sought.

As before, the C_p can be calculated using standard algorithms for optimal pairwise alignment, and C' may be chosen as the cost of any tree-alignment of $S_1, \cdots, S_n$, or guessed. Any set of x_p satisfying (10)-(12) then yields an upper bound on $c_2(\beta_q)$, which can be used to eliminate cells from the dynamic programming lattice. The smaller the bound, the more cells that can be eliminated. We thus consider which set of x_p optimizes the bound of (15).

6. Circular programming. The problem is to choose x_p satisfying inequalities (10)-(12) that minimize

$$\left(C' - \sum_{p \neq q} C_p x_p \right) \Big/ x_q. \tag{16}$$

This problem is the same as the classic linear programming problem (Dantzig [8]), with the following two differences: x_q is required to be positive, and the *objective function* (16) is not a linear function of the x_p. However, with the proviso that $x_q > 0$, the *feasible solutions* are still the interior and boundaries of a convex polyhedron in the hyperspace whose axes are the x_p. Still insisting that $x_q \neq 0$, the points in hyperspace where the objective function has constant value k are given by

$$k x_q + \sum_{p \neq q} C_p x_p = C'. \tag{17}$$

This is a hyperplane that intersects the hyperplane $x_q = 0$ at the hyperline

$$x_q = 0, \qquad \sum_{p \neq q} C_p x_p = C'. \tag{18}$$

Changing the value of k has the effect of rotating the hyperplane (17) about the hyperline (18) (Fig. 3). By an argument similar to that used for linear programming (Dantzig [8]), an optimal feasible solution must exist at one of the vertices of the polyhedron that bounds the feasible region. Moving along an edge of the polyhedron,

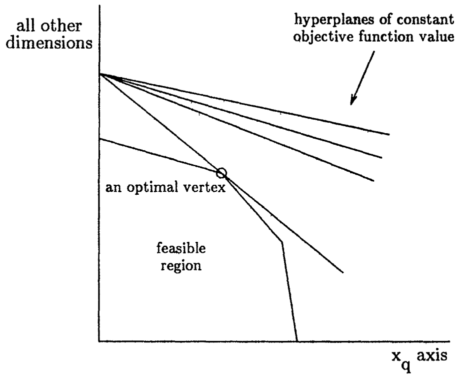


FIG. 3. *A circular programming problem. All hyperplanes of constant objective function value meet at a hyperline. There is always an optimal solution at a vertex of the feasible region.*

the objective function always increases, decreases, or remains unchanged. Therefore, with suitable minor modifications, the simplex algorithm for linear programming (Dantzig [8]) solves our minimization problem. An interesting question is whether other algorithms for linear programming, such as Khachian's algorithm [19] or Karmarkar's algorithm [18], can be generalized to solve the optimization problem studied in this section.

7. Time complexity. The time required for the complete algorithm to find optimal tree-alignments is the sum of the times required (i) to find bounds on the projection costs of an optimal alignment; (ii) to calculate the lattice these bounds imply; (iii) to execute Sankoff's algorithm [23] on this reduced lattice. The better the bounds found in (i) the less time required by (ii) and (iii). One question is whether the time spent minimizing (16) in part (i) is justified later by the time saved in parts (ii) and (iii). It is possible there is a heuristic for choosing the x_p that obviates solving the circular programming problem but still places a good upper bound on the cost of β_q.

In general, however, the time required for parts (ii) and (iii) grows exponentially with the number of sequences considered, and improving the bound on the cost of a single β_q by even one can halve the number of points in the reduced lattice. In the cases of interest the circular programming problem for minimizing this bound is small. For example, with six sequences there are 15 variables and no more than nine constraints of form (12). Of course there are also 15 circular programming problems to solve, one for each pair of sequences. Nevertheless, if the average time complexity of the simplex algorithm for circular programming is similar to that for linear programming (Adler et al. [1]), the time spent calculating bounds on the cost of the β_p in part (i) should be small compared to the time required for parts (ii) and (iii).

8. Star-alignments. Since star-alignments are tree-alignments, the simplex algorithm applied to the circular programming problem described above will find a feasible vertex that minimizes the bound of formula (15). However, the special symmetry of a star allows a simple classification of all possible optimal vertices. For fewer than nine sequences, it may be easier simply to evaluate all the resulting bounds than to employ the machinery of the simplex algorithm.

LEMMA 1. *For a star-alignment, any vertex of the convex polyhedron defined by the equations* (11) *and* (12) *has all coordinates x_p equal to zero, $\frac{1}{2}$, or one.*

Outline of the proof. Any vertex of the polyhedron is the simultaneous solution of $|P|$ of the inequalities (11) and (12), converted to equalities. Any equations of the form (11) can be used to reduce the number of variables in any equations of the form (12). There then remain k equations in k unknowns. A given x_p occurs (with coefficient 1) in at most two of these equations because each pair of input sequences are connected by two edges of the star. It can then be shown that, eliminating all other variables by simple substitution, x_p can be written in the form $2x_p = i$, where i is an integer. x_p is thus half integral, and the inequalities (11) and (12) immediately imply that x_p has the value zero, $\frac{1}{2}$, or one.

Lemma 1 allows us to associate a graph $G(v)$ with each feasible vertex v. The nodes of the graph represent the input sequences $S_1, \cdots, S_n$, and the number of edges between a pair p of nodes is equal to twice x_p. By classifying the graphs that can possibly correspond to an optimal vertex, we limit the search for such a vertex to a specific set of points in hyperspace.

A graph represents a feasible point in hyperspace if and only if it has an edge between the nodes associated with the input sequences of pair q (i.e., $x_q > 0$), and has no more than two edges touching any node. While such a graph always represents a feasible point in hyperspace, it need not represent a feasible vertex.

Define a *cycle* of a graph to be a connected subgraph all of whose nodes touch exactly two edges (Fig. 4). The *size* of a cycle is the number of nodes it contains.

LEMMA 2. *The graph $G(v)$ associated with a feasible vertex v cannot contain an even cycle of size other than* 2.

Proof. Suppose $G(v)$ contains a cycle γ of even size greater than 2 (Fig. 5(a)). Consider the two different graphs obtainable from $G(v)$ by removing alternating edges of γ and doubling the edges of γ that remain (Fig. 5(b), (c)). These two graphs

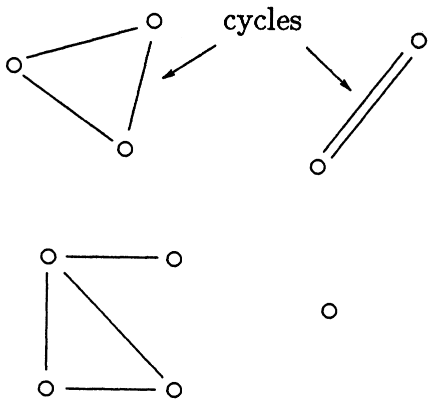


FIG. 4. *A graph on ten nodes. This graph has four connected components, two of which are cycles.*

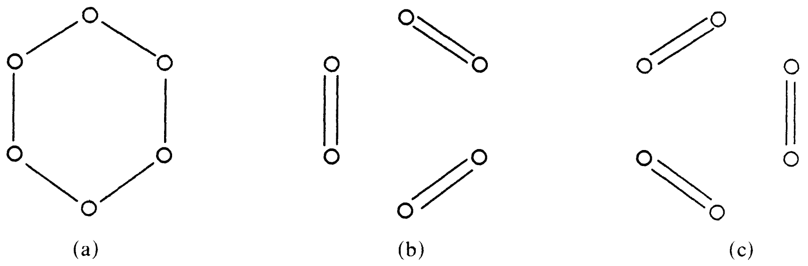


FIG. 5. (a) *A cycle γ of size six.* (b), (c) *The two different graphs obtainable from γ by removing alternating edges and doubling the edges that remain.*

correspond to different feasible points. v lies midway between these points and therefore cannot be a vertex of the feasible region.

LEMMA 3. *There exists an optimal vertex v such that at most one (isolated) node of $G(v)$ is not part of a cycle.*

Proof. Suppose the graph of a feasible vertex v has two nodes that touch fewer than two edges. The graph obtained by adding an edge between these nodes corresponds to a feasible point. Furthermore, the objective function at this point is at least as small as it is at v. Therefore there is an optimal vertex whose graph has at most one node that touches fewer than two edges. Since all other nodes of $G(v)$ touch exactly two edges, this node must be isolated. Thus at most one (isolated) node of $G(v)$ is not part of a cycle.

LEMMA 4. *For more than three sequences, there exists an optimal vertex whose graph consists only of cycles.*

Proof. Let v be an optimal vertex satisfying Lemma 3. Suppose N is an isolated node of $G(v)$. For more than three sequences, there exists a node M_1 not equal to N or the two nodes corresponding to the sequences of pair q. Also, there exists an edge of $G(v)$ between M_1 and some other node M_2 not equal to N. Consider the graph obtained from $G(v)$ by replacing this edge with an edge between N and M_1 and an edge between N and M_2. This graph corresponds to a feasible vertex. Because of the triangle inequality for pairwise distances implied by (6), the objective function at this point is at least as small as it is at v.

THEOREM 1. *In the feasible region corresponding to the inequalities* (10) *to* (12) *implied by a star, there always exists a vertex v minimizing* (16) *such that the following hold*:

(i) *The two nodes of $G(v)$ corresponding to the sequences of pair q are adjacent.*

(ii) *If the star has three edges, $G(v)$ is either a cycle of size three, or a cycle of size two and an isolated node.*

(iii) *If the star has more than three edges, $G(v)$ consists of cycles of size odd or equal to two.*

Proof. The theorem follows directly from observations above, and from Lemmas 2, 3, and 4.

For $n \leqq 10$ nodes with one distinguished pair, the number $f(n)$ of distinct graphs that satisfy (i)-(iii) of Theorem 1 are recorded in Table 1. Without further analysis, this many bounds must be computed in order to minimize (16). The problem may be reduced in some cases by examining the specific C_p. For n greater than eight, the

TABLE 1
The number of graphs on n nodes that satisfy Theorem 1(i)-(iii).

n	$f(n)$
3	2
4	1
5	10
6	7
7	217
8	277
9	9082
10	20905

simplex algorithm is preferable to the brute force examination of every graph described in Theorem 1. But of course for n this large the basic algorithm, of which calculating bounds on the projection costs is only a part, becomes impractical.

9. Generalizations. As discussed above, biological considerations frequently make it desirable to charge a cost for the existence of a gap in addition to that charged for aligning each letter of the gap with a null (Fitch and Smith [13]). It remains possible to define multiple alignment cost by formula (1) or (8), but if this is done the dynamic programming method for finding optimal alignments encounters difficulties (Altschul [2]). With a slight compromise, we can define a multiple alignment cost that avoids these difficulties and satisfies formula (1) or (8) in most cases. For *all* alignments the corresponding inequality applies (Altschul [2]):

$$c(\alpha) \geqq \sum_{i<j} c_{ij}(\alpha_{ij}), \tag{19}$$

$$c(\alpha) \geqq \sum_{e \in E} c_2(\alpha_e). \tag{20}$$

Carrillo and Lipman's argument [7] is still valid if (1) is replaced by (19), as is the argument in the tree-alignment section above if (8) is replaced by (20).

So far we have confined our attention to using pairwise distances to bound the projection costs of an optimal alignment. Examining three-sequence distances as well would likely lead to better bounds. A generalization of the argument in the tree-alignment section allows distances for more than two sequences to be taken into account as well. Of course this requires that such distances be calculated, which may not be justified by the time subsequently saved. The best strategy may be to use only certain of the three-sequence distances. Alternately some sort of recursive method might prove effective, using pairwise distances to find three-sequence distances, these to find four-sequence distances, etc. Such improvements on the basic approach remain to be investigated.

It is possible to allow each edge of a tree alignment to have a different weight or to use a different cost function, so that (8) becomes

$$c(\alpha) = \sum_{e \in E} c_e(\alpha_e). \tag{21}$$

If this is done, it is necessary to redefine the projection of an alignment onto two input sequences to include the reconstructed sequences at all intermediate nodes of the tree. For instance, in Fig. 2, a projection of a multiple alignment onto input sequences A and B would include the reconstructed sequences X and Y. We need to place bounds on the cost of such projections. The C_p required to do this are no longer simple pairwise distances: C_x is now the cost of an optimal tree-alignment whose tree is derived from the original tree by removing all edges e for which $\delta(x, e) = 0$.

Even if each edge uses the same pairwise cost function c_2, it can be advantageous to use the foregoing definition of a pairwise projection. Specifically, if the cost of aligning a sequence with itself can be greater than zero, as with the PAM-250 matrix (Dayhoff et al. [10]), this definition of the distance between input sequences yields better bounds. Furthermore, it is possible to include in the analysis distances between input sequences and unknown internal nodes, which may be nonzero.

10. Conclusion. Information derived from a multiple-sequence alignment can be useful in a number of molecular biology applications, including protein engineering and the study of molecular evolution. The computational requirements of the dynamic programming algorithm, the preferred method for pairwise comparison, has until recently [7] precluded its use for the comparison of more than three sequences. Although the SP-alignment cost that Carrillo and Lipman [7] employ is useful in certain applications, our extension of their results to tree-alignments makes this more biologically realistic measure available as well.

REFERENCES

[1] I. Adler, N. Megiddo, and M. J. Todd, *New results on the average behavior of simplex algorithms*, Bull. Amer. Math. Soc. (N.S.), 11 (1984), pp. 378-382.

[2] S. F. Altschul, *Gap costs for multiple sequence alignment*, submitted for publication.

[3] S. F. Altschul and B. W. Erickson, *Optimal sequence alignment using affine gap costs*, Bull. Math. Biol., 48 (1986), pp. 603-616.

[4] ———, *A nonlinear measure of subalignment similarity and its significance levels*, Bull. Math. Biol., 48 (1986), pp. 617-632.

[5] ———, *Locally optimal subalignments using nonlinear similarity functions*, Bull. Math. Biol., 48 (1986), pp. 633-660.

[6] D. J. Bacon and W. F. Anderson, *Multiple sequence alignment*, J. Molec. Biol., 191 (1986), pp. 153-161,

[7] H. Carrillo and D. Lipman, *The multiple sequence alignment problem in biology*, SIAM J. Appl. Math., 48 (1988), pp. 1073-1082.

[8] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[9] M. O. Dayhoff, *Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3*, National Biomedical Research Foundation, Washington, DC, 1978.

[10] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, *A model of evolutionary change in proteins*, in Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3, M. O. Dayhoff, ed., National Biomedical Research Foundation, Washington, DC, 1978, pp. 345-352.

[11] J. Felsenstein, *Numerical methods for inferring evolutionary trees*, The Quart. Rev. Biol., 57 (1982), pp. 379-404.

[12] J. W. Fickett, *Fast optimal alignment*, Nucl. Acids Res., 12 (1984), pp. 175-180.

[13] W. M. Fitch and T. F. Smith, *Optimal sequence alignments*, Proc. Nat. Acad. Sci. U.S.A., 80 (1983), pp. 1382-1386.

[14] D. J. Galas, M. Eggert, and M. S. Waterman, *Rigorous pattern-recognition methods for* DNA *sequences—analysis of promoter sequences from escherichia coli*, J. Mol. Biolec., 186 (1985), pp. 117-128.

[15] O. Gotoh, *An improved algorithm for matching biological sequences*, J. Molec. Biol., 162 (1982), pp. 705-708.

[16] M. S. Johnson and R. F. Doolittle, *A method for the simultaneous alignment of three or more amino acid sequences*, J. Molec. Evol., 23 (1986), pp. 267-278.

[17] E. A. Kabat and T. T. Wu, *Attempts to locate complementarity determining residues in the variable positions of light and heavy chains of immunoglobulin chains*, Ann. New York Acad. Sci., 190 (1971), pp. 382-393.

[18] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373-395.

[19] L. G. Khachian, *A polynomial algorithm in linear programming*, Soviet Math. Dokl., 20 (1979), pp. 191-194.

[20] W. Miller and E. W. Myers, *Sequence comparison with concave weighting functions*, Bull. Math. Biol., 50 (1988), pp. 97-120.

[21] M. Murata, J. S. Richardson, and J. L. Sussmann, *Simultaneous comparison of three protein sequences*, Proc. Nat. Acad. Sci. U.S.A., 82 (1985), pp. 7657-7661.

[22] S. B. Needleman and C. D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequences of two proteins*, J. Molec. Biol., 48 (1970), pp. 443-453.

[23] D. Sankoff, *Minimal mutation trees of sequences*, SIAM J. Appl. Math., 28 (1975), pp. 35-42.

[24] D. Sankoff and R. J. Cedergren, *Simultaneous comparison of three or more sequences related by a tree*, in Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, D. Sankoff and J. B. Kruskal, eds., Addison-Wesley, Reading, MA, 1983, pp. 253-263.

[25] P. H. Sellers, *On the theory and computation of evolutionary distances*, SIAM J. Appl. Math., 26 (1974), pp. 787-793.

[26] ———, *Pattern recognition in genetic sequences by mismatch density*, Bull. Math. Biol., 46 (1984), pp. 501–514.

[27] T. F. SMITH AND M. S. WATERMAN, *Comparison of biosequences*, Adv. in Appl. Math., 2 (1981), pp. 482–489.

[28] E. SOBEL AND H. MARTINEZ, *A multiple sequence alignment program*, Nucl. Acids Res., 14 (1986), pp. 363–374.

[29] E. N. TRIFONOV AND G. BOLSHOI, *Open and closed* 5 *S ribosomal* RNA, *the only two universal structures encoded in the nucleotide sequences*, J. Molec. Biol., 169 (1983), pp. 1–13.

[30] E. UKKONEN, *On approximate string matching*, in Proc. Internat. Conference on Foundations of Computing Theory, Lecture Notes in Computer Science 158, Springer-Verlag, New York, 1983, pp. 487–496.

[31] M. S. WATERMAN, *Multiple sequence alignment by consensus*, Nucl. Acids Res., 14 (1986), pp. 9095–9102.

[32] M. S. WATERMAN, T. F. SMITH, AND W. A. BEYER, *Some biological sequence metrics*, Adv. in Math., 20 (1976), pp. 367–387.

[33] W. J. WILBUR AND D. J. LIPMAN, *Rapid similarity searches of nucleic acid and protein data banks*, Proc. Nat. Acad. Sci. U.S.A., 80 (1983), pp. 726–730.

[34] T. T. WU AND E. A. KABAT, *An analysis of the sequences of the variable regions of Bence Jones proteins and myeloma light chains and their implications for antibody complementarity*, J. Exp. Med., 132 (1970), pp. 211–250.