Proc. Natl. Acad. Sci. USA
Vol. 89, pp. 6090–6093, July 1992
Biochemistry

Parametric sequence comparisons

MICHAEL S. WATERMAN†, MARK EGGERT†, AND ERIC LANDER‡

†Departments of Mathematics and Molecular Biology, University of Southern California, Los Angeles, CA 90089-1113; and ‡Whitehead Institute for Biomedical Research, 9 Cambridge Center, Cambridge, MA 02142

Communicated by Gian-Carlo Rota, January 2, 1992

ABSTRACT **Current algorithms can find optimal alignments of two nucleic acid or protein sequences, often by using dynamic programming. While the choice of algorithm penalty parameters greatly influences the quality of the resulting alignments, this choice has been done in an ad hoc manner. In this work, we present an algorithm to efficiently find the optimal alignments for all choices of the penalty parameters. It is then possible to systematically explore these alignments for those with the most biological or statistical interest. Several examples illustrate the method.**

Computer algorithms for DNA and protein sequence comparisons have proved increasingly valuable. Unexpected relationships have been found between sequences, where some of the best known are between viral and host DNA (1–3). To study sequence relationships by computer, dynamic programming was introduced for similarity alignment by Needleman and Wunsch (4), for distance by Sellers (5), for best aligning segments by Smith and Waterman (6), for linear gap weights by Gotoh (7), and for *k*th best-aligning segments by Waterman and Eggert (8). There are many other variants, including alignment with nonoverlapping inversions (9). Recently, rapid data base searches are made by using the hashing-based approach of Wilber and Lipman (10) and Lipman and Pearson (11) and by the newly described method BLAST (12). These useful approaches do not always find optimal alignments, and dynamic programming-based methods continue to be used. In fact, hashing-based programs limit the regions of interest and then apply dynamic programming to these regions. BLAST, on the other hand, locates alignments with no insertions or deletions (indels). The subject of this paper is dynamic programming sequence algorithms with linear alignment score.

One of the central difficulties of dynamic programming sequence comparison algorithms is the choice of algorithm penalty parameters—that is, the score given to aligned pairs of letters and to gaps (also called insertions or deletions). In some cases, small changes in amino acid weights or in the indel penalty function create large changes in the resulting alignments. In other cases, the alignments are very robust to changes in these algorithm parameters. We feel that there is no one set of "correct" parameters: parameters that will find significant matches of one statistical quality are not useful for another type of matching. We have come, therefore, to consider sequence comparison for a large set of parameter values and to study the associated statistical problems of multiple hypothesis testing. In the work of Waterman and Gordon (13), the relationship between penalty parameters and the statistical distribution of the score of the best-matching segments is studied, along with the multiple hypothesis testing problem.

Ideally, we would like to compute optimal alignments for the infinite set of all possible parameter values. At first glance, this would seem to require an infinite number of sequence comparisons and therefore appears to be a completely unrealistic goal. In a pioneering study, however, Fitch and Smith (14) were able to find all alignments of two hemoglobin genes where the penalty for a gap of length k is $w(k) = \alpha + \beta k$, and α and β vary. There are 11 regions or subsets of parameters that give distinct sets of optimal alignments. Fitch and Smith arrived at their regions by an insightful application of the basic alignment procedure, but they gave no systematic algorithm. In ref. 15, a similar decomposition of parametric space is given. No one has taken up the problem of creating an algorithm to automatically delineate such regions of constant optimal alignments. In this paper, we describe an algorithm to do this.

We begin by formulating Smith–Waterman algorithms for one- and two-dimensional parameter spaces, but our technique will succeed for any scoring function linear in the algorithm parameters. The score H is a function of the parameters, as we hold the sequences fixed. H has special properties that are useful to know about. H is piecewise linear, and on each linear piece there is a unique set of optimal alignments. In this way, the problem can be seen to be finite. We approach the problem by intersecting hyperplanes. As a conceptual and computational device, we employ infinitesimals, much in the sense of the Leibniz infinitesimals of elementary calculus (16). This gives an efficient algorithm in the one-dimensional case. Moving to a two-dimensional parameter space requires introduction of another order of infinitesimal. The generalization to k-dimensional parameter spaces is easy to describe but is computationally expensive.

One-Dimension Parameter Sets

First, we present an elementary version of the Smith–Waterman alignment algorithm (6). Let $\mathbf{a} = a_1a_2 \ldots a_n$ and $\mathbf{b} = b_1b_2 \ldots b_m$ be the sequences we compare. The score of aligning letters a and b is $s(a, b) = 1$ if $a = b$ and $s(a, b) = -\mu$ if $a \neq b$. The penalty $w(k)$ for deletion or insertion of k letters is given by $w(k) = \delta k$. [This is a special case of $w(k) = \alpha + \beta k$.] The algorithm proceeds recursively by finding the best score H_{ij} ending at a_i and b_j by the equation

$$H_{ij} = \max\{H_{i-1,j-1} + s(a_i,b_j); H_{i-1,j} - \delta; H_{i,j-1} - \delta;0\}. \quad [1]$$

The algorithm is initialized by $H_{0,j} = H_{i,0} = 0$ for $0 \leq i \leq n$, $0 \leq j \leq m$. The score of the best segments of **a** aligned with **b** is of course

$$H = H(\mathbf{a}, \mathbf{b}) = \max_{i,j} H_{i,j}. \quad [2]$$

To further simplify this algorithm, take $\mu = 2\delta$ so that the alignment score is a function of one parameter, $\lambda = \delta$. We emphasize the dependence on the parameter λ by writing $H = H(\lambda)$. The sequences are held constant and the parameter varies in what follows.

Next, we establish some elementary properties of $H(\lambda)$. $H'(\lambda)$ is nonpositive, increasing, and piecewise constant. Clearly, the number of alignments A is finite, and each alignment A has r identities, s_A mismatches, and t_A single-

The publication costs of this article were defrayed in part by page charge payment. This article must therefore be hereby marked "*advertisement*" in accordance with 18 U.S.C. §1734 solely to indicate this fact.

letter indels. Let $s = 2s_A + t_A$, the number of misaligned letters. Therefore $S(A)$, the score of single alignment A, satisfies

$$\begin{aligned} S(A) &= r - s_A\mu - t_A\lambda \\ &= r - (2s_A + t_A)\lambda \\ &= r - s\lambda. \end{aligned}$$

Therefore,

$$H(\lambda) = \max\{r - s\lambda: \text{ alignment } A\}.$$

The very large but finite number of alignments implies that $H(\lambda)$ is piecewise linear and continuous.

Since $r - s\lambda$ is decreasing in λ, so is $H(\lambda)$. If $i = 1, 2, \ldots$ is the index of the maximizing lines as λ increases from 0, r_i, and s_i. Suppose at $\lambda_i < \lambda_{i+1}$, we have

$$r_i - s_i\lambda_i > r_{i+1} - s_{i+1}\lambda_i,$$

and

$$r_i - s_i\lambda_{i+1} < r_{i+1} - s_{i+1}\lambda_{i+1}.$$

It is easy to show $s_i > s_{i+1}$ and $r_i > r_{i+1}$. We have shown more than simply that $H'(\lambda) = -s_i$ is an increasing function of λ. As i increases, not only does the number of unmatched bases, s_i, decrease but the number of identities r_i also decreases.

The algorithm we now describe calculates all values of $H(\lambda)$ on $(0, \infty)$. Recall that Eq. 1 allows us to find $H(\lambda)$ for any fixed λ. A brief sketch of the algorithm is given next. It is easy to find $H(0)$ and $H(\infty)$. The line segment through $[\infty, H(\infty)]$ is easy to find, because $H(\infty)$ = length of longest exact match between **a** and **b**. Since many alignment lines often satisfy $H(0) = r - s\cdot 0$, it is very useful to choose the one with the minimum score s. An algorithm to find the minimum s is given below. Thus, we find the leftmost and rightmost segments of $H(\lambda)$, $0 \le \lambda \le \infty$. If their intersection (x, y) satisfies $H(x) = y$, we know the entire function $H(\lambda)$. Otherwise $H(x) > y$. Computing $H(x)$ allows us to find all lines through $[x, H(x)]$. Below, we show how to find the line that dominates, to the left, all these lines. It is part of the final solution $H(\lambda)$. We continue intersections with our line containing $[0, H(0)]$ until we have found the line segment L_1 of H that intersects at $[x_1, H(x_1)]$ with that containing $[0, H(0)]$. Then, we take $[x_1, H(x_1)]$ and L_2, the line segment of H found just before L_1, and repeat the procedure.

The algorithm described above depends on our ability to find which alignment line through $[x, H(x)]$ is optimal to the left (or right) of x. To illustrate the problem, Fig. 1 shows the multiple lines satisfying $H(0) = r$ for a small example with sequences of length 20. To choose the line dominant for $\lambda > 0$, we introduce the idea of infinitesimal ε. Here think of $\varepsilon > 0$ as a small number, so small that any finite multiple remains smaller than any number that occurs in the algorithms described above. Our new numbers will have the form $u + v\varepsilon$, where $u, v \in R$. The idea, for example, is that we will run the algorithm for $\lambda = \varepsilon$ and find the line maximizing all those through $[0, H(0)]$.

Before explicitly describing the infinitesimal version of the maximum segments algorithm, it is necessary to define a lexicographic linear order on the numbers $u + v\varepsilon$ (16). Let $x_1 = u_1 + v_1\varepsilon$ and $y_2 = u_2 + v_2\varepsilon$. If $u_1 > u_2$ then $x_1 > y_2$. If $u_1 = u_2$ and $v_1 > v_2$, then $x_1 > y_2$. Of course, if $u_1 = u_2$, and $v_1 = v_2$, then $x_1 = y_2$. Thus, the new numbers are linearly ordered. Also, addition is easily defined $x_1 + y_1 = (u_1 + u_2) + (v_1 + v_2)\varepsilon$. Of course, scalar multiplication is defined by $cx_1 = (cu_1) + (cv_1)\varepsilon$.

For $\lambda = u + v\varepsilon$, the algorithm of Eqs. 1 and 2 can be used to compute $H = H(\lambda)$. It is clear that the algorithm is well defined: only addition, subtraction, and maximums are involved. This algorithm will be referred to as the infinitesimal algorithm. Notice that ε is never specified and that in this sense this is symbolic computation. Since the order on infinitesimals is consistent with the order on reals, it is easy to show that if $H(u + v\varepsilon) = a + b\varepsilon$, then $H(u) = a$. In this way, the usual algorithm is a special case of the infinitesimal algorithm.

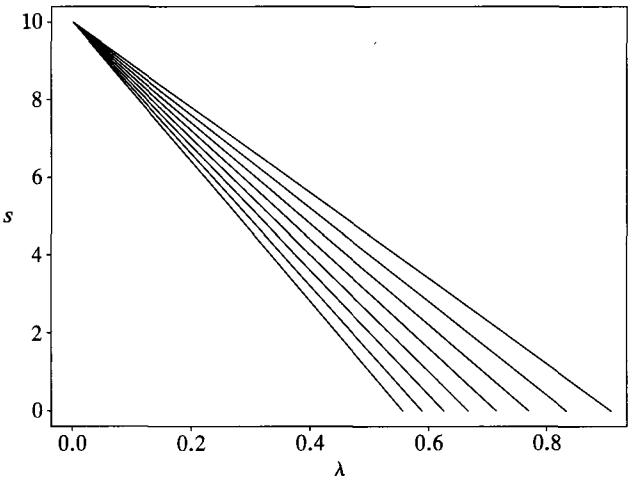

FIG. 1. The eight alignment lines through $[0, H(0)] = (0, 10)$ for **a** = gtaaagtcggacaactagct and **b** = cgcgagtctacgtttggggc. Here $\lambda = \mu = \delta$. The dominant line $10 - 11\lambda$ has 48 alignments, $10 - 12\lambda$ has 1208 alignments, $10 - 13\lambda$ has 10,480 alignments, $10 - 14\lambda$ has 46,076 alignments, $10 - 15\lambda$ has 115,916 alignments, $10 - 16\lambda$ has 169,240 alignments, $10 - 17\lambda$ has 133,412 alignments, and $10 - 18\lambda$ has 43,908 alignments.

Fig. 1 shows several alignment lines through $[0, H(0)]$ for a one-dimensional problem. Notice that if we move just to the right of $\lambda = 0$, to $\lambda = \varepsilon = 0 + 1\varepsilon$, we can find the optimal line. The idea then is to run the algorithm for H with the penalty set slightly larger than 0—that is, at $\lambda' = \varepsilon$. The values of $H_{i,j}(\lambda') = u + v\varepsilon$, and at $\lambda = 0$, $H_{ij} = u$. Just as it is routine to run the new algorithm, $H(\lambda') = H(\varepsilon) = \max_{1\le i\le n, 1\le j\le m} H_{ij}(\lambda')$ is easily calculated. For the length 20 sequences in Fig. 1, $H(\varepsilon) = 10 - 11\varepsilon$. The infinitesimal algorithm for H and the scalar algorithm for H are consistent, as can be seen by $H(0) = 10$. The eight lines in Fig. 1 represent 520,288 distinct alignments. It is this multiplicity of alignments that motivates the development of the infinitesimal algorithm. Otherwise, we would just directly calculate all alignments at a given parameter point.

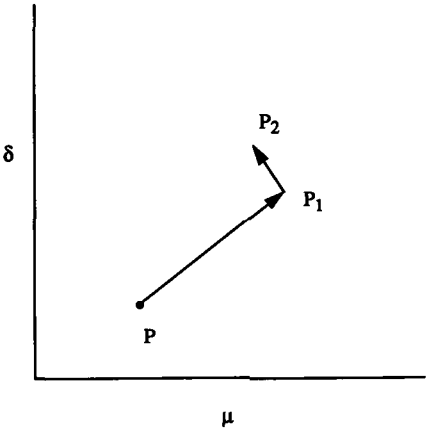

FIG. 2. From $P = (\mu, \delta)$ we obtain $P_1 = P + V_1\varepsilon_1$, and then $P_2 = P_1 + V_2\varepsilon_2$, where V_1 and V_2 are orthogonal.

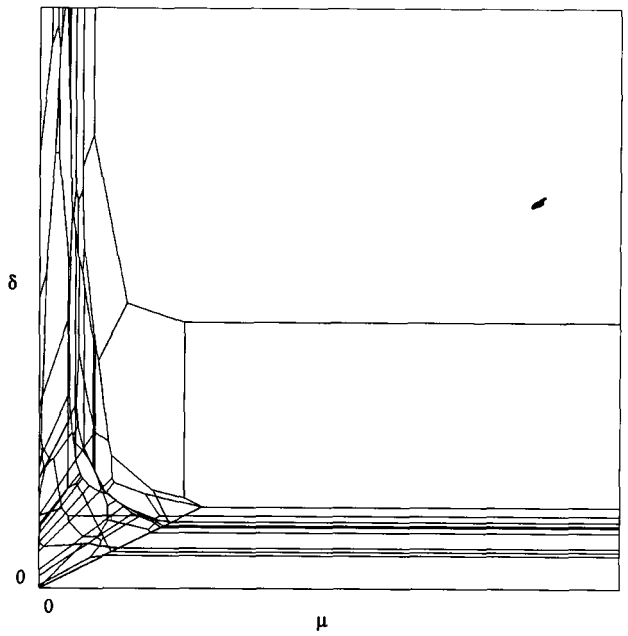

FIG. 3. The two-dimensional algorithm applied to two random DNA sequences of length 400.

Our use of the infinitesimal algorithms is not limited to the purely lexicographic. Below, it will be of interest to calculate the rightmost point of an alignment interval. One way is to proceed as described above, begin at ∞, and intersect toward the required point. To bound away from ∞, look at a traceback through the infinitesimal matrix that produces an optimal line. This alignment must remain optimal throughout the interval. Take the maximum value of ε that will allow all these choices to remain the same. This value can be used instead of ∞, and this is an arithmetic use of infinitesimals.

Two Dimensions and Beyond

Next, we face the task of finding all alignment scores for the two-dimensional (μ, λ) parameter space. In the one-dimensional parameter space, $[\lambda, H(\lambda)]$ is a piecewise linear, convex function, while in the two-dimensional parameter space $H(\mu, \lambda)$ is a piecewise linear, convex surface in three-dimensional space. Recall that our alignment scores satisfy

$$S(A) = r - s\mu - t\lambda, \qquad [3]$$

where r = number of identities, s = number of mismatches, and t = number of indels. The function $f(\mu, \lambda) = r - s\mu - t\lambda$ is referred to as an alignment hyperplane. The simplicity of our one-dimensional algorithm does not carry over here because of the increase in dimension. It is necessary to introduce another order of infinitesimal and to impose a linear order on our new numbers. Then, we derive a technique to find the unique optimal alignment hyperplane adjacent (to the left or right) to any infinitesimal vector from a given point (μ, λ). This algorithm is the basis of our method to find all convex polygons in (μ, λ) space, where the interior has a unique optimal alignment hyperplane.

First, we extend our numbers to include two orders of infinitesimals, ε_1 and ε_2. Let $x = u_1 + v_1\varepsilon_1 + w_1\varepsilon_2$ and $y = u_2 + v_2\varepsilon_1 + w_2\varepsilon_2$. If $u_1 > u_2$, then $x > y$. If $u_1 = u_2$ and $v_1 > v_2$ then $x > y$. If $u_1 = u_2$, $v_1 = v_2$, and $w_1 > w_2$, then $x > y$. Of course if $u_1 = u_2$, $v_1 = v_2$, and $w_1 = w_2$, then $x = y$. As before, no finite multiple of ε_1 can exceed $u_1 \neq 0$, and no finite multiple of ε_2 can exceed ε_1. Addition and subtraction are defined in the obvious way.

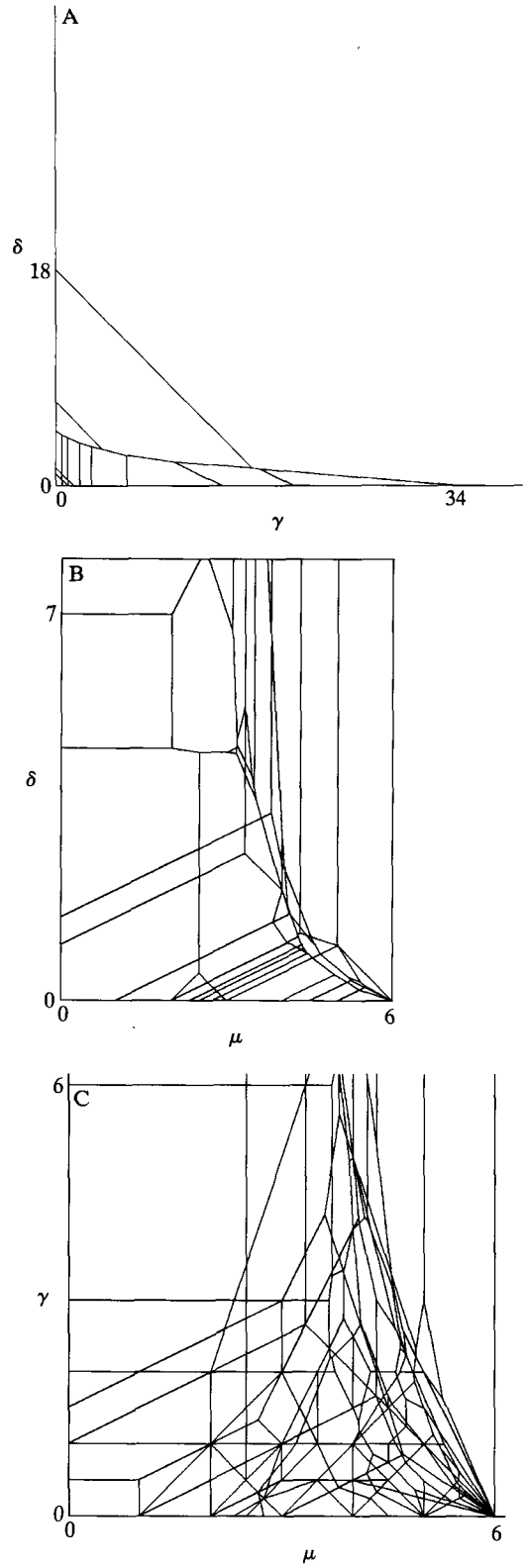

FIG. 4. The two-dimensional algorithm applied to two immunoglobulin protein sequences, L1HUNM or $FABV_H$ is a λ chain V-I region sequence (103 residues) and G1HUNM or $FABV_L$ is a heavy chain V-II region sequence (117 residues). (*A*) Algorithm applied to parameters α and β: $w(k) = \gamma + \delta k$. The 15 polygons in this quadrant are shown. Algorithm applied to $\mu \geq 0$, subtracted from each element in the PAM 250 matrix and δ, where $w(k) = \delta k$. Some of the 72 polygons in this quadrant are shown. (*C*) Algorithm applied to $\mu \geq 0$ (as in *B*) and constant gap cost γ, where $w(k) = \gamma$ for all $k \geq 1$. Some of the 242 polygons in this quadrant are shown.

Our basic algorithm finds the unique optimal alignment hyperplane in the direction (a, b) from (μ, λ). While the surface is in three-dimensional space, we are in the two-dimensional parameter space. We are allowing a point to represent the vector from (0, 0) to that point. It is possible that the (a, b) direction coincides with an intersection of optimal alignment hyperplanes. To ensure uniqueness we must move a small distance perpendicular to (a, b) that is in direction $(-b, a)$ or $(b, -a)$. The direction (a, b) is of length $\varepsilon_1(a^2 + b^2)^{1/2}$, while the directions $(-b, a)$ or $(b, -a)$ are of length $\varepsilon_2(a^2 + b^2)^{1/2}$. Therefore, the parameters are

$$(\mu^*, \lambda^*) = (\mu, \lambda) + \varepsilon_1(a, b) + \varepsilon_2(-b, a), \quad [4]$$

or

$$(\mu^*, \lambda^*) = (\mu, \lambda) + \varepsilon_1(a, b) + \varepsilon_2(b, -a).$$

See Fig. 2 for a graphic representation of the parameters.

To find the convex polygons of constant alignment hyperplane in $(0, \infty) \times (0, \infty)$, think of the parameter space as a rectangle with four edges and four vertices. Begin at a vertex v_0, $v_0 = (0, 0)$, say. From v_0, use the basic two-dimensional algorithm along the line L in the counterclockwise direction (say). Initially, L is the line from (0, 0) to $(\infty, 0)$. The two-dimensional algorithm can find the alignment hyperplane $f_0(\mu, \lambda) = r_0 - s_0\mu - t_0\lambda$ immediately adjacent to v_0 in this direction. The goal is to trace out the convex polygon in (μ, λ) associated with this hyperplane, with vertex/edge labels $(v_0, e_0, v_1, e_1 \ldots, v_n = v_0)$. By a method similar to the one-dimensional algorithm, it is easy to find the vertex v_1 of the first corner point on line L. To find edge e_1, determine the alignment hyperplane f_1 adjacent to the line beyond v_1 on L. The intersection $l = f_0 \cap f_1$, which is a line, has optimal alignment hyperplane f_2 immediately adjacent and counterclockwise. If $f_2 = f_0$ then l is the equation of the line containing the edge e_1. Otherwise, intersect f_0 and f_2, repeating the process until the intersection contains e_1. The process is repeated along e_2 and continued until $v_n = v_0$.

Having traced out the vertices and edges of one of the convex polygons of constant-alignment hyperplane, it can be removed from $(0, \infty)^2$. The procedure is repeated at a vertex on the boundary of the remaining figure until all convex polygons have been characterized.

How expensive is the method we have described? On the average, assume we do η iterations to locate a vertex on a line and ξ iterations to find the line adjacent to the vertex. We do not have theoretical estimates of η and ξ, but they do not appear to grow with n = sequence length. If $\|P_i\| = \eta\#\{\text{edges}\} + \xi\#\{\text{vertices}\}$ for a polygon P_i, then the complexity of our method is $O(nm\Sigma\|P_i\|)$. Clearly, we have not implemented the most efficient method, and we hope that a method can be established to run in time $O(nm\#\text{polygons})$.

To extend our methods to higher-dimensional parameter spaces is of course possible. For k-dimensional parameter spaces, we need $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k)$, where $\varepsilon_k < \varepsilon_{k-1} < \ldots < \varepsilon_1$. It is routine to describe the relevant vectors that generalize (4).

Practical Implementation

We have coded the one- and two-dimensional algorithms. Actually, we can use any two of three parameters: one mismatch and two indel parameters. We illustrate the two-dimensional algorithm by two random DNA sequences of length 400 with a uniform distribution on {A, C, G, T}. The results are shown in Fig. 3, where each planar region indicates a region of constant optimal hyperplane. As we saw in Fig. 1, a large number of alignments can go into a hyperplane, but the optimal hyperplane itself is unique. On the lines of intersection, of course, two or more hyperplanes are optimal.

We then applied the algorithm to the alignment of protein sequences, defining an appropriate three-dimensional parameter space. Amino acid comparisons were based on the Dayhoff matrix PAM250 with a variable offset, $D - \mu = (d_{ij} - \mu = (d_{ij} - \mu)$ and the gap penalty $w(k) = \gamma + \delta k$. We studied two immunoglobulin sequences–namely, the variable domains of the light and heavy chains of the Fab antibody, which we chose because Barton and Sternberg previously (17) studied the effect of different gap penalties on the alignment of these protein domains. Fig. 4 gives three two-dimensional views of this parameter system.

The protein comparison problem indicates a need for future research. Our methods handle two-dimensional systems, and with some patience a three-dimensional system can be studied with the current software. It might become practical to automate three-dimensional problems. However, the Dayhoff matrix has 210 parameters. This is far beyond the practical limits of our methods, and entirely new ideas must be devised for such problems. The current software allows us to systematically study the relationships between penalty parameters and the biological significance of the resulting optimal alignments for the first time.

Note Added in Proof. Since completing the above work, it has come to our attention that Gusfield *et al.* (18) have also considered parametric sequence comparison.

This work was supported by grants from the National Institutes of Health and the National Science Foundation.

1. Weiss, T. (1983) *Nature (London)* **304,** 12.
2. Doolittle, R. F., Hunkapiller, M. W., Hood, L. E., Devare, S. G., Robbins, K. C., Aaronson, S. A. & Antoniades, H. M. (1983) *Science* **221,** 275–276.
3. Naharro, G., Robbins, K. C. & Reddy, E. P. (1964) *Science* **223,** 63–66.
4. Needleman, S. B. & Wunsch, C. D. (1970) *J. Mol. Biol.* **48,** 443–453.
5. Sellers, P. (1974) *SIAM J. Appl. Math.* **26,** 787–793.
6. Smith, T. F. & Waterman, M. S. (1981) *J. Mol. Biol.* **147,** 195–197.
7. Gotoh, O. (1982) *J. Mol. Biol.* **162,** 705–708.
8. Waterman, M. S. & Eggert, M. (1987) *J. Mol. Biol.* **197,** 723–728.
9. Schoniger, M. & Waterman, M. S. (1992) *Bull. Math. Biol.* **54,** 521–536.
10. Wilber, W. J. & Lipman, D. J. (1983) *Proc. Natl. Acad. Sci. USA* **80,** 726–730.
11. Lipman, D. J. & Pearson, W. R. (1985) *Science* **227,** 1435–1441.
12. Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990) *J. Mol. Biol.* **214,** 1–8.
13. Waterman, M. S. & Gordon, L. (1990) in *Computers & DNA*, eds. Bell, G. I. & Marr, T. G. (Addison-Wesley, New York), pp. 127–135.
14. Fitch, W. & Smith, T. F. (1983) *Proc. Natl. Acad. Sci. USA* **80,** 1382–1386.
15. Gotoh, O. (1990) *Bull. Math. Biol.* **52,** 359–373.
16. Keisler, J. H. (1976) *Foundations of Infinitesimal Calculus* (Prindle, Weber & Schmidt, Boston).
17. Barton, G. J. & Sternberg, M. J. E. (1987) *Protein Eng.* **1,** 89–94.
18. Gusfield, D., Balasubramanian, K. & Naor, D. (1992) *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, January 1992*, pp. 432–439.