

Fast probabilistic analysis of sequence function using scoring matrices

Thomas D. Wu^{1,3,*}, Craig G. Nevill-Manning² and Douglas L. Brutlag¹

¹Department of Biochemistry, Stanford University School of Medicine, Stanford, CA, USA and ²Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Received on April 6, 1999; revised on August 4, 1999; accepted on September 10, 1999

Abstract

Motivation: We present techniques for increasing the speed of sequence analysis using scoring matrices. Our techniques are based on calculating, for a given scoring matrix, the quantile function, which assigns a probability, or p , value to each segmental score. Our techniques also permit the user to specify a p threshold to indicate the desired trade-off between sensitivity and speed for a particular sequence analysis. The resulting increase in speed should allow scoring matrices to be used more widely in large-scale sequencing and annotation projects.

Results: We develop three techniques for increasing the speed of sequence analysis: probability filtering, lookahead scoring, and permuted lookahead scoring. In probability filtering, we compute the score threshold that corresponds to the user-specified p threshold. We use the score threshold to limit the number of segments that are retained in the search process. In lookahead scoring, we test intermediate scores to determine whether they will possibly exceed the score threshold. In permuted lookahead scoring, we score each segment in a particular order designed to maximize the likelihood of early termination. Our two lookahead scoring techniques reduce substantially the number of residues that must be examined. The fraction of residues examined ranges from 62 to 6%, depending on the p threshold chosen by the user.

These techniques permit sequence analysis with scoring matrices at speeds that are several times faster than existing programs. On a database of 12 177 alignment blocks, our techniques permit sequence analysis at a speed of 225 residues/s for a p threshold of 10^{-6} , and 541 residues/s for a p threshold of 10^{-20} .

In order to compute the quantile function, we may use either an independence assumption or a Markov assumption. We measure the effect of first- and second-

order Markov assumptions and find that they tend to raise the p value of segments, when compared with the independence assumption, by average ratios of 1.30 and 1.69, respectively. We also compare our technique with the empirical 99.5th percentile scores compiled in the BLOCKSPUS database, and find that they correspond on average to a p value of 1.5×10^{-5} .

Availability: The techniques described above are implemented in a software package called EMATRIX. This package is available from the authors for free academic use or for licensed commercial use. The EMATRIX set of programs is also available on the Internet at <http://motif.stanford.edu/ematrix>.

Introduction

Large-scale projects in sequencing and annotation, including the many genome projects now underway, require increasing speed in sequence analysis. Speed is important not only because of the increasing numbers of query sequences, but also because of the growing size of pattern databases that are used to match queries. In addition, sequence analyses are most informative when they attach probability values to results, so that accurate inferences may be drawn. In this paper, we develop techniques for analyzing sequences rapidly and probabilistically using scoring matrices. Our techniques are implemented in a package of computer programs called EMATRIX, which analyzes sequences several times more quickly than existing programs.

Our approach is novel mainly due to allowing the user to trade off speed and sensitivity explicitly by specifying a probability, or p , threshold. This threshold indicates the desired level of statistical significance for a particular analysis. A low p threshold corresponds to a highly specific analysis, limited to those hits that are statistically very significant. Such an analysis should produce relatively few false positives, but may miss more distant sequence relationships. On the other hand, if

*To whom correspondence should be addressed.

³Present address: Genentech, Inc., 1 DNA Way, South San Francisco, CA 94080, USA. E-mail: twu@gene.com

the user specifies a high p threshold, he can analyze a particular sequence with high sensitivity. In our approach, the trade-off occurs because specific analyses can be performed at relatively high speed, whereas sensitive analyses require more time.

For example, at a p threshold of 10^{-6} , our program attains speeds of 225 residues/s; at a p threshold of 10^{-20} , the speed rises to 541 residues/s. (These speeds are based on a database of 12 177 alignment blocks, and would change linearly with the number of blocks.) In this paper, we show that our program is several times faster than other programs that use scoring matrices, such as BLIMPS (Wallace and Henikoff, 1992) and BLOCKSEARCH (Fuchs, 1993, 1994). Accordingly, we envision that our approach can be used successfully for a wide range of applications: both for genome-wide analyses—which require both high specificity and high speed—and for individual sequence analyses—which usually call for more sensitivity at the expense of speed.

Our program EMATRIX performs sequence analysis using *scoring matrices*, which are used widely in bioinformatics. A scoring matrix represents a nucleic acid or protein segment in a family of related sequences; other terms for this type of construct include weight matrices (Staden, 1990; Stormo and Hartzell, 1989), profiles (Gribskov *et al.*, 1987), and position-specific scoring matrices (Henikoff, 1996). A scoring matrix S represents a gapless local alignment of a sequence family. The alignment consists of several contiguous positions; each position is represented by a column in the scoring matrix. In turn, each column j consists of a vector of scores $S_j(a)$, one score for each possible residue a . (We use the term ‘residue’ to refer to an amino acid or nucleotide, depending on whether we are analyzing protein or nucleic acid sequences, respectively.)

A scoring matrix can be used in sequence analysis by sliding the matrix along the sequence and computing segmental scores. Each segmental score is simply the sum of the appropriate matrix entries, with each residue corresponding to a score in a column of the matrix. Specifically, for a sequence consisting of the residues a_1, \dots, a_L , and a segment of width J beginning at position k ($1 \leq k \leq L - J + 1$), the segmental score is

$$T = \sum_{j=1}^J S_j(a_{k+j-1}). \quad (1)$$

Intuitively, a higher segmental score indicates a greater likelihood that the sequence matches the given scoring matrix. The precise nature of this relationship is a key issue in this paper, and, in fact, the explicit computation of this relationship makes our speed-up techniques possible.

Several databases of alignment blocks are now available, including BLOCKS (Henikoff and Henikoff, 1991),

PRINTS (Attwood and Beck, 1994), PFAM (Bateman *et al.*, 1999), PRODOM (Corpet *et al.*, 1999) and DOMO (Gracy and Argos, 1998). These databases make it possible to identify the function of a sequence by comparing it against every alignment block. To do this with scoring matrices, each alignment block must first be converted into a scoring matrix. Many methods for this conversion have been developed (Brown *et al.*, 1993; Gribskov *et al.*, 1987; Henikoff and Henikoff, 1996; Henikoff *et al.*, 1995; Lawrence *et al.*, 1993; Sjölander *et al.*, 1996; Tatusov *et al.*, 1994; Wu *et al.*, 1999). For a given alignment block, each conversion method produces a slightly different scoring matrix. Thus, conversion methods may be judged based on their ability to represent accurately the alignment block and the underlying family of sequences. In fact, one of the contributions of the EMATRIX project, described elsewhere in (Wu *et al.*, 1999), has been to develop a conversion method based on minimal-risk estimation, and to show that the resulting minimal-risk scoring matrices are more accurate than other types of scoring matrices.

However, in this paper, we focus on a different aspect of the EMATRIX project: our techniques for increasing the speed of sequence analysis. These techniques can be applied to any scoring matrix, regardless of how it was constructed, and are therefore independent of the conversion method. Therefore, in this paper, we will not discuss issues of scoring-matrix construction, but assume that we have converted a set of alignment blocks to a set of scoring matrices. Once we have constructed the scoring matrices, we have essentially determined the segmental scores for any given sequence.

To give those scores a probabilistic interpretation, we need to compute the relationship between segmental scores and probability, or p , values. These p values represent the probability of obtaining the given score in a random segment. Our model of randomness can be defined by using either an independence assumption or a Markov assumption. In the independence assumption, which has been used most often, we assume that residue frequencies at each position in a random segment are independent of other positions. In contrast, in the Markov assumption, we assume that residue frequencies at each position depend on the residues at one or more neighboring positions. We may apply each assumption to a given scoring matrix to compute its relationship between segmental scores and p values. This relationship is called a *quantile function*.

A major contribution of this paper is to show that the quantile function can be used in conjunction with the user-specified p threshold to speed up sequence analysis. For each scoring matrix, this p threshold corresponds to a score threshold, which we can then exploit to achieve faster speeds. We have developed three techniques for this type of speed-up: (1) *significance filtering*, (2) *lookahead scoring*, and (3) *permuted lookahead scoring*.

In significance filtering, we need only store those segments and scoring matrices where the segmental score exceeds the score threshold for that matrix. This technique speeds up sequence analysis by eliminating the need to store and sort large numbers of potential hits.

In lookahead scoring, we add a test step to significance filtering. For each intermediate score of a segment, we test whether the final score could possibly exceed the score threshold. If we can predict that a segment will fail to achieve the score threshold, then we can terminate scoring early and proceed to the next segment. To implement lookahead scoring, we compute a set of intermediate score thresholds for each scoring matrix, one threshold for each column.

Permuted lookahead scoring is similar to lookahead scoring, except that we score each segment by evaluating the residues in a particular order. The order is designed to maximize the likelihood that we will terminate scoring early. The earlier we can terminate the scoring process, the faster we can perform sequence analysis.

Methods

Quantile function

To perform significance filtering, we must first relate each score T to its p value. This relationship is called the *quantile function*, which gives the score that corresponds to a given p value. We compute the quantile function through its inverse, the *complementary cumulative distribution function* (complementary CDF). In turn, we compute the complementary CDF by performing a summation over the *probability mass function* (PMF).

Let us denote the segmental score by the random variable X , and the PMF of X by $f(x)$. Then, for a particular segmental score T , the p value is

$$G(t) = \Pr\{X \geq t\} = \sum_{x=t}^{\infty} f(x)dx \quad (2)$$

The value of $G(T)$ is the probability of observing a score that is greater than or equal to the observed score T , under the null hypothesis. The null hypothesis in our case is that the segment is random.

Several methods for computing the PMF have been proposed. One method is to apply the scoring matrix empirically to a database of random sequences, such as SWISSPROT (Bairoch and Apweiler, 1996), and then tabulate the relative frequency of scores. This method is used by the curators of the BLOCKS database. Each alignment block in the BLOCKS database reports a score at the 99.5th percentile, where the percentiles are tabulated by taking the maximum segmental score for each sequence.

Another method is to compute the probability recursively through each column of the scoring matrix

(McLachlan, 1983; Staden, 1989; Tatusov *et al.*, 1994). In this method, each position in a random sequence is represented by a background frequency vector

$$\mathbf{q} = \langle q(1), \dots, q(|A|) \rangle$$

where A is the set of possible residues. The background frequencies can be obtained from the relative frequencies of residues in a large sequence database. We then compute the PMF recursively, column by column:

$$f^{(0)}(x) = \delta(x) \quad (3)$$

$$f^{(j)}(x) = \sum_a q(a) f^{(j-1)}(x - S_j(a)) \quad (4)$$

$j = 1, \dots, J$

$$f(x) = f^{(J)}(x) \quad (5)$$

where the function $\delta(x)$ is equal to 1.0 for $x = 0$ and zero otherwise. Essentially, we begin with the entire probability mass at score $x = 0$. Then, at each column j , we obtain a revised version of the PMF, $f^{(j)}(x)$, based on the previous PMF. We look back for scores in the previous PMF that could generate x ; these scores are simply $(x - S_j(a))$ for all possible residues a . The final iteration yields the desired PMF $f(x)$.

This method assumes that positions in a random sequence are independent and identically distributed. We now extend the recursive PMF method to handle random sequences under a Markov assumption, where the probability distribution at each position depends on the context of previous positions. In the Markov assumption, each position is represented by a contextual frequency vector \mathbf{q}_C , with elements $q_C(a)$, where C is a certain number c of upstream residues. For example, in a second-order Markov model, the context C would consist of the two residues upstream of the given position. The contextual frequency vectors \mathbf{q}_C can be compiled by tabulating oligomers in a large sequence database.

Under the Markov assumption, we distribute the recursive process over all possible contexts. We maintain $|A|^c$ distributions at each stage j , with the exception that the first few stages, for $j < c$, will have only $|A|^j$ distributions. Each distribution corresponds to a different possible substring C . Let us denote the left shift operation by 'sh', so that 'sh(C, a)' contains the rightmost $(c - 1)$ characters in the concatenation of C and a . Then, we can compute the PMF using the following recursion:

$$f^{(0)}(x) = \delta(x) \quad (6)$$

$$f_{\text{sh}(C,a)}^{(j)}(x) = \sum_C \sum_a q_C(a) f_C^{(j-1)}(x - S_j(a)), \quad (7)$$

$j = 1, \dots, J$

$$f(x) = \sum_C f_C^{(J)}(x) \quad (8)$$

At each stage j of the recursion, the function $f_C^{(j)}$ is not a PMF, because its total mass will be less than 1. However, the sum of the functions $f_C^{(j)}$ over all C will contain total mass of 1. The PMF is represented by summing over all contexts C .

Once we have the PMF $f(x)$, we can compute the complementary CDF $G(T)$ by summation [equation (2)]. Then, we have the quantile function, which is the inverse function, $G^{-1}(p)$. Given a p threshold p^* , the quantile function generates a *score threshold* $T^* = \lceil G^{-1}(p^*) \rceil$. Because we have computed $G(T)$ for all possible segmental scores, we can determine the score threshold readily for any given p threshold.

Significance filtering

In significance filtering, we use the quantile function for each scoring matrix to convert the user-specified p threshold p^* into a score threshold T^* . Then, we retain a segment only if its score is equal to or exceeds T^* and discard it otherwise.

To implement significance filtering, we would like to store the quantile function for each scoring matrix. However, it would be prohibitively expensive to store the quantile function for all values of p^* . Therefore, we store the quantile function only at certain intervals. Our current implementation stores the quantile function at multiples of 10 from 10^{-1} through 10^{-40} .

Figure 1 shows the quantile function for an example scoring matrix, stored at these intervals. Suppose the user specifies a p threshold of 10^{-6} . Then, the corresponding score threshold is -148 . If we desire a more specific sequence analysis, at $p^* = 10^{-10}$, the score threshold rises to $T^* = 182$, thereby allowing fewer segments to match.

If a segment does exceed the score threshold, indicating a positive hit, we would like to report its p value. For this task, we need the complementary CDF $G(T)$, so we should be able to use its inverse, the stored quantile function. However, because we have stored the quantile function only at certain intervals, we must perform interpolation. To determine the p value of a given score T , we use log-linear interpolation on the stored points T_0 and T_1 :

$$G(T) \approx \exp \left[\log G(T_0) + \frac{T - T_0}{T_1 - T_0} (\log G(T_1) - \log G(T_0)) \right] \quad (9)$$

Lookahead scoring

With significance filtering, we compare the final segmental score with a score threshold T^* . However, it is possible to evaluate the acceptability of a segment midway through

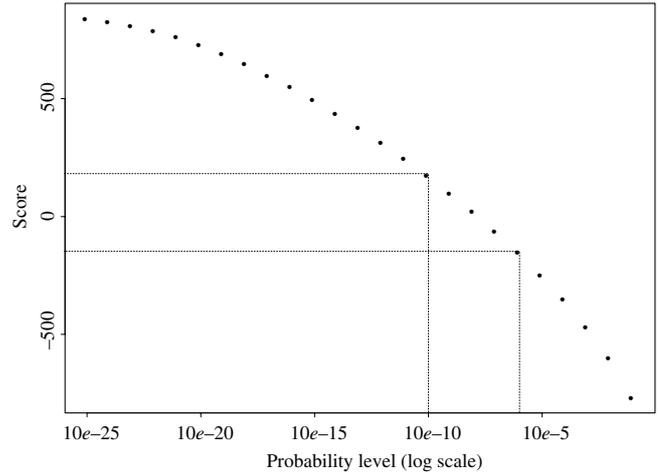


Fig. 1. Quantile function for a scoring matrix. The scoring matrix was derived from block 21B (kringle domain proteins) from BLOCKSPLUS 11.0. This block contains 32 protein segments, each having a width of 18 amino acids. The scoring matrix was computed by the square-error minimal-risk method described in (Wu *et al.*, 1999), using the BLOSUM62 substitution matrix. Quantiles were computed using a first-order Markov assumption. Two points are highlighted with dashed lines, corresponding to p thresholds of 10^{-6} (score = -148) and 10^{-10} (score = 182).

its computation. In lookahead scoring, we compare intermediate scores with intermediate score thresholds. We can derive intermediate score thresholds by knowing the maximum possible score for the remainder of the segment. If the intermediate score is less than the required threshold at any point, then we can terminate the scoring process early. The idea behind lookahead scoring is similar in spirit to the A^* algorithm for search problems (Hart *et al.*, 1968), although here we are trying to find satisfactory scores rather than optimal scores.

The intermediate score threshold at column j is based on the maximum possible score in columns $(j + 1)$ through J . We call this quantity the *maximal remainder score*:

$$Z^{(j)} = \sum_{k=j+1}^J \max_a S_k(a) \quad (10)$$

which is a summation of the maximum scores in columns $(j + 1)$ through J . This quantity can be pre-computed and stored for each scoring matrix. Then, the intermediate score threshold at column j is $(T^* - Z^{(j)})$. At each column j , the intermediate score $T^{(j)}$ must exceed this quantity. If this lookahead condition fails, we can terminate scoring.

Note that as T^* increases, the intermediate score thresholds all increase in parallel, making it more likely that the lookahead condition fails earlier in the process. This con-

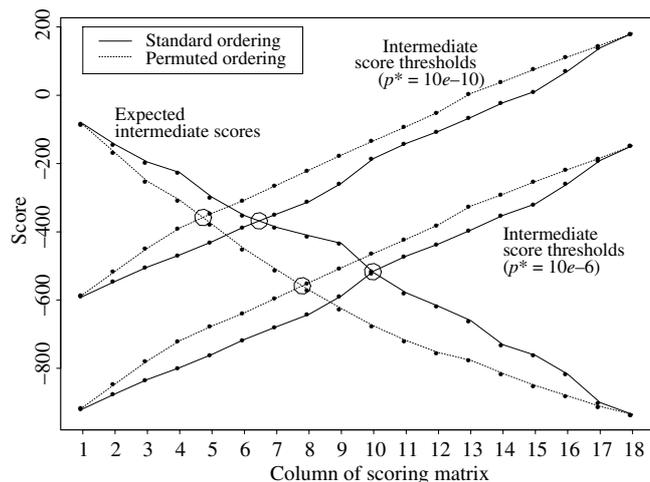


Fig. 2. Intermediate scores and score thresholds. Intermediate score thresholds are shown for block 21B from BLOCKSPLUS 11.0, for p thresholds of 10^{-10} and 10^{-6} . These two sets of thresholds differ only in their starting score thresholds of 182 and -148 in their rightmost column, obtained from Figure 1. Intermediate score thresholds are shown both for the standard ordering of columns in the scoring matrix (solid curves) and for a permuted ordering (dashed curves). The expected intermediate scores are also shown, with the crossover points circled. For a p threshold of 10^{-6} , the crossover points occur at columns 10.0 (standard ordering) and 7.8 (permuted ordering). For 10^{-10} , crossover occurs at columns 6.5 and 4.7, respectively.

cept is shown in Figure 2, which shows two parallel sets of intermediate score thresholds, corresponding to values of T^* for $p^* = 10^{-6}$ and $p^* = 10^{-10}$. The figure also shows the expected scores along the scoring matrix. The lookahead condition fails when the two curves cross, allowing early termination of scoring. As the value of T^* increases, the average crossing point moves leftward, resulting in earlier terminations.

At first glance, it would appear that the additional cost of lookahead scoring is one subtraction per column (to compute the intermediate score threshold ($T^* - Z^{(j)}$)) and one comparison operation per column (to compare the intermediate score with the threshold). However, we can reduce this cost to just the comparison operation by subtracting T^* from the intermediate scores and comparing that value with $-Z^{(j)}$. This subtraction can essentially be performed by starting the segmental scoring process with $-T^*$. Then, if the intermediate scores all exceed their respective thresholds, meaning a positive hit, we can correct the segmental score by adding back T^* .

Permuted lookahead scoring

Previously, we have used the values in scoring matrices sequentially from position 1 through position J . However,

we may evaluate the residues in a given segment in any order. With lookahead scoring, the sooner we can reject a segment, the better. Therefore, we investigate the possibility of evaluating scoring matrices in a permuted order, giving rise to the strategy of permuted lookahead scoring.

Suppose that we have a permutation $\pi = \langle \pi_1, \dots, \pi_J \rangle$, where π_j indicates the position to be evaluated at step j . Then, we simply compute both the intermediate scores and the intermediate score thresholds in this order. As before, if the intermediate score is less than the corresponding threshold, we can terminate scoring early.

It would seem that the additional cost of the permuted lookahead scoring, compared with standard lookahead scoring, would be two look-ups in the permutation π , one to find the right residue in the segment and one to find the right entry in the scoring matrix. However, we can eliminate the latter step by storing the scoring matrix in permuted order.

The issue then remains of how to compute the permutation π . Each column in a scoring matrix has a maximal score and an expected score, respectively:

$$M_j = \max_a S_j(a) \quad (11)$$

$$E_j = \sum_a S_j(a)q_a \quad (12)$$

where the values of $q(a)$ are the background frequencies discussed previously. The key statistic should be the difference between these scores. If the expected score for a column is low relative to the maximal score, then it is more likely that the column will cause the lookahead condition to fail. Because we would like to know as early as possible whether the segment will fail to achieve the desired score threshold, we should order the columns according to their differences in expected and maximal scores, ($E_j - M_j$). We compute this difference for each column in the scoring matrix and create the permutation by ordering the columns from largest difference to smallest difference.

As Figure 2 (dashed curves) shows, this permutation causes the crossover point to move earlier, from column 6.4 to 4.7 for $p^* = 10^{-10}$, and from column 10.0 to 7.8 for $p^* = 10^{-6}$. This difference contributes somewhat to faster sequence analysis, by an amount that we quantify in the Results section.

Implementation

The auxiliary information needed for the above techniques can be computed for each scoring matrix in advance, because databases of alignment blocks change relatively infrequently. We pre-compute and store the auxiliary information as part of the process of converting a set of alignment blocks into a set of scoring matrices. The EMATRIX package is designed to perform both the conversion and pre-analysis steps, using the programs

EMATRIX-MAKER and EMATRIX-BUILD, respectively. The EMATRIX package also includes software for performing the actual sequence analysis, in the program EMATRIX-SEARCH. If the user has a particular alignment block of interest, he can use the program EMATRIX-MAKER to create a scoring matrix, and the program EMATRIX-SCAN to compare the scoring matrix against a set of sequences, such as SWISSPROT (Bairoch and Apweiler, 1996).

The EMATRIX package is written in the programming languages C and Perl. The EMATRIX-BUILD program runs EMATRIX-MAKER on a database of alignment blocks to create a database of permuted scoring matrices. This program also computes, for each scoring matrix, a set of score thresholds for various p thresholds and a set of maximal remainder scores, one for each column in the matrix. These data are converted into binary files for fast input. Each alignment block also has a brief description, and these descriptions are stored in a separate binary file, along with pointers, so that descriptions may be accessed randomly. This implementation detail fits well with our significance filtering strategy, which requires that only a small fraction of descriptions be read, thereby making random access useful.

The conversion process of EMATRIX-BUILD requires several hours for current databases, largely because of the time needed to compute the complementary CDF for each scoring matrix. This time is lengthened considerably by the use of higher-order Markov assumptions, because the computing time increases roughly as $O(|A|^c)$, where c is the order of the Markov process. For large alphabets A , such as the set of 20 amino acids, this computational behavior can be limiting. For example, computing the complementary CDFs for all blocks in the current BLOCKSPLUS database requires 76 min under the independence assumption. Under the first-order Markov assumption, the computation requires 17 h. The second-order assumption would take approximately 20 times longer, and would be prohibitively expensive to compute for large databases.

Once computed, however, the new databases can be used repeatedly for rapid sequence analysis. Because standard databases of alignment blocks exist, we can apply EMATRIX-BUILD to produce standard databases of scoring matrices and auxiliary data structures. The standard EMATRIX databases are updated in sync with new releases of alignment blocks. Currently, the standard EMATRIX database is kept in sync with BLOCKSPLUS (Henikoff *et al.*, 1999), which contains alignment blocks taken from the BLOCKS, PRINTS, PFAM, DOMO, and PRODOM databases. Non-standard or specialized databases of alignment blocks can also be processed with EMATRIX-BUILD, and the resulting scoring matrices can be used by EMATRIX-SEARCH to perform sequence analysis.

Results

We perform four experiments to quantify the efficacy of our techniques. First, we assess the speed of our techniques and compare them with one another, as well as with existing programs. Second, we analyze why lookahead scoring is so effective. Third, we compare the probability distributions obtained using Markov-based computations versus independence-based computations. Finally, we give a probabilistic interpretation to existing empirical methods for scoring matrix analysis.

Speed of sequence analysis

In this test, we compiled sequence analysis programs on a Silicon Graphics O2 machine with an R10000 processor at 175 MHz, using 32-bit, level-2 optimization. We measured the total processing time required to analyze all 100 sequences, using the user CPU time reported by the Unix `time` command. The accuracy of this timing procedure can be estimated roughly from repeated runs of EMATRIX-SEARCH. We used 36 measurements in this experiment (described later) for which the running time was essentially independent of the p threshold. We obtained a standard error of 0.26 s compared with a mean of 370.4 s, which suggests a relative error of approximately 0.1%.

We selected 100 sequences at random from the SWISSPROT database, version 36. These sequences contained a total of 63 496 amino acids, or an average of 635 amino acids per sequence. For sequence analysis, we used the BLOCKSPLUS database, version 11.0 (November 1998), which contains 12 177 alignment blocks. The width of the alignment blocks varies from 5 to 55 positions, with an average of 24.2 positions.

We tested the BLIMPS program, version 3.2.5 (January 1999), previously called PATMAT (Wallace and Henikoff, 1992). This program required a total of 4001.2 s to analyze the sequences, or 15.9 residues/s. We performed the same analysis using the BLOCKSEARCH program (Fuchs, 1993, 1994), version 2.1 (December 1993). This program required a total of 582.4 s to analyze the sequences, or 109.0 residues/s.

We then analyzed the same 100 sequences using the EMATRIX-SEARCH program, under the three methods of significance filtering, lookahead scoring, and permuted lookahead scoring. We selected 36 p thresholds from 10^{-5} to 10^{-40} at multiples of 10. With significance filtering, the 36 different p thresholds had little effect on speed. (Also, as we noted previously, these CPU times allowed us to estimate the precision of our timing procedure). Significance filtering required 370.4 s (171.4 residues/s) to analyze all 100 sequences.

In contrast, the speed of lookahead scoring and permuted lookahead scoring depended greatly on the p

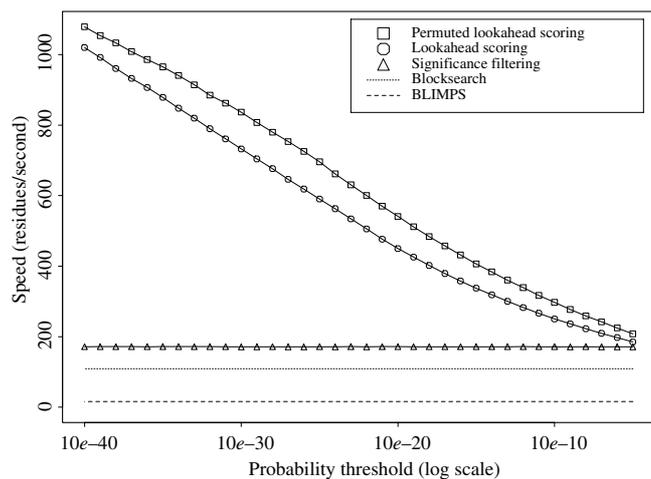


Fig. 3. Speed of sequence analysis as a function of p threshold. For all programs, speeds were measured on a set of 100 randomly selected protein sequences, containing a total of 63 496 residues, against the 12 177 blocks in BLOCKSPLUS 11.0. For significance filtering, lookahead scoring, and permutated lookahead scoring, measurements were performed on p thresholds ranging from 10^{-5} to 10^{-40} at multiples of 10. The dashed line shows an asymptotic level of 4.1%, derived from the average width of 24.2 over all alignment blocks in BLOCKSPLUS 11.0.

threshold. For lookahead scoring, the total time ranged from 342.9 s (185.2 residues/s) for $p^* = 10^{-5}$, to 62.3 s (1019.2 residues/s) for $p^* = 10^{-40}$. For permutated lookahead scoring, the total time ranged from 305.2 s (208.0 residues/s) to 58.9 s (1078.0 residues/s) for $p^* = 10^{-40}$. Over all p thresholds, permutated lookahead scoring was between 5.8 and 20.6% faster than sequential lookahead scoring, with an average speed increase of 15.6%. The total set of speeds is plotted in Figure 3.

Analysis of lookahead scoring

To understand why lookahead scoring and permutated lookahead scoring are so powerful, we performed further analysis of these techniques. Both lookahead scoring methods allow our program to examine only a fraction of the residues. We therefore analyzed the savings afforded by the lookahead scoring methods, in terms of the fraction of residues examined.

We modified our EMATRIX-SEARCH program to report the total number of residues that it examined for each scoring matrix, as well as the potential number of residues that it would have examined without lookahead scoring. We measured the results for p thresholds from 10^{-5} to 10^{-40} . For the input sequence, we concatenated the 100 sequences used in our previous analysis into a single sequence.

The analysis is shown in Figure 4. For each p threshold,

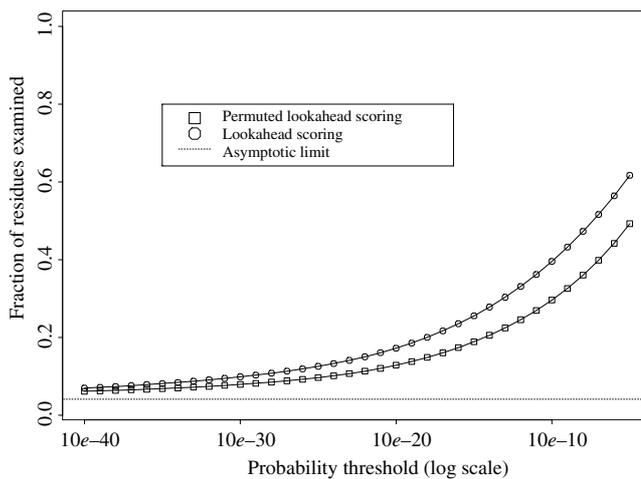


Fig. 4. Efficiency of lookahead scoring. The graph shows the fraction of residues examined, where the fraction is determined relative to the number of residues examined by significance filtering. The performance of both lookahead scoring and permutated lookahead scoring were analyzed, on a set of 100 randomly selected protein sequences. Measurements were performed at probability thresholds ranging from 10^{-5} to 10^{-40} at multiples of 10.

the figure shows the fraction of residues examined, averaged over all scoring matrices, for the lookahead scoring and permutated lookahead scoring techniques. Lookahead scoring examines only 62% of the residues at a p threshold of 10^{-5} ; 40% at 10^{-10} ; and 17% at 10^{-20} . Permutated lookahead scoring examines even fewer residues: 49, 30, and 13%, respectively. Both methods reach an asymptotic limit as p^* becomes smaller. This limit occurs because the methods must evaluate at least one residue in each segment of length J , whereas significance filtering evaluates all J residues. The scoring matrices in BLOCKSPLUS have an average width of 24.2 residues, meaning that the savings must reach an asymptotic limit of $1/24.2$, or 4.1%.

Markov computation of probability values

In this paper, we have introduced a method for computing the probability mass function and quantile function under a Markov assumption. Our method extends existing methods for calculating the PMF under an independence assumption. The independence and Markov assumptions yield different complementary CDFs, and hence different p values for a given score. We performed an analysis to compare the results of using a first-order and a second-order Markov assumption against an independence assumption.

We chose 100 blocks at random from the BLOCKSPLUS database. For each block, we computed the PMF and the corresponding complementary CDF under the independence, first-order, and the second-order assump-

tions. In these computations, we used the marginal frequencies $q(a)$ and the first-order and the second-order Markov frequencies $q_C(a)$ observed over all sequences in SWISSPROT, version 36.

In order to compare the methods, we compared p values between the Markov and the independence assumptions by composing the quantile function under the independence assumption with the complementary CDF under the Markov assumption. In other words, the relationship between p values was given by

$$p' = G_{\text{Markov}}(G_{\text{indep}}^{-1}(p))$$

where G_{indep}^{-1} is the quantile function under the independence assumption and G_{Markov} is the complementary CDF under the Markov assumption. We used p values ranging from 10^{-5} to the smallest p value found for that scoring matrix. We computed the difference between the logarithms of the Markov-based p values and the independence-based p values.

For the first-order Markov assumption, the difference in $\log p$ values ranged from -0.168 to 0.942 , with an average of 0.115 . For the second-order Markov assumption, the difference in $\log p$ values ranged from -0.275 to 1.683 , with an average of 0.229 . The averages are positive, indicating that Markov assumptions tend to raise the p value of a given score. In other words, Markov methods tend to evaluate segments as being less significant, compared with the independence method. The average differences in $\log p$ values correspond to multiples of 1.30 and 1.69 in p values. The distributions of differences in $\log p$ values are shown in Figure 5.

Probabilistic interpretation of empirical scores

Our methods for assessing segmental scores enable us to give a probabilistic interpretation to existing empirical methods for scoring matrix analysis. The empirical approach is embodied in the BLOCKSPLUS database, where each alignment block is labeled with its 99.5th percentile score over all sequences in SWISSPROT.

The BLOCKSPLUS database essentially provides only a single quantile score, out of an entire distribution of scores. In addition, the semantics of empirical quantile scores are different from that of our quantile scores. The empirical distribution is computed by tabulating the maximum score for each sequence in SWISSPROT. Therefore, an extreme value distribution (Castillo, 1988) is inherent to the empirical scores, and this distribution depends on the distribution of lengths of sequences in SWISSPROT. On the other hand, our quantile scores are based on the application of a scoring matrix to a single segment, and therefore represent the original distribution, before any consideration of extreme values.

With these differences in mind, we performed a

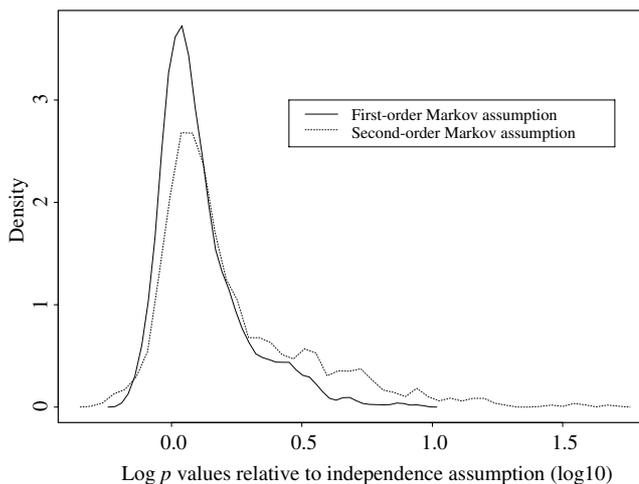


Fig. 5. Comparison of probability values under Markov and independence assumptions. The graph shows the distribution of the difference in $\log p$ values between Markov assumptions and the independence assumption. The comparison of p values was performed by composing the quantile function from the independence assumption with the complementary CDF of the Markov assumption. The distribution is shown as a smoothed density function, using a Gaussian window with a standard error of 0.025 .

probabilistic analysis of the empirically derived 99.5th percentile scores in BLOCKSPLUS. First, we computed the scoring matrices corresponding to the alignment blocks in BLOCKSPLUS. These scoring matrices are computed using a ‘position-specific’ method (Henikoff and Henikoff, 1996), which adds $5N$ pseudocounts to each position, where N is the number of distinct amino acids observed in the position. We computed each scoring matrix using the program PSSM, available from the authors of BLOCKSPLUS, and then applied our methods to the resulting scoring matrix.

For each scoring matrix, we computed the complementary CDFs under the independence, first-order, and second-order assumptions. We then looked up the probability value corresponding to the 99.5th percentile score listed in the BLOCKSPLUS database. For the second-order assumption, we computed the complementary CDF only for one-tenth of the scoring matrices, because the computation time for all blocks would have required several days. We compiled probability values separately for the five databases contained within BLOCKSPLUS, and computed statistics using both raw p values and their logarithms, base 10.

The mean and standard deviation values are shown in Table 1. The table shows that the 99.5th percentile corresponds on average to a p value of 1.45×10^{-5} under

Table 1. Probabilistic interpretations of empirically determined scores. The table shows the statistics of p values corresponding to the scores compiled in the BLOCKSPLUS database. For details of the comparison, see the text and the legend to Figure 6. SD = standard deviation, log = log p values, base 10

Order	Statistic	BLOCKS	DOMO	PFAM	PRINTS	PRODOM	Overall
0	Minimum	5.87×10^{-10}	2.73×10^{-8}	3.60×10^{-11}	1.30×10^{-5}	9.92×10^{-11}	3.60×10^{-11}
	Maximum	2.60×10^{-5}	2.63×10^{-5}	2.50×10^{-5}	2.08×10^{-5}	9.62×10^{-5}	9.62×10^{-5}
	Mean	1.22×10^{-5}	1.20×10^{-5}	1.25×10^{-5}	1.71×10^{-5}	1.19×10^{-5}	1.45×10^{-5}
	SD	0.41×10^{-5}	0.41×10^{-5}	0.41×10^{-5}	0.10×10^{-5}	0.52×10^{-5}	0.41×10^{-5}
	Mean(log)	-4.96	-4.97	-4.96	-4.77	-4.98	-4.87
	SD(log)	0.29	0.27	0.34	0.03	0.35	0.25
1	Minimum	6.27×10^{-10}	4.11×10^{-8}	3.90×10^{-11}	1.30×10^{-5}	1.06×10^{-10}	3.90×10^{-11}
	Maximum	2.64×10^{-5}	2.56×10^{-5}	2.81×10^{-5}	2.88×10^{-5}	9.87×10^{-5}	9.87×10^{-5}
	Mean	1.25×10^{-5}	1.22×10^{-5}	1.27×10^{-5}	1.76×10^{-5}	1.22×10^{-5}	1.48×10^{-5}
	SD	0.41×10^{-5}	0.41×10^{-5}	0.41×10^{-5}	0.15×10^{-5}	0.52×10^{-5}	0.42×10^{-5}
	Mean(log)	-4.95	-4.96	-4.94	-4.76	-4.97	-4.86
	SD(log)	0.29	0.26	0.33	0.04	0.34	0.24
2	Minimum	1.79×10^{-9}	2.96×10^{-6}	2.28×10^{-6}	1.40×10^{-5}	5.95×10^{-6}	1.79×10^{-9}
	Maximum	2.24×10^{-5}	2.37×10^{-5}	2.63×10^{-5}	4.15×10^{-5}	2.02×10^{-5}	4.15×10^{-5}
	Mean	1.28×10^{-5}	1.22×10^{-5}	1.35×10^{-5}	1.81×10^{-5}	1.31×10^{-5}	1.53×10^{-5}
	SD	0.42×10^{-5}	0.39×10^{-5}	0.40×10^{-5}	0.23×10^{-5}	0.33×10^{-5}	0.42×10^{-5}
	Mean(log)	-4.95	-4.94	-4.90	-4.74	-4.90	-4.85
	SD(log)	0.35	0.18	0.17	0.05	0.12	0.24

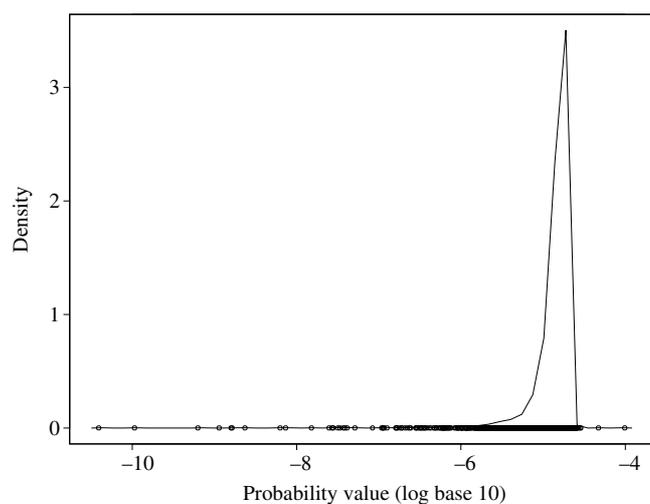


Fig. 6. Probabilistic interpretations of empirically determined scores. The graph shows the distribution of p values for scores stored in the BLOCKSPLUS 11.0 database. The p values are plotted on a logarithmic scale, so that the entire range of values could be represented. These scores were determined empirically to represent the 99.5th percentile over a sequence database. The p values are determined by computing the complementary CDF, and correspond to the probability of obtaining the score in a random segment. The distribution is shown as a smoothed density function, using a Gaussian window with a standard error of 0.03. Individual data points are plotted on the horizontal axis.

the independence assumption, 1.48×10^{-5} under the first-order assumption, and 1.53×10^{-5} under the second-order assumption. On the log base 10 scale, the averages are approximately -4.86 , which corresponds to a p value of 1.38×10^{-5} .

We show the density function for the first-order Markov assumption in Figure 6. The density functions for the independence assumption and second-order Markov assumption (not shown) are very similar. The graph shows that most of the p values lie in the range from -4.5 to -5.5 . However, there is a long tail towards the left, indicating that a few empirically determined scores correspond to relatively small p values.

Discussion

The architecture of the EMATRIX package differs in four main ways from existing packages for sequence analysis with scoring matrices. First, our approach depends on probabilistic distributions of segmental scores, such as the complementary CDF and quantile functions. Quantile functions essentially provide a way to calibrate scoring matrices. Scoring matrices have traditionally been calibrated by empirical methods, by computing the distribution of segmental scores against a real-world sequence database. For example, the curators of the BLOCKSPLUS database compile the distribution of the maximum segmental scores for each sequence, and store the 99.5th percentile score with each alignment block.

However, a single score can provide only a binary test of significance, whereas a stored quantile function permits a p value to be assigned to each hit.

Second, our approach permits accurate comparisons across scoring matrices. Score-based programs, such as BLIMPS and BLOCKSEARCH, typically rank hits according to their raw scores. However, a score from one scoring matrix may not compare well with a score from another scoring matrix. Raw scores depend heavily on such variables as the width of scoring matrices and the way in which they are scaled. Probability distributions provide a uniform standard with which we can compare hits across scoring matrices and interpret their statistical significance.

Third, our approach computes probability and other auxiliary information at compile time, rather than run time. We perform the necessary analysis of each scoring matrix when we convert a database of alignment blocks into a database of scoring matrices. We therefore perform most of the computing work in the pre-analysis stage, and store computations that would otherwise have to be performed for each sequence analysis. Thus, the EMATRIX system involves a close linkage between database construction and sequence analysis.

Finally, our approach introduces a trade-off between speed and sensitivity, by allowing the user to specify a desired p threshold in advance. Different thresholds can be used for different searching tasks, ranging from analyses of single sequences to entire genomes.

Previously, the fastest program for sequence analysis using scoring matrices was the BLOCKSEARCH program of Fuchs (1993, 1994). However, BLOCKSEARCH can potentially sacrifice accuracy, because it may fail to report some high-scoring segments. The program BLOCKSEARCH depends on the observation that many alignment blocks in BLOCKS contain singleton positions, those with only a single amino acid. The program exploits this observation by reporting only segments that contain the conserved amino acid, thereby achieving high speed. However, some segments may nevertheless achieve high scores, even if they do not contain the conserved amino acid, and these segments are not reported by BLOCKSEARCH. The technique of BLOCKSEARCH depends on a particular characteristic of conservation in the BLOCKS database, which derives from the particular way in which it is generated (Smith *et al.*, 1990). In contrast, our techniques make no assumptions about the presence of singleton positions or the method of scoring-matrix construction, and are more broadly applicable.

The BLIMPS program also performs sequence analysis by ranking segmental scores. This program maintains segmental scores in an ordered fashion by using a skiplist data structure (Pugh, 1990). The skiplist data structure is a variant of a linked list that allows one to insert each

new entry in $O(\log L)$ time, where L is the length of the linked list. Thus, for a total of N scoring matrices, the N segmental scores can be maintained in a linked list in $O(N \log N)$ time. However, if we simply sort the list of segmental scores at the end of the scoring process, rather than maintaining an ordered list throughout the scoring process, an implementation without skiplists would also require $O(N \log N)$ time. The EMATRIX-SEARCH program essentially avoids the sorting issue altogether by using significance filtering to limit the number of hits that must be stored and ranked.

Sequence analysis using scoring matrices is becoming increasingly popular. Recently, the program PSI-BLAST (Altschul *et al.*, 1997) has been developed to perform a sequence similarity search with a scoring matrix as a query. This program generates a scoring matrix through several iterations, with the first iteration starting with a single query sequence. The statistics of PSI-BLAST are based on the underlying method of BLAST (Altschul *et al.*, 1990). The task that PSI-BLAST performs is somewhat different from that performed by EMATRIX-SEARCH, where we match a query sequence against a database of known scoring matrices. Our analogue of PSI-BLAST is EMATRIX-SCAN, which matches a scoring matrix against a sequence database.

Virtually all sequence analysis programs must address the problem of multiple inference, which arises because numerous statistical tests must be performed during each analysis. The more tests that are performed, the more likely it is to achieve a low p score by chance. Thus, some correction is often performed to account for the multiple inferences performed. In our program, the p values computed by our procedure refer to a single application of a scoring matrix to a single segment, and therefore have not yet been corrected for multiple inferences.

There exist several methods for handling the multiple inference problem. One method is to compute an extreme value distribution that gives the p value for the maximum value from a series of segment scores (Castillo, 1988; Goldstein and Waterman, 1994). We have elected not to compute extreme value distributions, because we would have to store several distributions in anticipation of the different input sequence lengths. Moreover, the extreme value distribution still does not account for the fact that we are making inferences over multiple scoring matrices.

Another solution to the multiple inference problem is to use a Bonferroni correction to replace the p threshold p^* by $1 - (1 - p^*)^N$, where N is the number of tests performed. Here, for each sequence, the number of tests is equal to $B(L - \bar{J} + 1)$, where B is the number of scoring matrices, L is the length of the given sequence, and \bar{J} is the average width of the scoring matrices. The Bonferroni correction could be applied easily in our case by modifying the p threshold. In our experience,

our segment-based p values correspond to biologically meaningful results, given an appropriate threshold.

The EMATRIX package is an example of the pattern-based approach to sequence analysis, which contrasts with similarity search programs, such as FASTA (Pearson and Lipman, 1988) or BLAST (Altschul *et al.*, 1990). Specifically, EMATRIX uses patterns represented as scoring matrices. In previous work, we have explored the use of other pattern representations, such as discrete motifs. Our approach to discrete motifs is implemented in the EMOTIF package for sequence analysis, which we have described elsewhere (Nevill-Manning *et al.*, 1997, 1998).

Hence, the EMATRIX package performs the same functions as EMOTIF, except that it relies on scoring matrices instead of discrete motifs. One advantage of discrete motifs is that they can match segments at high speed. We have found that EMOTIF achieves speeds of approximately 1000 residues/s, using a database of 50 000 discrete motifs from BLOCKS, version 10.0. This speed exceeds those obtained by using scoring matrices, unless we specify extremely low p thresholds. However, because discrete matches are binary, rather than probabilistic, discrete motifs are typically less sensitive than scoring matrices. The techniques developed in this paper therefore make it possible to use scoring matrices at relatively high speed, and to apply them to large-scale projects in sequence analysis and annotation.

Acknowledgments

T.D.W is a Howard Hughes Medical Institute Physician Postdoctoral Fellow. C.G.N and D.L.B were supported by a grant from Smith-Kline Beecham and by NLM grant LM-05716.

References

- Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
- Altschul,S.F., Madden,T.L., Schäffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucl. Acids Res.*, **25**, 3389–3402.
- Attwood,T.K. and Beck,M.E. (1994) PRINTS—A protein motif fingerprint database. *Prot. Eng.*, **7**, 841–848.
- Bairoch,A. and Apweiler,R. (1996) The SWISS-PROT protein sequence data bank and its new supplement TREMBL. *Nucl. Acids Res.*, **24**, 21–25.
- Bateman,A., Birney,E., Durbin,R., Eddy,S.R., Finn,R.D. and Sonnhammer,E.L.L. (1999) Pfam 3.1: 1313 multiple alignments and profile HMMs match the majority of proteins. *Nucl. Acids Res.*, **27**, 260–262.
- Brown,M., Hughey,R., Krogh,A., Mian,I.S., Sjölander,K. and Hausler,D. (1993) Using Dirichlet mixture priors to derive hidden Markov models for protein families. In Hunter,L., Searls,D. and Shavlik,J. (eds), *Proceedings of the First International Conference on Intelligent Systems in Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 47–55.
- Castillo,E. (1988) *Extreme Value Theory in Engineering*. Academic Press, Boston.
- Corpet,F., Gouzy,J. and Kahn,D. (1999) Recent improvements of the ProDom database of protein domain families. *Nucl. Acids Res.*, **27**, 263–267.
- Fuchs,R. (1993) Block searches on VAX and Alpha computer systems. *Comput. Appl. Biosci.*, **9**, 587–591.
- Fuchs,R. (1994) Fast protein block searches. *Comput. Appl. Biosci.*, **10**, 79–80.
- Goldstein,L. and Waterman,M.S. (1994) Approximations to profile score distributions. *J. Comput. Biol.*, **1**, 93–104.
- Gracy,J. and Argos,P. (1998) Automated protein sequence database classification. I. Integration of compositional similarity search, local similarity search, and multiple sequence alignment. *Bioinformatics*, **14**, 164–173.
- Gribbskov,M., McLachlan,A.D. and Eisenberg,D. (1987) Profile analysis: Detection of distantly related proteins. *Proc. Natl Acad. Sci. USA*, **84**, 4355–4358.
- Hart,P.E., Nilsson,N.J. and Raphael,B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.*, **SSC-4**, 100–107.
- Henikoff,J.G. and Henikoff,S. (1996) Using substitution probabilities to improve position-specific scoring matrices. *Comput. Appl. Biosci.*, **12**, 135–143.
- Henikoff,S. (1996) Scores for sequence searches and alignments. *Curr. Opin. Struct. Biol.*, **6**, 353–360.
- Henikoff,S. and Henikoff,J.G. (1991) Automated assembly of protein blocks for database searching. *Nucl. Acids Res.*, **19**, 6565–6572.
- Henikoff,S., Henikoff,J.G., Alford,W.J. and Pietrokovski,S. (1995) Automated construction and graphical presentation of protein blocks from unaligned sequences. *Gene*, **163**, GC17–GC26.
- Henikoff,S., Henikoff,J.G. and Pietrokovski,S. (1999) Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, **15**, 471–479.
- Lawrence,C.E., Altschul,S.F., Boguski,M.S., Liu,J.S., Neuwald,A.F. and Wootton,J.C. (1993) Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.
- McLachlan,A.D. (1983) Analysis of gene duplication repeats in the myosin rod. *J. Mol. Biol.*, **169**, 15–30.
- Nevill-Manning,C., Sethi,K., Wu,T.D. and Brutlag,D.L. (1997) Enumerating and ranking discrete motifs. In *Proceedings of the Fifth International Conference on Intelligent Systems in Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 202–209.
- Nevill-Manning,C.G., Wu,T.D. and Brutlag,D.L. (1998) Highly specific protein sequence motifs for genome analysis. *Proc. Natl Acad. Sci. USA*, **95**, 5865–5871.
- Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.
- Pugh,W. (1990) Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, **33**, 668–676.
- Sjölander,K., Karplus,K., Brown,M., Hughey,R., Krogh,A. and Hausler,D. (1996) Dirichlet mixtures: A method for improved detection of weak but significant protein sequence homology. *Comput. Appl. Biosci.*, **12**, 327–345.

- Smith,H.O., Annau,T.M. and Chandrasegaran,S. (1990) Finding sequence motifs in groups of functionally related proteins. *Proc. Natl Acad. Sci. USA*, **87**, 826–230.
- Staden,R. (1989) Methods for calculating the probabilities of finding patterns in sequences. *Comput. Applic. Biosci.*, **5**, 89–96.
- Staden,R. (1990) Searching for patterns in protein and nucleic acid sequences. *Meth. Enzymol.*, **183**, 193–211.
- Stormo,G.D. and Hartzell III,G.W. (1989) Identifying protein-binding sites from unaligned DNA fragments. *Proc. Natl Acad. Sci. USA*, **86**, 1183–1187.
- Tatusov,R.L., Altschul,S.F. and Koonin,E.V. (1994) Detection of conserved segments in proteins: Iterative scanning of sequence databases with alignment blocks. *Proc. Natl Acad. Sci. USA*, **91**, 12091–12095.
- Wallace,J.C. and Henikoff,S. (1992) PATMAT: A searching and extraction program for sequence, pattern and block queries and databases. *Comput. Appl. Biosci.*, **8**, 249–254.
- Wu,T.D., Nevill-Manning,C.C. and Brutlag,D.L. (1999) Minimal-risk scoring matrices for sequence analysis. *J. Comput. Biol.*, **6**, 219–235.