JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 9, Number 1, 2002
Mary Ann Liebert, Inc.
Pp. 23–33

The Efficient Computation of Position-Specific Match Scores with the Fast Fourier Transform

S. RAJASEKARAN,[1] X. JIN,[1] and J.L. SPOUGE[2]

ABSTRACT

Historically, in computational biology the fast Fourier transform (FFT) has been used almost exclusively to count the number of exact letter matches between two biosequences. This paper presents an FFT algorithm that can compute the match score of a sequence against a position-specific scoring matrix (PSSM). Our algorithm finds the PSSM score simultaneously over all offsets of the PSSM with the sequence, although like all previous FFT algorithms, it still disallows gaps. Although our algorithm is presented in the context of global matching, it can be adapted to local matching without gaps. As a benchmark, our PSSM-modified FFT algorithm computed pairwise match scores. In timing experiments, our most efficient FFT implementation for pairwise scoring appeared to be 10 to 26 times faster than a traditional FFT implementation, with only a factor of 2 in the acceleration attributable to a previously known compression scheme. Many important algorithms for detecting biosequence similarities, e.g., gapped BLAST or PSIBLAST, have a heuristic screening phase that disallows gaps. This paper demonstrates that FFT algorithms merit reconsideration in these screening applications.

Key words: position-specific scoring matrices, fast Fourier transform, algorithm.

1. INTRODUCTION

The fast Fourier transform (FFT) is an astonishingly efficient algorithm (e.g., Press *et al.*, 1976). Accordingly, computational biologists have considered its use in at least two separate contexts. First, the Fourier transform detects periodicities within biosequences (Tavare and Giddings, 1989). In protein sequence analysis, e.g., it detects periodicities in profiles of amino acid hydrophobicity (Kubota *et al.*, 1981; McLachlan and Karn, 1983; Cornette *et al.*, 1987) and electron-ion potentials (Veljkovic *et al.*, 1985). Similarly, it detects compositional regularities in DNA sequences (Trifunov and Sussman, 1980; Silverman and Linsker, 1986; Chechetkin and Turygin, 1995; Korotkov *et al.*, 1997). Second, it detects similarities between pairs of biosequences (Felsenstein *et al.*, 1981; Liquori *et al.*, 1986; Benson, 1990; Cheever *et al.*, 1991). The detection of sequence similarity has become central to modern molecular biology. For example, the function of a newly sequenced gene is inferred when the corresponding protein sequence is similar to other, better known proteins in a sequence database (Needleman and Wunsch, 1970; Altschul

[1]Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611.
[2]National Center for Biotechnology Information, National Library of Medicine, Bethesda, MD 20894.

et al., 1997). It is therefore surprising, perhaps, that despite its astonishing efficiency, the FFT is rarely used for detecting biosequence similarity.

This apparent anomaly has a brief explanation. Historically, in computational biology, the FFT has been used almost exclusively to count the number of exact letter matches between two biosequences, when the sequences are aligned in an arbitrary offset with gaps disallowed (e.g., Gusfield, 1999). (See Fig. 1.) This usage may reflect computational biologists' early interest in edit distance, the minimum number of deletions, insertions, or letter substitutions that transform one sequence into another (Ukkonen, 1985).

On the other hand, experience has shown that to improve sensitivity, similarity searches in protein databases need to consider chemical and evolutionary relationships between pairs of amino acids (Feng *et al.*, 1984; Altschul *et al.*, 1994). Thus, protein searches use pairwise scoring matrices rather than exact matches to quantify similarity (Altschul, 1991, 1993). Scoring matrices also improve the sensitivity of nucleotide similarity searches (States *et al.*, 1991). Sensitivity also improves when gaps are allowed in sequence alignments (Altschul *et al.*, 1997; Altschul, 1999). Accordingly, because similarity searches using exact matches and disallowing gaps are known to be relatively insensitive, the FFT has become discredited as a useful tool for detecting biosequence similarities.

When scrutinized, however, the discredit has not been closely justified. In support of this statement, we present an FFT algorithm that can incorporate arbitrary scoring matrices for detecting pairwise similarity between biosequences. Although extant FFT algorithms can be modified to this end, they either make undesirable approximations (Liquori *et al.*, 1986; Cheever *et al.*, 1991) or are extremely inefficient by comparison (Felsenstein *et al.*, 1981; Gusfield, 1999) (see the results section). More importantly for current

A

a	a	t	c	a	g

B

t	c	t	g	t

S

a	-4	-4	-4	-4	-4
c	-4	5	-4	-4	-4
g	-4	-4	-4	5	-4
t	5	-4	5	-4	5

FIG. 1. A gapless global alignment between two sequences **A** = *aatcag* and **B** = *tctgt*. In the alignment shown, **B** is offset 2 to the right from **A**. The alignment shown has a total match count of 3 from the exact match **A** = *aa***tc***a***g** and **B** = **tc***t***g***t*. The edit distance between **A** and **B** is 4, with the corresponding edit operations as follows. Delete the initial two *a*'s in **A**, delete the final *t* in **B**, and substitute *g* for the third *a* in **A**. For pairwise scoring, the old BLAST default for DNA sequences gives a score 5 to an exact letter match and −4 to a mismatch. The global alignment shown matches four letters for a pairwise match score of $5+5-4+5=11$. The figure also shows a PSSM **S**, which corresponds to the old BLAST default for pairwise scoring when a sequence is aligned with **B**. Thus, the alignment shown between **A** and **S** has a PSSM score of $5+5-4+5=11$, which equals the pairwise score (as indeed it must). If the final three letters *tgt* in **B** (and these alone) are shifted one position to the right, an alignment allowing gaps is generated. If we consider only the subsequences *cag* in **A** and *ctg* in **B**, and ignore the relationships and scores for any other letters, a local alignment is generated. This paper considers only global alignments disallowing gaps: local alignments or alignments allowing gaps are not explicitly considered.

areas of interest, however, our FFT algorithm can match a protein sequence against the position-specific scoring matrix (PSSM) profiling a protein family (Gribskov *et al.*, 1987; Gribskov *et al.*, 1990; Krogh *et al.*, 1994; Barrett *et al.*, 1997; Karplus *et al.*, 1998). PSSM applications constitute an unexplored possibility for the FFT in computational biology.

Throughout this paper, the sequence similarities we describe involve global matches (alignments involving the entire length of two biosequences), as opposed to local matches (alignments of pairs of arbitrary subsequences). FFTs have a reputation of being restricted to global matching, although the reputation is again unmerited. The FFT can be adapted to local matching (Cheever *et al.*, 1991; Rajasekaran *et al.*, 2000).

Unfortunately, like all extant FFT algorithms, ours disallows gaps in its sequence alignments. On the other hand, many important algorithms for detecting biosequence similarities, e.g., gapped BLAST or PSIBLAST (Altschul *et al.*, 1997; Altschul and Koonin, 1998; Altschul, 1999) have a heuristic screening phase that likewise disallows gaps when searching for sequence similarity. This paper demonstrates that FFT algorithms merit reconsideration in these screening applications.

The remainder of the introduction gives an abbreviated formal description of the FFT in the context of the exact matching problems that it addressed historically. Thus grounded, our methods section then describes our PSSM modification to the FFT algorithm. The PSSM modification handles pairwise similarity matrices as a special case. The results section then uses timing tests to compare our PSSM-modified FFT algorithm to other FFT variants, and a discussion follows. Our Appendix contains some unimplemented FFT algorithms, including a potentially useful integer compression scheme.

We now briefly describe the FFT (Press *et al.*, 1976, p. 304). Consider $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$, two vectors of n complex numbers. The cyclic cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ is defined to be the complex vector $\mathbf{z}_{cyc} = (z_0, z_1, \ldots, z_{n-1})$, where $z_j = \sum_{k=0}^{n-1} x_{j+k} y_k$ and the subscript $j + k$ in the summation is taken modulo n (Gusfield, 1999). Less formally, set up a circular alignment of the vectors $\mathbf{x}$ and $\mathbf{y}$, i.e., place x_k directly over y_k in consecutive evenly spaced positions clockwise around a circle ($k = 0, 1, \ldots, n-1$). We can compute z_j, if we hold $\mathbf{x}$ in place and rotate $\mathbf{y}$ j positions clockwise, multiply the aligned coordinates (x_{j+k}, y_k) ($k = 0, 1, \ldots, n-1$), and then sum the resulting products. Some analogy with biosequence alignments is already apparent.

Given $\mathbf{x}$ and $\mathbf{y}$, we can compute the cyclic correlation $\mathbf{z}_{cyc}$ efficiently with the FFT. If $\omega = \exp(2\pi i/n)$ denotes the nth root of unity, the FFT of the complex vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ is the vector $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ whose coordinates are $X_j = x_0 + x_1\omega^j + \ldots + x_{n-1}\omega^{j(n-1)}$ ($j = 0, 1, \ldots, n-1$). (Following custom, we denote the FFT of a vector with the corresponding capital letter.) Let $\mathbf{y}_R = (y_{n-1}, y_{n-2}, \ldots, y_0)$ be the reverse of $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$, and let $\mathbf{x} \odot \mathbf{y} = (x_0 y_0, x_1 y_1, \ldots, x_{n-1} y_{n-1})$ denote the product of $\mathbf{x}$ and $\mathbf{y}$ coordinate by coordinate. It is well known (e.g., Horowitz *et al.*, 1998; Gusfield, 1999) that the FFT of $\mathbf{z}_{cyc}$ satisfies the equation $\mathbf{Z}_{cyc} = \mathbf{X} \odot \mathbf{Y}_R$, where $\mathbf{Y}_R$ denotes the FFT of $\mathbf{y}_R$ (not the reverse of the transform $\mathbf{Y}$). In summary, to find the cyclic cross-correlation $\mathbf{z}_{cyc}$, we reverse $\mathbf{y}$, find the FFTs $\mathbf{X}$ and $\mathbf{Y}_R$ of $\mathbf{x}$ and $\mathbf{y}_R$, take the product $\mathbf{Z}_{cyc} = \mathbf{X} \odot \mathbf{Y}_R$ of $\mathbf{X}$ and $\mathbf{Y}$ coordinate by coordinate, and then perform an FFT inversion. Although there are other mathematical transforms for detecting sequence periodicity, e.g., the Walsh transform (Tavare and Giddings, 1989), the FFT seems unique in its relation to cyclic cross-correlations.

Because linear alignments are more common than circular alignments in computational biology, linear cross-correlation is of greater interest to us than its circular cousin. Given two vectors $\mathbf{x} = (x_0, x_1, \ldots, x_{l-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{m-1})$, possibly of different lengths, pad $\mathbf{x}$ on the left with $m-1$ 0's and $\mathbf{y}$ on the right with $l-1$ 0's to form two vectors $\mathbf{x}' = (0, 0, \ldots, 0, x_0, x_1, \ldots, x_{l-1})$ and $\mathbf{y}' = (y_0, y_1, \ldots, y_{m-1}, 0, 0, \ldots, 0)$ of length $n = l + m - 1$. The (linear) cross-correlation $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ is the cyclic cross-correlation $\mathbf{z}'_{cyc}$ of $\mathbf{x}'$ and $\mathbf{y}'$. Less formally, set up a linear alignment of the vectors $\mathbf{x}$ and $\mathbf{y}$, i.e., place $\mathbf{x}$ on a line and $\mathbf{y}$ underneath it, with x_0 directly over y_{m-1} and the coordinates evenly spaced. We can compute z_j, if we hold $\mathbf{x}$ in place and move $\mathbf{y}$ j positions to the right, multiply the aligned coordinates from $\mathbf{x}$ and $\mathbf{y}$, and then sum the resulting products.

Because the computation of the cross-correlation $\mathbf{z}$ apparently requires $O(n^2)$ operations, the next fact is somewhat astonishing on first encounter. The FFT and FFT inversions above can be computed in $O(n \log n)$ time. The divide-and-conquer strategy that accomplishes this magic is standard textbook material (e.g., Press *et al.*, 1976; Horowitz *et al.*, 1998; Gusfield, 1999). An FFT algorithm for computing the cross-correlation $\mathbf{z}$, therefore, only requires $O(n \log n)$ time.

The FFT has been of some interest in biosequence analysis, because it solves the match-count problem for letter strings efficiently (Gusfield, 1999). Consider two sequences $\mathbf{A} = A_0A_1 \ldots A_{l-1}$ and $\mathbf{B} = B_0B_1 \ldots B_{m-1}$ drawn from an alphabet Λ with L letters, and select two (possibly equal) letters a and b from Λ. For each (linear) alignment between two sequences $\mathbf{A}$ and $\mathbf{B}$, compute a match count for (a, b), i.e., count the number of times the letter pair (a, b) occurs in the alignment. Here, we set up a (linear) alignment of the sequences $\mathbf{A}$ and $\mathbf{B}$, with $\mathbf{A}$ on a line and $\mathbf{B}$ underneath, starting with A_0 directly over B_{m-1} and the letters evenly spaced. Hold $\mathbf{A}$ in place and move $\mathbf{B}$ to the right j positions (this position is "the jth alignment," for $j = 0, 1, \ldots, n-1$). Count the number of times z_j that the letter pair (a, b) occurs in the alignment. The vector $\mathbf{z}_{ab} = (z_0, z_1, \ldots, z_{n-1})$ gives the match counts.

Our notation intentionally suggests that the match count is a special type of cross-correlation. Let $\mathbf{x}_a$ be an indicator vector of length l, with 1's wherever the letter a appears in the sequence $\mathbf{A}$. Let $\mathbf{y}_b$ be the corresponding indicator vector for the letter b appearing in the sequence $\mathbf{B}$. The cross-correlation $\mathbf{z}_{ab}$ of $\mathbf{x}_a$ and $\mathbf{y}_b$ gives the match counts for (a, b). In Fig. 1, e.g., consider the match count of the letter pair (a, t) for the DNA sequences $\mathbf{A} = aatcag$ ($l = 6$) and $\mathbf{B} = tctgt$ ($m = 5$). The relevant indicator vectors are $\mathbf{x}_a = (1, 1, 0, 0, 1, 0)$ and $\mathbf{y}_t = (1, 0, 1, 0, 1)$, and their cross-correlation $\mathbf{z}_{at} = (1, 1, 1, 1, 2, 1, 1, 0, 1, 0)$ gives the match count of (a, t) in $\mathbf{A}$ and $\mathbf{B}$.

Usually, the total match count $\mathbf{z}_{total} = \sum_{a \in \Lambda} \mathbf{z}_{aa}$ is of primary interest, e.g., for DNA sequences $\mathbf{z}_{total} = \mathbf{z}_{aa} + \mathbf{z}_{cc} + \mathbf{z}_{gg} + \mathbf{z}_{tt}$. In Fig. 1, e.g., the total match count is $\mathbf{z}_{total} = (0, 0, 1, 0, 1, 0, 3, 0, 0, 0)$. As shown above, a transformed match count like $\mathbf{Z}_{aa}$ can be computed with two FFTs. Thus, for a general alphabet of L letters, the computation of the total match count appears to require $2L$ FFTs. Because FFT inversion is a linear operation, however, only a single FFT inversion is required to compute $\mathbf{z}_{total} = \sum_{a \in \Lambda} \mathbf{z}_{aa}$ from the transformed sum $\mathbf{Z}_{total} = \sum_{a \in \Lambda} \mathbf{Z}_{aa}$.

A clever encoding scheme reduces the FFT operations required for computing match counts for several letter pairs (Cheever *et al.*, 1991). Instead of using a separate indicator vector for each letter, the scheme indicates the positions of two letters in a single vector, indicating the letters with $i = \sqrt{-1}$ as well as 1. In our example from Fig. 1, where $\mathbf{A} = aatcag$ and $\mathbf{B} = tctgt$, let $\mathbf{x}_{at} = (1, 1, i, 0, 1, 0)$ indicate the positions of the letters a and t in sequence $\mathbf{A}$, and similarly, $\mathbf{x}_{cg} = (0, 0, 0, 1, 0, i)$. The indicator vectors for sequence $\mathbf{B}$ are similar, except that $-i$ replaces i, so $\mathbf{y}_{at} = (-i, 0, -i, 0, -i)$ and $\mathbf{y}_{cg} = (0, 1, 0, -i, 0)$. The total match count $\mathbf{z}_{total}$ is the real part of $\mathbf{z}_{at} + \mathbf{z}_{cg}$, where $\mathbf{z}_{at}$ ($\mathbf{z}_{cg}$) is the cross-correlation of $\mathbf{x}_{at}$ and $\mathbf{y}_{at}$ ($\mathbf{x}_{cg}$ and $\mathbf{y}_{cg}$). The total match count $\mathbf{z}_{total}$ can therefore be obtained in $L = 4$ FFTs and one FFT inversion. If certain approximations are acceptable, the number of FFTs can be reduced even further (Cheever *et al.*, 1991).

For reasons given above, however, modern similarity searches rarely count exact matches. Rather, they use scores to indicate different degrees of mismatching. Let $S(a, b)$ be the element of an $L \times L$ scoring matrix that quantifies the similarity of the letters a and b. For DNA sequences, e.g., the old BLAST program defaulted to scoring an exact nucleotide match as 1 and a mismatch as -3; i.e., $S(a, a) = \ldots = S(t, t) = 1$ and $S(a, c) = S(a, g) = \ldots = S(t, g) = -3$ (see *http://www.ncbi.nlm.nih.gov/blast/html/blastcgihelp.html#other_advanced* for the new defaults). For proteins, the customary PAM or BLOSUM matrices reflect evolutionary relationships by assigning closely related amino acid pairs a positive score (Dayhoff *et al.*, 1978; Henikoff and Henikoff, 1993). The pairwise match score $\mathbf{z}_{pair} = (z_0, z_1, \ldots, z_{n-1})$ for sequences $\mathbf{A}$ and $\mathbf{B}$ is then of interest, where z_j is the sum of the scores of all letter pairs in the jth alignment.

The pairwise match score provides a convenient benchmark for our FFT algorithms. Note the equation $\mathbf{z}_{pair} = \sum_{a \in \Lambda} \sum_{b \in \Lambda} S(a, b)\mathbf{z}_{ab}$, where $\mathbf{z}_{ab}$ is the match count of the letter pair (a, b) in $\mathbf{A}$ and $\mathbf{B}$. "**Algorithm 0**," our benchmark FFT algorithm from the literature (Benson, 1990), calculates $\mathbf{z}_{pair}$ from this equation: compute the match counts $\mathbf{z}_{ab}$ with the FFT and then compute their $S(a, b)$-weighted sum. Its time is dominated by the L^2 FFTs required to compute $\mathbf{z}_{ab}$ for each letter pair (a, b) (because the transformed sum $\mathbf{Z}_{pair} = \sum_{a \in \Lambda} \sum_{b \in \Lambda} S(a, b)\mathbf{Z}_{ab}$ only requires one FFT inversion). Thus, Algorithm 0 requires $O(L^2 n \log n)$ time.

In our context, the pairwise match score $\mathbf{z}_{pair}$ can be considered a specialized position-specific match score, which we define as follows. Let $\mathbf{A} = A_0A_1 \ldots A_{l-1}$ be a sequence, drawn as usual from our alphabet Λ of L letters. Let $S_0, S_1, \ldots, S_{m-1}$ be a sequence of scoring functions, so that S_k assigns a score $S_k(a)$ to each letter $a \in \Lambda$. Now, view the values $S_k(a)$ ($a \in \Lambda$) as the kth column of a position-specific scoring matrix (PSSM) $\mathbf{S} = (S_0, S_1, \ldots, S_{m-1})$. The position-specific match score $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ for the

sequence **A** and the PSSM **S** can be derived just like the match count or score for two sequences **A** and **B**. (See Fig. 1.) With $\mathbf{S} = (S_0, S_1, \ldots, S_{m-1})$ as a formal sequence of column vectors, construct the jth linear alignment of **A** and **S** as described above. The jth position-specific match score is the sum of $S_{k'}(A_k)$ over all aligned pairs $(A_k, S_{k'})$.

More formally, let $n = l + m - 1$ as usual. With the coordinate index j running from 0 to $n - 1$,

$$z_j = \sum_k S_{k'}(A_k), \tag{1}$$

where within the sum, the difference $k' - k$ is fixed at $m - 1 - j$, and the remaining, free index k runs from $\max\{0, j - (m - 1)\}$ to $\min\{l - 1, j\}$. For brevity, later equations do not mention the indices and their ranges. They are always the same.

Note that the PSSM score contains all historical uses of the FFT as special cases. Let I be the pairwise scoring matrix for exact matching, so $I(a, b) = 1$ if $a = b$, and $I(a, b) = 0$ otherwise. Note that the matrix I is symmetric: $I(a, b) = I(b, a)$. Match counts are the specialized PSSM $S_k(a) = I(a, B_k)$ $(k = 0, 1, \ldots, m - 1)$, where $\mathbf{B} = B_0 B_1 \ldots B_{m-1}$ is the second match sequence. More generally, pairwise match scores are the specialized PSSM $S_k(a) = S(a, B_k)$. Equation (1) therefore provides a formal definition for match counts and pairwise match scores.

With this preamble, the methods section will give our PSSM-modified FFT algorithms. When applied to pairwise scoring, they compute $\mathbf{z}_{pair}$ in $O(Ln \log n)$ time, a factor of L faster than the previous $O(L^2 n \log n)$ algorithms.

2. METHODS

2.1. Description of the algorithms

Algorithm 1 is as follows. Fix $a \in \Lambda$. Let $\mathbf{x}_a = (I(a, A_0), I(a, A_1), \ldots, I(a, A_{l-1}))$ be the indicator vector of the letter a in the sequence **A**, and let $\mathbf{y}_a = \mathbf{S}_a = (S_0(a), S_1(a), \ldots, S_{m-1}(a))$ be the corresponding row of the PSSM **S**. The cross-correlation $\mathbf{z}_a = (z_{a0}, z_{a1}, \ldots, z_{a,n-1})$ of $\mathbf{x}_a$ and $\mathbf{y}_a$ equals the position-specific match score attributable to the letter a. The jth coordinate of the position-specific match score attributable to the letter a is

$$z_{aj} = \sum_k I(a, A_k) S_{k'}(a). \tag{2}$$

Reversal of summation order gives $\sum_{a\in\Lambda} z_{aj} = \sum_{a\in\Lambda} \sum_k I(a, A_k) S_{k'}(a) = \sum_k S_{k'}(A_k) = z_j$ in Equation (1). Thus, the position-specific match score $\mathbf{z} = \sum_{a\in\Lambda} \mathbf{z}_a$.

Algorithm 1 can be implemented with the FFT as described in the Introduction. For each $a \in \Lambda$, the transform $\mathbf{Z}_a$ of $\mathbf{z}_a$ can be computed with one FFT. One FFT inversion is required to recover the position-specific match score $\mathbf{z}$ from $\mathbf{Z} = \sum_{a\in\Lambda} \mathbf{Z}_a$. Thus, the computational time is $O(Ln \log n)$, as advertised.

Algorithm 2 is a variant of Algorithm 1, using the compression scheme of Cheever *et al.* (1991) that indicates the positions of two letters with a single vector (hence Algorithm "two"). The Appendix gives a variant compression scheme, Algorithm 2.1.

Fix two different letters $a, b \in \Lambda$. In this encoding, the indicator vector $\mathbf{x}_{ab}$ for the letters a and b in sequence **A** has the coordinates $x_{ab,j} = I(a, A_j) + iI(b, A_j)$, and with $\mathbf{S}_a$ and $\mathbf{S}_b$ defined as for Algorithm 1, $\mathbf{y}_{ab} = \mathbf{S}_a - i\mathbf{S}_b$. The cross-correlation $\mathbf{z}_{ab} = (z_{ab,0}, z_{ab,1}, \ldots, z_{ab,n-1})$ of $\mathbf{x}_{ab}$ and $\mathbf{y}_{ab}$ has coordinates

$$z_{ab,j} = \sum_k \{I(a, A_k) S_{k'}(a) + I(b, A_k) S_{k'}(b)\} - i \sum_k \{I(a, A_k) S_{k'}(b) - I(b, A_k) S_{k'}(a)\}. \tag{3}$$

Comparison with Equation (2) shows that the real part $\mathbf{z}_{ab}$ equals $\mathbf{z}_a + \mathbf{z}_b$ from Algorithm 1. Again, the computational time is $O(Ln \log n)$.

2.2. Implementation of the algorithms

We now give our implementations of Algorithms 0, 1, and 2, which we call Implementations 0, 1, and 2. Note that Implementation 0 was a traditional approach (Benson, 1990), which computes one FFT for every letter pair. We describe our implementations for the protein alphabet ($L = 20$ letters), although they clearly generalize. Throughout the discourse, we therefore consider two protein sequences $\mathbf{A} = A_0A_1 \ldots A_{l-1}$ and $\mathbf{B} = B_0B_1 \ldots B_{m-1}$.

Implementation 0 was an implementation of Algorithm 0, which computes a match score for each amino acid pair (a, b). For each (a, b), we took two linear arrays $\mathbf{x} = (x_0, x_1, \ldots, x_{n-2})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-2})$ $(n = l + m - 1)$. The array $\mathbf{x}$ indicated the appearance of a in $\mathbf{A}$, whereas the array $\mathbf{y}$ indicated the appearance of b in $\mathbf{B}$. Both $\mathbf{x}$ and $\mathbf{y}$ were initialized with 0's. For each k from 0 to $l - 1$, the number x_{m-1+k} was set to 1 if $A_k = a$. Similarly, for each k from 0 to $m - 1$, the number y_k was set to 1 if $B_k = b$. The array $\mathbf{x}$ was therefore padded with 0's on the left like $\mathbf{x}'$ in the introduction; $\mathbf{y}$, on the right like $\mathbf{y}'$.

Next, as described in the introduction, we computed the cyclic cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ with a textbook C++ program for the FFT (Horowitz *et al.*, 1997). The resulting match count was stored in an array $\mathbf{z}$. All elements in the array $\mathbf{z}$ were then multiplied by the similarity score $S(a, b)$, so $\mathbf{z}$ now contained the contribution to the match score of $\mathbf{A}$ and $\mathbf{B}$ from the amino acid pair (a, b). This computation was repeated for every pair (a, b) of amino acids, with the overall match score accumulating in a separate array.

Implementation 1 was an implementation of Algorithm 1. We took two linear arrays $\mathbf{x}$ and $\mathbf{y}$ of length $L(n - 1)$ each. The array $\mathbf{x}$ contained $n - 1$ structures $\mathbf{x}_k$, indexed by k from 0 to $n - 2$. Each structure $\mathbf{x}_k$ was a linear array of $L = 20$ numbers, indexed from 1 to $L = 20$. The entire array $\mathbf{x}$ was initialized with 0's. As in Implementation 0, the structure $\mathbf{x}_{m-1+k}$ corresponded to the letter A_k $(k = 0, \ldots, l - 1)$. The exact correspondence was established as follows.

First, we imposed an arbitrary order on the $L = 20$ amino acids. The alphabetical order

```
ACDEFGHIKLMNPQRSTVWY
```

of the standard single-letter codes was convenient. Accordingly, alanine (A) was the first amino acid, cysteine (C) was the second amino acid, etc. Then, if A_k was the jth amino acid, we set the jth element of the $\mathbf{x}_{m-1+k}$ to 1. If sequence $\mathbf{A}$ began with DGA, e.g., the three structures $\mathbf{x}_{m-1+k}$ $(k = 0, 1, 2)$ were

```
ACDEFGHIKLMNPQRSTVWY
00100000000000000000

ACDEFGHIKLMNPQRSTVWY
00000100000000000000

ACDEFGHIKLMNPQRSTVWY
10000000000000000000
```

where the structures appear on separate lines underneath the alphabetical order.

The array $\mathbf{y}$ also contained $n - 1$ structures $\mathbf{y}_k$, indexed by k from 0 to $n - 2$. Each structure $\mathbf{y}_k$ contained the following $L = 20$ elements: $S_k(A), S_k(C), S_k(D), \ldots, S_k(Y)$, the scores of amino acids $A, C, D, \ldots, Y$ (alanine, cysteine, glutamate, ..., tyrosine) when matched against the amino acid B_k. The cyclic cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ is the match score. Note that in this implementation of Algorithm 1, only one product in $L = 20$ from the cross-correlation actually contributes to the match score. Algorithm 1 has a natural implementation with the theoretical time $O(Ln \log n)$, slightly better than the theoretical time of $O\{Ln \log(Ln)\}$ for Implementation 1. Implementation 1 was for us more convenient, however.

Implementation 2 is similar to Implementation 1 except that we used complex indicators to reduce the lengths of $\mathbf{x}$ and $\mathbf{y}$. We took two linear arrays $\mathbf{x}$ and $\mathbf{y}$ of length $\frac{1}{2}L(n - 1)$ each. The array $\mathbf{x}$ contained $n - 1$ structures $\mathbf{x}_k$, indexed by k from 0 to $n - 2$. Each structure $\mathbf{x}_k$ was a linear array of $\frac{1}{2}L = 10$

numbers, indexed from 1 to $\frac{1}{2}L = 10$. The entire array $\mathbf{x}$ was initialized with 0's. As in Implementation 0, the structure $\mathbf{x}_{m-1+k}$ corresponded to the letter A_k $(k = 0, \ldots, l-1)$. The exact correspondence was established as follows.

First, we paired the amino acids in alphabetical order. The following shows the first letter of each pair above the second.

```
ADFHKMPRTW
CEGILNQSVY
```

On one hand, if A_k was the $(2j-1)$th amino acid, then we set the jth element of the $\mathbf{x}_{m-1+k}$ to 1. If A_k was the $(2j)$th amino acid, then we set the jth element of the $\mathbf{x}_{m-1+k}$ to i. If sequence $\mathbf{A}$ began with DAG, e.g., then the three structures $\mathbf{x}_{m-1+k}$ $(k = 0, 1, 2)$ were

```
aDFHKMPRTW  ADFHKMPRTW  ADFHKMPRTW
01000000000 00i0000000 1000000000
CEGILNQSVY  CEGILNQSVY  CEGILNQSVY
```

where the structures (separated by spaces here) appear between the alternate halves of the alphabetical order.

The array $\mathbf{y}$ also contained $n-1$ structures $\mathbf{y}_k$, indexed by k from 0 to $n-2$. The structure $\mathbf{y}_k$ contained the following $\frac{1}{2}L = 10$ elements: $S_k(A) - iS_k(C)$, $S_k(D) - iS_k(E, \ldots, S_k(W) - iS_k(Y)$. The real part of the cyclic cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ is the match score. Note again that only one product in $\frac{1}{2}L = 10$ from the cross-correlation actually contributes to the match score.

Like Implementation 1, Implementation 2 has time $O\{Ln\log(Ln)\}$ and is convenient but slower than the theoretical time $O(Ln\log n)$ for Algorithm 2.

3. RESULTS

We timed Implementations 0, 1, and 2 with randomly generated sequences as well as sequences downloaded from various NIH databases. This section gives the timing results.

3.1. Simulated protein sequences

Figure 2 gives our timing of the three Implementations with simulated protein sequences. Our FFT implementation always pads its array sizes up to an integer power of 2. Without the padding, the arrays $\mathbf{x}$ and $\mathbf{y}$ needed in Implementation 0 are of size $n-1$; in Implementation 1, of size $L(n-1)$; and in Implementation 2, of size $\frac{1}{2}L(n-1)$. Here, $L = 20$ and $n = l + m - 1$ is one less than twice the sequence length.

Implementation 0 therefore uses FFT array sizes of 64 for sequences of lengths 25 and 32. Implementation 1 uses array sizes of 1,024 for sequences of length 25, doubling to 2,048 for sequences of length 32. Implementation 2 always requires half the array size of Implementation 1. When sequence lengths were scaled up by factors of 8, 16, 32, and 64 (i.e., powers of 2), the required array sizes scaled accordingly. The corresponding choices of sequence lengths (shown in Fig. 2) permit a comparison of Implementation 0 and 1 under conditions that favor first one implementation and then the other.

Figure 2 can be summarized as follows. Implementation 1 is between about 5 to 12 times faster than Implementation 0, depending on whether the sequence lengths favor it or not. Implementation 2 is about twice as fast as Implementation 1. All comparisons between the implementations are consistent, regardless of the scale of sequence length.

3.2. Real protein sequences

We downloaded six proteins from *http://www.ncbi.nlm.nih.gov.* Their accession numbers, followed in parentheses by their sequence lengths, were AAD14597.1 (440), CAA56071.1 (445), BAA34431.1 (622),

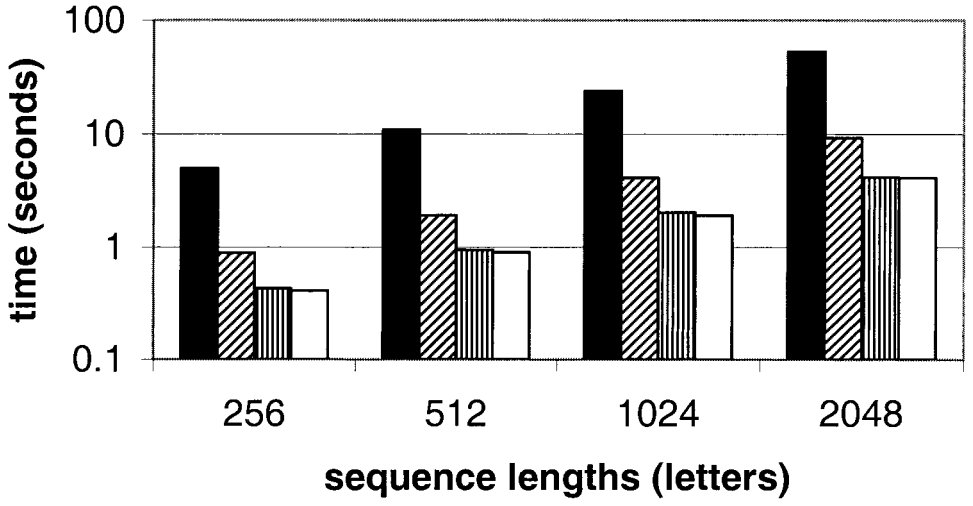

FIG. 2. A timing comparison of the three Implementations with simulated protein sequences. Sequence lengths are shown on the X-axis; the corresponding timing results, on the logarithmic Y-axis. All times shown are the averages over ten random pairs. For each of the sequence lengths 256, 512, 1024, and 2048, ten pairs of random sequences were generated. The corresponding times are given by the black bars for Implementation 0, the diagonally striped bars for Implementation 1, and by the white bars for Implementation 2. For reasons given in the text, ten pairs of random sequences were also generated for each of the sequence lengths 200, 400, 800, and 1600. The corresponding times for Implementation 1 are given by the vertically striped bars.

AAB65242.1 (670), AAB60937.1 (1224), and BAA13219.1 (1496). We selected these protein sequences because they form three pairs of similar sizes: (440,445), (622,670), and (1224,1496).

Figure 3, which gives our timing of the three Implementations with these real protein sequences, can be summarized as follows. Implementation 1 is about 12 times faster than Implementation 0. Implementation 2 is about 26 times faster than Implementation 0. All comparisons between the implementations are consistent, regardless of the sequence lengths.

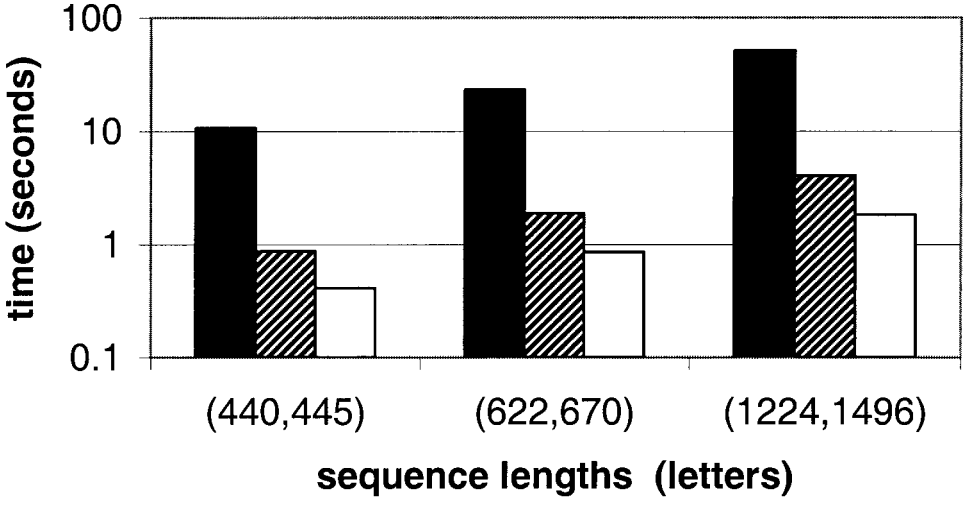

FIG. 3. A timing comparison of the three Implementations with real protein sequences. The pairs of sequence lengths are shown on the X-axis; the corresponding timing results, on the logarithmic Y-axis. The corresponding times are given by the black bars for Implementation 0, by the diagonally striped bars for Implementation 1, and by the white bars for Implementation 2.

4. DISCUSSION

In this paper, we modified the FFT for the efficient computation of PSSM scores. Our PSSM-modified FFT algorithm computes PSSM scores simultaneously over all alignment offsets between a sequence and a PSSM. Within the context of our algorithm, match counts and pairwise scoring are merely special PSSMs. Traditional FFT algorithms cannot handle PSSMs. Thus, we needed to use traditional FFT algorithms with pairwise scoring as benchmarks for our PSSM-modified FFT algorithm. Our FFT algorithm for pairwise scoring appeared to be 5 to 12 times faster than the traditional algorithm (compare Implementation 0 and Implementation 1 in the results section).

A previously known compression scheme (Cheever *et al.*, 1991) made our FFT algorithm twice as fast as before: other improvements to our implementations are also readily available. For example, the appendix gives Algorithm 3, a compression scheme. In addition, although the Introduction describes the FFT operation within the complex number field, the FFT can also be performed within the field of integers modulo a prime number. We did not explore the efficiencies of integer arithmetic, but it sometimes speeds FFT computations considerably (Benson, 1990). Because specialized, ultra-fast hardware is also available for digital signal processing, our PSSM-modified FFT algorithm has considerable potential for further acceleration.

Although our PSSM-modified FFT algorithm was presented in the context of global matches, FFT algorithms can be adapted to local matching (Cheever *et al.*, 1991; Rajasekaran *et al.*, 2000). Thus, the disallowance of gaps in FFT matching remains the main obstacle to wide application in bioinformatics. On the other hand, many important algorithms for detecting biosequence similarities, e.g., gapped BLAST or PSIBLAST (Altschul *et al.*, 1997; Altschul and Koonin, 1998; Altschul, 1999), have a heuristic screening phase that also disallows gaps. The FFT might find uses in such applications.

APPENDIX

Algorithm 2.1 is one of many possible variants of Algorithm 2. Define the vectors $\mathbf{x}_{ab}$ and $\mathbf{S}_a$ just as in Algorithm 2, but now define $\mathbf{y}_{ab} = \mathbf{S}_{ab} = \frac{1}{2}(1+i)\mathbf{S}_a + \frac{1}{2}(1-i)\mathbf{S}_b$. The cross-correlation $\mathbf{z}_{ab} = (z_{ab,0}, z_{ab,1}, \ldots, z_{ab,n-1})$ of $\mathbf{x}_{ab}$ and $\mathbf{y}_{ab}$ has coordinates

$$\begin{aligned} z_{ab,j} = {} & \frac{1}{2}(1+i)\sum_k \{I(a, A_k)S_{k'}(a) + I(b, A_k)S_{k'}(b)\} \\ & + \frac{1}{2}(1-i)\sum_k \{I(a, A_k)S_{k'}(b) - I(b, A_k)S_{k'}(a)\}. \end{aligned} \tag{4}$$

The real and imaginary parts of $z_{ab,j}$ sum to $\sum_k \{I(a, A_k)S_{k'}(a) + I(b, A_k)S_{k'}(b)\} = z_{aj} + z_{bj}$ from Equation (2).

Our point here is that indicators can be encoded into complex vectors in many ways. Algorithm 2 is simpler than (and therefore preferable to) Algorithm 2.1, however.

Algorithm 3 is a variant of Algorithm 1 that applies to PSSMs $\mathbf{S}$ with integer entries $S_k(a)$ ($a \in \Lambda$ and $k = 0, 1, \ldots, m-1$). Algorithm 3 might be especially useful when the FFT is performed within the field of integers modulo a prime number (Benson, 1990). In a search for local similarity (Cheever *et al.*, 1991), $n = l + m - 1$ might be particularly small. For convenience and without loss of generality, we replace the L letters $a \in \Lambda$ by the numbers $a = 0, 1, \ldots, L-1$.

Algorithm 3 is a compression scheme modulo R, where the integer radix R bounds the range of the cross-correlations z_{aj} in Equation (2), i.e., $0 \le \sum_k \max_a S_{k'}(a) - \sum_k \min_a S_{k'}(a) < R$. Such bounds are easily estimated.

Subtract the constant $c_k = \min_a S_k(a)$ from each column in $\mathbf{S}$ to form a matrix $\mathbf{S}'$, i.e., $S'_k(a) = S_k(a) - c_k$ for $a = 0, 1, \ldots, L-1$. The elements of the matrix $\mathbf{S}'$ are all zeros or counting numbers, because $\min_a S'_k(a) = \min_a S_k(a) - c_k = 0$. The matrix $\mathbf{S}'$ also satisfies $0 = \sum_k \min_a S'_{k'}(a) \le \sum_k \max_a S'_{k'}(a) < R$.

Define vectors $\mathbf{x}$ and $\mathbf{y}$ with integer coordinates $x_k = \sum_{a=0}^{L-1} I(a, A_k)R^{L-1-a}$ $(k = 0, 1, \ldots, l-1)$ and $y_k = \sum_{a=0}^{L-1} S'_k(a)R^a$ $(k = 0, 1, \ldots, m-1)$. Define also the vector $\mathbf{y}_c = (c_0, c_1, \ldots, c_{m-1})$. Then the cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$, where the integer

$$z_j = \sum_k \left\{ \sum_{a=0}^{L-1} I(a, A_k)R^{L-1-a} \right\} \left\{ \sum_{a=0}^{L-1} S'_{k'}(a)R^a \right\} = \sum_{a=0}^{L-1} \left\{ \sum_k S'_{k'}(a) \right\} R^{L-1+a-A_k}. \tag{5}$$

Because $0 \le \sum_k S'_{k'}(a) < R$ for all k' and all $a = 0, 1, \ldots, L-1$, the Lth digit of z_j in base R (i.e., the term $a = A_k$ from the right side of Equation (5)) is $\sum_k S'_{k'}(A_k)$. Let $\mathbf{z}_{(L)}$ be the vector of Lth digits from z_j in base R, and let $\mathbf{z}_c$ be the cross-correlation of $(1, 1, \ldots, 1)$ and $\mathbf{y}_c$. The coordinates of the vector $\mathbf{z}_{(L)} + \mathbf{z}_c$ are the desired PSSM scores $\sum_k S_{k'}(A_k) = \sum_k \{S'_{k'}(A_k) + c_{k'}\}$.

The time required for Algorithm 3 is the sum of $O(Lm)$ for the computation of $\mathbf{S}'$ and $O(n \log n)$ for the FFTs, as long as the denser encoding does not increase the time for computing the integer additions and multiplications.

REFERENCES

Altschul, S.F. 1991. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.* 219, 555–565.

Altschul, S.F. 1993. A protein alignment scoring system sensitive at all evolutionary distances. *J. Mol. Evol.* 36, 290–300.

Altschul, S.F. 1999. Hot papers—Bioinformatics—Gapped BLAST and PSI-BLAST: A new generation of protein database search programs by S.F. Altschul, T.L. Madden, A.A. Schaffer, J.H. Zhang, Z. Zhang, W. Miller, D.J. Lipman—Comments. *Scientist.* 13, p. 15.

Altschul, S.F., Boguski, M.S., Gish, W., and Wootton, J.C. 1994. Issues in searching molecular sequence databases. *Nature Genet.* 6, 119–129.

Altschul, S.F., and Koonin, E.V. 1998. Iterated profile searches with PSI-BLAST—a tool for discovery in protein databases. *Trends Biochem. Sci.* 23, 444–447.

Altschul, S.F., Madden, T.L., Schaffer, 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.

Barrett, C., Hughey, R., and Karplus, K. 1997. Scoring hidden Markov models. *Computer Appl. Biosci.* 13, 191–199.

Benson, D.C. 1990. Fourier methods for biosequence analysis. *Nucl. Acids Res.* 18, 6305–6310.

Chechetkin, V.R., and Turygin, A.Y. 1995. Search of hidden periodicities in DNA-sequences. *J. Theor. Biol.* 175, 477–494.

Cheever, E.A., Overton, G.C., and Searls, D.B. 1991. Fast Fourier transform-based correlation of DNA sequences using complex plane encoding. *Comput. Appl. Biosci.* 7, 143–154.

Cornette, J.L., Cease, K.B., Margalit, H., Spouge, J.L., Berzofsky, J.A., and Delisi, C. 1987. Hydrophobicity scales and computational techniques for detecting amphipathic structures in proteins. *J. Mol. Biol.* 195, 659–685.

Dayhoff, M.O., Schwartz, R.M., and Orcutt, B.C. eds. 1978. A model of evolutionary change in proteins, 345–352. *Atlas of Protein Sequence and Structure*, Supp. 3, National Biomedical Research Foundation, Silver Spring, MD.

Felsenstein, J.S., Sawyer, S., and Kochin, R. 1981. An efficient method for matching nucleic acid sequences. *Nucl. Acids Res.* 10, 133–139.

Feng, D.F., Johnson, M.S., and Doolittle, R.F. 1984. Aligning amino acid sequences: comparison of commonly used methods. *J. Mol. Evol.* 21, 112–125.

Gribskov, M., Luthy, R., and Eisenberg, D. 1990. Profile analysis. *Methods Enzymol.* 183, 146–159.

Gribskov, M., McLachlan, A.D., and Eisenberg, D. 1987. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA* 84, 4355–4358.

Gusfield, D. 1999. *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge.

Henikoff, S., and Henikoff, J.G. 1993. Performance evaluation of amino acid substitution matrices. *Proteins* 17, 49–61.

Horowitz, E., Sahni, S., and Rajasekaran, S. 1997. *Computer Algorithms/C++*, W. H. Freeman Press, New York.

Horowitz, E., Sahni, S., and Rajasekaran, S. 1998. *Computer Algorithms*, W. H. Freeman Press, New York.

Karplus, K., Barrett, C., and Hughey, R. 1998. Hidden Markov models for detecting remote protein homologies. *Bioinformatics* 14, 846–856.

Korotkov, E.V., Korotkova, M.A., and Tulko, J.S. 1997. Latent sequence periodicity of some oncogenes and DNA-binding protein genes. *Comput. Appl. Biosci.* 13, 37–44.

Krogh, A., Brown, M., Mian, I.S., Sjolander, K., and Haussler, D. 1994. Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.* 235, 1501–1531.

Kubota, Y., Takahashi, S., Nishikawa, K., and Ooi, T. 1981. Homology in protein sequencs expressed by correlation coefficients. *J. Theor. Biol.* 91, 347–361.

Liquori, A.M., Ripamonti, A., Sadun, C., Ottani, S., and Braga, D. 1986. Pattern recognition of sequence similarities in globular proteins by Fourier analysis: A novel approach to molecular evolution. *J. Mol. Evol.* 23, 80–87.

McLachlan, A.D., and Karn, J. 1983. Periodic features in the amino acid sequence of nematode myosin rod. *J. Mol. Biol.* 164, 605–626.

Needleman, S.B., and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48, 443–453.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. 1976. *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Rajasekaran, S., Nick, H., Pardalos, P.M., Sahni, S., and Shaw, G. 2000. Efficient algorithms for local alignment search. *J. Combinatorial Optimization.* 5, 117–124.

Silverman, B.D., and Linsker, R. 1986. A measure of DNA periodicity. *J. Theor. Biol.* 118, 295–300.

States, D.J., Gish, W., and Altschul, S.F. 1991. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods* 3, 66–70.

Tavare, S., and Giddings, B. 1989. Some statistical aspects of the primary structure of nucleotide sequences. In M.S. Waterman, eds., *Mathematical Methods for DNA Sequences*, 117–132, CRC Press, Boca Raton, FL.

Trifunov, E.N., and Sussman, J.L. 1980. The pitch of chromatin DNA is reflected in its nucleotide sequence. *Proc. Natl. Acad. Sci.* 77, 3816–3820.

Ukkonen, E. 1985. Algorithms for approximate string matching. *Information and Control* 64, 100–118.

Veljkovic, V., Cosic, I., Dimitrijevic, B., and Lalovic, D. 1985. Is it possible to analyze DNA and protein sequences by the methods of digital signal processing? *IEEE Trans. Biomed. Eng.* 32, 337–341.

Address correspondence to:
J.L. Spouge
National Center for Biotechnology Information
National Library of Medicine
Bethesda, MD 20894

E-mail: spouge@nih.gov