

Novel FFT Techniques for Sequence Analysis

Sanguthevar Rajasekaran

Xioming Jin

John L. Spouge

Dept. of CISE, Univ. of Florida

NCBI, NLM, NIH

Gainesville, FL 32611

Bethesda, MD 20894

Abstract. Algorithms for computing Fast Fourier Transforms (FFTs) have been employed in the past for sequence analysis. However these techniques had to perform a large number of FFT computations. In this paper we present algorithms for sequence analysis that reduce the number of FFT computations drastically. We also provide experimental evaluation of the new algorithms. The speedups obtained are impressive.

1. Introduction

The problem of identifying similarities among biological sequences has numerous applications such as inferring the functionality of a newly sequenced gene [1]. The similarity between two given biological sequences can be defined in a number of ways. For example, we can use the minimum edit distance between two sequences as a measure of their similarity. Here the minimum edit distance refers to the least number of deletions, insertions, or replacements needed to transform one sequence into the other.

Another measure of similarity can be computed as follows. There is a matrix M that assigns a score for every pair of bases. Given two sequences A and B , for each possible alignment between the two we compute the total score and pick the alignment with the maximum score. FFT algorithms can be employed to compute the score for each possible alignment.

The similarity measure can either be global or local. Global similarity refers to the similarity between the two sequences as a whole. But often, the biological similarity

between two sequences is dictated by smaller subsequences. If we use a global measure of similarity, local similarities might get unnoticed. For this reason, it is desirable to compute all the subsequence pairs for which the local similarities are high.

In particular, given two sequences A and B , we are interested in identifying all the pairs (A', B') where A' is a subsequence of A , B' is a subsequence of B , both A' and B' are of the same length, the similarity score between A' and B' is at least S (for some specified S), and these two subsequences are maximal, i.e., they can neither be expanded or shrunk to increase the similarity score. Any such pair is called a *Maximal Segment Pair* (MSP). We assume that a similarity score matrix such as PAM [2] has been given. BLAST is a system that is widely used to obtain MSPs [3].

Though it is usually believed that FFT techniques can only be used for computing global similarities, it has been shown in [4] that FFT techniques can also be utilized for identifying MSPs. Traditional approaches to sequence analysis using FFTs call for a large number of applications of the FFT algorithm. In particular, if A is the alphabet size of the sequences under concern, previous approaches will call the FFT algorithm A^2 times. Thus these approaches take a very long time especially when A is large (protein sequences being an example). In this paper we provide novel compression schemes that reduce the number of FFT calls to just A . We also provide an experimental comparison of the earlier algorithms with ours. The results are impressive.

2. Fast Fourier Transforms

Let $b = b_0, b_1, \dots, b_n$ be any sequence of elements from a field. We can associate a polynomial of degree n with any such sequence and vice-versa. For example, b can be associated with $f(x) = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$. The FFT of b is defined to be $B = B_0, B_1, \dots, B_n$ where $B_i = f(w^i)$, for $0 \leq i \leq n$, w being the primitive n th root of unity.

If $d = d_0, d_1, \dots, d_n$ is another sequence, then the convolution of b and d , denoted $b \otimes d$, is $e = e_0, e_1, \dots, e_n$ where $e_i = \sum_{j=0}^{n-i} b_{j+i} d_j$. Imagine b and d being perfectly aligned. If we keep b static and move d i -positions to the right, multiply the corresponding elements, and sum the products, we get e_i . Thus the convolution can be used to compute the similarities between two given sequences for each possible alignment.

We can compute $b \otimes d$ for a given b and d using efficient algorithms for FFT. It is known that $b \otimes d = FFT^{-1}(FFT(b) \bullet FFT(d^R))$ (see e.g., [5]). Here FFT^{-1} is the inverse FFT, d^R stands for the reverse of d , and \bullet is the dot product operator. (If $x = x_0, x_1, \dots, x_n$ and $y = y_0, y_1, \dots, y_n$ then $x \bullet y = x_0 y_0, x_1 y_1, \dots, x_n y_n$) The following Lemma is well known (see e.g., [5]).

Lemma 1. FFT and inverse FFT operations and hence the convolution on an n -element sequences can be performed in $O(n \log n)$ time.

Consider two sequences (e.g., protein or DNA) $x = x_0, x_1, \dots, x_n$ and $y = y_0, y_1, \dots, y_n$. We can use the convolution to perform the following task: For each possible alignment between x and y compute the number of matches and mismatches. In general there may be different scores associated with matches and mismatches. In BLAST, for DNA sequences, a match has a score of 5 and a mismatch has a score of -4. But any other scores are possible as well. For protein sequences, the PAM matrix is used. One way of performing this task is to do it in four stages, one for each possible base.

For example, let $x = g, g, t, a, c, c, t, g, a, a$ and $y = a, g, c, a, t, g, c, a, t, a$. We can compute the number of matching pairs of the base a for each alignment by performing a convolution of the sequences 0,0,0,1,0,0,0,0,1,1 and 1,0,0,1,0,0,0,1,0,1. The same can be repeated for the other bases as well. Thus there are eight FFT and one inverse FFT calculations involved. In the case of computing similarity scores for each possible alignment (where

the score for each possible base-pair can be different, notice that an FFT has to be computed for each pair of base pairs. Thus if A is the alphabet size, then the number of FFT computations is $\Omega(A^2)$.

For the case of computing the number of matching pairs for each alignment, we can reduce the number of FFT calculations using a clever encoding scheme (see e.g. [6]). Instead of using one sequence for each base we can use one sequence for two bases. The bases g and c in x can be encoded as $x_1 = 1,1,0,0,i,i,0,1,0,0$, where $i = \sqrt{-1}$. Similarly, the bases t and a in x can be encoded as $x_2 = 0,0,1,i,0,0,1,0,1,1$. We employ similar encodings for y except that the complex conjugate of each element is used. For instance, bases g and c in y get coded as $y_1 = 0,1,-i,0,0,1,-i,0,0,0$ and the bases t and a get coded as $y_2 = -i,0,0,-i,1,0,0,-i,1,-i$.

Clearly, the result we are interested in is

$$FFT^{-1}[FFT(x_1) \bullet FFT(y_1^R) + FFT(x_2) \bullet FFT(y_2^R)].$$

Thus we only have to perform four FFTs and one inverse FFT. Further reduction in the number of FFTs is possible with a sacrifice in accuracy [6]. One possibility is to use $1,i,-1,-i$ for the bases g,c,t , and a , respectively. The resultant scores will not be exact but might be acceptable approximations.

Since ultrafast hardwares are available for digital signal processing, FFT-based techniques have the potential of yielding superior performance. In this paper we provide ways of reducing the number of FFT computations drastically.

3. New Algorithms

In this section we give three different algorithms. As has been pointed out in Section 2,

the FFT permits an $O(n \log n)$ solution for finding the cyclic correlation $Z_j = \sum_{k=0}^{n-1} x_k y_{j+k}$

of two complex vectors $x = x_0, x_1, \dots, x_{n-1}$ and $y = y_0, y_1, \dots, y_{n-1}$. Here, all subscripts are taken modulo n , although with sufficient padding with zeroes on the ends of x and y , the cyclic correlation can perform non-cyclic correlation of x and y .

Consider two sequences, the database sequence and the query sequence. Our basic problem is to compute $Z_j = \sum_{k=0}^{n-1} S_k(a_{j+k})$, where $S_k(a)$ gives the score for an amino acid a in the database when matched to position k of a query ($k = 0, 1, \dots, n-1$), and the database consists of the sequence a_0, a_1, \dots, a_{n-1} . The total Z_j is the total score for the database when the query is shifted by j before the database is matched.

Algorithm I. Set up A different computations Z_{aj} ($a = 1, 2, \dots, A$), where A is the total number of amino acids. For each of the computations ($a = 1, 2, \dots, A$), fix amino acid a and let $\delta(a, j) = 1$ if amino acid a occurs at position j of the database, and $\delta(a, j) = 0$ otherwise. Then

$$(0.1) \quad Z_{aj} = \sum_{k=0}^{n-1} \delta(a, k) S_{j+k}(a),$$

which is the shifted total score attributable solely to the amino acid a in the database.

Clearly, $Z_j = \sum_{a=1}^A Z_{aj}$ because $S_{j+k}(a_k) = \sum_{a=1}^A \delta(a, k) S_{j+k}(a)$.

One way of implementing this algorithm is as follows. For a fixed a , we can compute all the Z_{aj} values ($0 \leq j \leq n-1$) using one FFT computation. This is repeated for every amino acid a . Thus the number of FFT computations is only $O(A)$, which is an improvement over the previous approaches by a factor of A .

For a fixed a , the computation proceeds as follows. If $x = x_0, x_1, \dots, x_{n-1}$ and $y = y_0, y_1, \dots, y_{n-1}$ are the database and query sequences respectively we can construct the two sequences $\delta(a, 0), \delta(a, 1), \dots, \delta(a, n-1)$ and $S_0(a), S_1(a), \dots, S_{n-1}(a)$. When these two

sequences are convolved (using FFT algorithms), we get all the Z_{aj} values ($0 \leq j \leq n-1$).

Algorithm II. This algorithm is a generalization of Algorithm I. Consider a compression scheme modulo R , where R is chosen as follows. Normalize the individual position scores $S_j(a)$ (assumed integers) by adding a constant to all of them (including zero-padded positions for non-cyclic correlations), so that the minimum value of any $S_j(a)$ is 0. Let the resulting totals Z_{aj} be integers falling in the range $0, 1, \dots, R-1$. Let $X(j) = \sum_{a=1}^A \delta(a, j) R^{1-a}$, and let $Y(j) = \sum_{a=1}^A S_j(a) R^{a-1}$. Then

$$(0.2) \quad Z_j = \sum_{k=0}^{n-1} \left\{ \sum_{a=1}^A \delta(a, k) R^{1-a} \right\} \left\{ \sum_{a=1}^A S_{j+k}(a) R^{a-1} \right\} = \sum_{k=0}^{n-1} \sum_{a=1}^A S_{j+k}(a) R^{a-a_k},$$

which has (because under the specified restrictions, the sums base R have no carrying)

the units digit of Z_j is $\sum_{k=0}^{n-1} S_{j+k}(a_k)$, the sum desired.

Algorithm III. Third, consider a complex compression scheme using $\frac{1}{2}A$ separate sums representing pairs of amino acids. Let $X_{ab}(j) = \delta(a, j) + i\delta(b, j)$ and let $Y_{ab}(j) = \frac{1}{2}(1+i)S_j(a) + \frac{1}{2}(1-i)S_j(b)$. Then

$$(0.3) \quad Z_{abj} = \frac{1}{2}(1+i) \sum_{k=0}^{n-1} \left\{ \delta(a, k) S_{j+k}(a) + \delta(b, k) S_{j+k}(b) \right\} \\ + \frac{1}{2}(1-i) \sum_{k=0}^{n-1} \left\{ \delta(a, k) S_{j+k}(b) + \delta(b, k) S_{j+k}(a) \right\},$$

so $\text{Re } Z_{abj} + \text{Im } Z_{abj} = Z_{aj} + Z_{bj}$.

As a slight variation of this scheme, consider a complex compression scheme using $\frac{1}{2}A$ separate sums representing pairs of amino acids. Let $X_{ab}(j) = \delta(a, j) + i\delta(b, j)$ and let $Y_{ab}(j) = S_j(a) - iS_j(b)$. Then

$$(0.4) \quad \hat{Z}_{abj} = \sum_{k=0}^{n-1} \left\{ \delta(a,k) S_{j+k}(a) + \delta(b,k) S_{j+k}(b) \right\} + i \sum_{k=0}^{n-1} \left\{ \delta(a,k) S_{j+k}(b) - \delta(b,k) S_{j+k}(a) \right\},$$

so $\text{Re } \hat{Z}_{abj} = Z_{aj} + Z_{bj}$.

4. Implementation Details

In this section we provide experimental results that compare three different methods for computing similarity the scores for each possible alignment of two given sequences. The first method is the traditional approach that performs an FFT computation for every pair of base-pairs. Method 2 is Algorithm II and Method 3 is Algorithm III. For the purpose of our experiments we have used protein sequences that were downloaded from various databases of NIH as well as protein sequences that were randomly generated.

In our implementation of Method 1, we computed the scores for each base-pair separately. Consider two sequences x and y of length n each. Say we are interested in computing the similarity scores with respect to the pair AC. We used two arrays A1 and A2. A1 corresponds to x and A2 corresponds to y . For every element in x is an A there will be a 1 in the corresponding position of A1. The other entries in A1 will be zeros. Similarly, there will be a 1 in A2 corresponding to every occurrence of C in y . Now A1 and A2 are convolved using an FFT algorithm. (A C++ program for FFT can be found in such texts as [7]). Now we have computed the score between the two sequences for each alignment with respect to the pair AC. This process is repeated for every pair. We use another array to accumulate the scores for each possible alignment. The arrays A1 and A2 were of length $2n$ each. Though there are only n elements in each of x and y , A1 and A2 had to be padded with n zeros in order to carry out the convolution.

Method 2 was an implementation of Algorithm II. If $x = x_0, x_1, \dots, x_{n-1}$ and $y = y_0, y_1, \dots, y_{n-1}$ are two given protein sequences, we create two arrays A1 and A2 of

size $20n$ each. Array A1 has n parts with 20 elements in each part. The i th part of A1 corresponds to $x_i, (0 \leq i \leq n-1)$. We order the amino acids as follows: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y. We refer to A as the first amino acid, C as the second amino acid, and so on. If x_i is the j th amino acid then the j th element of the i th part of A1 is set to 1; the other elements of this part are set to zeros (for $0 \leq i \leq n-1$ and $1 \leq j \leq 20$). As an example, if x is DAG, then the array A1 will look like:

The first 20 elements of array A1 (i.e., the first part of A1):

0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The second part of A1:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The third part of A1:

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Array A2 also has n parts with 20 elements in each part. The i th part of A2 will have the following 20 elements: $S_i(A), S_i(C), S_i(D), \dots, S_i(Y)$. Here $S_i(A)$ refers to the score of amino acid A when matched to the amino acid y_i . A convolution of A1 and A2 will yield the necessary scores. Note that we only need every 20th element of the convolution.

Method 3 is similar to Method 2 except that we employ complex encodings to reduce the lengths of A1 and A2. These arrays will be of size $10n$ each now. A1 has n parts with 10 elements in each part. We group the amino acids into 10 groups. A and C are in the first group, D and E are in the second group, and so on. A part of A1 now corresponds to a group. All the elements of A1 are initialized to zeros. If x_i is A then the first element of the i th part of A1 will be set to 1; if x_i is C then the first element of the i th part will be set to $-i$; if x_i is D then the second element of the i th part of A1 will be set to 1; if x_i is E then the second element of the i th part of A1 will be set to $-i$; and so on (for $0 \leq i \leq n-1$).

For example, if x is DAG, then the array A1 looks like:

The first part of A1:

0-0i	1-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i
------	------	------	------	------	------	------	------	------	------

The second part of A1:

1-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i
------	------	------	------	------	------	------	------	------	------

The third part of A1:

0-0i	0-0i	0-i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i	0-0i
------	------	-----	------	------	------	------	------	------	------

Array A2 has n parts with 10 elements in each part. The i th part of A2 corresponds to y_i ($0 \leq i \leq n-1$). The ten elements in the i th part will correspond to the 10 groups of amino acids. For example, consider the first element of the i th part. If u and v are the scores of y_i with A and C, respectively, then the first element of the i th part of A2 is set to $u+iv$; if e and f are the scores of y_i with D and E, respectively, then the second element of the i th part of A2 is set to $e+if$; and so on.

When we convolve A1 and A2 we get the desired result. Here again we only need every 10th item from the convolution. In general each element in the convolution will be a complex number. We pick only the real parts.

5. Experimental Results

We compared Method 1, Method 2, and Method 3 experimentally with various data. The data employed were protein sequences. We tested these methods on randomly generated sequences as well as sequences downloaded from various databases of the NIH. In this section we provide the results of our comparison.

5.1 Random Data

Table 1 gives a comparison of Method 1 and Method 2. Times shown are averaged over 10 random sequences of the given size. The FFT algorithms employed assume that the

input size is an integral power of 2. Note that for Method 1 the arrays A1 and A2 are of size $2n$ whereas for Method 2 these arrays are of size $40n$ each. All the input sizes in Table 1 are integral powers of 2 and hence favor Method 1. For Method 2 we have to choose an array size of $64n$.

TABLE 1

INPUT DATA	METHOD 1	METHOD 2	SPEED UP
256 (1024 byte)	4.95s	0.89s	5.55
512 (2048 byte)	10.88s	1.91s	5.7
1024(4096 byte)	23.80s	4.11	5.79
2048(8192 byte)	52.96s	9.23s	5.75

Table 2 presents a comparison of Method 1 and Method 2 for some other input sizes. These input sizes have been chosen to favor Method 2.

TABLE 2

INPUT DATA	METHOD 1	METHOD 2	SPEED UP
200 (800 byte)	4.95s	0.43s	11.51
400 (1600 byte)	10.88s	0.94s	11.57
800(3200byte)	23.80s	2.03s	11.72
1600(6400 byte)	52.96s	4.17s	12.70

An average of the above two speedups (shown in Tables 1 and 2) is perhaps a reasonable indicator of the typical speedups we can expect from Method 2. In Table 3 we show these averages.

TABLE 3

INPUT DATA	METHOD 1	METHOD 2	SPEED UP
I	4.95s	0.66s	7.5
II	10.88s	1.425s	7.64
III	23.80s	3.07s	7.75
IV	52.96s	6.7s	7.9

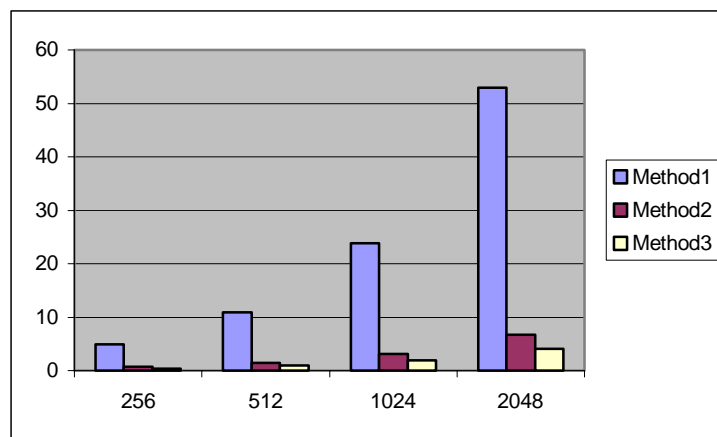
In Table 4 we compare Method 1 and Method 3. Note that the arrays A1 and A2 for Method 3 are of size $20n$ each. Since the array size has to be an integral power of 2, we use a size of $32n$.

TABLE 4

INPUT DATA	METHOD 1	METHOD 3	SPEED UP
256 (1024 byte)	4.95s	0.41s	12.07
512 (2048 byte)	10.88s	0.90s	12.1
1024(4096byte)	23.80s	1.91s	12.46
2048(8192 byte)	52.96s	4.12s	12.85

Chart 1 compares all the three methods.

CHART 1



5.2 NIH Data

We downloaded the following sequences from www.nih.gov:

Table 5

Protein's Unique Number	Data Size
AAD 14597.1	440 byte
CAA 56071.1	445 byte
AAB 65242.1	670 byte
BAA 34431.1	622 byte
BAA 13219.1	1496 byte
AAB 60937.1	1224 byte

We group the protein sequences as follows: (440,445), (670,622), and (1496,1224) because these pairs of data are of similar size. Let Data1, Data2, and Data3 represent these three pairs of data, respectively.

Table 6 shows the speed up of Method 2 over Method 1.

Table 6

INPUT DATA	METHOD 1	METHOD 2	SPEED UP
Data 1	10.67s	0.87s	12.26
Data 2	23.39s	1.88s	12.44
Data 3	51.2s	4.06s	12.56

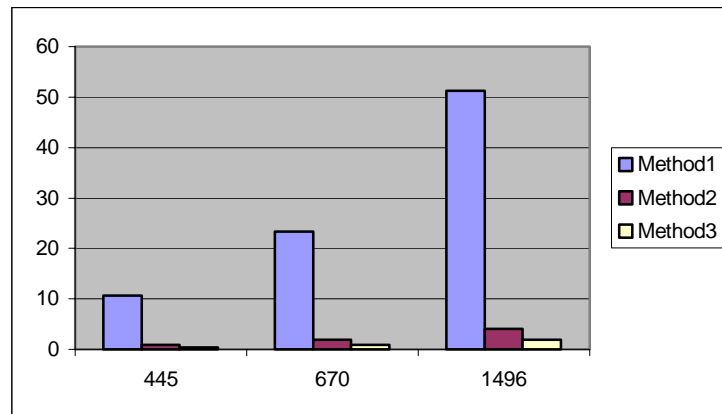
Table 7 shows the speed up of Method 3 over Method 1.

Table 7

Input Data	Method 1	Method 3	Speed Up
Data 1	10.67s	0.41s	26.1
Data 2	23.39s	0.86s	27.2
Data 3	51.2s	1.84s	27.3

Chart 2 compares Method 1, Method 2, and Method 3.

CHART2



6. Conclusions

In this paper we have introduced three new ways of employing FFT in the computation of similarity scores between two given sequences for each possible alignment. An experimental comparison has been done among the traditional approach and the new algorithms. These results indicate that the new algorithms are superior to existing ones in practice.

7. References

- [1] S. F. Altschul, M. S. Boguski, W. Fish, and J. C. Wootton, Issues in Searching Molecular Sequence Databases, *Nature Genetics* 6, 1994, pp. 119-128.
- [2] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, A Model of Evolutionary Change in Proteins, in *Atlas of Protein Sequence and Structure*, edited by M. O. Dayhoff, Volume 5, Supplement 3, National Biomedical Research Foundation, 1978, pp. 345-352.
- [3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, Basic Local Alignment Search Tool, *Journal of Molecular Biology* 215, 1990, pp. 403-410.
- [4] S. Rajasekaran, H. Nick, P. M. Pardalos, S. Sahni, and G. Shaw, Efficient Algorithms for Local Alignment Search, to appear in *Journal of Combinatorial Optimization*.
- [5] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, W. H. Freeman Press, 1998.
- [6] E. A. Cheever, G. C. Overton, and D. Searls, Fast Fourier Transform-Based Correlation of DNA Sequences Using Complex Plane Encoding, *CABIOS* 7(2), 1991, pp. 143-154.
- [7] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms/C++*, W. H. Freeman Press, 1997.