Efficient Algorithms For Local Alignment Search

S. Rajasekaran[1], H. Nick[2], P.M. Pardalos[3], S. Sahni[1], and G. Shaw[2]

Univ. of Florida

ABSTRACT. We present efficient algorithms for local alignment search in biological sequences. These algorithms identify maximal segment pairs (MSPs). Our algorithms have the potential of performing better than BLAST (Basic Local Alignment Search Tool) and also are efficiently parallelizable. We employ fast Fourier transforms (FFTs). Though several attempts have been made in the past to employ FFTs in sequence analysis, they fail to capture local similarities. Our algorithms employ FFTs in a novel way to identify local similarities. FFT-based techniques have the attractive feature of benefiting from ultrafast special purpose hardware available for digital signal processing.

1 Introduction

The problem of identifying similarities among biological sequences has numerous applications such as inferring the functionality of a newly sequenced gene [2]. 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.

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

[1]Department of Computer and Information Science and Engineering

[2]Department of Neuroscience

[3]Department of Industrial Systems Engineering

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 [6] has been given.

2 The BLAST System

A novel technique called Basic Local Alignment Search Tool (BLAST) has been proposed in [2]. This system can be accessed through the internet. In this section we provide a brief introduction to this algorithm.

BLAST takes as input a query sequence A and a database $\mathcal{DB}$ of sequences. It considers all the MSPs between A and each sequence in $\mathcal{DB}$, and outputs those MSPs that have a score of at least S (for some specified S). The database consists of thousands of sequences and the number of sequences increases by the day.

The critical idea behind BLAST is the observation that if two subsequences have a score of at least S then they should have subsequences of length w with a score of at least T (for some appropriate w and $T < S$). Clearly, T should be larger than the expected score between two random sequences of length w each.

There are three steps in the algorithm. In the first step, BLAST forms a list L of w-mers (i.e., sequences of length w) that score at least T with some w-mer in the query sequence. In the second step, it scans through the database $\mathcal{DB}$ to identify w-mers of $\mathcal{DB}$ that are also in L. For any such w-mer that is in L as well as in DB its occurrence in A together with its occurrence in $\mathcal{DB}$ is called a *hit*. The third step involves expanding the hits to get MSPs and output those MSPs that have a score of at least S.

A typical value for w is 12. For every base in the query sequence, there are around 50 words in L. These w-mers are generated in time proportional the number of words in L. Each w-mer can be thought of as an integer in the range $[1, 20^w]$ (in the case of protein sequences) or $[1, 4^w]$ (in the case of DNA sequences). The list L can be stored in an array $\mathcal{A}$ of size $[1, 20^w]$ (or $[1, 4^w]$). Any w-mer is stored in the index given by its

corresponding integer. The ith entry of the array has a list of all the occurrences of the corresponding w-mer in the query sequence.

In the second step, every w-mer in the data base $\mathcal{DB}$ is converted into an integer and using this as the index a search is made in $\mathcal{A}$. This search gives a list of all the occurrences of the w-mer in the query sequences. In other words, we get a list of hits corresponding to the w-mer. This second step can also be implemented using a finite state machine.

The third step consists of expanding every hit to a maximal segment pair and checking if the resultant score will be at least S. This expansion is done in the straight forward way.

The expected run time of BLAST is $aW + bN + cNW/20^w$ for protein sequences. Here W is the number of words generated in step 1, N is the number of residues in $\mathcal{DB}$, and a, b, and c are constants.

Recent Modifications. The BLAST system has recently undergone some changes to improve the speed and also to account for gaps in sequences. In the previous system, step 3 dominated the overall run time. It took around 90% of the total time. Modified BLAST uses a *two-hit* method where instead of looking for a single hit of score at least T, it looks for two smaller hits on the same diagonal within a distance of A (for some appropriate A). This modified version seems to have improved the run time by a factor of 3.

The ability to identify gapped alignments has also been added. This part of the system makes use of dynamic programming based techniques.

3 Fast Fourier Transforms

Let $b = b_0, b_1, b_2, \ldots, b_n$ 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 $f(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n$. The FFT of b is defined to be $B = B_0, B_1, \ldots, B_n$ where $B_i = f(w^i)$, for $0 \leq i \leq n$, w being the primitive n-th root of unity.

If $d = d_0, d_1, \ldots, d_n$ is another sequence, then the convolution of b and d, denoted $b \otimes d$, is $E = e_0, e_1, \cdots, e_n$ 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 $b \otimes d = FFT^{-1}(FFT(b) \odot FFT(d^R))$ (see e.g., [7]). Here FFT^{-1} is the inverse FFT, d^R stands for the reverse of d, and $\odot$ is the dot product operator. (If $x = x_0, x_1, \ldots, x_n$ and $y = y_0, y_1, \ldots, y_n$ then $x \odot y = x_0y_0, x_1y_1, \ldots, x_ny_n$.) The following Lemma is well known (see e.g., [7]).

Lemma 3.1 *FFT and inverse FFT operations and hence the convolution on n-element sequences can be performed in $O(n \log n)$ time.* □

Consider two DNA sequences $x = x_0, x_1, \ldots, x_n$ and $y = y_0, y_1, \ldots, y_n$. We can use 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.

But we can reduce the number of FFT calculations using a clever encoding scheme [5]. 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

$$FFT^{-1}\left[FFT(x_1) \odot FFT(y_1^R) + FFT(x_2) \odot FFT(y_2^R)\right]$$

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 [5]. 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.

4 The New Algorithm

The proposed algorithm uses FFT in a novel way to capture local similarities. Let $X = x_1, x_2, \ldots, x_n$ be the given query sequence. We partition X into subsequences of length m each. An appropriate choice for m is $n^{1-\epsilon}$ for some constant $0 < \epsilon < 1$. The value of m can be decided empirically and adjusted for the best performance. Let $X_1, X_2, \ldots, X_{n/m}$ be the subsequences. There are two phases in the algorithm.

Let Y be any sequence in the database $\mathcal{DB}$. In the first phase, for each subsequence X_i, $1 \leq i \leq \frac{n}{m}$, and for each possible alignment (i.e., position) j in Y, we compute the similarity score. If $|Y| = n'$, then for a fixed X_i, this computation can be performed in time $O(n' \log m)$ using FFTs. Thus, for a fixed X_i and for all the sequences in $\mathcal{DB}$ the time needed is $O(N \log m)$, where N is the total number of bases in $\mathcal{DB}$. As a consequence, for all the subsequences of X and all the sequences in $\mathcal{DB}$ this computation takes time $O(\frac{Nn}{m} \log m) = O(Nn^{\epsilon} \log m) = O(Nn^{\epsilon} \log n)$.

In the above computation we generate triples (i, j, k) such that the subsequence X_i when aligned in position j of sequence k results in a score of at least T (where T is an appropriate threshold). In the second phase, these triples are then analyzed to produce MSPs which score at least S.

The crucial difference between BLAST and the new algorithm lies in the fact that the new algorithm is more selective since it considers longer subsequences before picking candidate MSPs. The effect is that the number of triples generated in the first phase will be smaller than the number of hits that BLAST will generate in its step 2. Also in the second phase, expansion can benefit from the scores that have been computed in the first phase.

Note that since the query sequence is partitioned into smaller subsequences, the possibility of a local similarity getting unnoticed is minimized.

Let (i, j, k) be a triple generated in phase 1. Let $X_i = x_1,', x'_2, \ldots, x'_m$ and let $Y_j^k = y'_1, y'_2, \ldots, y'_m$ be the corresponding portion of the kth sequence in $\mathcal{DB}$. There are four possibilities.

The maximal scoring segment with respect to X_i and Y_j^k is interior to these two, i.e., the starting index $q > 1$ and the ending index $q' < m$. In this case, we do a prefix and suffix computation to throw out the unwanted portion. The prefix to be thrown out is such that its score is negative and the smallest. The same is true for the suffix. If T is chosen properly, then with high probability, the lengths of these prefixes and suffixes won't be very long.

The second possibility is that the MSP might extend to the left. The third possibility is that the MSP might extend to the right and the fourth possibility is that the MSP might extend in both directions.

If the MSP extends to the left, the expanded sequence will be such that it covers some integral number of subsequences (of length m each) and a portion of another subsequence. In this case also, the computational effort needed corresponds to only a fraction of a subsequence, i.e., $o(m)$. For the same reasons, in the third and fourth cases also, the additional effort can be expected to be small.

5 Other Heuristics

In this section we list some heuristics that can be used to speedup the system.

5.1 Random Sampling

Random sampling can be used to solve many problems related to local alignment search.

Problem 1. Let X and Y be two strings with $|X| = |Y| = q$. We can compute the score between X and Y by making q comparisons. Alternatively, we can estimate this score by picking only $o(q)$ symbols from each. To be precise, we pick $\frac{q}{q'} = o(q)$ symbols randomly from X and the corresponding symbols from Y, compute the score of the two subsequences and multiply the resultant score by q'. An appropriate choice for q' is q^ϵ, for some constant $0 < \epsilon < 1$.

Problem 2. Let X and Y be two sequences with $|X| = |Y| = q$. We want to identify a MSP with respect to X and Y. Possibly the whole sequences can be maximal. One way of solving this problem is to identify a prefix and a suffix with the least scores and throw them away. Instead, we could pick a sample from the two sequences as described

in Problem 1 and process this sample to identify the prefix and the suffix that have to be eliminated.

Problem 3. We have a query sequence and a data base sequence such that there is a region of hit (as in step 2 of BLAST). We want to expand the hit around the hit to obtain a MSP. We can combine the ideas from Problem 1 and Problem 2 to perform this task. We guess a certain distance d to expand and perform an estimate of the score using random sampling. As a result we would either decide that it does not help to expand further or realize that it helps to expand further. In the later case, increase the distance by another d, and so on.

In the least, this technique will be helpful in quickly eliminating hits that can not possibly yield a score of S in step 3 of BLAST.

5.2 Deterministic Sampling

Similar to random sampling deterministic sampling can also help. Instead of picking the sample randomly, here we pick elements that are the same distance apart.

Problem 4. This problem is the same as Problem 1. We can pick elements that are d apart from X and Y, compute the score of the sample, and multiply the result by d. For example, if $d = 2$, we can pick either elements in odd positions or elements in even positions.

Problems 5 and 6. These are similar to Problems 2 and 3, respectively. The same discussions hold here in the context of deterministic sampling.

5.3 Data Compression

Some simple static data compression methods such as Huffman coding can be used to speedup our system. For instance, we can scan through the data base to identify the most commonly occurring k-tuples (for some suitable k) and replace them with small codes.

6 Parallelism

Parallelism offers the potential of getting speedups for any system employing more than one processing elements. If we have P processing elements then we can get a maximum speedup of P. It may not be always possible to obtain this high a speedup. The challenge in designing parallel algorithms lies in getting speeds that are as close to P as possible.

Special purpose parallel machines with tens to thousands of processors are available in the market. But often, these machines tend to be high-priced and may not be affordable to all. As an attractive alternative, many software packages (such as the message passing interface (MPI) and the parallel virtual machine (PVM)) are available that can be used to connect heterogeneous machines together and achieve parallelism. For instance, workstations of varying power can be connected together. These packages are available in the public domain for free.

There are several parallel models of computing. The one we employ here is the parallel random access machine (PRAM). The algorithm, however, is general and not restricted to this model. A PRAM consists of a collection of processors that work synchronously. Interprocessor communication takes place with the help of a common block of memory. If processor i wants to communicate with processor j, then it writes a message in common memory cell j in one step, which will be read by processor j in the next time step. Thus for any i and j, processors i and j can communicate in two steps.

Depending on how read and write conflicts are handled, a PRAM can be classified into three. In an exclusive-read exclusive-write (EREW) PRAM no common cell can be accessed by more than one processor for either writing into or reading from at the same time. In a concurrent-read exclusive-write (CREW) PRAM, any number of processors can access the same cell at the same time for the purpose of reading from, but no more than one processor can write into the same cell at the same time. In a concurrent-read concurrent-write (CRCW) PRAM, both concurrent writes and concurrent reads are allowed. If more than one processor can write in the same cell at the same time, the processors can have different messages to write and we should determine which message gets written. Accordingly variants of the CRCW PRAM arise. In a common CRCW PRAM, concurrent writes into the same cell are permitted only if the conflicting processors have the same message to write. In an arbitrary CRCW PRAM, one of the messages will be written in cases of write conflicts and we don't know which one. In a priority CRCW PRAM write conflicts are resolved using priorities assigned to the

processors.

The proposed algorithm for local alignment search is easily parallelizable since efficient parallel algorithms exist for the computation of FFTs.

Lemma 6.1 *The FFT of a sequence of length n can be computed using $\frac{n}{\log n}$ EREW PRAM processors in $O(\log n)$ time [8].*

As a corollary, the following Lemma follows.

Lemma 6.2 *The first phase of the proposed algorithm can be completed in $O(\log n)$ time using Nn^{ϵ} EREW PRAM processors.*

Also, given two subsequences of length m each, we can compute the unwanted prefix and suffix in $O(\log m)$ time and $\frac{m}{\log m}$ EREW PRAM processors using the prefix computation algorithms. It should be possible to reduce this processor bound significantly since not the whole sequences have to be examined in order to identify the unwanted suffix and prefix. Note also that for each triple, the computation is independent of the other triples (given concurrent write processors).

Thus the whole algorithm can be run in $O(\log n)$ time, given enough processors. Even though it seems unreasonable that the Lemmas require so many processors, we can use the slow-down lemma to reduce the processor requirement so that the total work done is preserved. The machines available in the market have only a fixed number of processors.

References

[1] S. F. Altschul, M. S. Boguski, W. Gish, and J. C. Wootton, Issues in Searching Molecular Sequence Databases, *Nature Genetics* 6, 1994, pp. 119-128.

[2] 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.

[3] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs, *Nucleic Acids Research* 25(17), 1997, pp. 3389-3402.

[4] W. I. Chang and E. L. Lawler, Approximate String Matching in Sublinear Expected Time, *Proc. IEEE Symposium on Foundations of Computer Science*, 1990, pp. 116-124.

[5] 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.

[6] 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.

[7] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, W. H. Freeman Press, 1998.

[8] J. Já Já, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, 1992.

[9] S. Karlin and S. F. Altschul, Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes, *Proc. National Academy of Science, USA* 87, 1990, pp. 2264-2268.

[10] S. Karlin, A. Dembo, and T. Kawabata, Statistical Composition of High-Scoring Segments from Molecular Sequences, *The Annals of Statistics* 18(2), 1990, pp. 571-581.

[11] G. M. Landau and U. Vishkin, Fast Parallel and Serial Approximate String Matching, *Journal of Algorithms* 10, 1989, pp. 157-169.

[12] W. R. Pearson and D. J. Lipman, Improved Tools for Biological Sequence Comparison, *Proc. National Academy of Science, USA* 85, 1988, pp. 2444-2448.

[13] E. Ukkonen, On Approximate String Matching, *Proc. International Conference on Foundations of Computing Theory*, Springer-Verlag LNCS 158, 1983, pp. 487-496.

[14] M. S. Waterman, General Methods of Sequence Comparison, *Bulletin of Mathematical Biology* 46(4), 1984, pp. 473-500.