Volume 11 Number 13 1983 Nucleic Acids Research

An efficient method for finding repeats in molecular sequences

Hugo M.Martinez

Department of Biochemistry and Biophysics, University of California, San Francisco, CA 94143, USA

Received 4 April 1983; Revised 23 May 1983; Accepted 31 May 1983

ABSTRACT

The problem of finding repeats in molecular sequences is approached as a sorting problem. It leads to a method which is linear in space complexity and NlogN in expected time complexity. The implementation is straightforward and can therefore be used to handle large sequences with relative ease. Of particular interest is that several sequences can be treated as a single sequence. This leads to an efficient method for finding dyads and for finding common features of many sequences, such as favorable alignments.

INTRODUCTION

There are a number of significant problems in the analysis of molecular sequences (here regarded as strings over finite alphabets) which can be reduced to one of finding repeats. A necessary condition, for instance, that two or more sequences be homologous is that they contain a significantly large subsequence (substring) in common. If the sequences are placed end to end, then such a common subsequence is an instance of a special kind of repeat. Another example arises in the stem-oriented approach to determining the secondary structure of RNA. This approach first requires the finding of all the potential double helices (stems). Such helices are instances of dyads which in turn can be regarded as special repeats when the RNA sequence and its reverse complement are placed end to end. Still another important example occurs relative to searching for control signals. Sequences which are known to contain the signal can be placed end to end and the signal can be regarded as a special repeat.

Computer science has provided a number of solutions to the repeats problem, the most efficient of which appears to be the one based on the concept of position trees (1). It can find repeats with an expected time complexity which is linear in the length of the sequence but whose worst case gives a time complexity which depends quadratically on the length unless special precautions are taken. Such worst cases also involve a space complexity which depends quadratically on the sequence length, though linearly in the expected sense. Additionally, there is the complication that delineating specific repeats involves cumbersome though straightforward tracing of paths in the tree. The method we propose is linear in space complexity for both the expected and worst case, and it is of time complexity NlogN (N = sequence length) in the expected sense. Worst cases can give a time complexity which is quadratic, but special methods can be used to spot the unusual examples in which they arise and reduce them to essentially the expected case. Further, the method involves no path tracing. The repeats are immediate and are reported during the process of sorting to be explained below. An entire tree does not have to first be constructed.

DESCRIPTION OF THE METHOD

Our approach is exceedingly simple and requires no more than the repeated application of a sorting algorithm. The overall speed is essentially determined by the speed of the sorting algorithm employed.

There is first constructed a sequence P of pointers such that pointer value P[i] is the location of the ith element in the sequence S. We then sort P so that it constitutes an ordering of S. That is, P[i] < P[j] or P[i] > P[j] or P[i] = P[j] according to whether S[P[i]] < S[P[j]] or S[P[i]] > S[P[j]] or S[P[i]] = S[P[j]] respectively. With such a sorting of P all the pointer values which point to the same kind of element in S are grouped together. If there are m letters in the alphabet over which S is defined, then there will be at most m groups of pointer values in this first sorting.

We next sort each of these groups of P so that in the resulting subgroups two pointer values belong to the same one if and only if the elements immediately following the ones they point to are equal. Each of these subgroups is then sorted according to the elements twice removed from the elements pointed to, etc.

When no subgroups contain more than one pointer value the process is complete and there results an ordering of P with the following characteristic. Starting at each element of S there is a unique substring (sequence of contiguous elements) which distinguishes it from any other element of S; that is, no other element of S is the start of such a substring. These substrings can be lexicographically ordered and it is precisely this ordering which the final ordering of P represents.

The repeats are generated during the sorting procedure in the following manner. Suppose that we have just produced a group of pointer values and that it is the result of k sorts. The elements pointed to then have the property that the substrings of length k of which they are the start are instances of a repeat of length k provided that it cannot be extended in length. To be extended in length means that an additional sorting of the group of pointers does not break it up into subgroups. Thus, every time a group of pointer values breaks up it signals the finding of a repeat, and where it occurs are the pointer values in that group.

A distinct merit of this approach is that no significant storage space is required beyond that necessary for the sequence S and its pointer sequence P. We also note that the sorting scheme is very similar to how one would go about the lexicographic ordering of a finite number of sequences defined over a finite alphabet. If there are m elements in this alphabet we would first group the sequences (pointers to them) into at most m groups G1,G2,..,Gm. The pointers in group G1 are those pointing to the sequences beginning with the first letter of the alphabet, those in group G2 are the ones pointing to the sequences starting with the second letter of the alphabet, etc. Each of these groups is now independently divided into at most m groups Gi1,Gi2,..,Gim such that Gi1 are those pointers in group Gi which point to sequences whose second element is the first letter of the alphabet, those in Gi2 are the ones of Gi pointing to sequences whose second element is the second letter of the alphabet, etc. If the average length of the sequences is k and if there are n sequences, then the expected time to accomplish their full lexicographic ordering is just k(cN), in which cN is the time required to sort the N elements into m groups.

The basic difference between this sorting scheme and the one used for obtaining the repeats is that we do not have separately defined sequences. There is just one. But if its length is N, then it is as though we had N sequences of average length $\log_m M$. To see this equivalence, we note that the first sorting of the N elements results, on the average, in m groups of size N/m. If it takes time cN to sort N elements into m groups, then it will take time cN to sort all of the m groups of size N/m. This second sorting results, on the average, in m^2 groups of size M/m^2, for which the combined resorting time will again be cN, etc. The highest power k for which $N/m^k > 1$ sets an upper limit to the number of such full sorts. We can therefore take k as logN to the base m.

IMPLEMENTATION CONSIDERATIONS AND APPLICATIONS

A considerable improvement in speed can be achieved with a little pruning. For example, a group of element positions which can be extended backwards must necessarily be included in a set of repeats obtained by the forward extension rule. We therefore test a group of potential repeats for backward extendability prior to its resorting. If the test is positive it is disregarded thereafter.

In the case of finding repeats in multiple sequences we take advantage of the constraint that a potential repeat group must necessarily contain an instance of the repeat from all of the sequences. Sizeable groups which would otherwise require repeated resorting can thus be eliminated.

We have constructed four implementations of the algorithm, called qrepeats, qdyads, qstems and qalign. The first finds all the repeats in one or more sequences, the second is qrepeats specialized to accept but one sequence, construct its reverse complement, and then find the repeats common to these two sequences. Attention in qdyads is also given to filtering out duplicates which necessarily arise because of the mirror image effect produced by working with the reverse complement of a sequence.

The third implementation, qstems, is qdyads specialized to allow for internal U-G base pairing as required for secondary structure in RNA. The programs qdyads and qstems are necessarily geared to the DNA and RNA alphabets, but qrepeats is quite general. No alphabet need be specified. This is achieved by using a sorting algorithm which does not depend upon sequence structure imposed by an alphabet. The slight loss in speed is, for the most part, amply compensated for by the increased flexibility.

Another feature of qrepeats is to allow for intersymbol spacing. For instance, we normally regard repeats as referring to a sequence of contiguous elements which occurs at more than one place. But it is sometimes important to consider not strict contiguity but also a sequence of 'every other element' and hence a single spacing between elements. An appropriate parameter is used to select any periodic spacing desired. This flexibility gives a direct implementation of searching for control signals consisting, for instance, of sequences whose elements are a helical turn apart. But in addition it provides a means of investigating complex repeat structures viewed as the combination of simple, periodic ones.

The fourth implementation, qalign, offers a somewhat new approach to the general problem of determining to what extent two or more sequences are homologous. This means that simultaneous alignments must allow for insertions and/or deletions. Our approach to this problem is a generalization of the approach to the RNA secondary structure problem which first finds the potential stems and then pieces them together to find a combination which minimizes the total free energy. Thus, we first find the potential common substrings and then find compatible combinations of these whose total length is as large as possible. This latter optimization is equivalent to a "shortest path" kind of a problem and therefore has a solution of time complexity which is the number of common substrings squared (1). In applying the qalign program to a specific problem the option is given of selecting the minimum length which is to be allowed for a common substring. This gives control over what are to be regarded as statistically significant common substrings and hence on the number of common substrings from which to select compatible combinations. The optimization algorithm employed allows for the weighting of gaps, if desired, and mention should also be made of the option for finding near optimal alignments. The specific implementation of this latter feature finds all the alignments which lie within a specified distance of the optimal alignment as measured in units of the optimizing function (such as total number of matches in an alignment).

We have run a number of tests as to actual speed. Typical figures obtained on a VAX 11/750 computer are:

Seq. length	qrepeats	qdyads	$cN\log_4 N$
1024	4.0 secs	16.0 secs	4.0 secs
2048	9.4	27.9	8.8
4096	22.1	103.4	19.2
8192	51.9	136.2	41.6

The qrepeat figures refer to repeats of length 5 or greater and those for qdyads to dyads for which each half is of length 5 or greater. The column headed $cN\log_4 N$ is obtained from the qrepeats column by assuming the 4.0 second figure to correspond to $cN\log_m N$. The constant c is thus evaluated and used to calculate what the computation speeds would be at the remaining sequence lengths if the expected time were $cN\log_m N$. That the actual computation times are more than this is attributed to the use of a sorting algorithm which has an expected time complexity of NlogN rather than cN.

The implementations are written in the C language. They are available separately or as part of the UCSF Biomathematics Computation Laboratory sequence analysis package, which provides a comprehensive set of programs geared to a UNIX operating system environment.

ACKNOWLEDGEMENT

I am indebted to Professor Frederick Blattner of the University of Wisconsin for his very helpful suggestions during the initial implementations of the algorithm and for his making a 27 kilobase DNA sequence available as an impetus to develop efficient methods for handling large sequences.

This work was in part supported by NSF grant # PCM 8022006.

REFERENCES

1. Aho,V.A,Hopcroft,J.E and Ulman,J.D. (1975) The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading.