Volume 14 Number 1 1986 Nucleic Acids Research

Algorithms for the search of amino acid patterns in nucleic acid sequences

Hannu Peltola, Hans Söderlund[1] and Esko Ukkonen

Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki, and [1]Recombinant DNA-laboratory, University of Helsinki, Valimotie 7, SF-00380 Helsinki, Finland

Received 18 July 1985

ABSTRACT

Some algorithms are described for the search of regions in a nucleic acid sequence that, when translated into amino acids, are homologous to a given amino acid pattern. All algorithms are modifications of the dynamic programming method for sequence comparison such that the translation of codons is taken into account. One of the algorithms has been implemented as a FORTRAN 77 program. The program operates on files that follow the format of the EMBL Nucleotide Sequence Data Library.

INTRODUCTION

Several algorithms and computer programs have been developed during the past few years for comparing the primary structure of two biopolymers such as amino acid sequences or nucleic acid sequences. Typically, two sequences are compared as a whole to find different types of homologies between them. Another important question is to find all regions in a "long" sequence (or in a data base of sequences) that are homologous to a given "short" sequence. Both problems can be solved using the dynamic programming method developed in different variations e.g. in (1,2); Sankoff and Kruskal (3) give a good overview of the algorithms in this class.

Here we consider a related problem of comparing an amino acid sequence to a nucleic acid sequence. Since the translation from nucleic acids to amino acids is not one-to-one, it is clear that the methods for comparing amino acid sequences or nucleic acid sequences do not work without modifications.

The ANSEARCH program is available for non-commercial reseach use by paying postage and handling; re-distribution is prohibited. For further information contact Hannu Peltola.

More specifically, this paper discusses methods for the search of amino acid patterns in nucleic acid sequences. All methods are modifications of the dynamic programming algorithm for sequence comparison. The simplest of the methods has been implemented as a FORTRAN 77 program.

SEARCH ALGORITHMS

The problem

Let **a** and **b** be two sequences such that **a** is an amino acid sequence and **b** is a nucleic acid sequence.

Sequence **a** is represented as a character string a(1)a(2)...a(n) where each a(i) can be an amino acid (denoted as A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y) or a stop codon (denoted as *). Moreover, a(i) can be letter X denoting an unknown amino acid. The possibility of using X is convenient when only partial information is available on sequence **a**. This is the case e.g. when **a** is produced by radio sequencing (4,5). Then, in fact, X is not completely unknown since X must be different from all the explicitly known amino acids of **a**. For example, if **a** = AXCX has been produced by radio sequencing it is known that the second and the fourth element of **a** are any amino acids different from A and C (and also different from all the other already sequenced amino acids not present in **a**).

Sequence **b** is represented as a character string b(1)b(2)...b(m) where each b(i) can be a nucleic acid (denoted T/U,A,G,C).

The search problem we want to solve is to find in sequence **b** all regions that, when translated to amino acids, are sufficiently homologous to the amino acid sequence **a**. Both sequences can contain small sequencing errors such as short insertions, deletions, and changes.

The homologies between such **a** and **b** can be found by dynamic programming algorithms. The idea behind such methods is that we fix a set of basic transformation operations on sequences (typical operations are deletion, insertion, and change of a character). Each operation has an associated penalty value. Dynamic programming algorithms find regions in **b** that can be transformed into **a** with minimal sum penalty by successively

using basic operations (2). The translation of codons to amino acids adds a constraint which has to be taken into account in the detailed formulation of the dynamic programming algorithm.

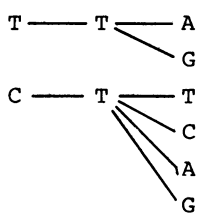
There are two approaches to the problem created by the codon-to-amino-acid translation. The first is to backtranslate sequence **a** into nucleic acids. This means reducing the problem to the comparison between nucleic acids. The symmetric possibility is, of course, to reduce the problem to a comparison between amino acids by translating sequence **b** into amino acids.

Reduction to a comparison between nucleic acid sequences

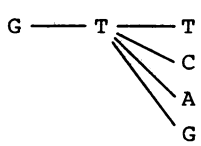
A brute-force solution would be to backtranslate **a** in all possible ways into a nucleic acid sequence and compare each of the sequences against **b**. However, the resulting algorithm is immediately seen intolerably slow. This is because up to six codons may translate into the same amino acid which means that already relatively short amino acid sequences backtranslate in enormously many different ways.

Fortunately, a more efficient solution is possible. The idea of this algorithm is to avoid explicit backtranslation into separate sequences by comparing all alternatives in parallel.

For example, consider an amino acid sequence **a** = LVM. Since L (leusine) has 6, V (valine) has 4, and M (methionine) has 1 codons, this sequence can be backtranslated in $6 \cdot 4 \cdot 1 = 24$ different ways. A compact graphical representation of the six codons of L, namely TTA, TTG, CTT, CTC, TTA, CTG, is as follows:

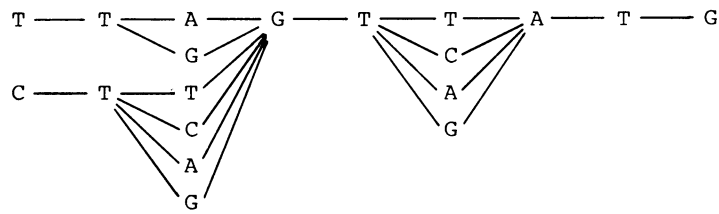


The similar network for V is as follows:



and for M as follows: A —— T —— G

Combining the three networks by connecting all rigthmost nucleotides of a network to all leftmost nucleotides of the next network gives a network for LVM:



Every left-to-right path in the network, such as the (lowermost) path CTGGTGATG, gives a backtranslation of LVM.

Obviously, the backtranslations of any amino acid sequence have this kind of **network representation.** Also the backtranslations of the symbol X can be represented in this way. So the problem of comparing **a** to **b** reduces to the problem of comparing the network representation of **a** to **b.** A generalization of the dynamic programming method by (2) to handle this situation is straightforward in principle, as described in a general setting in (6); the adaptation of the method for the needs of the present application is simple and we do not give details.

The resulting algorithm is reasonably efficient: The number of computational steps is propotional to k·m where k is the sum of the number of the backtranslations of the individual elements in a (for LVM we get k = 6+4+1 = 11) and m is the length of b. This is because the algorithm works by evaluating a k·m matrix. If an upper bound for the acceptable total penalty is given a priori, evaluation of some matrix elements can be skipped. This trick, which is based on techniques from (7,8) still improves the efficiency.

Maybe the most important feature of the delineated algorithm is its mathematical soundness: The search of the occurences of the pattern **a** in sequence **b** is based solely on minimizing the total penalty of transforming a possible occurence into a nucleic acid sequence which is a backtranslation of **a.** On the negative side, the programming of the algorithm (which we have not done) seems rather tedious.

Reduction to a comparison between amino acid sequences

A simple solution would be to first translate sequence **b** into amino acids in the three reading frames in both directions. Then the six resulting amino acid sequences are searched for the pattern **a**. This, however, ignores that **b** can contain errors such as nucleotide insertions and deletions which may mix the reading frames. To cope with this, more flexible changing of the reading frame ought to be possible during sequence comparison. Again, this can be achieved by a suitable form of dynamic programming. We have used the following algorithm.

Algorithm 1.

To find all occurences of pattern **a** = a(1)a(2)...a(n) in sequence **b** = b(1)b(2)...b(n), an (n+1)·(m+3) matrix E must be evaluated. The rows of matrix E are indexed as 0,1,...,n and the columns are indexed as -2,-1,0,1,...,m. The matrix is initialized as follows:

$E(0,j) = \infty$ for $j = -2, -1$;

$E(0,j) = 0$ for $j = 0, \ldots, m$;

$E(i,j) = \infty$ for $i = 1, \ldots, n$ and $j = -2,-1,0,1,2$.

The remaining elements of E are evaluated using the recurrence equation

$$E(i,j) = d(a(i), aa(j)) + \min \begin{cases} E(i-1,j-1) + del_{-2} \\ E(i-1,j-2) + del_{-1} \\ E(i-1,j-3) \\ E(i-1,j-4) + del_{1} \\ E(i-1,j-5) + del_{2} \end{cases}$$

where

aa(j) denotes the amino acid that corresponds to the codon b(j-2)b(j-1)b(j);

d(a(i), aa(j)) denotes the penalty of matching amino acids a(i) and aa(j);

del_{-1} denotes the penalty of one missing nucleotide in sequence **b** before codon b(j-2)b(j-1)b(j). Therefore the reading frame has been moved for one position to the left;

del_{1} denotes the penalty of one extra nucleotide in sequence **b** before codon b(j-2)b(j-1)b(j). Therefore

the reading frame has been moved for one position to the right;

del_{-2} (del_2) is like del_{-1} (del_1) but now there are two missing (extra) nucleotides and the reading frame has been moved for two positions to the left (to the right).

Using the recurrence equation above, matrix E can be evaluated, say, column by column.

The interpretation of the evaluated matrix E is standard: The last row of E (=row n) gives the total penalties of matching **a** against different parts of **b**. In fact, E(n,j) is the total penalty of matching **a** against the part b(i)b(i+1)...b(j) of **b** where i is such that there is in E a "minimizing path" from E(0,i-1) to E(n,j).

To select the cases where the homology between **a** and the corresponding b(i)b(i+1)...b(j) is good enough the algorithm uses a threshold value for the total penalty. Whenever E(n,j) is found to be equal to or less than the threshold value, the algorithm outputs the corresponding (translated) b(i)b(i+1)...b(j) as well as the alignment of **a** against b(i)b(i+1)...b(j), as suggested by the minimizing path.

The number of computing steps of Algorithm 1 is proportional to n·m. The tricks from (7,8) can again be used in improving the efficiency.

Proper choice of the penalty and the threshold values in Algorithm 1 is crucial, of course. In our implementation the following values have worked quite well:

$$d(a(i), aa(j)) = \begin{cases} 0 & \text{if } a(i) = aa(j) \text{ or } a(i) = X \\ & \text{(or has been produced by radio} \\ & \text{sequencing and } a(i) = X \text{ and} \\ & aa(j) \text{ does not appear in a)}, \\ 3 & \text{otherwise (i.e., } a(i) \text{ and } aa(j) \\ & \text{are different)}, \end{cases}$$

$del_{-1} = 2$,
$del_{-2} = 4$,
$del_1 = 2$,
$del_2 = 4$,
Threshold value for total penalty = n.

Using more sophisticated penalty functions in Algorithm 1 is also possible. In particular, the penalties d(a(i), aa(j)) of comparing amino acids could be based on the PAM250 matrix of (9).

Algorithm 1 allows insertion and deletion of one or two nucleotides between two codons of **b** when searching for regions that translate to **a**. This is rather restrictive but gives a reasonably fast algorithm. More homologies can be found by using in Algorithm 1 the recurrence equation

$$E(i,j) = \min \begin{cases} E(i-1,j-1) + e(a(i), b(j)) , \\ E(i-1,j-2) + e(a(i), b(j-1)b(j)) , \\ E(i-1,j-3) + e(a(i), b(j-2)b(j-1)b(j)) , \\ E(i-1,j-4) + e(a(i), b(j-3)...b(j)) , \\ E(i-1,j-5) + e(a(i), b(j-4)...b(j)) . \end{cases}$$

Here e(a(i), w), where w is a nucleotide sequence of length between 1 and 5, is the penalty of matching amino acid a(i) against w. A natural method to calculate these penalties is to determine the minimum base change distance of w from each codon for a(i), and take the minimum of the distances. To improve efficiency, the penalties should be precomputed and stored in a table.

This algorithm always matches an amino acid against a sequence of nucleotides of length from 1 to 5, which means that at most two insertions of deletions are allowed in **b** per a matching amino acid in **a**. The retriction to two nucleotides is consistent with our assumption that **a** and **b** can contain only small sequencing errors. If one wants to allow arbitrary long blocks of deletions and insertions when searching for homologies, for example the following form of the recurrence equation can be used:

$$E(i,j) = \min \begin{cases} E(i-1, j) + e(a(i), \emptyset) , \\ E(i-1,j-1) + e(a(i), b(j)) , \\ E(i-1,j-2) + e(a(i), b(j-1)b(j)) , \\ E(i-1,j-3) + e(a(i), b(j-2)b(j-1)b(j)) , \\ E(i ,j-1) + e(\emptyset , b(j)). \end{cases}$$

where $\emptyset$ denotes the empty sequence, and penalty function e is as earlier, with obvious extensions.

THE PROGRAM

Algorithm 1 has been implemented as an interactive program, called ANSEARCH, using standard FORTRAN 77 language. The penalty values are as described after Algorithm 1 in the previous section.

The program operates on a file of nucleotide sequences from which sequence **b** is retrieved using an identifier given by the user as the retrieval key. The file must follow the format used in the EMBL Nucleotide Sequence Data Library.

The program then asks the user for sequence **a**, to be given from the computer terminal. Also the symbol X may be used in partially known sequences. Finally, the program asks for the threshold value of the total penalty. A default value can also be used.

Then the search algorithm is performed and all regions of **b** that approximately translate to **a**, within the limits given by the threshold value, are shown to the user terminal.

The maximum length of sequence **a** in the current version of the program is 50. Since **b** is used through buffers, its length is not limited.

DISCUSSION

Algorithm 1 as well as program ANSEARCH were originally designed for locating genes during a sequencing project in newly sequenced nucleotide sequences. If one wants to make sure that rigth gene is cloned, and the gene product is (partially) known, a possibility is to determine the nucleotide sequence and check with program ANSEARCH that the sequence can code the known amino acid pattern. Also, program ANSEARCH is useful in locating DNA sequencing errors on regions where the amino acid sequence is known by other means. Currently the program is used in searching for nucleotide sequences that code certain intresting pentapeptides.

Algorithm 1 is almost symmetric with respect to **a** and **b**. To search in **a** for all regions that are homologous to the translation of **b**, one has only to interchange 0 and ∞ in the initialization of E. The total penalty is now read from the last column (instead of the last row) of E.

ACKNOWLEDGEMENTS

The ANSEARCH program was written by Markku Korhonen. The comments by Ale Närvänen are gratefully acknowledged. This work was supported by a grant from Foundation for Biotechnical and Industrial Fermation Research.

REFERENCES

1. Needleman, S.B. and Wunsch, C.D. (1970) J. Mol. Biol. 48, 443-453.
2. Sellers, P.H. (1980) J. Algor. 1, 359-373.
3. Sankoff, D. and Kruskal J.B. (Ed.) (1983) Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, Reading, Mass.
4. Kalkkinen, N., Laaksonen, M., Söderlund, H. and Jörnvall, H. (1981) Virology 113, 188-195.
5. Jörnvall, H., Kalkkinen, N., Luka, J, Kaiser, R., Carlquist, M. and von Bahr-Lindström, H. (1983) in Modern Methods in Protein Chemistry, Review articles, Ed. H. Tschesche, (Nordic Biochemical Society Meeting, September 27-29, 1982) Walter de Gruyter, New York.
6. Kruskal J.B. and Sankoff, D. (1983), pp. 265-310 in (3).
7. Ukkonen, E. (1986) Information and Control, to appear.
8. Peltola, H., Söderlund, H. and Ukkonen, E. (1984) Nucl. Acids Res. 12, 307-321.
9. Schwartz, R.M. and Dayhoff, M.O. (1978) in Atlas of Protein Sequence and Structure, Dayhoff, M.O. Ed., Vol. 5 suppl. 3, p.354, National Biomedical Research Foundation, Washington.