CABIOS

Vol.8. no.1. 1992
Pages 57 – 64

A search for common patterns in many sequences

M.A.Roytberg

Abstract

A new approach to search for common patterns in many sequences is presented. The idea is that one sequence from the set of sequences to be compared is considered as a 'basic' one and all its similarities with other sequences are found. Multiple similarities are then reconstructed using these data. This approach allows one to search for similar segments which can differ in both substitutions and deletions/insertions. These segments can be situated at different positions in various sequences. No regions of complete or strong similarity within the segments are required. The other parts of the sequences can have no similarity at all. The only requirement is that the similar segments can be found in all the sequences (or in the majority of them, given the common segments are present in the basic sequence). Working time of an algorithm presented is proportional to $n \cdot L^2$ when n sequences of length L are analyzed. The algorithm proposed is implemented as programs for the IBM-PC and IBM/370. Its applications to the analysis of biopolymer primary structures as well as the dependence of the results on the choice of basic sequence are discussed.

Introduction

Various biological problems require a simultaneous comparison of several (more than two) DNA or protein sequences. Two main advantages of multiple sequence comparison were recognized 20 years ago (Fitch, 1970). First, multiple comparison reveals similarities too weak to be separated from random ones by pairwise comparison. Second, multiple comparison allows one to choose the best alignment even when pairwise comparisons yield many alignments with almost the same scores.

Two approaches to the problem of multiple sequence comparison are possible (see Friedmann, 1988; Waterman, 1989 for review): (i) a multiple alignment of sequences as a whole; and (ii) a search for all reasonable local similarities.

Most publications on multiple sequence comparison are devoted to the first approach. Multiple sequence alignment is useful when sequences under considerations (or at least a subset of them) have long similar regions. The alignment is generally constructed in two ways: (i) using various modifications of the dynamic programming method which yields an optimal multiple alignment; and (ii) using a 'step-by-step' procedure, each step being an alignment of two sequences only.

The first versions of the dynamic programming method (Murata *et al.*, 1985; Gotoh, 1986) required time proportional to the product of the lengths of all aligned sequences and was suitable for alignment of 3–5 not very long sequences. More effective implementations of dynamic programming (Carillo and Lipman, 1988; Altschul and Lipman, 1989; Lipman *et al.*, 1989) allow as many as ten sequences to be aligned in a reasonable time.

The step-by-step procedures give more acceptable times, but the result generally depends on the order in which the sequences are aligned. Various implementations differ with possibility of multiple revision of sequences, their preliminary classification into clusters, taking into account the three-dimensional structures, etc. See, for example, Martinez (1988) and Waterman (1989) for review.

However, if the sequences from the set under consideration have only short regions of local similarities (in the extreme case, only one segment of similarity per sequence) the multiple alignment approach has no sense and the second (segment-based) approach should be used. Published algorithms solve this problem effectively only when similar regions meet some restrictions. For example, the similar segments can differ by mismatches, but not by insertions/deletions (Bacon and Anderson, 1986), or they should be situated at almost the same distances from the start of the sequence (Johnson and Doolittle, 1986; Waterman, 1986; Vihinen, 1988; Vingron and Argos, 1989), or they should contain a relatively long subsegment of perfect similarity (Sobel and Martinez, 1986).

Here I propose a new algorithm for searching for local similarities in several sequences that is free from these restrictions. This algorithm is also general in the sense that segments found can be 'similar' not only in a 'textual' way with any arbitrary scoring table, but in any other way according to the biological problem to be solved.

Informal description of the approach

Our goal is to find common patterns, i.e. to locate the segments that occur (possibly with some variations) in all the sequences compared. We shall not consider in detail the problem of aligning the segments found. These segments are, as a rule, similar and short enough to be well aligned by one of the known

Research Computer Center, Pushchino, Moscow Region, 142292, USSR

methods. The heuristic method implemented in the MSC program is informally described below.

Let us call a segment of a sequence 'fundamental' if each other sequence contains at least one segment similar to it. The idea is to select one sequence as 'basic' and to search for the fundamental segments on it. Then the patterns common to all the sequences are reconstructed using the fundamental segments found on the basic sequence.

To find these segments, the basic sequence is compared successively with all the other sequences. The result of each comparison is a set of all the segments of a basic sequence that are similar to any segment of the other sequence. After all these comparisons have been made the fundamental segments of the basic sequence are constructed by intersections of the segments revealed by pairwise comparisons.

Thus, common segments from all $n + 1$ sequences are found from a series of n pairwise comparisons of a basic sequence with all other ('serial') sequences. 'Similarity' of two sequences may be defined arbitrarily. The way of finding local pairwise similarities may be considered as a replaceable part (parameter) of the algorithm (cf. Taylor, 1987; Martinez, 1988). For example, instead of the textual similarity adopted in the example below, it is possible to consider two segments as similar on a basis of their common symbol usage, internal symmetries or three-dimensional structures.

Example 1

Let us consider the basic sequence

$$B = \text{'AGTATACATTCGAAAA'}$$

and a series of two sequences

$$S(1) = \text{'GTTCCGAACTATAC'}$$

and

$$S(2) = \text{'GGTATAGATTGGAAA'}$$

Here only exactly coinciding segments of lengths no less than 4 are considered as 'similar'. Then B and $S(1)$ have one similarity.

$$B[3,7] = \text{'TATAC'} \ <=> \ S(1)[10,14] = \text{'TATAC'}$$

where $V[x,y]$ is a segment of a sequence V from the xth to the yth symbol inclusively. B and $S(2)$ have two similarities:

$$B[2,6] = \text{'GTATA'} <=> S(2)[2,6] = \text{'GTATA'}$$

and

$$B[12,15] = \text{'GAAA'} <=> S(2)[12,15] = \text{'GAAA'}$$

Therefore a sequence B contains one fundamental segment, $B[3,6[$ = 'TATA'. The corresponding segments in the serial sequences are $S(1)[10,13]$ = 'TATA' and $S(2)[3,6]$ = 'TATA' respectively.

If local similarities are sought by an algorithm allowing one difference (mismatch, insertion or deletion), then B contains one more fundamental segment, $B[9,14]$ = 'TTGGAA'. This segment corresponds to the similarities

$$B[9,14] = \text{'TTCGAA'} <=> S(1)[2,8] = \text{'TTCCGAA'}$$

and

$$B[9,14] = \text{'TTCGAA'} <=> S(2)[9,14] = \text{'TTGGAA'}$$

End of example.

This approach allows an effective multiple similarity search. If the search for all the local similarities in two sequences requires the time T, the whole time of comparison of the basic sequence with a series of n sequences is porportional to $n \cdot T$ and the intersections necessary to obtain the final result can be done much faster. The memory required depends mainly on the algorithm of comparison of two sequences (see below).

An 'asymmetry' of the approach originating from the choice of a basic sequence does not seem to be a severe limitation. It is possible to choose various sequences as basic for the same set of sequences and to compare the results obtained. Sometimes the asymmetry is natural, e.g. when a new sequence is compared with a series of known sequences. We shall also consider a more 'symmetrical' version of the approach (see Problem 2 below).

Another modification of this approach allows one to find 'subfundamental' segments, i.e. segments that are similar in some but not all serial sequences.

Strict formulation of problems

A pairwise local similarity may be defined in many ways. Therefore, there are many possible formalizations and algorithms for the search for all the reasonable local similarities in two sequences (Altschul and Ericson, 1986; Sellers, 1984; Hall and Myers, 1988; Waterman, 1989). Any algorithm may be used for our goal. The algorithm adopted here is briefly described in the Implementation section.

By a local similarity of two sequences W and U we mean a pair (d,d') of segment d on sequence W and segment d' on sequence U. If (d,d') is a local similarity, we say that d is an analog of d' and vice versa. Some algorithms can also find for every pair of similar segments an alignment and other characteristics (likelihood weight, similarity score, etc.)

Let us choose the basic sequence B, the series of n 'serial' sequence $S(1), \ldots, S(n)$, the algorithm H, finding local similarities in two sequences, and a value D_{min}, which is the minimum length of similar fragments.

Segment F of the sequence B is called fundamental with respect of $S(1), \ldots, S(n)$, algorithm H and value D_{min} if

(i) for each $j \in \{1, \ldots, n\}$, F is a part of a segment d having an analog in $S(j)$;

(ii) segment F is 'maximal', i.e. is not contained entirely in any other segment of B satisfying to requirement (i);

(iii) the length of F is not smaller than D_{min}.

Thus, for every fundamental segment F of the basic sequence B there is a corresponding similar segment in each serial sequence. The main problem here is as follows.

Problem 1

(i) To find all fundamental segments of the basic sequence B with respect to sequences $S(1), \ldots, S(n)$ with a given search algorithm H and value of D_{min}.

(ii) To find all analogs of each fundamental segment in the serial sequences $S(1), \ldots, S(n)$.

Problem 1(ii) should be reformulated if local similarities are sought by an algorithm that does not align similar segments (an algorithm by Zharkikh and Rzhetskiy, 1989, for example). In this case it is unclear what should be meant by an analog of the fundamental segment. A suitable modification is the problem of finding for each fundamental segment F all local similarities (E,E') such that F lies inside E or overlaps this by a length not smaller than D_{min}.

Note that the weight of similarity of the fundamental segment F and its analog F' may be smaller than the initial similarity (d,d') weight. We believe it is interesting to take into acocunt such weak subsimilarities of strong similarities, which is seen in the statement of Problem 1. But in some situations it is more natural to neglect them in further analysis of the fundamental fragments found (see Implementation).

The relation of local similarity is not transitive: two segments from two different serial sequences that are similar to a given fundamental segment cannot be similar between themselves (see example below). Therefore, the following problem is of interest.

Problem 2

To find all segment systems $\{d(0), \ldots, d(n)\}$ such that

(i) $d(0)$ is a fundamental segment of the basic sequence;

(ii) $d(j)$ is a analog of $d(0)$ in the jth serial sequence $(j \in \{1, \ldots, n\})$;

(iii) for all $i,j \in \{1, \ldots, n\}$ segments $d(i)$ and $d(j)$ are similar according to the adopted definition of a local similarity.

This problem resembles the problem of searching for similar segments if the sequences compared are considered equally right, i.e. none of them is picked out.

Example 2

Let us consider again the basic sequence

$$B = \text{'AGTATACATTCGAAAA'}$$

and the series of two sequences:

$$S(1) = \text{'GTTCCGAACTATAC'}$$

and

$$S(2) = \text{'GGTATAGATTGGAAA'}$$

Let us treat as similar those segments that differ by no more than one mutation (mismatch, insertion or deletion); here $D_{min} = 4$. As we noted in the previous section, the basic sequence possesses two fundamental segments $B[3,6] = S(1)[10,13] = S(2)[3,6] = \text{'TATA'}$ and $B[9,14] = \text{'TTCGAA'}$, which is similar to $S(1)[2,8] = \text{'TTCCGAA'}$ and $S(2)[9,14] = \text{'TTGGAA'}$. Here the solution of Problem 2 is only the triplet

$$<B[3,6],\ S(1)[10,13],\ S(2)[3,6]>$$

while the other triplet is not a solution, because $S(1)[2,8] = \text{'TTCCGAA'}$ differs from $S(2)[9,14] = \text{'TTGGAA'}$ by more than one mutation. End of example.

In many situations it is important to single out those segments that have similarity with some of the sequences compared, but not with all of them (cf., for instance, Martinez, 1988). This situation leads to the Problem 3 below, but first some definitions.

Segment E of the basic sequence B is subfundamental with respect to the series $S(1), \ldots, S(n)$ if E is a fundamental segment with respect to a subseries V of $S(1), \ldots, S(n)$, V contains not less than $n - f$ sequences (f is a parameter).

Problem 3

(i) To find all subfundamental segments of the basic sequence B with respect to the series $S(1), \ldots, S(n)$ for a given search algorithm H and values of parameters D_{min} and f.

(ii) To find all analogs of these subfundamental segments in the serial sequences $S(1), \ldots, S(n)$.

Surely, in this case the results depend strongly on the choice of basic sequence. This is a weak point of the approach presented. A possible way to overcome it is to try each sequence as a basic one (of course, this will take more time). Note that if all the sequences contain fragments that are very similar to one another, then a choice of the basic sequence is not crucial.

These three problems are formalizations of the approach outlined in the previous sections. Their development may consist in applying additional restrictions on the class of fundamental segments we are analyzing. These restrictions may be caused by mutual interpositions of similar segments on the sequences compared, their space structures, physical and chemical properties, etc.

Algorithms

Let B be the basic sequence with length L; $S_1, \ldots, S_n$ be the series of sequences with the lengths $L_1, \ldots, L_n$. Let LOC_SIM be the algorithm used for search of local similarities in two sequences, a form of which is shown in Figure 1.

We begin with a detailed description of the main ALG1A algorithm for solving Problem 1(i). The algorithm is presented in Figure 2; its subroutines SETSYS1 and FUND are shown in Figures 3 and 4 respectively. The algorithm FUND is the main part of the algorithm ALG1A. This algorithm computes new value of SYS from its preceding value and the value of SYS1 already generated. To be more accurate, FUND generates all such segments F, that

```
Algorithm LOC__SIM (V1, V2: sequence; . . . );
begin
  INIT__LOC__SIM;
  while (processing of V1 and V2 is not completed) do
    GET__NEXT__SIM (E1, E2: segment; . . .)
  end__while;
end.
```

Fig. 1. A common form of an algorithm for searching for local similarities in two sequences. INIT LOC SIM is an initialization subroutine; GET__NEXT__SIM generates a current pair of similar segments, E1 of V1 and E2 of V2. The type 'segment' represents a sequence segment and is a record with two integer components, FIRST and LAST, indicating the ends of the segment. In all alogrithmic texts dots represent additional parameters. The additional output data of the GET__NEXT__SIM program, for example, may be an alignment of E1 and E2.

```
Algorithm ALG1A;
begin
  INIT__1A;
  for NSEQ:=1 to n do begin
    read next serial sequence;
    'Construct SYS1 (see Figure III)'
    SETSYS 1 (SYS1, NYS1);

    if (NYSY1=0) then      'no similarity between basic'
      NSYS:=0;             'and current serial sequence'
      exit
    end__if;

    'Assign a new value to SYS'
    if (NSEQ=1) then       'processing of 1-st serial'
      SYS:=SYS1;           'sequence'
      NSYS:=NSYS1;
    else
      FUND(SYS, NSYS, SYS1, NSYS1)   'see Figure IV'
    end__if;
    if (NSYS=0) then exit;
  end__for;
end.
```

Fig. 2. Algorithm ALG1. INIT__1A is an initialization subroutine reading the basic sequence and preprocessing it according to the INIT__LOC__SIM subroutine of the LOC__SIM algorithm (see Figure 1). Here n is a number of serial sequences. The array SYS1 of length NSYS1 contains all segments of the basic sequence B that have a similar segment in a current serial sequence S_j and are not a part of any other such segment. Array SYS of length NSYS contains all the segments of the sequence B fundamental with respect to current subseries $\{S_1, \ldots, S_j\}$. The segments in the arrays are ordered according to their starting points.

(i) $F = F_1 \cap F_2$, where F_1 is an element of SYS1 and F_2 is an element of the preceding value of SYS;

(ii) F is not shorter than D_{min};

(iii) F lies in no other segment satisfying previous requirements.

The execution time and memory needed for algorithm FUND are proportional to a number of elements in the systems intersected and are in our case proportional to the length of the basic sequence.

The solutions of Problems 1(ii), 2 and 3 are based on the algorithm ALG1A. To avoid a lot of technical details, we will only sketch these solutions.

Problems 1(ii) and 2 are each solved in three stages. The first stage is computation of an array SYS of fundamental segments,

```
Algorithm SETSYS1(SYS1,NSYS1);
begin
  'Initialisation'
  INIT__LOC__SIM;
  for j:=1 to L do MEND[j]:=0;

  'Computing of MEND'
  '(the loop below is a modified loop from LOC__SIM)'

  while (processing of sequence B and S is not completed) do
    GET__NEXT__SYM (E1, E2, . . .);

    b:=E1.FIRST; e:=E1.LAST;
    if (e - b + 1≥ Dmin) & (MEND[b] < e) then
      MEND[b]:=e;
    end__if;

  end__while;

  'Computing of SYS1 and NSYS1'
  NSYS1:=0;
  z:=0;
  for j:=1 to M do
    if (MEND[j] > z) then
      z:=MEND[j];
      NSYS1:=NSYS1+1;
      SYS1[NSYS1].FIRST:=j;
      SYS1[NSYS1].LAST:=z;
    end__if
  end__for;

end.
```

Fig. 3. Algorithm SETSYS1. The algorithm computes a value of an array SYS1 and its length NSYS1. Elements of SYS1 are segments represented with records having two integer components, FIRST and LAST. INIT__LOC__SIM and GET__NEXT__SIM are subroutines of the LOC__SIM algorithm (see Figure 1). B and S are arrays containing basic and current serial sequences respectively. E1 and E2 are similar segments (E1 and B and E2 from S) generated by the GET__NEXT__SIM subroutine. An element MEND[i] of an auxilliary array MEND contains the largest number r such that the segment $[i,r]$ of the basic sequence has a similar segment in S (according to the LOC__SIM algorithm) and 0 if there are no such segments.

as in Problem 1(i). In addition, array SYS is used to create two auxiliary arrays FIRSTSEG and LASTSEG of length L. FIRSTSEG $[x]$ is the least number of a segment $[a,b]$ from system SYS such that $b - x + 1 \geq D_{min}$, and LASTSEG[y] is the largest number of a segment $[a,b]$ from system SYS such that $y - a + 1 \geq D_{min}$.

It is evident that if $[x,y]$ is a segment of the basic sequence, it has non-empty intersection of a length not smaller than D_{min} with only those fundamental segments that are numbered from FIRSTSEG[x] to LASTSEG[y] in SYS. It can easily be seen that the time taken for computation of arrays FIRSTSEG and LASTSEG is of the order of L.

The second stage consists, like the first, in computation of the local similarities between the basic sequence B and each sequence of the series $S_1, \ldots, S_n$. For each similarity (E,E') found, we search for fundamental segments that have an intersection with E of a length not smaller than D_{min}. If the set of such fundamental segments is non-empty, the local similarity is saved and is referred to from the appropriate fundamental

```
Algorithm FUND (SYS, NSYS);
begin
  'Initialisation'
  K1:=; K2:=1;   'current segments of SYS1 and SYS2'
  NSYS:=0:       'current length of SYS'

  while (K1 ≤ NSYS1 & K2 ≤ NSYS2) do

    'Determination of an active segment'
    'of next good pair'
    if (SYS1[K1].FIRST ≤ SYS2[K2].FIRST)
      then Act:=1; Pass:=2;
      else Act:=2; Pass:=1;
    end_if;

    'Caculation of a segment of SYSPass that'
    'complete SYSAct[KAct] to a good pair'
    KPass:=the maximal NZ for which
        SYSPass[NZ].FIRST ≤ SYSAct[KAct].FIRST;
    'Now SYSAct[KAct] and SYSPass[KPass] form'
    'a good pair. Is their intersection long enough?'
    b:=SYSAct[KAct].FIRST;
    e:=min(SYS1[K1].LAST, SYS1[K2].LAST)
    if (e−b+1 ≥ Dmin) then
      'Store the next element of SYS''
      NSYS:=NSYS+1;
      SYS[NSYS].FIRST:=b; SYS[NSYS].LAST:=e;
    end_if

    'increase values of K1 and/or K2;'
    'an element of a good pair having smaller end'
    'cannot belong to any following good pair'
    if (SYS1[K1].LAST=e) then K1:=K1+1;
    if (SYS2[K2].LAST=e) then K2.=K2+1;
  end_while

end.
```

Fig. 4. Algorithm FUND. The FUND algorithm computes a new value of the array SYS from values of an auxiliary array SYS1 (see SETSYS1, Figure 3) and array SYS2 containing the previous value of SYS. NSYS, NSYS1, NSYS2 are the numbers of elements in corresponding arrays. Elements of the arrays are segments (see Figure 3) ordered according to their beginnings. Therefore, they are also ordered according to their ends (otherwise some segment would be part of another one, which contradicts the definition of fundamental segments). The algorithm looks over arrays SYS1 and SYS2 in a 'parallel' way to find 'good' pairs of segments, i.e. pairs of such segments SYS1[K] and SYS2[K2] whose intersection does not lie in an intersection of any other pair of segments from SYS1 and SYS2. The element of a good pair with a larger beginning is called active (if the beginnings are equal we call the segment of SYS1 active for convenience). Here we use the following notation. The variable Act contains the number (1 or 2) of the array that includes the active segment of the current good pair; the variable Pass contains a number of the other (passive) array, so that Pass = 3 − Act. Thus SYSAct identifies the array SYS1 if Act = 1 and the array SYS2 if Act = 2. Identifiers SYSPass, KAct, KPass, etc., are used analogously. For example, KPass means K1 if the array SYS2 is active, and K2 otherwise.

segments. If desired, all weights of subsimilarities of fundamental segments are computed and subsimilarities of low weight are rejected.

In the third stage the local similarities found are printed (Problem 1(ii)) or further analyzed (Problem 2).

It remains to consider Problem 3. We will only describe the algorithm for the solution of Problem 3(i); Problem 3(ii) is solved on the basis of this, just as Problem 1(ii) is solved using Problem 1(i).

	Identifier	Start of the Coding sequence
1.	HSACTH	681
2.	HSENKE	950
3.	HSIL05	1263
4.	HSTNFA	615
5.	HSIIFI56	1157
6.	HSGMCSFG	620
7.	HSVWF123	836
8.	HSEGFR1	839
9.	HSBSF2	1123
10.	HSINSRB	1254

Fig. 5. The following human genes were analyzed: common precursor of corticotropin and β-lipotropin (identifier in 20th release of EMBL database, HSACTH; accession number, V01510), preproenkephalin (HSENKE; X00187), interleukin-2 (HSIL05; X00695, X00200, X00201, X00202), α-tumor necrosis factor (HSTNFA; X02910, X02159), interferon-inducible gene IFI-56K (HSIIFI56; X06559, Y00986), granulocyte−macrophage colony stimulating factor (HSGMCSFG; X03021), von Willebrand factor (HSVWF123; X06828), epidermal growth factor receptor (HSEGFR1; X06370), hepatocyte stimulating factor BSF-2/IL6 (HSBSF2; Y00081), insulin receptor (HSINSRB; J03466).

The algorithm, ALG3A, for solving Problem 3(i) is structurally similar to ALG1A (see Figure 2). Let f be a maximal number of sequences in which a subfundamental segment can have no similarities. The difference between these two algorithms is that ALG1A deals with the array of segments SYS and ALG3A deals with $f + 1$ analogous arrays SYS(0), . . . , SYS(f). When r first sequences of a series are processed, the array SYS(j) contains all segments that have no similarities exactly in j sequences ($j = 0, \ldots, f$) or r sequences considered.

Implementation

The algorithms described above were implemented in the program MuSCo (Multiple Sequence Comparison). The program exists in two versions. The first is implemented in the FORTRAN-77 language for IBM/370 computers (operating system VM) and is a part of the SAMSON package for analyses of biosequences (Vernoslov *et al.*, 1989). The second version is written in C for IBM-PCs.

Both DNA and protein sequences can be compared. The number of sequences in a set is up to 30 and the maximum sequence length is 1000.

The program has five options. The first is to compare a basic sequence with every serial sequence independently. This option is useful to choose the appropriate values of parameters. The four other options are to solve Problems 1(i), 1(ii), 3(i), 3(ii) respectively. The data obtained when these problems are solved can be further analyzed to solve Problem 2 or to select the multiple similarities that meet some conditions.

To find local similarities in two sequences, the following heuristic method is implemented. First, the program finds all the 'primary blocks', i.e. pairs of coinciding segments of the sequences. The length of the segments cannot be less than a specified threshold. The program then links the blocks into

Identifer	From	To	Location	Alignment
HSTNFA	−237	−219	——*——	G G A ___ G ___ C A G G G A G G A _ T G ___ G G G A
HSENKE	−443	−422	---*------	G G g ___ G c c C A G G G A G G A ___ G g c G G G A
HSGMCSFG	−240	−217	------*---	a G A c t G c c C A G G G A G G g c T G ___ G a G A
HSEGFR1	−344	−328	----*----	G G A ___ a _____ A G G G _ G G A _ a G ___ G G G A
HSINSRB	−437	−418	---*------	G G c ___ a ___ C A G G G A G G c ___ G ___ G G G A
				G G a g C A G G G A G G a N G G G G A

Fig. 6. A multiple similarity between 5 of 10 considered sequences. The top sequence was selected as basic. The columns 'From' and 'To' contain distances from the transcription initiation site to the ends of the segment found. The 'Location' column indicates the place of the segment on the initial sequence. Every position in this column denotes 50 nucleotides. The last column shows the multiple alignment of the segments. Capital letters indicate that the majority of the five sequences contain the same letter in this position. The bottom line in the 'Alignment' column contains a 'consensus' site. A capital letter in this line means that it presents at least in three-quarters of the rows. A lower-case letter indicates a simple majority only.

Identifier	From	To	Location	Alignment
HSEGFR1	− 42	− 29	————*	T C C C T C C ___ T C C T C _ C C
HSACTH	−507	−492	-*————	T C C C T C C c c T C C T C _ C C
HSENKE	−141	−129	———*--	a C C g a C C _____ C C T C _ C C
HSTNFA	− 48	− 35	————*	g C C C T C C ___ T C _ T C g C C
HSGMCSFG	− 73	− 61	———*-	T C C C _ C C ___ g C C T C _ C C
HSINSRB	−254	−243	------*----	g C C C T C _____ T C _ T C _ C C
				t C C C T C C T C C T C C C

Fig. 7. A multiple similarity between 6 of 10 considered sequences. The notations are explained in Figure 6.

'chains' and marks those chains that have a weight more than another threshold and meet some 'maximality' conditions (cf. Sellers, 1984; Goad and Kanehisa, 1982). A weight of a chain is defined as follows.

Some 'weight' $W(R) > 0$ is assigned to each block R and a linkage penalty $P(R_1, R_2)$ is assigned to each pair of blocks R_1 and R_2 that can be linked due to their locations on the sequences compared. The linkage penalty is analogous to the gap penalty in the traditional definitions of an alignment weight (Roytberg, 1984; Waterman, 1984; Miller and Myers, 1988). The weight of a chain $C = \{R_1, \ldots R_k\}$ is a difference between the sum of weights of blocks $W(R_i)$ and the sum of the linkage penalties $P(R_i, R_{i+1})$, where $i = 1, \ldots, k$.

This method of pairwise comparison is very fast, but does not guarantee finding an 'optimal' local similarity. A similar technique was propsoed by Sobel and Martinez (1986) to construct directly a multiple alignment of many sequences.

When applied to a pair of sequences this method can detect short, even single-element primary blocks, and therefore can find weak similarities. If one searches for blocks common to all n sequences considered (as in Sobel and Martinez, 1986), their number increases proportionally to L^n when the average sequence length L increases. Therefore, the minimal block length cannot be too small.

The weight of the primary block in the MuSCo program may be defined as sum of match weights of its symbols or as a function (linear, or exponential) of the block's length. A linkage penalty usually is a linear or logarithmic function on the distance between the blocks, but more complicated ways of defining it are also possible.

Every detected set of the similar segments consists of a (fundamental) segment of the basic sequence and, similar to it, segments of the serial sequences. These segments are aligned as follows. First, we align the segments according to their pairwise alignments with the segment of the basic sequence: the positions corresponding to the same position of the fundamental segment are placed in the same column. Then we modify the obtained multiple alignment with some local transformations to increase the number of multiple matches and to decrease the number of gaps (the details are discussed separately).

The total memory used for multiple comparison depends mainly on the maximum possible number of primary blocks when two sequences are compared. In the IBM/370 version this number is 25 000, therefore the memory used is ~1 Mbyte (including 640K for data). In the PC version of the program the maximum possible number of primary blocks is determined dynamically according to available memory and can be up to 5000 for a 640K RAM computer. Note that the maximum number of primary blocks determines the minimum possible length of a block; 25 000 primary blocks allows, as a rule, the use of one-symbol blocks for comparison of DNA sequences of length 300 or protein sequences of length 700.

The execution time also depends on the number of primary blocks generated and some other characteristics (maximum possible distance between blocks, for example). Solving Problem 1(i) on the IBM/370 computer for the basic sequence of length 100 and nine serial DNA sequences of the same length (all are parts of the sequences shown in Figure 5) takes ~100 s when minimum length of primary block is 1 and maximum distance between linking blocks is 3, and 12 s when minimum length of two primary block is 2. If the parameters of comparison are fixed and all serial sequences have the same lengths, the execution time grows approximately proportionally to a product of the lengths of the basic and serial sequences.

Discussion

The principal advantage of the method of searching for multiple local similarities proposed here over methods published before is that it can find 'weak similarities', i.e. similar fragments that (i) can contain no regions of exact match, (ii) can differ with deletions/insertions as well as with replacements, and (iii) can be situated at arbitrary positions on the sequences compared. The possibility of using various concepts of pairwise similarity is also important. The following is a brief review of other methods.

The search for short similar segments in the analyzed sequences is a preliminary step in some multiple alignment algorithms (Waterman *et al.*, 1984; Bains, 1986; Johnson and Doolittle, 1986; Waterman, 1986; Vingron and Argos, 1989). For our goal, however, the methods used at this step are of independent interest. Their common feature is that they find similar segments only if they are almost equidistant from the sequences' beginnings. This is reasonable for the purpose of construction of a multiple alignment. In the case of a local similarity search, however, usually there are no reasons to assume that similar segments are situated at similar positions.

Several papers consider the problem of searching for multiple local similarities directly (Queen *et al.*, 1982; Bacon and Anderson, 1986; Krishnan *et al.*, 1986; Sobel and Martinez, 1986; Taylor, 1986; Vihinen, 1988).

The method of Queen *et al.* (1982) is based on a preliminary search for exact matches of all the sequences, which may be done very quickly using special tables (Aho *et al.*, 1974). Therefore, if the set of even very similar segments contains no absolutely conserved positions, this method cannot find it. This appproach also requires rather extended exact matches, otherwise their number and the computation time are too large.

The method of Sobel and Martinez (1986), which searches for exact matches and then links them into chains, has already been discussed in the Implementation section. In a recent paper Martinez (1988) has proposed the generalization of the method to produce a multiple alignment.

The method of Bacon and Anderson (1986) processes sequences successively. The result for the r first sequences is M sets of segments (one segment from each sequence) with the largest similarity weights (M is a parameter of the algorithm). Unfortunately, this attractive method does not allow insertions/deletions to be taken into account.

A method of Taylor (1986) is designed for the comparison of proteins. It starts from 'templates' created using the data on a structure, and a function of separate segments of a part of the compared proteins. Templates are then refined in the course of analysis of the other proteins.

Krishnan *et al.* (1986) and Vihinen (1988) have sought for similar segments using overlapped similarity matrices (dot-matrices). The weight of a dot in the resulting matrix is a sum of the weights of the corresponding points in the initial matrices. Therefore, the points in a resulting matrix with weights exceeding some threshold mark the local similarities among all (or the majority) of sequences compared.

Although very similar, these two dot-matrix approaches have some differences. Krishnan *et al.* generate dot-matrices for all possible pairs of sequences (when these are long, their segments of fixed lengths are compared). Vihinen's method is designed to compare proteins, especially their secondary structures, though it may be used to compare any sequences. As in our method, it picks up a basic sequence, whose dot matrix is overlapped with that of every other sequence. These are initially aligned with the basic sequence to take into account their possible shifts with respect to the basic sequence.

An important difference of these methods from ours is that they use overlapped dot-matrices, while our method uses overlapped projections of similar segments onto the basic sequence. Therefore, dot-matrix methods can find only the segments of the basic sequence that have similar segments in the other sequences at approximately the same positions with respect to the starts of the compared sequences (Krishnan *et al.*, 1986) or 'reference points' found during preliminary two-sequence alignments (Vihinen, 1988).

To summarize, note that all thesse methods cannot search for 'weak similarities'. Therefore, the advantages of our method are particularly important in searching for short patterns that may differ at various sequences with both mismatches and insertions/deletions and whose positions at various sequences may differ significantly.

The main peculiarity of the method (like all step-by-step methods) is its 'asymmetry', i.e. the dependence of the results on a choice of the basic sequence. To illustrate this let us consider again Example 1, where the similarity

$$B[9,14] = \text{'TTCGAA'} <=> S(1)[2,8] = \text{'TTCCGAA'} <=> S(2)[9,14] = \text{'TTGGAA'}$$

is shown. It was found with basic sequence B and a threshold for a pairwise comparison equal to one (i.e. only one difference—mismatch or deletion—is allowed). The distance between $S(1)[2,8]$ = 'TTCCGAA' and $S(2)[9,14]$ = 'TTGGAA' is 2 (see Example 2), therefore the similarity would not be found if the sequence $S(1)$ or $S(2)$ is considered as basic. But the asymmetry is not a fatal drawback. For instance, in the previous situation we can overcome it by increasing the threshold to 2. Note that this 'asymmetry' of the method may reflect the asymmetry between an ancestral sequence and the successor ones. Some other problems associated with a choice of the basic sequence have already been discussed in the previous sections.

To illustrate the abilities of our method we present data on a local similarities search in 5'-regulatory regions of ten protein-coding loci of *Homo sapiens* (Figure 5). Six hundred nucleotides preceding a transcription initiation site were considered at each locus. No obvious common biological functions and no

significant pairwise similarities were found for all these sequences, so each sequence was regarded as basic. The minimum primary block length was 2 and the maximum distance between blocks was 3.

Data in Figures 6 and 7 show multiple alignments for the best multiple similarities found in this set of sequences. Of course, we do not claim that these similarities reflect any common function, but an analogous search in a series of random sequences did not reveal similarities of comparable quality.

Unfortunately, we cannot estimate precisely the statistical significance of similarities that our method yields. This is a common problem with all the methods that allow gaps. A routine way to overcome this difficulty is either to use very crude estimates of significance or to do the same comparison with several series of 'random' (having the same frequencies of symbols, or, preferably, of pairs of adjacent symbols) sequences of the same lengths.

To estimate the significance of similarities presented in Figures 6 and 7 these ideas were combined. First, an approximate value of the probability that a random site of given length has a similarity in a random sequence was obtained by Monte Carlo simulation (in the case of the similarity shown in Figure 6, the length of the site is 20, the length of the sequence is 600 and the probability is 0.0022). Then an estimate is made of the probability of multiple similarity from the probability, obtained from the Mone Carlo modeling (in the case above the probability is less than 0.001).

Acknowledgements

I thank A.S.Kondrashov for useful discussions and for suggesting a set of sequences for analysis, V.V.Levitin who took part in programming of the IBM-PC, S.A.Shabalina and T.V.Astakhova for valuable discussions, and A.V.Finkelstein whose reading of the manuscript led to many improvements. I also thank the referees for their criticisms.

References

Aho,A., Hopcroft,J. and Ullman,J. (1976) *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

Aleksandrov,N.A. and Mironov,A.A. (1989) Pattern recognition and a computer analysis of nukelotide sequences. *Molek. Biol.*,**23**, set 5, 1248–1262.

Altschul,S.F. and Erickson,B.W. (1986) Optimal sequence alignment using affine gap costs. *Bull. Math. Biol*, **48**, 603–616.

Altschul,S.F. and Lipman,D.J. (1989) *SIAM J. Appl. Math.*, **49**, 197–209.

Bacon,D.J. and Anderson,W.J. (1986) Multiple sequence alignment. *J. Mol. Biol.*, **191**, 153–161.

Bains,W. (1986) MULTAN: a program to align multiple DNA sequences. *Nucleic Acids Res.*, **14**, 1, 159–177.

Barton,G.J. and Sternberg,M.J. (1987) A strategy for the rapid multiple alignment of protein sequences. Confidence levels tertiary structure comparison. *J. Mol. Biol.*, **198**, 327–338.

Carillo,H. and Lipman,D. (1988) The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, **48**, 1073–1082.

Fitch,W.M. (1970) Further improvements in the method of testing for evolutionary homology among proteins. *J. Mol. Biol.*, **49**, 1–14.

Friedmann,T. (1988) Alignment of multiple DNA and protein sequence data. *Comput. Applic. Biosci.*, **4**, 231–214.

Goad,W.B. and Kanehisa,M.I. (1982) Pattern recognition in nucleic acid sequences. I. A general method for finding local homologies and symmetries. *Nucleic Acids Res.*, **10**, 247–278.

Gotoh,O. (1986) Alignment of three biological sequences with an efficient traceback procedure. *J. Theor. Biol*, **121**, 327–337.

Hall,J.D. and Myers,E.W. (1988) A software tool for finding locally optimal alignments in protein and nucleic acid sequences. *Comput. Applic. Biosci.*, **4**, 35–40.

Higgins,D.G. and Sharp,P.M. (1989) Fast and sensitive multiple sequence alignments on a microcomputer. *Comput. Applic. Biosci.*, **5**, 151–154.

Johnson,M.J. and Doolittle,R.F. (1986) A method for the simultaneous alignment of three of more amino acid sequences. *J. Mol. Evol.*, **23**, 267–277.

Krishnan,G., Kaul,R.K. and Jagadeeswaran,P. (1986) DNA sequence analysis: a procedure to find homologies among many sequences. *Nucleic Acids Res.*, **14**, 543–550.

Lipman,D.J., Altschul,S.F. and Kececioglu,J.D. (1989) A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. USA*, **86**, 4412–4415.

Martinez,H.M. (1988) A flexible multiple sequence alignment program. *Nucleic Acids Res.*, **16**, 1683–1691.

Miller,W. and Myers,E.W. (1988) Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, **50**, 91–120.

Murata,M., Richardson,J.S. and Sussman,J.L. (1985) Simultaneous comparison of three protein sequences. *Proc. Natl. Acad. Sci. USA*, **82**, 3073–3077.

Myers,E.W. (1986) An O(ND) difference algorithm and its variations. *Algorithmica*, **1**, 251–266.

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.

Queen,C., Wegman,M.N, and Korn,L.J. (1982) Improvements to a program for DNA analysis: a procedure to find homologies among many sequences. *Nucleic Acids Res.*, **10**, 449–456.

Roytberg,M.A. (1984) An algorithm for finding homologies among primary structures. *Pushchino* (in Russian).

Sellers,P.N. (1974) On the theory and computation of evolutionary distance. *SIAM J. App, Math.*, **26**, 787–793.

Sellers,P.N. (1984) Pattern recognition in genetic sequences by mismatch density. *Bull. Math. Biol*, **46**, 501–514.

Sobel,E. and Martinez,H.M. (1986) A multiple sequence alignment program. *Nucleic Acids Res.*, **14**, 363–374.

Tajima,K. (1988) Multiple DNA and protein sequence alignment on a work station and a super computer. *Comput. Applic. Biosci.*, **4**, 467–471.

Taylor,W.R. (1986) Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol.*, **188**, 233–258.

Taylor,W.R. (1987) *Comput. Applic. Biosci.*, **3**, 81–88.

Vernoslov,S.Ye., Kondrashov,A.S., Roytberg,M.A., Shabalina,S.A., Yurieva,O.V. and Nazipova,N.N. (1989) 'SAMSON' program package to analyze primary biopolymer structures. *Pushchino*, (in Russian).

Vingron,M. and Argos.P. (1989) A fast and sensitive multiple sequence alignment algorithm. *Comput. Applic. Biosci.*, **5**, 115–122.

Vihinen,M. (1988) An algorithm for simultaneous comparison of several sequences. *Comput. Applic. Biosci.*, **4**, 89–92.

Waterman,M.S. (1984a) Efficient sequence alignment algorithm. *J. Theor. Biol*, **108**, 333–337.

Waterman,M.S. (ed.) (1984b) General methods of sequence comparison. *Bull. Math. Biol.*, **46**, 473–500.

Waterman,M.S., Arratia,R. and Galas,D.J. (1984) Pattern recognition in several sequences: consensus and alignment. *Bull. Math. Biol.*, **46**, 515–527.

Waterman,M.S. (1986) Multiple sequence alignment by consensus. *Nucleic Acids Res.*, **14**, 9095–9102.

Waterman,M.S. and Perlwitz,M.D. (1984) Line geometries for sequence comparisons. *Bull. Math. Biol*,**46**, 567–578.

Waterman,M.S. (ed.), (1989) *Mathematical Methods for DNA Sequence Analysis*. CRC Press, Boca Raton, FL.

Waterman,M.S. and Perlwitz,M.D. (1984) Line geometries for sequence comparisons. *Bull. Math. Biol.*, **46**, 567–578.

Zharkikh,A.A. and Rzhetskiy,A.Yu. (1989) Rapid oligonucleotide frequency estimation of homologies among nucleotide sequences. *Dok. Akad. Nauk SSSR*, **308**, 1232–235 (in Russian).

Received on September 21, 1990; accepted on June 12, 1991