Programming Research Group

IMPROVING THE SENSITIVITY OF MULTIPLE-SEQUENCE ALIGNMENTS BY INCORPORATING PRIOR KNOWLEDGE

Sumedha Gunewardena
Peter Jeavons

PRG-RR-02-01

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Programming Research Group

IMPROVING THE SENSITIVITY OF MULTIPLE-SEQUENCE ALIGNMENTS BY INCORPORATING PRIOR KNOWLEDGE

Sumedha Gunewardena
Peter Jeavons
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK
S.Gunewardena,P.Jeavons@comlab.ox.ac.uk

PRG-RR-02-01

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

In this paper, we present efficient modifications to the well-established progressive alignment algorithm for biological sequences. These modifications are designed to allow the user to incorporate prior knowledge about the sequences and so greatly improve the sensitivity of the resulting alignments. The first modification increases the probability that certain biologically distinguishable structures are preserved during the alignment process. The second modification increases the probability that specified sequence segments will align with each other.

We have implemented both of these modifications in an interactive multiple-sequence alignment tool (IMSA). IMSA takes a two-stage approach to the alignment process. The initial or pre-processing stage takes as input sets of sequence segments defined on DNA, RNA or protein sequences. These sets of sequences represent biologically distinguishable features, which could be derived from known homologies, or known structural or functional elements. The sequences to be aligned are efficiently annotated based on this additional information, and the program then computes an alignment which is adjusted to take account of this annotation.

1 Introduction

One common way to extract information from biological sequence data is to construct an alignment of a group of similar sequences. These alignments are often computed using dynamic programming techniques [9, 13, 17]. As the number of sequences to be aligned grows, the computational challenge of efficiently calculating useful alignments also increases.

It is possible to solve this multiple alignment problem using a direct extension of the standard two dimensional dynamic programming algorithm for pairwise alignment. However, this approach is generally considered impractical due to its $O(2^k N^k)$ time complexity (for k sequences of length N). The inherent computational difficulty of this problem has been demonstrated by Wang and Jiang [23], who proved that the optimal sum of pairs alignment problem is NP-complete. As a result many sub-optimal strategies have been suggested for the multiple-sequence alignment problem [1, 5, 22], the most common of which is the progressive alignment technique [3, 7, 11, 24], which works by constructing a succession of pair-wise alignments. The progressive alignment technique works as follows: initially, two sequences are chosen and aligned using the standard dynamic programming algorithm. This alignment is then fixed and the third sequence is aligned with it. This process is iterated until all the sequences have been aligned. In general, one can extend the process by calculating a pair-wise alignment between two sequences, between a sequence and an alignment, or between two alignments. This method of aligning multiple sequences is not guaranteed to be mathematically optimal in the alignments that it produces, but can efficiently align a large number of sequences.

Various different strategies have been proposed to improve the sensitivity of this progressive alignment technique. Many of the existing programs perform an initial ordering of the sequences being aligned. CLUSTALW [22] for instance uses a guide tree constructed using the neighbour-joining clustering algorithm by Saitou & Nei [16]. MULTAL [20] uses a sequential branching method which aligns the two closest sequences first and builds on it by progressively adding the next closest sequence to those already aligned. MULTALIGN [3] and PILEUP [8] constructs a guide tree using the UPGMA method [18]. Many programs also incorporate iterative strategies to improve on the initial alignment [3, 4, 10]. As well as the initial sequence ordering, CLUSTALW incorporates sequence weighting, position specific gap penalties and the dynamic selection of the residue comparison matrix based on the degree of identity of the sequences, in order to improve its alignments. Programs such as DIALIGN [12] construct multiple sequence alignments based on segment to segment comparisons (instead of single residue comparisons) where alignments are constructed from pairs of equal length gapless segments called diagonals. SAGA [14] is a genetic algorithm based multiple sequence alignment program that uses a consistency based objective function for alignment evaluation called COFFEE [15] in order to increase the accuracy of the multiple sequence alignments. The COFFEE score reflects the level of consistency between a multiple sequence alignment and a library containing pair-wise alignments of the same sequences.

Current sequence alignment algorithms, though inspired by natural evolutionary models of genetic formation, generally do not take into account any additional specific information which is available for the sequences being aligned. Many different functional and structural elements (e.g. structural elements, binding sites, hypersensitive sites, etc.) are now known and well documented from detailed studies of DNA, RNA and protein sequences. This prior knowledge is often readily available to the sequence analyst, and tends to be incorporated by manually adjusting the alignments after they are computed. We demonstrate below that by simply facilitating the automatic alignment of such known biological features one can often obtain much more useful initial sequence alignments.

For example, Figure 1 shows an alignment of segments of seven Immunoglobulin protein sequences produced using the standard pair-wise alignment algorithm. The PAM 100 substitution matrix [6] was used for scoring residue pairs, but this was the only biological information used in the algorithm.

```
              012345678901234567890123456789012345678901234567890

FABVL         ASVLTQPPSVSGAPGQRVTISCTGSSSNIGAGHNVKWYQ--QLPGT-APK-
FB4VL         QSVLTQPPSASGTPGQRVTISCSGTSSNIGSS-TVNWYQ--QLPGM-APKL
FB4VH         EVQLVQSGGGVVQPGRSLRLSCS-SSGFIFSSYAMYWVR--QAPGK-GLEW
FABVH         AVQLEQSGPGLVRPSQTLSLTCTVSGTSF-DDYYWTWVR--QPPGR-GLEW
FABCL         QPKAAPSVTLFPPSSEELQANKATLVCLISDFYPGA-VT--VAWKA-DGSP
FABCH1        A---STKGPSVFPLAPTAALGCLVKD-YFPEPVTVSWNSGALTSGVHTFPA
FCCH3         QPR-EPQVYTLPPSREEMTKNQVSLTCLVKGFYPSD---IAVEWESNGQPE
```

Figure 1: Alignment of 7 immunoglobulin sequence segments produced by the standard algorithm.

The marked regions within the sequences in Figure 1 correspond to the known β-strands of these Immunoglobulin domains [3]. It is clear that the standard alignment does not correctly align these known structural features, and in some cases even introduces gaps within these features.

In this paper we introduce an interactive multiple sequence alignment tool called IMSA that provides a convenient means of incorporating prior knowledge about sequence elements, such as these structural features. By giving this program the sequences for just *one* of the known collection of homologous β-strands (the first block marked in Figure 1), we obtain the alignment shown in Figure 2, which is clearly much more likely to be of biological significance.

```
         01234567890123456789012345678901234567890123456789012345

FABVL    ----ASVLTQPPSVSGA-PGQRVTISCTGSSSNIGAGHNVKWYQ---QLPGTAPK-
FB4VL    ----QSVLTQPPSASGT-PGQRVTISCSGTSSNIGSS-TVNWYQ---QLPGMAPKL
FB4VH    ----EVQLVQSGGGVVQ-PGRSLRLSCS-SSGFIFSSYAMYWVR---QAPGKGLEW
FABVH    ----AVQLEQSGPGLVR-PSQTLSLTCTVSGTSF-DDYYWTWVR---QPPGRGLEW
FABCL    QPKAAPSVTLFPPSSEELQANKATLVCLISD-FYPGAVTVAWK--------ADGSP
FABCH1   ASTKGPSVFPLA--------PTAALGCLVK-DYFPEPVTVSWNSGALTSGVHTFPA
FCCH3    QP-REPQVYTLPPSREEMTKNQVSLTCLVKG-FYPSDIAVEWE-------SNGQPE
```

Figure 2: Alignment of the same 7 immunoglobulin sequence segments of Figure 1 produced by IMSA

2 Standard Pairwise Alignment

The starting point for IMSA is the standard pairwise dynamic programming alignment algorithm described below.

Definition 2.1 *Let $S_1, \cdots, S_k$ be sequences over some finite alphabet Φ, of lengths $l_{S_1}, \cdots, l_{S_k}$.*

A multiple sequence alignment *of these k sequences is a function that maps each sequence S_i to a sequence S'_i of fixed length L, for $i = 1, 2, \ldots, k$. Each sequence S'_i is obtained by inserting* gaps *(represented by the character $'-' \notin \Phi$) between certain elements of sequence S_i.*

An alignment A, can be specified by a $k \times L$ matrix whose rows represent the new sequences $S'_1 \cdots S'_k$. Note that alignments are constrained so that the associated matrix does not contain any columns containing only gaps.

A single sequence will also be viewed as an alignment of one sequence. The jth column in an alignment A will be denoted $A[j]$, and the length of alignment A will be denoted l_A.

Definition 2.2 *An* optimal *alignment maximises the positional similarity over all columns of the resulting alignment according to a given similarity scoring function.*

The standard approach to calculating an optimal alignment is to use a dynamic programming algorithm based on a recurrence equation for the optimal alignment score. A full description of a suitable dynamic programming algorithm is given in the Appendix.

The dynamic programming algorithm makes use of three functions $G(i,j)$, $F(i,j)$, and $E(i,j)$, which are calculated for each entry in the dynamic programming table.

Definition 2.3 *Let A_x and A_y be alignments of lengths l_{A_x} and l_{A_y} respectively. For $1 \leq i \leq l_{A_x}$ and $1 \leq j \leq l_{A_y}$:*

- *$G(i,j)$ is the maximum possible score for aligning $A_x[1], \ldots, A_x[i]$ with $A_y[1], \ldots, A_y[j]$, given that $A_x[i]$ is aligned with $A_y[j]$;*
- *$F(i,j)$ is the maximum possible score for aligning $A_x[1], \ldots, A_x[i]$ with $A_y[1], \ldots, A_y[j]$, given that $A_x[i]$ is aligned with a gap;*

- *$E(i,j)$ is the maximum possible score for aligning $A_x[1],\ldots,A_x[i]$ with $A_y[1],\ldots,A_y[j]$, given $A_y[j]$ is aligned with a gap.*

In the next section we will show how the standard alignment algorithm can be modified to allow the incorporation of prior knowledge simply by adjusting the values of these three functions.

3 Incorporating Prior Knowledge

For many sequences of biological interest, certain specified subsequences have already been identified as corresponding to significant structural or functional elements. For example, certain subsequences may be known to act as ligand binding sites in protein sequences, or as transcription factor binding sites in DNA sequences. As another example, certain regions of a DNA sequence may have been found experimentally to be hypersensitive sites for enzymatic degradation.

To allow this kind of prior knowledge to be incorporated in the sequence alignment process we make the following definition.

Definition 3.1 *A set of sequences which are regarded as sharing some significant feature that makes them likely to align together will be called a* motif set*. Each element of a motif set is called a* motif*.*

Example 3.2 Figure 3 shows a collection of motif sets, labelled A to G. Each of these motif sets contains 8 different sequences corresponding to a particular β-chain of the Immunoglobulin domain [3] across a variety of proteins. Each motif set has being obtained experimentally by determining the 3-dimensional structure of these proteins. □

A	B	C	D	E	F	G
VLTQPP	TISCTG	NVKWY	SVSKS	TSATLAI	YYCQSY	VFG
VLTQPP	TISCSG	TVNWY	SGSKS	ASASLAI	YYCAAW	VFG
QLVQSG	RLSCSS	AMYWV	TISRN	NTLFLQM	YFCARD	YWG
QLEQSG	SLTCTV	YWTWV	TMLVN	NQFSLRL	YYCARN	VWG
SVFLFP	EVTCVV	KFNWY	KTKPR	VVSVLTV	YKCKVS	IEK
SVTLFP	TLVCLI	TVAWK	GVETT	ASSYLSL	YSCQVT	VEK
SVFPLA	ALGCLV	TVSWN	GVHTF	LSSVVTV	YICNVN	VDK
QVYTLP	SLTCLV	AVEWE	NYKTT	LYSKLTV	FSCSVM	TQK

Figure 3: 7 motif sets for immunoglobulins

Note that we are not assuming that any of the motifs in a motif set are necessarily present in the sequences being aligned. In some cases it will be convenient to simply specify an entire database of relevant motif sets and let the program determine which of the motifs actually occur. For example, if we are dealing with regulatory regions of DNA sequences, then it may be useful to specify a motif set for each known transcription factor, containing the possible binding sites for that factor. Note also that we are not assuming that the motifs in every motif set are necessarily short. In some cases it may be useful to include quite large sequences in a motif set in order to identify large sequence features.

Once we are given information about motifs, we can modify the sequence alignment process to attempt to preserve the structure of these motifs; in other words, to reduce the likelihood of inserting a gap within a motif. We do this by adjusting the definition of the functions $F(i,j)$ and $E(i,j)$ introduced in the previous section.

The modified definitions of $F(i,j)$ and $E(i,j)$ are as follows:

$$F'(i,j) = F(i,j) - \beta(A_y[j-1], A_y[j]) \times W_b \tag{1}$$

$$E'(i,j) = E(i,j) - \beta(A_x[i-1], A_x[i]) \times W_b \tag{2}$$

Where the function β is defined as follows:

Definition 3.3 *The function $\beta(A[i-1], A[i])$ is a Boolean function that returns the value 1 if and only if for some motif m, and some sequence $S \in A$, there exists a subsequence $S' = S[j], S[j+1], \cdots, S[j+n]$ of S such that $m = S'$ and $j < i \leq j+n$.*

With these modified definitions of the functions F and E , the score for an alignment which splits a motif is decreased by a factor of W_b, which makes such a split less likely to happen.

Furthermore, we can also modify the sequence alignment process to ensure that sequence regions associated with motifs in the same motif set are more likely to be aligned together. We do this by adjusting the definition of the function $G(i,j)$ introduced in the previous section. In order to describe the modifications to $G(i,j)$ more concisely we introduce the following terminology:

Definition 3.4 *Let A be an alignment of k sequences, and let M be a motif set.*

For any column $A[i]$ of A, and any motif m of M, we say that $A[i]$ intersects *m if for some sequence $S \in A$, m occurs as a subsequence of S and this subsequence includes $S[i]$.*

Any column $A[i]$ of A is said to be associated with a motif set *M if it intersects any motif m of M.*

We say that two columns are strongly related *if they are both associated with the same motif set.*

Example 3.5 Consider the protein sequence shown in Figure 4 and the motif sets defined in Example 3.2. As the shading in Figure 4 indicates, columns 6 to 11 *intersect*

```
012345678901234567890123456789012345678
SREEMTVLTQPPSASGTPGQRVSVTLFPPSSEELQANKA
```

Figure 4: A protein sequence containing two motifs

motif $VLTQPP$ which is contained in motif set A and columns 22 to 27 *intersect* motif $SUTLFP$ which is also contained in motif set A. Hence, these columns are all *associated with motif set* A, and so are all *strongly related* to each other. □

Example 3.6 Consider the alignment shown in Figure 1 and the motif sets defined in Example 3.2.

Columns 2 to 7 of the alignment intersect motifs $VLTQPP$, $QLVQSG$ and $QLEQSG$ which are all contained in motif set A. Similarly, columns 6 to 11 intersect motifs $SVTLFP$, $QVYTLP$ and columns 9 to 14 intersect motif $SVFPLA$, and these motifs are also all contained in motif set A. Hence columns 2 to 14 are all associated with motif set A, and so are all strongly related to each other. □

In order to increase the probability that columns which are associated with the same motif set are aligned together, we modify the definition of the function $G(i, j)$ as follows:

$$G'(i,j) \quad = \quad G(i,j) + \phi(A_x[i], A_y[j]) \times W_t \tag{3}$$

Where the function ϕ is defined as follows:

Definition 3.7 *The function $\phi(u, v)$ is a Boolean function that returns the value* 1 *if and only if both its arguments are strongly related to each other.*

With this definition, the score for aligning such columns together is increased by a factor of W_t, which makes it more likely to happen.

Example 3.8 The effect on the alignment process of these modifications has already been illustrated in Figures 1 and 2.

The alignment in Figure 2 is obtained simply by giving the modified algorithm a description of motif set A, defined in Example 3.2. Notice that the resulting alignment not only aligns this motif set correctly, but also obtains an improved alignment of motif sets B and C, which were not given as prior knowledge, but are simply marked in the final alignment. This result indicates the way in which incorporating some prior knowledge can lead to improved alignments which facilitate the discovery of further significant features in sequences. □

4 Implementation of IMSA

The modifications we have described in the previous section are implemented in the program IMSA which is available on request from the authors.

To calculate $F'(i, j)$ and $E'(i, j)$, we make use of what we call a *binary sequence tag*.

Definition 4.1 *A* binary sequence tag *B is an array of Boolean values, with one entry for each column of an alignment. Each entry $B[i]$ of B records the value of $\beta(A[i-1], A[i])$ (see Definition 3.3) for adjacent columns $i-1, i$ $(i > 1)$ of alignment A.*

Two binary sequence tags, B_x and B_y are created for the two alignments A_x and A_y respectively. We can then compute the functions $F'(i, j)$ and $E'(i, j)$ in terms of B_x and B_y as follows:

$$F'(i,j) \quad = \quad F(i,j) - B_y[j] \times W_b \tag{4}$$

$$E'(i,j) \quad = \quad E(i,j) - B_x[i] \times W_b \tag{5}$$

To calculate $G'(i, j)$, we make use of what we call a *sequence tag*, defined as follows:

Definition 4.2 *A* sequence tag *T is an array of numerical values, with one entry for each column of an alignment. The value of each entry $T[i]$, represents annotation information encoding all motif sets associated with the column, $A[i]$.*

Two sequence tags, T_x and T_y are created for the two alignments A_x and A_y respectively. We can then compute the function $G'(i,j)$ in terms of T_x and T_y as follows:

$$G'(i,j) \quad = \quad G(i,j) + \Delta(T_y[i], T_x[j]) \times W_t \tag{6}$$

Where $\Delta(u,v)$ is a function that returns the value 1 if there is some motif set M which is encoded by both u and v. It returns 0 otherwise.

The encoding that is used for the sequence tags has a great impact on the efficiency and storage requirements of the resulting program. Encoding of the binary sequence tags is straightforward: a single bit can be used to represent a tag for each column of the alignment, setting this bit to 1 or 0 as appropriate. Encoding of the general sequence tags is a little more complicated, and we will discuss this in more detail.

One obvious encoding scheme for a general sequence tag is to use a vector of n bits, where each bit represents a different motif set. The association of a motif set with a column $A[i]$ can then be recorded by setting the appropriate bit in the tag vector $T[i]$. However, using this encoding scheme means that each tag consists of a vector of bits whose length depends on the number of motif sets. The storage requirements for this form of encoding therefore grow linearly with the number of motif sets. Since we want to allow the possibility of specifying large numbers of motif sets, (such as a database of transcription factor binding sites), this bit-wise encoding scheme is not ideal.

Example 4.3 For a realistic example with, say, 100 motif sets, and sequences of length 2000, we would need $(2 \times 2000 \times 100)/8 = 50,000$ *bytes* for the sequence tags of both alignments. □

To simplify the storage of the sequence tags, and reduce the overall storage requirements, we propose an alternative encoding scheme that generates tag numbers as required, in such a way that a new number is generated only if an alignment column is associated with a novel collection of motif sets, not previously encountered. This is a more robust encoding scheme, since each tag simply holds a numerical value, and the size of the tags can be fixed to, say, 2 bytes. An implementation using this scheme can allow large numbers of motif sets to be specified, by exploiting the fact that, for typical data sets, only a very small proportion of the possible combinations of motif sets actually occur.

The relationship between the tag numbers is stored in an *attraction matrix*, which is used to hold the values of the function Δ referred to in Equation 6. Since this attraction matrix is symmetric, we can reduce the storage requirements further by using a triangular matrix and storing only half of the entries.

Example 4.4 For example, if we again consider 100 motif sets and sequences of length 2000, with this encoding scheme we only need $2 \times 2000 \times 2 = 8000$ *bytes* for the tags (using 2 *bytes* for each tag).

A reasonable empirical estimate after observing sequence data suggests that around 400 different combinations of associated motif sets will actually occur. Hence the scheme would use around 400 distinct tag numbers, and create a 400×400 attraction matrix. Storing one half of this symmetric matrix requires a further $(\frac{400^2}{2} + \frac{400}{2})/8 =$ $10,025$ *bytes*, giving a total storage requirement of $18,025$ *bytes*. □

An algorithm for generating the tags and the corresponding attraction matrix is described in Figure 5. In this algorithm, the tag values for the alignments A_x and A_y are stored in the integer vectors T_x and T_y. The attraction matrix is stored in the triangular Boolean matrix Δ. The integer variable n is used to keep track of the tag number which is assigned to the motif set currently being processed, and the integer variable $nextn$ is used to hold the next available tag number. The integer vector R is used to record the tag numbers that have so far been assigned to motif set combinations which include the current motif set. The variables Δ, n, $nextn$, and R are used as global variables in the procedure SetTag, which determines the sequence tag values for a given alignment and a given motif.

We now give an example to illustrate how the algorithm in Figure 5 computes the sequence tags for a pair of short sequences and a small number of motif sets.

Example 4.5 Consider the following two sequences:

Sequence 1: VFPACTKLCTGCTRTGACKQD

Sequence 2: QCACKALCPTGART

and the following four motif sets:

Motif set 1: { ACTK, GCT, ACKA }

Motif set 2: { TKLC, TGA }

Motif set 3: { LCTGC }

Motif set 4: { PACT, KLCT, GCTR }

Using the algorithm in Figure 5, the two sequences will be tagged using the four motif sets, one set at a time. Each position in the sequence which intersects a motif is tagged with the tag number for the combination of motif sets it is associated with. If that combination has been tagged before then the same tag number is used. If it is a first occurrence of a combination, then a new tag number is generated for it and the column is tagged with the new number. Tagging for this example proceeds as follows:

Sequence 1:

Sequence:	V	F	P	A	C	T	K	L	C	T	G	C	T	R	T
set 1:	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0
set 2:	0	0	0	1	1	3	3	2	2	0	1	1	1	0	2
set 3:	0	0	0	1	1	3	3	5	5	4	6	6	1	0	2
set 4 (final tag):	0	0	7	8	8	9	9	10	10	11	12	12	8	7	2

Sequence cont ...:	C	K	Q	D	G	A
set 1:	0	0	0	0	0	0
set 2:	2	2	0	0	0	0
set 3:	2	2	0	0	0	0
set 4 (final tag):	2	2	0	0	0	0

Sequence 2:

Sequence:	Q	C	A	C	K	A	L	C	P	T	G	A	R	T
set 1:	0	0	1	1	1	1	0	0	0	0	0	0	0	0
set 2:	0	0	1	1	1	1	0	0	0	2	2	2	0	0
set 3:	0	0	1	1	1	1	0	0	0	2	2	2	0	0
set 4 (final tag):	0	0	1	1	1	1	0	0	0	2	2	2	0	0

The final attraction matrix looks like this:

```
program CreateTags
% Create sequence tags T_x and T_y and attraction matrix Δ
% for alignments A_x and A_y, using given collection of motif sets

   integer vector   T_x, T_y                  % Sequence tags
   boolean matrix   Δ                         % Attraction matrix
   integer          n                         % Tag number for current motif set
   integer          nextn                     % Next available tag number
   integer vector   R                         % Tag numbers for combinations

   input A_x, A_y, and motif sets

   for all 1 ≤ i ≤ l_{A_x} do T_x[i] := 0     % Initialise sequence tags for A_x
   for all 1 ≤ i ≤ l_{A_y} do T_y[i] := 0     % Initialise sequence tags for A_y
   Δ[0,0] := FALSE                            % Initialise attraction matrix
   nextn := 1                                 % Initialise next available tag number

   for all motif sets M do
      n := nextn                              % Set tag number for current motif set
      for all motifs m of M do
         call SetTag(T_x, A_x, m)             % Set sequence tags for alignment A_x
         call SetTag(T_y, A_y.m)              % Set sequence tags for alignment A_y

procedure SetTag(T, A, m)
% Set sequence tags T and attraction matrix Δ for alignment A using motif m

   for all sequences S of A do
      for all occurrences of m in S do
         index := start position of m in S
         if n = nextn then                    % If first occurrence of new motif set...
            R[0] := n                         % ...initialise the reference vector R
            for all 1 ≤ i < n do R[i] := 0
            Δ[n,n] := TRUE                    % ...add row to attraction matrix
            for all 0 ≤ i < n do Δ[n,i] := FALSE
            nextn := nextn + 1                % ...increment next available tag number
         for all 0 ≤ j ≤ l_m do               % For each character in motif m
            t := T[index+j]                   % Store current tag value
            if t < n then                     % If tag value needs updating...
               if R[t] = 0 then               % ...if no tag value for this combination...
                                              % ......add row to attraction matrix
                  for all n ≤ i ≤ nextn do Δ[nextn,i] := TRUE
                  for all 0 ≤ i < n do Δ[nextn,i] := Δ[max(t,i), min(t,i)]
                  R[t] := nextn               % ......record chosen value in reference vector
                  nextn := nextn + 1          % ......increment next available tag number
               T[index + j] := R[t]           % ...set tag value to new tag number
```

Figure 5: Algorithm to generate sequence tags and attraction matrix

0	0	**1**	**2**	3	**4**	5	6	**7**	8	9	10	11	12
0	F	F	F	F	F	F	F	F	F	F	F	F	F
1		**T**	F	T	F	F	T	F	T	T	F	F	T
2			**T**	**T**	F	T	F	F	F	T	T	F	F
3				**T**	F	T	T	F	T	T	T	F	T
4					**T**	**T**	**T**	F	F	F	T	T	T
5						**T**	**T**	F	F	T	T	T	T
6							**T**	F	T	T	T	T	T
7								**T**	**T**	**T**	**T**	**T**	**T**
8									**T**	**T**	**T**	**T**	**T**
9										**T**	**T**	**T**	**T**
10											**T**	**T**	**T**
11												**T**	**T**
12													**T**

The alignment which then results from using these sequence tags and this attraction matrix is as follows:

Sequence 1:	V	F	P	A	C	T	K	L	C	T	G	C	T	R	T	G	A
Sequence 2:	-	Q	C	A	C	K	A	L	-	-	-	-	C	P	T	G	A

Sequence 1:	C	K	Q	D
Sequence 2:	R	T	-	-

□

By examining the loops in the program shown in Figure 5, we can determine its time complexity. Alignments are tagged by checking for all matches between each motif and each of the sequences making up the alignments. Hence, the time complexity of tagging the alignments grows linearly with the total number of motifs in all motif sets, and the number of sequences in each alignment. The overall time complexity is critically dependent on the string matching algorithm used for finding the matching subsequences. An algorithm such as the exact set matching algorithm of Aho & Corasick [2] can be used to find the presence of each motif in time which is linear in the length of the sequences plus the total length of all the motifs.

We note that each entry in the attraction matrix is modified only once, when it is created, so the additional time complexity of computing the attraction matrix is linear in the size of the final matrix.

The current implementation of IMSA is written in C and uses a primitive string matching algorithm, rather than a more advanced linear time algorithm. It is still able to compute all of the alignments shown in this paper in less than 3 seconds on a standard PC with Pentium processor running at 450 MHz.

5 Applications

In this section we will give three examples to show how IMSA may be used to obtain improved alignments, compared to the basic dynamic programming algorithm. The first two examples involve protein sequences, and the third involves DNA sequences.

In all of the examples we use a gap opening cost of 8 and a gap extension cost of 2. The binding penalty (W_b) was set to 5 and the attraction weight (W_t) was set to 25. These were the minimum parameter values that were consistent in producing good sequence alignments in all of the examples tested. For the protein sequences we used the PAM 100 substitution matrix [6] for scoring amino acid residue pairs. For the DNA sequences we used a standard scoring matrix awarding +4 for a match and −3 for a mismatch.

Example 5.1 The first example concerns a set of seven Globin sequences taken from Barton & Sternberg [3]. The seven sequences are the human haemoglobin α-strand (HAHU), the human haemoglobin β-strand (HBHU), the horse haemoglobin

α-strand (HAHO), the horse haemoglobin β-strand (HBHO), the sperm whale myoglobin (MYWHP), the sea lamprey cyanohaemoglobin (P1LHB) and the root nodule leghaemoglobin (LGHB). These sequences belong to the α/α class of proteins and have a common fold that is highly conserved even in proteins from distantly-related organisms.

An alignment of these sequences produced by the standard dynamic programming pairwise alignment algorithm is shown in Figure 6. The regions in the sequences boxed and marked A, B, C, E, F, G and H are known to correspond to α-helices.

An alignment of the same sequences produced using IMSA is shown in Figure 7. The motif information supplied to IMSA was the motif sets marked E,F,G and H for just the first three sequences: $P1LBH$, $MYWHP$ and $LGHB$. (These input motifs are highlighted in the alignment shown in Figure 7.) Using this information, IMSA aligns all seven α-helices across all seven sequences with no misalignments or gaps, as shown by the marked regions in Figure 7. Contrast this with the alignment produced by the standard algorithm shown in Figure 6 where there are a number of serious misalignments and spurious gaps inserted in all seven α-helices. □

Example 5.2 The second example concerns a set of 8 immunoglobulin sequences, also taken from Barton & Sternberg [3]. The 8 sequences are as follows (using Brookhaven Data Bank codes):

Four from 3FAB:	(a) light chain constant region $C\lambda$(FABCL)
	(b) light chain variable region $V\lambda$(FABVL)
	(c) heavy chain constant reg. 1 C_γ(FABCH1)
	(d) heavy chain variable region V_γ(FABVH)
Two from 1FC1:	(a)heavy chain constant reg. 2 $C_\gamma 2$(FCCH2)
	(b) heavy chain constant reg. 3 $C_\gamma 3$(FCCH3)
Two from 1FB4:	(a) light chain variable region $V\lambda$(FB4VL)
	(b) heavy chain variable region V_γ (FB4VH)

These sequences belong to the β/β class of proteins. Although the overall fold of these domains is conserved, they exhibit considerable sequence variation particularly between the variable and constant domains.

An alignment of these sequences produced by the standard dynamic programming algorithm is shown in Figure 8. The regions in the sequences boxed and marked A, B, C, D, E, F and G are known to correspond to β-strands.

An alignment of the same sequences produced using IMSA is shown in Figure 9. The motif information supplied to IMSA was the motif sets marked A,C,D,E,F and G for just the four sequences: $FCCH2$, $FABCL$, $FABCH1$ and $FB4VL$. (These input motifs are highlighted in the alignment shown in Figure 9.) Using this information, IMSA obtains a perfect alignment of the five β-strands A, B, C, E and F and only a single residue shift of the motif TQK of the β-strand marked G. The only serious misalignment which remains is for the β-strand marked D. Contrast this with the alignment produced by the standard algorithm shown in Figure 8 where none of the eight β-strands are properly aligned, and there are spurious gaps inserted in several of these known structural features. □

Our final example concerns DNA sequences. Tools for the alignment of DNA or RNA sequences are generally more primitive than those for protein sequences, because much less information can be inferred from the individual residues. However,

Figure 6: Alignment of 7 Globin sequences using the standard algorithm.

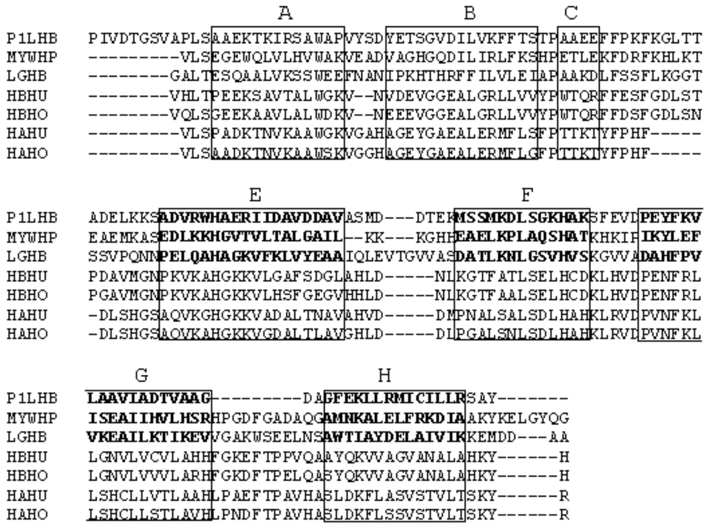

Figure 7: Alignment of 7 Globin sequences using IMSA.

Figure 8: Alignment of 8 immunoglobulin sequences using the standard algorithm.

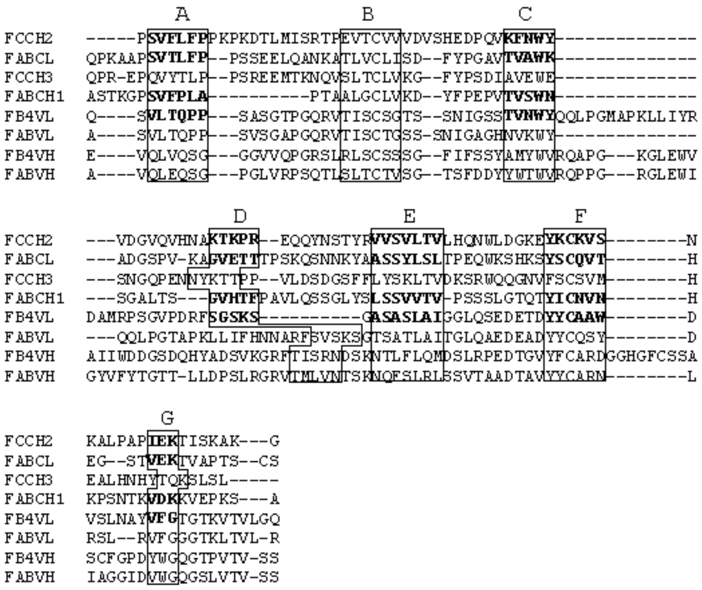

Figure 9: Alignment of 8 immunoglobulin sequences using IMSA.

the techniques we have described above for incorporating prior knowledge of motifs can be applied just as easily to DNA or RNA sequences as to protein sequences. The program IMSA is designed to work with sequences over any specified alphabet.

Example 5.3 This example concerns a set of 8 DNA sequences obtained by reverse translation from the SH2 fragments, of the set of Reference 2 alignments of the BALiBASE benchmark alignment database [21]. The DNA translations were taken from the EMBL Nucleotide sequence database [19].

The SH2 fragments of the set of Reference 2 alignments of the BALiBASE benchmark alignment database consist of 19 protein sequences. We used the DNA translations of the first eight of these sequences to give an illustrative example of the use of IMSA. The eight sequences were the fragments for the lymphocyte specific adapter protein lnk (lnk_rat), crk-like protein (crkl_mouse), cytoplasmic protein nck1 (nck_human), tyrosine-protein kinase htk16 (ht16_hydat), 1-hosphatidylinositol-4 5-bisphosphate phosphodiesterase gamma 2 (pip5_human), proto-oncogene tyrosine-protein kinase fer (fer_human), tyrosine-protein kinase fps85d (fps_drome) and megakaryocyte-associated tyrosine-protein kinase (ctk_rat).

An alignment of these sequences by the standard dynamic programming algorithm is shown in Figure 10. The regions in the sequences boxed and marked A, B, C, D, E and F correspond to the core blocks of these sequences identified in the BALiBASE benchmark alignment database.

An alignment of the same sequences using IMSA is shown in Figure 11. The motif information supplied to IMSA was the motif sets marked B,C,D,E and F for just the four sequences: lnk_rat, $pip5\_human$, fer_human and fps_drome. (These input motifs are highlighted in the alignment shown in Figure 11.) Using this information, IMSA obtains a perfect alignment of the core blocks [21] marked A, D and E, and a good alignment of the other three core blocks B, C and F. Contrast this with the standard alignment of the same DNA sequences shown in Figure 10 where many of the core blocks are very poorly aligned. □

6 Conclusion

In this paper we have described a novel computational approach to the multiple sequence alignment problem which allows the user to easily incorporate prior knowledge. We have shown that by annotating the sequences using this prior knowledge, and then modifying the scoring functions, it is possible to obtain alignments which are much more biologically informative.

We have demonstrated the effectiveness of this approach by using it on protein sequences with known structural features, and on DNA sequences coding for proteins with known homologous features. The same approach can also be used for investigating many other types of biological sequence data. For example, it could be used to align non-coding regions of DNA, using motif sets consisting of known transcription factor binding sites.

At present the program is implemented using a naive string matching algorithm to find the motif positions, but it is already sufficiently fast to be a useful and efficient alignment tool. We plan to develop the implementation using more sophisticated algorithmic techniques, and to investigate the application of this new tool to other types of biological sequences.

```
                       A                            B
lnk_rat     TACCCCTGGTTCCACGGCCCCATCTCCAGGGTGAGGGCTGCACAGCTGGTTCAGCTGCAG
crkl_mouse  TCTGCCTGGTACATGGGGCCAGTGACTCGCCAGGAGGC------GCAGACTCGTCTCCAA
nck_human   ------TGGTATTATGGCAAAGTCACCAGGCATCAAGCAGAAATGGCATTAAATGAAAGA
htl6_hydat  ------TGGTATCATGGCAAAATTACACGTGAGGTTGCTG---TGCAAGTTCTCCTTAGA
pip5_human  AAGCCGTGGTACTATGACAGCCTGAGCCGCGGAGAGGCAG---AGGACATGC---TGATG
fer_human   ------TGGTACCATGGTGCAATTCCCAGAATAGAAGC------TCAAGAACTGTTAAAA
fps_drome   GAGGAATGGTTCCATGGCGTTCTGCCGCGCGAGGAAG--------------TGGTTCGA
ctk_rat     ATGCCATGGTTCCATGGCAAGATTTCCGGTCAGGAAGC---------CATACAGCAGCTG

                                      C
lnk_rat     GGCCCTGATGCCCACGGTGTGTTCCTGGTGCGGCAGAGTGAGTCCCGGAGAGGGGA-ATA
crkl_mouse  GG---CCAGCGCCATGGCATGTTCCTAGTCCGGGACTCATCTACCTGCCCCGGGGA-CTA
nck_human   GGACATGAAGGGGA------TTTCCTCATTCGTGATAGTGAATCTTCGCCAAATGA-TTT
htl6_hydat  AAGGGTGGTAGAGATGGATTTTTTCTGATTAGAGATTGTGGAAACGCTCCTGAAGA-TTA
pip5_human  AGGATTCCCCGGGACGGGGCCTTCCTGATCCGGAAGCGAGAG---GGGAGCGACTC-CTA
fer_human   AAACAAG---------GAGACTTTTTGGTGCGAGAGAGTCATGGGAAACCTGGTGA-ATA
fps_drome   TTGCTGAATAACGATGGTGACTTCCTGGTCCGCGAAAC-GATTCGAAACGAGGAGAGCCA
ctk_rat     CAGCCACCCGAGGATGGGCTGTTCCTCGTGAGGGAATCGGCCCGTCACCCTGGTGA-CTA

                   D                         E
lnk_rat     TGTACT-CACATTCAACTTACAGGGGAGAGCCAAGCACCTCCGCCTGGTGCTCACAGAGC
crkl_mouse  TGTACT-GTCCGTGTCCGAGAACTCGCGTGTCTCGCACTACATCATCAACTCCCTGCCCA
nck_human   CTCAGT-ATCACTAAAAGCACAAGGGAAAAACAAGCATTTTAAAGTCCAACTAAAAGAGA
htl6_hydat  TGTTCT-CTCAATGATGTTTAGATCTCAAATATTGCATTTTCAAATAAACTGTTTAGGTG
pip5_human  TGCCAT-CACCTTCAGGGCTAGGGGCAAGGTAAAGCATTGTCGCATCAACCG---GGACG
fer_human   TGTCCT-TTCTGTATATTCTGATGGACAGAGGAGACATTTT---ATCATACAATATGTTG
fps_drome   GATTGTGCTCAGTGTCTGTTGGAATGGCCAT-AAGCACTTCATTGTCCAGACCACCGGAG
ctk_rat     TGTCCTG-TGTGTCAGTTTCGGCCGTGATGTCATCCACTACCGTGTC---TTGCATCGAG

                                                          F
lnk_rat     GTGGACAGTGCCGGGTGCAACA-CCTGCACTTCCCCTCGGT----GGTGGATATGCTCCG
crkl_mouse  ACCGCCGCTTTAAGATCGGGGA-CCAGGAGTTTGACCATTT----GCCGGCCTTGTTAGA
nck_human   CTGTCTACT---GCATTGGGCA-GCGTAAATTCAGCACCAT----GGAAGAACTTGTAGA
htl6_hydat  ATAATAAGTTTTCAATTGACAATGGACCAATTTTCCAAGGTCTCGATATGCTTATTTCTT
pip5_human  GCCGGCACTTTGTGCTGGGGACCTCCGCCTATTTTGAGAGTCTGGTGGAGC--TCGTCAG
fer_human   ATAACATGTATCGATTCGAGGGCACTG--GGTTTTCAAACATTCCTCAACTTATAGATCA
fps_drome   AGGGTAATTTCCGGTTCGAGGGACC-GCCATTTGCCAGCATC-CAGGAGCTGATCATGCA
ctk_rat     ATGGGCACCTCACCATTGATGAGGCCGTGTGTTTCTGTAATCTGATGGACATGGTAGAGC

lnk_rat     CCACTTCC-AGCGCTCC
crkl_mouse  GTTCTACA-AGATCCAC
nck_human   ACATTACA-AAAAGGCA
htl6_hydat  ATTATAAAGTAATTTCA
pip5_human  TTACTAC-GAGAAGCAT
fer_human   TCACTATACAACA-AAA
fps_drome   TCAGTAT-CACTCGGAA
ctk_rat     ACTACACCAGGACAAG
```

Figure 10: Standard alignment of the DNA sequences.

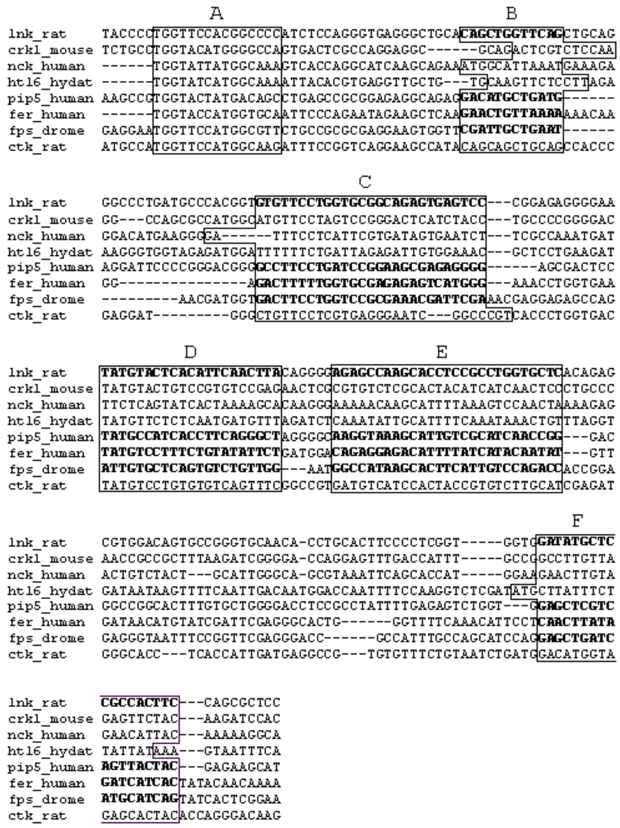

Figure 11: Alignment of DNA sequences using IMSA.

Acknowledgement

The authors wish to thank John Peden for pointing out the potential usefulness of a tool such as IMSA in many areas of biological sequence analysis.

Appendix: Dynamic Programming Algorithm

Let A_x and A_y be alignments of lengths l_{A_x} and l_{A_y} respectively, and let $V(i,j)$ be the score for an optimal alignment of the sub-alignment $A_x[1], \cdots, A_x[i]$ with the sub-alignment $A_y[1], \cdots, A_y[j]$. The score for the optimal alignment of A_x and A_y is then given by $V(l_{A_x}, l_{A_y})$.

The dynamic programming matrix for the alignment of A_x and A_y is computed by starting from a basis and recursively building on it to compute all values $V(i,j)$ where $0 \le i \le l_{A_x}$ and $0 \le j \le l_{A_y}$, in order of increasing i and j. The basis and the set of recurrence equations are as follows:

Basis :

$V(0,0) \;:=\; 0$
$\lambda(0,0) \;:=\; 0$
for *all* $1 \le i \le l_{A_x}$ **do**
  $V(i,0) \;:=\; 0$
  $\lambda(i,0) \;:=\; \lambda(i-1,0)+1$
  $\Theta(i,0) \;:=\; VERTICAL$

for *all* $1 \le j \le l_{A_y}$ **do**
  $V(0,j) \;:=\; 0$
  $\lambda(0,j) \;:=\lambda(0,j-1)+1$
  $\Theta(0,j) \;:=\; HORIZONTAL$

Where $\lambda(i,j) \in \mathbb{N}$ is the "gap status", and $\Theta(i,j) \in \{DIAGONAL,\ VERTICAL,\ HORIZONTAL\}$ is the "direction vector" of cell (i,j).

Recurrence :

for *all* $0 < i \le l_{A_x}$ **do**
  for *all* $0 < j \le l_{A_y}$ **do**
    if $max(G(i,j), F(i,j), E(i,j)) \;=\; G(i,j)$ **then**
      $V(i,j) \;:=\; G(i,j)$
      $\lambda(i,j) \;:=\; 0$
      $\Theta(i,j) \;:=\; DIAGONAL$

    if $max(G(i,j), F(i,j), E(i,j)) \;=\; F(i,j)$ **then**
      $V(i,j) \;:=\; F(i,j)$
      $\lambda(i,j) \;:=\; \lambda(i-1,j)+1$
      $\Theta(i,j) \;:=\; VERTICAL$
    if $max(G(i,j), F(i,j), E(i,j)) \;=\; E(i,j)$ **then**
      $V(i,j) \;:=\; E(i,j)$
      $\lambda(i,j) \;:=\; \lambda(i,j-1)+1$
      $\Theta(i,j) \;:=\; HORIZONTAL$

Where:

$$\begin{aligned}
G(i,j) &= V(i-1,j-1) + \sigma'(A_x[i], A_y[j]) \\
F(i,j) &= V(i-1,j) - !(\lambda(i-1,j)) \times W_g - \gamma(\lambda(i-1,j)) \times W_s \\
E(i,j) &= V(i,j-1) - !(\lambda(i,j-1)) \times W_g - \gamma(\lambda(i,j-1)) \times W_s
\end{aligned}$$

In these equations, W_g and W_s are the gap start-up cost and gap extension cost respectively.

The function γ allows different scores to be associated with gaps of different lengths. In the standard affine gap model we are using, γ simply returns the constant value 1.

The operator $!(x)$ for integer x is equivalent to the logical negation operator in the programming language C, and is defined as follows:

$$\begin{aligned} !(0) &= 1 \; and \\ !(x) &= 0 \; for \; all \; x \neq 0 \end{aligned}$$

The scoring function, $\sigma'(A_x[i], A_y[j])$, which gives the total score for aligning the i^{th} column $A_x[i]$ of alignment A_x with the j^{th} column $A_y[j]$ of alignment A_y is given by:

$$\sigma'(A_x[i], A_y[j]) = \sum_{c=1}^{K_{A_x}} \sum_{d=1}^{K_{A_y}} \sigma(A_x^c[i], A_y^d[j]) / (K_{A_x} K_{A_y})$$

Where $\sigma(x, y)$ is a function that assigns a score for aligning x with y, where x and y are each a single residue or a gap, and:

$$\begin{aligned} K_{A_x} &= \text{the number of sequences in alignment } A_x \\ K_{A_y} &= \text{the number of sequences in alignment } A_y \\ A_x^c[i] &= \text{the } i^{th} \text{ residue of sequence } c \text{ in alignment } A_x \\ A_y^d[j] &= \text{the } j^{th} \text{ residue of sequence } d \text{ in alignment } A_y \end{aligned}$$

To solve the above equations, we only need to store the score $V(i, j)$, the gap-status $\lambda(i, j)$, and the direction vector $\Theta(i, j)$, for each cell in the dynamic programming matrix. The final alignment is given by the path through the dynamic programming matrix starting from $V(l_{A_x}, l_{A_y})$ which is indicated by the direction vector.

References

[1] Churchill G. A. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.

[2] A Aho and M Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[3] G. J Barton and M. J. E Sternberg. A strategy for the rapid multiple alignment of protein sequences. *Journal of Molecular Biology*, 198:327–337, 1987.

[4] M. P Berger and P. J Munson. A novel randomised iterative strategy for aligning multiple protein sequences. *Computer Applications in the Biosciences*, 7:479, 1991.

[5] K Binder and D. W Heerman. *Monte Carlo simulation in Statistical Physics*. Springer-Verlag, 1988.

[6] M. O Dayhoff, R. M Schwartz, and B. C Orcutt. A model of evolutionary change in protein. In *In Dayhoff, M. O, ed., Atlas of Protein Sequence and Structure*, volume 5 of *3*. National Biomedical Research Foundation, Washington D.C, 1978.

[7] D Feng and R. F Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360, 1987.

[8] Genetics Computer Group. Wisconsin Package v.8.

[9] O Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[10] O Gotoh. Significant improvement in accuracy of multiple protein alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264:823, 1996.

[11] P Hogeweg and B Hesper. The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of Molecular Evolution*, 20:175–186, 1984.

[12] B Morgenstern, A Dress, and T Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, 1996.

[13] S. B Needleman and C. D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[14] C Notredame and D.G Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24:1515–1524, 1996.

[15] C Notredame, L Holm, and D.G Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14:407–422, 1998.

[16] N. Saitou and M Nei. The neighbour-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[17] T. F Smith and M. S Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, 1981.

[18] R.R Sokal and C.D Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[19] G Stoesser, W Baker, A Van den Broek, E Camon, M Garcia-Pastor, C Kanz, T Kulikova, R Leinonen, Q Lin, V Lombard, R Lopez, N Redaschi, P Stoehr, M Tuli, K Tzouvara, and K Vaughan.

[20] W.R Taylor. A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28:161–169, 1988.

[21] J Thompson, F Plewniak, and O Poch. BALiBASE: A benchmark alignments database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, 15:87–88, 1999.

[22] J. D Thompson, D. G Higgins, and T.J Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22, 1994.

[23] L Wang and T Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.

[24] M. S Waterman and M. D Perlwitz. Line geometries for sequence comparisons. *Bulletin of Mathematical Biology*, 46:567–577, 1984.