CABIOS

Vol. 13 no. 6 1997
Pages 565-581

A genetic algorithm for multiple molecular sequence alignment

Ching Zhang and Andrew K.C. Wong

Department of Systems Design Engineering, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

Received on December 12, 1996 revised on March 17, 1997, accepted on May 21, 1997

Abstract

Motivation: *Multiple molecular sequence alignment is among the most important and most challenging tasks in computational biology. The currently used alignment techniques are characterized by great computational complexity, which prevents their wider use. This research is aimed at developing a new technique for efficient multiple sequence alignment.*

Approach: *The new method is based on genetic algorithms. Genetic algorithms are stochastic approaches for efficient and robust searching. By converting biomolecular sequence alignment into a problem of searching for optimal or near-optimal points in an 'alignment space', a genetic algorithm can be used to find good alignments very efficiently.*

Results: *Experiments on real data sets have shown that the average computing time of this technique may be two or three orders lower than that of a technique based on pairwise dynamic programming, while the alignment qualities are very similar.*

Availability: *A C program on UNIX has been written to implement the technique. It is available on request from the authors.*

Contact: *E-mail: czhang@watnow.uwaterloo.ca*

Introduction

Multiple molecular sequence alignment is characterized by very high computational complexity. The dynamic programming method (Sankoff, 1972; Gotoh, 1982; Waterman, 1984) has been accepted as an effective method for two-sequence optimal alignment. In aligning two sequences of lengths m_1 and m_2, the straightforward implementations of this method need $O(m_1m_2)$ time and $O(m_1m_2)$ space. An improved algorithm of dynamic programming by Hirschberg (1975) can produce an optimal alignment in $O(m_2)$ space (Myers and Miller, 1988), where m_2 is the shorter sequence length. However, when the dynamic programming method is used for simultaneous multiple sequence alignment, the computational complexity is $O(m^n)$ theoretically, where n is the number of sequences and m is the length of the sequences.

A number of methods have been developed to align multiple sequences with reasonable computer resources. Many of them use heuristics to find good alignments that are not necessarily optimal. Existing heuristic methods for multiple sequence alignment may be categorized into five different approaches: (i) the subsequence approaches; (ii) the tree approaches; (iii) the consensus sequence approaches; (iv) the clustering approaches; (v) the template approaches. For a comprehensive survey, see Chan *et al.* (1992). Of the more recent methods, many combine heuristic techniques with pairwise dynamic programming. These methods include those: (i) aligning every pair of sequences, (ii) aligning each sequence with a pre-selected 'basic' sequence, (iii) aligning sequences in an arbitrary order or (iv) aligning sequences following the branching order in a phylogenetic tree (e.g. Feng and Doolittle, 1987; Chan, 1990; Gotoh, 1990; Vingron and Argos, 1991; Roytberg, 1992; Vihinen *et al.*, 1992; Thompson *et al.*, 1994). Some of the methods involve creating a sequence graph (or tree) with each edge representing a pairwise alignment (Gusfield, 1993; Miller, 1993). Others involve grouping sequences according to their structural similarity or species origins and then conducting intra- and inter-cluster alignments (Miller, 1993). The results of these methods may have deviations from the optimal and most of the methods are still very costly. When aligning n sequences, they usually require at least $(n - 1)$ pairwise alignment processes. The high costs have limited wider use of the methods. More efficient methods are required.

In this paper, a new method for simultaneous multiple sequence alignment is presented. This method is characterized by very high efficiency in both computing time and memory space. Its quality, measured in a series of experiments, proves to be satisfactory for most analysis purposes. The method is based on genetic algorithms.

Methods and system

An overview of genetic algorithms

Genetic algorithms are a set of stochastic algorithms for efficient and robust searching. They are developed in a way to simulate biological evolutionary process and genetic oper-

ations on chromosomes. Unlike the majority of the conventional search algorithms, a genetic algorithm starts with a population of points (states) in the problem space instead of a single one. In each step of a search, it generates a new and usually a better generation of points. Searching towards the better points may improve efficiency, and searching with a population may minimize the chance of falling into local extrema (Goldberg, 1989a; Michalewicz, 1992). Genetic algorithms are suitable for problems with large, complex, and poorly understood search spaces (De Jong, 1988).

Since the pioneering work by Holland (1975), research has been conducted to develop genetic algorithms. Considerable progress has been achieved in the theoretical work, e.g. by Bethke (1981), Grefenstette and Fitzpatrick (1985), Holland (1987), Goldberg (1987, 1989a,b), and Buckles and his colleagues (1990). Meanwhile, genetic algorithms have been successfully applied in many fields, e.g. rule-based intelligent systems (Holland, 1986), symbolic learning (Spears and De Jong, 1990) and complex control system optimization (Grefenstette, 1986). In the fields of computational biology, genetic algorithms have been used as effective tools for protein and RNA structure studies, including modeling the evolution of the zinc finger sequence motif of protein (Dandekar and Argos, 1992), simulating protein folding (Unger and Moult, 1993) and predicting RNA secondary structures (van Batenbury *et al.*, 1995; Gultyaev *et al.*, 1995).

Concepts, components and operations of a genetic algorithm

The major components of a genetic algorithm include a string representation of points (states) in the problem space, a fitness function, and three operations on the strings: reproduction, crossover and mutation.

The string representation of the points is denoted as:

$$A = a_1\, a_2 \ldots a_l \tag{1}$$

where a_i ($1 \le i \le l$) is the ith string elements and l is the length of the string. The elements are taken from a domain $\mathfrak{D}$ which is a collection of symbols, i.e. $a_i \in \mathfrak{D}$. Usually $\mathfrak{D}$ is {0, 1}. A string of l elements has l positions on it, which are occupied by elements $a_1, a_2, \ldots$, and a_l, respectively. Let Ω denote the problem space and $\omega \in \Omega$ be the generic representation of points in Ω. An encoding procedure g is needed to encode the points into strings:

$$A = g(\omega) \tag{2}$$

The fitness function f is used to evaluate fitness (goodness) of strings. It can be expressed as:

$$f(A) = r \tag{3}$$

where $r \in R$ and R is a set of real numbers.

A search process starts with a population of strings. In a standard genetic algorithm, strings are of the same length. The size of the population may be constant. In each step of the search, the three operations are applied to the strings to create a new generation.

Reproduction duplicates strings of high fitness values. For string A_i, the number of duplicates is calculated as:

$$\text{round}\,(Q \cdot p_i) \tag{4}$$

where Q is the population size and p_i is the probability for string A_i to be duplicated, which is calculated as:

$$p_i = \frac{f(A_i)}{\sum_{j=1}^{Q} f(A_j)} \tag{5}$$

The duplicated strings are placed in a mating pool for creating a new generation of strings.

Crossover is conducted on the strings in the mating pool. The operation mates two strings to create two new ones and the frequency of crossover is controlled by a crossover probability p^c. There are three steps in a cross-over operation:

1. Randomly select two strings, denoted by A_1 and A_2, from the mating pool.
2. For string A_i ($1 \le i \le 2$), randomly select a cutting position k_i ($1 \le k_i < l_i$) on it where l_i is the length of A_i. Cut between the k_ith and (k_i + 1)th elements to split the string into two substrings (head and tail).
3. Create two new strings by exchanging A_1 and A_2's heads.

Mutation is the alternation of the values of string elements. In a mutation operation, a string is randomly selected from the mating pool and a mutation position on it is selected between position 1 and position l inclusively. The element value at the mutation position is changed. The frequency of mutation is controlled by a mutation probability p^m.

A search may repeat the three operations until the process converges. At convergence, all the strings are exactly the same. A search may terminate when a certain condition is satisfied.

Schema is an important concept in genetic algorithms, which is helpful in studying the law of string growth. A schema is a similarity template describing a subset of strings which have the same elements at certain positions on the strings. Schemata are represented by the symbols in $\mathfrak{D}$ plus the symbol * which denotes 'do not care', i.e. $\mathfrak{D} \cup \{*\}$. Strings described by a schema are instances of it. In a schema, the positions where the elements are symbols in $\mathfrak{D}$ are called fixed positions. For example, if $\mathfrak{D}$ is the set of binary digits, i.e. $\mathfrak{D}$ = {0, 1}, schemata on $\mathfrak{D}$ are represented by using 0, 1 and *. '1 1 1 1 *' is a schema on $\mathfrak{D}$ which describes all the strings in which elements at the first four positions are ones and the element at the fifth position may be 0 or 1.

Two important properties of a schema are order and defining length. The order of schema H, denoted by o(H), is the

number of fixed positions. The defining length of schema H, denoted by $\delta(H)$, is the distance between the first and last fixed positions.

According to Goldberg (1989a), the expected number of instances of schema H in the next generation under reproduction, crossover and mutation can be given as:

$$n_H(t+1) = n_H(t) \cdot \left\{ \frac{f(H)}{\bar{f}} \cdot (1-p^c \cdot \frac{\delta(H)}{l-1})(1-p^m \cdot o(H)) \right\} \quad (6)$$

where $n_H(t)$ is the number of instances of H in generation t, $\bar{f}$ is the average fitness of the whole population and $f(H)$ is the average fitness of the instances of H.

Equation (6) indicates that the strings with high fitness values, short defining lengths and low orders have good probabilities to grow. This knowledge is useful in designing a genetic algorithm.

Use of a genetic algorithm for sequence alignment

The basic idea of using a genetic algorithm for multiple sequence alignment is to represent possible alignments as points (states) in a space and to search for an optimal or near-optimal point. This process involves coding the points into a population of genetic strings and associating each string with a fitness value, and applying the three genetic operations to the strings. In a search process, good strings are duplicated and their segments are combined to form better strings. Being able to search with a group of alignments and move in a 'good' direction, a genetic algorithm might be more efficient than many of the conventional search methods and may have less chances of falling into local extrema.

In practical applications, the molecular sequences to be aligned, such as DNA, RNA and protein sequences, are usually very long and numerous possible alignments may be constructed. For an alignment, its fitness value is determined not only by the number of matched subunits, but also by the numbers of insertions, deletions and substitutions. The effects of state transition on fitness values can hardly be predicted. For example, the increase in fitness value obtained by matching subunits at one position may be offset by the loss of substitutions at other positions. Thus, the multiple sequence alignment problem is characterized by large and complex search spaces. The conventional search methods may be inefficient in dealing with such a problem and may easily fall into local extrema. Genetic algorithms may provide better approaches for such a problem (De Jong, 1988).

The system

The computer system used for this research is an IBM RISC 6000 32H workstation with a processor rated at 32.2 MIPS and 11.7 MFLOPS. The main memory is 32 megabytes. The operating system is AIX (an IBM version of UNIX).

The algorithm for multiple sequence alignment

Conversion of sequence alignment into space search

In our approach, sequence alignment is conducted in two steps: (i) identifying matches and (ii) identifying mismatches (i.e. deletions, insertions and substitutions). The input of the first step is the sequences to be aligned and the output is the matched subunits. The matched subunits are organized in a form called pre-alignment. The pre-alignment is the input of the second step. The final result is an alignment. A genetic algorithm is designed for the first step.

To apply a genetic algorithm, the task of identifying matches is converted into a search problem. Before we discuss the conversion, necessary representations are given in the following. To avoid possible confusion, some terms are clarified: a biomolecular sequence consists of subunits. The subunits are represented by characters in domain $\mathcal{A}$. A string in a genetic algorithm consists of elements which are represented by symbols in domain $\mathfrak{D}$.

It is assumed that there are n sequences to be aligned. They are:

$$\begin{aligned} X^1 &= x_1^1 x_2^1 \ldots x_{m_1}^1 \\ X^2 &= x_1^2 x_2^2 \ldots x_{m_2}^2 \\ &\ldots \\ X^n &= x_1^n x_2^n \ldots x_{m_n}^n \end{aligned}$$

where $x_i^k \in \mathcal{A}$ is a subunit, superscript k indicates that the subunit belongs to the kth sequence, subscript i indicates that it is the ith subunit in the sequence, $1 \le i \le m_k$, m_k is the length of the kth sequence, and $1 \le k \le n$. m_a is used to denote the average length.

To take into account mismatches, we use φ to represent 'blank' elements. Thus, $\mathcal{A}' = \mathcal{A} \cup \{\varphi\}$ and x_0^k is used for φ, i.e., $x_0^k = \varphi$.

An alignment of the n sequences, denoted as $X^1 \# X^2 \# \ldots \# X^n$, is:

$$X^1 \# X^2 \# \ldots \# X^n = \begin{matrix} x^1_{u_{1,1}} & x^1_{u_{1,2}} & \cdots & x^1_{u_{1,m}} \\ | & | & \ldots & | \\ x^2_{u_{2,1}} & x^2_{u_{2,2}} & \cdots & x^2_{u_{2,m}} \\ | & | & \ldots & | \\ \cdots & \cdots & \cdots & \cdots \\ | & | & \ldots & | \\ x^n_{u_{n,1}} & x^n_{u_{n,2}} & \cdots & x^n_{u_{n,m}} \end{matrix} \quad (7)$$

where $x^k_{u_{k,i}} \in \mathcal{A}'$, | indicates a match or a mismatch, and $m \ge \max(m_1, m_2, \ldots, m_n)$.

An alignment has m columns and n rows. It satisfies the following conditions. (i) The subunits included in the ith row must be subunits in the ith sequence ($1 \le i \le n$). (ii) The align-

ment includes all the subunits in the input sequences. (iii) A subunit can appear only once in the alignment. (iv) The order of subunits in a sequence is preserved in the alignment. (v) There does not exist a column in which all the subunits are φ.

An alignment may also be denoted as a set: $X^1 \# X^2 \# ... \# X^n = \{(x^1_{u_{1,i}}, x^2_{u_{2,i}}, ..., x^n_{u_{n,i}})\}_{i=1}^{m}$ where $(x^1_{u_{1,i}}, x^2_{u_{2,i}}, ..., x^n_{u_{n,i}})$ may include matches and mismatches.

A pre-alignment of the n sequences, denoted as $X^1 \#' X^2 \#' ... \#' X^n$, is:

$$X^1 \#' X^2 \#' ... \#' X^n = \begin{array}{cccc} x^1_{u_{1,1}} & x^1_{u_{1,2}} & \cdots & x^1_{u_{1,m1}} \\ \| & \| & ... & \| \\ x^2_{u_{2,1}} & x^2_{u_{2,2}} & \cdots & x^2_{u_{2,m1}} \\ \| & \| & ... & \| \\ \cdots & \cdots & \cdots & \cdots \\ \| & \| & ... & \| \\ x^n_{u_{n,1}} & x^n_{u_{n,2}} & \cdots & x^n_{u_{n\,m1}} \end{array} \quad (8)$$

where $x^k_{u_{k,i}} \in \mathcal{A}$, $\|$ indicates a match, and m_1 is the length of X^1.

A pre-alignment has m_1 columns and n rows. It satisfies the following conditions. (i) The subunits included in the ith row must be subunits in the ith sequence ($1 \le i \le n$). (ii) A subunit can appear only once in the alignment. (iii) The order of subunits in a sequence is preserved in the alignment.

A pre-alignment is a preliminary state for generating an alignment. In a pre-alignment, match relationships have been identified which are usually considered as the most important relationships in an alignment. A column may be a match or entirely φ. The latter represents possible mismatches. By comparing the two definitions, we note that a pre-alignment allows columns of entire φ and may not include all the subunits in the input sequences. Such columns and the missing subunits will be processed in the subsequent step. The number of columns in a pre-alignment can be the length of any other sequence. Because, in a pre-alignment, only matches are processed, all the sequences can be considered to have the same length. Therefore, the choice of X^1 is arbitrary, and will not affect the quality of the alignment.

The following example is used to clarify the difference between a pre-alignment and an alignment, and explain some concepts involved in the conversion. For simplicity, reproduction and mutation are not concerned in the example.

Example:

(i) Three sequences to be aligned:

1. $C^1_1\ A^1_2\ C^1_3\ G^1_4\ T^1_5\ C^1_6$
2. $G^2_1\ A^2_2\ A^2_3\ T^2_4\ C^2_5$
3. $G^3_1\ A^3_2\ C^3_3\ T^3_4\ G^3_5\ C^3_6$

(ii) Two pre-alignments constructed from the three sequences:

$$\begin{array}{cccccc} G^1_1 & A^1_2 & \phi & \phi & \phi & C^1_6 \\ \| & \| & \| & \| & \| & \| \\ G^2_1 & A^2_2 & \phi & \phi & \phi & C^2_5 \\ \| & \| & \| & \| & \| & \| \\ G^3_1 & A^3_2 & \phi & \phi & \phi & C^3_3 \end{array}$$

$$\begin{array}{cccccc} \phi & \phi & \phi & \phi & T^1_5 & C^1_6 \\ \| & \| & \| & \| & \| & \| \\ \phi & \phi & \phi & \phi & T^2_4 & C^2_5 \\ \| & \| & \| & \| & \| & \| \\ \phi & \phi & \phi & \phi & T^3_4 & C^4_6 \end{array}$$

(iii) A good pre-alignment generated from the two pre-alignments by a crossover operation

$$\begin{array}{cccccc} G^1_1 & A^1_2 & \phi & \phi & T^1_5 & C^1_6 \\ \| & \| & \| & \| & \| & \| \\ G^2_1 & A^2_2 & \phi & \phi & T^2_4 & C^2_5 \\ \| & \| & \| & \| & \| & \| \\ G^3_1 & A^3_2 & \phi & \phi & T^3_4 & C^3_6 \end{array}$$

(iv) An alignment generated from the pre-alignment in (iii) by discovering a substitution and two insertions.

$$\begin{array}{ccccccc} G^1_1 & A^1_2 & C^1_3 & G^1_4 & T^1_5 & \phi & C^1_6 \\ | & | & | & | & | & | & | \\ G^2_1 & A^2_2 & A^2_3 & \phi & T^2_4 & \phi & C^2_5 \\ | & | & | & | & | & | & | \\ G^3_1 & A^3_2 & C^3_3 & \phi & T^3_4 & G^3_5 & C^3_6 \end{array}$$

To convert a sequence alignment problem into a search problem, a search space is defined which is called a pre-alignment space. A pre-alignment space, denoted as Ω, is an m_1-dimensional space where m_1 is the length of sequence X^1. Each position on X^1 defines an axis of the space. A pre-alignment corresponds to a point in Ω. The pre-alignment in equation (8) corresponds to point

$$\omega = \begin{array}{cccc} \widehat{u_{1,1}} & \widehat{u_{1,2}} & \cdots & \widehat{u_{1,m_1}} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m_1} \\ \cdots & \cdots & \cdots & \cdots \\ \underbrace{u_{n,1}} & \underbrace{u_{n,2}} & \cdots & \underbrace{u_{n,m_1}} \end{array} \quad (9)$$

A point representation has m_1 columns. The ith column consists of the subscripts in the ith column in equation (8), and is used as the ith coordinate of ω. The pre-alignment in (iii) of the above example corresponds to the following point:

$$\begin{array}{cccccc} \widehat{1} & \widehat{2} & \widehat{0} & \widehat{0} & \widehat{5} & \widehat{6} \\ 1 & 1 & 0 & 0 & 4 & 5 \\ \underbrace{1} & \underbrace{1} & \underbrace{0} & \underbrace{0} & \underbrace{4} & \underbrace{6} \end{array}$$

As illustrated in the above example, for a set of input sequences, a group of pre-alignments can be constructed which

correspond to the points in Ω. To use a genetic algorithm to search for a point corresponding to a good pre-alignment, the points must be represented as genetic strings. In the following, the procedure of constructing genetic strings is described.

In constructing genetic strings, the match tuples must first be found. A match tuple is a tuple of subscripts of n matched subunits with each from an input sequence. For example, the match tuple for

$$x^1_{v_{1,i}} = x^2_{v_{2,i}} = \ldots = x^n_{v_{n,i}}$$

is

$$(v_{1j}, v_{2j}, \ldots, v_{nj})$$

where v_{kj} are subscripts, $k = 1, 2, \ldots, n$, and $1 \le v_{kj} \le m_k$.

The match tuples are grouped into a collection of series which satisfy conditions (ii) and (iii) in equation (8). In the above example, two series are formed from the input sequences:

$$\begin{array}{lll} \widehat{1} & \widehat{2} & \widehat{6} \\ 1 & 2 & 5 \\ \underbrace{1} & \underbrace{2} & \underbrace{3} \end{array} \quad , \qquad \begin{array}{ll} \widehat{5} & \widehat{6} \\ 4 & 5 \\ \underbrace{4} & \underbrace{6} \end{array} \tag{10}$$

In each series, for each position on X^1, if there is not a match tuple whose first subscript is equal to the position number, a tuple of zeros is inserted. We thus generate the point representation of a pre-alignment. For instance, in the first series in equation (10), since there are no match tuples whose first subscripts are 3, 4 and 5, three tuples of zeros are inserted:

$$\begin{array}{llllll} \widehat{1} & \widehat{2} & \widehat{0} & \widehat{0} & \widehat{0} & \widehat{6} \\ 1 & 2 & 0 & 0 & 0 & 5 \\ \underbrace{1} & \underbrace{2} & \underbrace{0} & \underbrace{0} & \underbrace{0} & \underbrace{3} \end{array} \tag{11}$$

This is the point representation of the first pre-alignment in (ii) of the example.

Formally, the generation of a point representation from a series can be expressed as:

$$\begin{array}{cccc} \widehat{v}_{1,1} & \widehat{v}_{1,2} & \cdots & \widehat{v}_{1,q} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ \underbrace{v_{n,1}} & \underbrace{v_{n,2}} & \cdots & \underbrace{v_{n,q}} \end{array} \rightarrow \begin{array}{cccc} \widehat{u}_{1,1} & \widehat{u}_{1,2} & \cdots & \widehat{u}_{1,m_1} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m_1} \\ \cdots & \cdots & \cdots & \cdots \\ \underbrace{u_{n,1}} & \underbrace{u_{n,2}} & \cdots & \underbrace{u_{n,m_1}} \end{array} \tag{12}$$

where

$$u_{ik} = \begin{cases} v_{kj} & \text{if } \exists j (i = v_{1j}) \\ 0 & \text{otherwise} \end{cases}$$

and $1 \le i \le m_1$, $1 \le j \le q$, and $1 \le k \le n$.

The match tuples in the point representation are then grouped into match blocks. A match block consists of the largest possible number of successive match tuples. For example,

$$\begin{array}{ll} \widehat{1} & \widehat{2} \\ 1 & 2 \\ \underbrace{1} & \underbrace{2} \end{array}$$

is a match block in equation (11).

Formally, a match block of length r is

$$\begin{array}{cccccc} \widehat{u}_{1,i} & \widehat{u}_{1,i+1} & \cdots & \widehat{u}_{1,i+r-2} & \widehat{u}_{1,i+r-1} \\ u_{2,i} & u_{2,i+1} & \cdots & u_{2,i+r-2} & u_{2,i+r-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \underbrace{u_{n,i}} & \underbrace{u_{n,i+1}} & \cdots & \underbrace{u_{n,i+r-2}} & \underbrace{u_{n,i+r-1}} \end{array} \tag{13}$$

such that

$$u_{kj} + 1 = u_{kj+1}, u_{kj+1} + 1 = u_{kj+2}, \ldots, u_{kj+r-2} + 1 = u_{kj+r-1}$$

and $(u_{1j} - 1, u_{2j} - 1, \ldots, u_{nj} - 1)^T$ and $(u_{1j+r-1} + 1, u_{2j+r-1} + 1, \ldots, u_{nj+r-1} + 1)^T$ are not match tuples, for $k = 1, 2, \ldots, n$.

To be uniform, isolated single match tuples are also represented as blocks. Then, we make genetic strings out of match blocks. A genetic string corresponding to point ω is a string of match blocks, i.e.:

$$A = a_1 a_2 \ldots a_l$$

where a_i is the ith match block in ω and $1 \le l \le m_1$.

A block is an indecomposable element in the search process. Thus, the order and defining length of a schema should be calculated in terms of blocks. For a point of a fixed number of tuples (columns), more tuples of successive values result in less blocks. The point thus would be of lower order and shorter defining length. Usually, such a point represents a good alignment. Recall the discussion in the section on 'Concepts, components and operations of a genetic algorithm', a lower order and a shorter defining length may increase the probability for the string to survive.

By means of the procedure discussed above, a set of genetic strings can be constructed out of input sequences. They form the first generation. By applying the genetic operations of reproduction, cross-over and mutation to the population, a new generation which contains the same number of hybrid strings is generated. This process is repeated until a termination condition is satisfied. The best string in the last generation is used to make the output pre-alignment.

When applying crossover to two strings, we must check whether the strings are matable. Two strings are matable at given cutting positions if the two new strings satisfy conditions (ii) and (iii) in equation (8).

Algorithm parameters

Determining the appropriate population sizes and the mutation position is important in applying genetic algorithms. This section covers a discussion on the two issues as well as the condition for terminating the search processes. The fitness function is described.

Fitness function. The fitness value of a string is equal to the number of match tuples it contains.

Population size. The population size largely affects algorithm performance in terms of convergence and computational costs. If the size is too small, the risk of falling into a local maximum becomes high. However, if it is too big, the

```
                                                @@
fankt--------------------------------qnvllvaqyqfdfglrpsiaytkskakdvegigdvdlvny
fankt--------------------------------qnfeavaqyqfdfglrpslgyvlskgkdiegigdedlvny
fankatqfeavaqyqfsfglrpslgylskgkdieggsqn--------------------------------edlvny
```

Fig. 1. A 'harmful' element, which is marked by '@@'.

computation might be very costly. As Goldberg (1989b) suggested, relatively small population sizes are appropriate for serial implementations of genetic algorithms and large sizes are appropriate for parallel implementations.

Since the algorithm is coded as a serial program, we use small populational sizes. In an alignment task, the population size is determined when choosing a number of series to form the first generation. It was observed that of the series, there exists a small group of strings with high fitness values, while the rest have much lower values (<1/10 of the highest). Since strings of very low fitness values have small probabilities to survive, we use only the highly fit series to form the first generation. It is also observed that the number of highly fit series is proportional to the product of m_a and n, and the number is usually <1/100 of the product. For an alignment task, we choose the population size Q as:

$$Q = m_a n/100 \tag{14}$$

Selection of mutation positions. Mutation plays an important role in the algorithm. It is used to remove some 'harmful' elements from the strings. To pick up such elements, we define four distances.

For two successive blocks B_1 and B_2:

$$B_1 = \begin{matrix} \overbrace{u_{1,r}} & \overbrace{u_{1,r+1}} & \cdots & \overbrace{u_{1,s}} \\ u_{2,r} & u_{2,r+2} & \cdots & u_{2,s} \\ \cdots & \cdots & \cdots & \cdots \\ \underbrace{u_{n,r}} & \underbrace{u_{n,r+1}} & \cdots & \underbrace{u_{n,s}} \end{matrix} \quad B_2 = \begin{matrix} \overbrace{u_{1,t}} & \overbrace{u_{1,t+1}} & \cdots & \overbrace{u_{1,w}} \\ u_{2,t} & u_{2,t+2} & \cdots & u_{2,w} \\ \cdots & \cdots & \cdots & \cdots \\ \underbrace{u_{n,t}} & \underbrace{u_{n,t+1}} & \cdots & \underbrace{u_{n,w}} \end{matrix}$$

where $u_{i,r} \le u_{i,s}$, $u_{i,s} < u_{i,t}$, and $u_{i,t} \le u_{i,w}$ $(i = 1,\ldots, n)$.

The average left distance of B_2 is:

$$\mathrm{ALD}_{B_2} = [\sum_{i=1}^{n}(u_{i,t}-u_{i,s})]/n \tag{15}$$

The average right distance of B_1 is:

$$\mathrm{ARD}_{B_1} = \mathrm{ALD}_{B_2} \tag{16}$$

The maximum left distance of B_2 is:

$$\mathrm{MLD}_{B_2} = \max_i(u_{i,t}-u_{i,s}) \tag{17}$$

The maximum right distance of B_1 is:

$$\mathrm{MRD}_{B_1} = \mathrm{MLD}_{B_2} \tag{18}$$

For a string selected for mutation, the operation removes the element which has the largest D value and the D value is greater than a threshold, where:

$$D = [(\mathrm{MLD} - \mathrm{ALD}) + (\mathrm{MRD} - \mathrm{ARD})]/\mathrm{L} \tag{19}$$

where L is the length of the element (block).

In most cases, an element with a large D value is a 'harmful' one which prevents the string from mating with others to generate strings of high fitness values. The string element 'qn' in Figure 1 illustrates such an element.

Termination condition. The termination condition used in this algorithm stops the search process when the population has been converged to a string or when there is no change in the highest fitness value for 10 successive generations.

The procedure

Matching. Finding match blocks is the first step in the alignment algorithm. Conceptually, match tuples are first identified and match blocks are then formed by grouping them. While in implementing the algorithm, the match blocks are directly built for improving efficiency. The following is the procedure for building a match block when a match tuple is found:.

```
If ((v1,j,v2,j,..., vn,j) is a match tuple)
  Form a new match block which includes (v1,j, v2,j,..., vn,j);
  j = 1;
  While ((v1,j + j, v2,j + j,..., vn,j + j) is a match tuple)
    Add (v1,j + j, v2,j + j,..., vn,j + j) to the match block;
    j = j + 1;
  EndWhile
EndIf
```

The match tuples included in the match block will not participate in the subsequent searching for match blocks.

Construction of pre-alignment. Here is the genetic algorithm for pre-alignment generation. The sequences to be aligned are $X^1, X^2,\ldots$, and X^n which have lengths $m_1, m_2,\ldots$, and m_n, respectively. The output is pre-alignment $X^1\#' X^2\#' \ldots \#' X^n$.

procedure PRE-ALIGN (**Input:** $X^1, X^2,\ldots,X^n$; **Output:** $X^1\#' X^2\#' \ldots \#' X^n$)
begin
1. Find match blocks from $X^1, X^2,$, and X^n.
2. Create genetic strings from the match blocks (I – number of match blocks, J – number of strings):
   $J = 0$

For i = 1 through I do
For j = 1 through J do
If (A_j satisfies conditions (ii) and (iii) in (8) after inserting B_i into it)
Insert B_i into A_j;
Quit the inner loop;
EndIf
EndFor
If (B_i is not inserted)
$J = J + 1$;
Create A_J to contain B_i;
EndIf
EndFor

3. Form the first generation of string population

$$G^0 = \{A_1, A_2, \ldots, A_Q\}$$

where A_i ($i = 1, 2, \ldots, Q$) are the genetic strings of the highest fitness values and Q is the population size.

4. Apply reproduction to G^t where t denotes generation number ($t = 0, 1, \ldots$).
For i = 1 through Q do
Calculate the fitness value for $A_i \in G^t$;
Determine the number of duplicates for A_i using
Equation (4);
If (the number is greater than or equal to 1)
Duplicate A_i and place the duplicate(s) into matepool M^t;
EndIf
EndFor
The matepool is $M^t = \{A'_1, A'_2, \ldots, A'_Q\}$ where A'_i ($1 \le i \le Q$) is duplicated from a string in G^t.

5. Apply mutation to M^t:
Calculate the number of element(s) to be mutated;
Randomly select strings for mutation;
In each selected string, determine the mutation position using (15)–(19), and remove the element at the position;

6. Apply crossover to M^t to generate G^{t+1}:
While (There are strings in M^t to mate) do
Randomly select A_i and A_j from M^t;
Randomly select cutting positions on A_i and A_j,
If (A_i and A_j are matable at the positions)
Perform crossover on A_i and A_j to create two new strings;
Replace A_i and A_j with the new strings;
EndIf
EndWhile
$G^{t+1} = M^t$;

7. If the termination condition is not satisfied,
Go to step 4.

8. Select the best string from the population.

9. If there exist substrings of successive zeros longer than $m_a/20$ in the best string
Call PRE-ALIGN to align the substrings.

10. Produce X^1 #' X^2 #' ... #' X^n from the selected string.

end procedure

From the procedure and the description in the previous section, it can be observed that the genetic algorithm developed in this research deviates from a standard one. The major deviations are in the string coding and the selection of mutation position. The use of match blocks as string elements instead of binary digits is more expressive and efficient. Besides, the strings in a population may be of different lengths and this makes generating highly fit pre-alignments possible. In this algorithm, mutation positions are selected by using a set of distances [equations (15)–(18)], such that some harmful elements can be removed. Mutation operation plays a very important role in the new method.

Construction of alignment. This subsection describes the second step in multiple sequence alignment, i.e. identifying mismatches.

A pre-alignment is a series of match blocks and there exists a gap between every two adjacent blocks. A gap consists of one or more tuples of φ, which represent subunits missing from the pre-alignment. In constructing an alignment from a pre-alignment, we use the missing subunits to fill the gaps between match blocks. For every two adjacent match blocks, we do the following:

1. Find the missing subunits from the input sequences which should be in the gap.
2. Find sequence X^i which has the maximum number of such subunits ($1 \le i \le n$).
3. Fill row i of the gap of the pre-alignment with the subunits of X^i.
4. For j = 1 through n and $j \ne i$
Place the missing subunits $x^j{}_{u_{j,l}}, x^j{}_{u_{j,2}}, \ldots, x^j{}_{u_{j,s}}$ at the lower end of the gap in row j;
For $k = s$ down to 1.
Move $x^j{}_{u_{j,k}}$ to the column of $x^i{}_{v_{i,l}}$ in the free space, such that $F(x^j{}_{u_{j,k}}, x^i{}_{u_{i,l}})$ is the maximum for all the subunits of X^i in the free space;
EndFor
EndFor

The following are the evaluation functions F for protein, DNA and RNA. The matrix in equation (20) is the function for evaluating the scores between amino acids, in which $\mathcal{A} = \{c, s, t, p, a, g, n, d, e, q, h, r, k, m, i, l, v, f, y, w\}$. An asterisk in the matrix represents a negative integer and −1 is used in this paper for gap penalty. The scores, except those for φ, are from the SG (Structure-Genetic) Matrix (Feng *et al.*, 1985). According to Feng, this scoring system has taken into account the structure similarity of amino acids, as well as the likelihood of interchanges. The function for evaluating the scores between DNA or RNA nucleotides is given in equation (21), in which $\mathcal{A} = \{a, g, c, t\}$ for DNA and $\mathcal{A} = \{a, g, c, u\}$ for RNA. The scores, except the diagonal ones, are from Chan (1990).

Note that the evaluation functions are used in handling the mismatches between match blocks and in calculating the scores of the final alignment results, while the fitness function for the genetic algorithm is defined as the number of match tuples contained in a pre-alignment.

	ϕ	c	s	t	p	a	g	n	d	e	q	h	r	k	m	i	l	v	f	y	w
ϕ	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
c		6	4	2	2	2	3	2	1	0	1	2	2	0	2	2	2	2	3	3	3
s			6	5	4	5	5	5	3	3	3	3	3	3	1	2	2	2	3	3	2
t				6	4	5	2	4	2	3	3	2	3	4	3	3	2	3	1	2	1
p					6	5	3	2	2	3	3	3	3	2	2	2	3	3	2	2	2
a						6	5	3	4	4	3	2	2	3	2	2	2	5	2	2	2
g							6	3	4	4	2	1	3	2	1	2	2	4	1	2	3
n								6	5	3	3	4	2	4	1	2	1	2	1	3	0
d									6	5	4	3	2	3	0	1	1	3	1	2	0
e										6	4	2	2	4	1	1	1	4	0	1	1
q											6	4	3	4	2	1	2	2	1	2	1
h												6	4	3	1	1	3	1	2	3	1
r													6	5	2	2	2	2	1	1	2
k														6	2	2	2	3	0	1	1
m															6	4	5	4	2	2	3
i																6	5	5	4	3	2
l																	6	5	4	3	4
v																		6	4	3	3
f																			6	5	3
y																				6	3
w																					6

(20)

	ϕ	a	g	c	$t(u)$
ϕ	0	0	0	0	0
a		6	1	1	1
g			6	1	1
c				6	1
$t(u)$					6

(21)

Algorithm complexity

In this subsection, the algorithm complexity is analyzed in terms of computing time and memory space. The complexity is compared with pairwise dynamic programming which is the basis of many existing alignment approaches.

The time complexity of the genetic algorithm can be estimated using a model developed by Ankenbrandt (1991), which is based on the schema theorem for genetic algorithms (Goldberg, 1989a). As Ankenbrandt formulated and proved, the time required for a genetic algorithm to converge is:

$$O(Q \log Q) \tag{22}$$

where Q is the population size. That is, there exist constants c_0 and Q_0 such that the time required is $c_0 \cdot (Q \log Q)$ for all $Q > Q_0$. For a specific task, the time is affected by string lengths and the fitness ratio which is defined as:

$$r = \frac{\bar{f}}{\bar{f}'} \tag{23}$$

where $\bar{f}$ is the average fitness of the strings which have particular symbols at certain positions and $\bar{f}'$ is the average fitness value of all the other strings in the population. It will be seen in the section 'Results of aligning a protein sequence set' that equation (22) fits well with the experimental results.

Since $Q = m_a n/100$ (see the discussion in the previous section 'The procedure') we have:

$$Q \log Q = (m_a n/100) \log(m_a n/100)$$

When using a dynamic programming method to align two sequences of length m_1 and m_2, the computing time is $O(m_1 m_2)$. When the two sequences have similar lengths, we use m_a, the average length, to approximate them. We can thus estimate the time as $O(m_a^2)$. In aligning n sequences, pairwise dynamic programming involves at least $(n-1)$ times of two-sequence alignment. Assuming the n sequences are of similar lengths, the computing time can be estimated as:

$$O[m_a^2(n-1)] \tag{24}$$

Pairwise dynamic programming has greater time complexity than the genetic algorithm: For $n \geq 2$ and $n \ll m_a$, we have:

$$\frac{m_a^2(n-1)}{(m_a n/100)\log(m_a n/100)} = \frac{m_a \frac{(n-1)100}{n}}{\log(m_a n/100)} > \frac{m_a}{\log(m_a n/100)} > 1 \tag{25}$$

which indicates that the longer the sequences, the greater the ratio.

The major data structure of the genetic algorithm is a two-dimensional array which is used to store G (population) and M (matepool). The array is of size $l_{\max}Q$ where $l_{\max}$ is the maximum length of the strings and Q is the population size. When $Q = m_a n/100$, the total memory space required for the array is $l_{\max} m_a n/100$. Recall that the length of a string is the number of match blocks. In most cases, we have $l_{\max} \ll m_a$ and $n < 100$. The requirement for memory can be satisfied by most computer systems.

Experimental results

In the rest of the paper, we use 'the genetic algorithm' to refer to the whole algorithm for constructing pre-alignments and alignments. The algorithm has been implemented as a C program and applied to align several sets of molecular sequence data. The results are encouraging in terms of time efficiency and alignment quality. In the following, the results of aligning sequences in a protein data set, an mRNA data set and 11 other DNA or RNA data sets are presented and analyzed. Also, the results are compared with those obtained by using CLUSTALW, a most widely used program for multiple molecular alignment (Thompson *et al.*, 1994). The experimental results have shown that the genetic algorithm is dramatically more efficient.

Table 1. Time and quality measurements of the alignments of the protein data set by the two programs

No. of sequences	Average length	Methods	Processing time	No. of matches	ScoreG	ScoreC
2	335	G	0s312ms	208	1505	1184
		C	8s524ms	211	1687	1312
3	333	G	0s499ms	191	5048	4149
		C	14s257ms	189	5224	4320
4	336	G	0s764ms	187	10 279	8785
		C	21s860ms	187	10 654	9207
5	338	G	0s876ms	153	15 539	11 817
		C	29s881ms	155	16 936	14 176
6	341	G	1s120ms	148	23 425	16 980
		C	39s 12ms	151	25 081	21 625
7	341	G	1s344ms	139	32 625	23 148
		C	46s125ms	145	34 836	28 014
8	341	G	1s390ms	139	43 708	31 346
		C	53s663ms	143	46 457	37 414

Results of aligning a protein sequence set

The protein data set has eight sequences of bacterial porin (Jeanteur *et al.*, 1991). The lengths of the sequences range from 329 to 347 subunits. This data set is discussed because the alignment results can be presented in detail. Alignments of longer sequences (up to 12 000 subunits long each) will be discussed later.

The genetic algorithm was used to align two, three, ... and eight of the protein sequences. The processing time and quality measurements of the results are listed in Table 1. Two types of scores, ScoreG and ScoreC (to be discussed later), are used to evaluate alignment quality. CLUSTALW was applied to the same data for comparison. The time and quality measurements of its results are also listed. In the table, as well as in the following discussion, we use 'G' to indicate the genetic algorithm and 'C' to indicate CLUSTALW.

The 'processing time' listed in Table 1 is elapse time measured when the program, i.e. the genetic algorithm or CLUSTALW, was the only one executed on the computer. For the genetic algorithm, this includes the time required for the steps from finding match blocks to constructing the alignments and evaluating the scores (ScoreG). For CLUSTALW, this includes the time for pairwise alignment, multiple alignment and evaluation of quality.

The 'number of matches' is the number of tuples of:

$$(x^1{}_{u_{1,i}}, x^2{}_{u_{2,i}}, , x^n{}_{u_{n,i}}) \in X^1 \# X^2 \# \ldots \# X^n$$

such that

$$x^1{}_{u_{1,i}} = x^2{}_{u_{2,i}} = \ldots = x^m{}_{u_{m,i}}$$

'ScoreG' and 'ScoreC' in the table are goodness measurements of alignments. The two programs use different scoring systems for aligning sequences. For the purpose of comparison, the alignments by a program are also evaluated by using the other's scoring system (time for the 'mutual' evaluation is not included in the processing time).

'ScoreG' is calculated using the following formula:

$$\text{ScoreG} = \Sigma_{i=1}^{m} \Sigma_{j,k} F(x^j{}_{u_{j,i}}, x^k{}_{u_{k,i}}) \qquad (26)$$

where m is the length of the alignment, $1 \le j \le n$, $1 \le k \le n$, $j < k$, $n \ge 2$, and F is a function for evaluating the score between any two characters. The evaluation function F for protein, DNA and RNA is defined in the matrixes of equations (20) and (21), respectively.

'ScoreC' is calculated by using the evaluation method of CLUSTALW. The evaluation function for calculating scores of 'ScoreC' is equation (27) in the Appendix. Gap penalties of this scoring system are not included in the matrix. Details of the gap definition and the gap penalty can be found in the article by Thompson *et al.* (1994). When using CLUSTALW, we choose 10 as gap penalty.

Table 2 lists the ratios of the measurements of the genetic algorithm over those of CLUSTALW from Table 1. The following facts can be observed from the alignment results of the two programs.

The processing time required by the genetic algorithm is about two orders lower than the time required by CLUSTALW. As the number of sequences increases, the processing time required by both programs increases. However, the difference in processing time between the two programs also increases. This is indicated by the decrease in the ratios of the processing time.

Table 2. G/C ratios of the measurements in Table 1. The columns marked with an asterisk contain the ratios

No. of sequences	Processing time*	No. of matches*	ScoreG*	ScoreC*
2	3.7×10^{-2}	0.99	0.89	0.90
3	3.5×10^{-2}	1 01	0.97	0.96
4	3.5×10^{-2}	1.00	0.97	0.95
5	2.9×10^{-2}	0.99	0.92	0.83
6	2.9×10^{-2}	0.98	0.93	0.79
7	2.9×10^{-2}	0.96	0.94	0.83
8	2.6×10^{-2}	0.97	0.94	0.84
Average	3.1×10^{-2}	0.99	0.94	0.87

The number of matches identified by the genetic algorithm is no less than 96% of the number by CLUSTALW. On average, the number of matches identified by the genetic algorithm is 99% of those by CLUSTALW. Overall, the alignments by the two programs have very similar numbers of matches.

The scores of the results by using the genetic algorithm are slightly lower. On average, the ratio in ScoreG is 94% and in ScoreC is 87%. Since ScoreG and ScoreC are implemented in the two programs, respectively, there would be some advantages when the alignments generated by one program are evaluated by the original scoring system. Thus, the actual performance difference of the two programs should be evaluated as the average of the two ratio averages, which is ~90%.

Figures 2 and 3 illustrate the eight protein sequences aligned by the two programs. For easy comparison, the alignments are subdivided into successive segments, with each containing 60 subunits of the first sequence. An asterisk is used to indicate a match. It can be observed that both methods have correctly identified most of the matched subunits and discovered the overall relationships between the sequences. The alignment generated by the genetic algorithm is slightly less compact than that by CLUSTALW. The former has six more columns.

It can be noticed that in the alignment generated by the genetic algorithm, there is not any long match block. The longest match block is seven subunits long. In aligning this kind of sequence, the genetic algorithm has to deal with a large number of short match blocks (shorter than or equal to three subunits). This is a relatively time-consuming task for a genetic algorithm because short match blocks usually lead to a large number of long strings. Searching with long strings requires more time, even though the genetic algorithm is still much more efficient and achieves reasonably good results. As will be seen in the following two subsections, when sequences have long match blocks, the genetic algorithm is more advantageous than CLUSTALW in terms of efficiency.

When aligning all the eight sequences in the data set, 49 pre-alignment series were created and 24 of the series were used to form the first generation. The search process stopped at the 46th generation. Figure 4 illustrates the maximum numbers of matches in the generations, which may help understand the search towards the final result. The maximum number in a generation is the number of matches in the string which is the most fit.

Results of aligning an mRNA sequence set

The mRNA data set has 10 sequences, each of which has a length of ~12 000 subunits. In this subsection, the results of aligning the 10 mRNA sequences of different lengths by the two programs are presented and compared in terms of processing time and alignment quality. The ratios of the measurements are also presented. Then the time required by the genetic algorithm to align different numbers of sequences is presented.

Table 3 lists the processing time and the quality measurements of the alignments from the 10 sequences. Table 4 gives the ratios of the measurements. The alignments were obtained from subsequences of different lengths. For example, length '1000' indicates the first 1000 subunits of the 10 sequences. The CLUSTALW method is not able to align sequences longer than 9000 subunits.

For this data set, the genetic algorithm produces alignments of very similar quality as CLUSTALW in terms of the number of matches and the scores. The difference in time efficiency is even bigger. For instance in aligning the 10 sequences of length 9000, while CLUSTALW took >14 h to produce a result which has 61 (0.85%) more matches, the genetic algorithm spent only 1 min and 11 s to align the sequences. In analyzing this set of results, the following facts can be observed.

The average processing time taken by the genetic algorithm to align the 10 sequences of different lengths is approximately three orders lower than the time taken by CLUSTALW. As the length of the 10 sequences increases, the processing time required by the genetic algorithm increases, but the difference in processing time between the two programs also increases. This is also indicated by the decrease of the ratios in processing time: when the length increases from 1000 to 9000, the ratio in processing time decreases from 9.0×10^{-3} to 1.4×10^{-3}.

The averages of the ratios in number of matches is 1.00. The averages of the ratios in ScoreG and ScoreC are 1.00 and 0.97, respectively, and the average of these two is 0.99. Overall, the genetic algorithm is able to generate alignments of very similar quality in about 1/1000 of the time required by CLUSTALW.

Fig. 2. Alignment of eight protein sequences by the genetic algorithm. An asterisk is used to indicate a match.

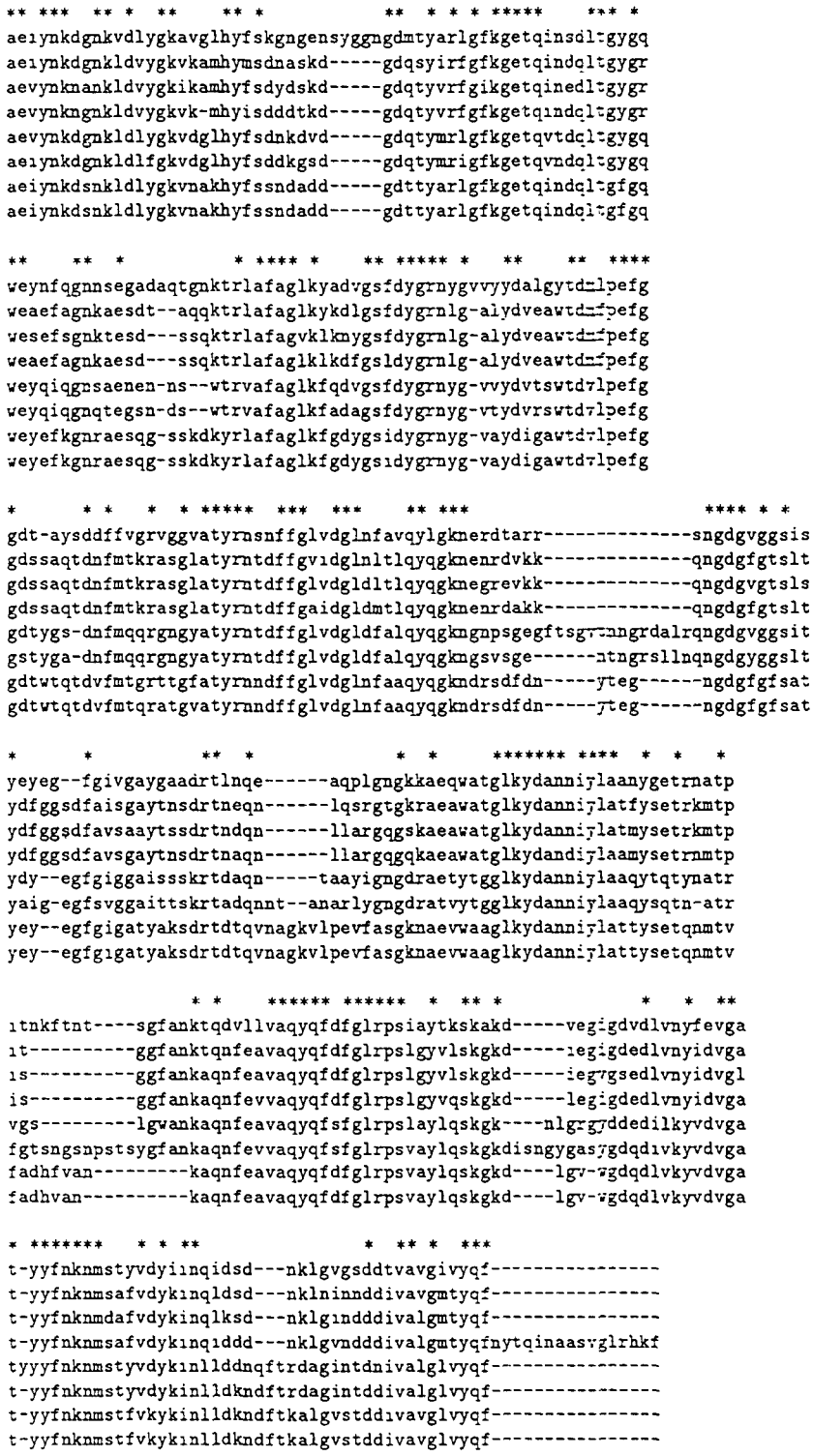

Fig. 3. Alignment of the same protein sequences by CLUSTALW.

Table 3. Time and quality measurements of the alignments of the mRNA data set by the two programs

Length	Methods	Processing time	No. of matches	ScoreG	ScoreC
1000	G	5s092ms	797	257 766	284 173
	C	9m24s352ms	797	257 813	288 591
2000	G	14s461ms	1504	501 230	545 930
	C	37m50s329ms	1508	501 539	556 687
3000	G	24s276ms	2319	758 391	807 544
	C	1hr25m46s520ms	2322	759 854	828 861
4000	G	31s552ms	3130	1 012 239	1 086 277
	C	2hr41m30s783ms	3143	1 015 712	1 116 731
5000	G	43s679ms	3922	1 286 592	1 372 633
	C	4hr17m53s542ms	3877	1 265 871	1 390 669
6000	G	48s344ms	4700	1 512 500	1 600 349
	C	6hr25m16s070ms	4732	1 522 469	1 659 179
7000	G	55s757ms	5544	1 776 488	1 896 154
	C	8hr41m23s699ms	5587	1 784 108	1 956 777
8000	G	1m04s230ms	6328	2 030 002	2 164 507
	C	11hr04m55s431ms	6383	2 040 507	2 237 351
9000	G	1m10s844ms	7205	2 285 279	2 421 534
	C	14hr03m13s902ms	7266	2 300 673	2 512 553
10 000	G	1m17s227ms	8038	2 547 223	2 715 173
	C	NA	NA	NA	NA
11 000	G	1m23s472ms	8906	2 806 860	2 990 517
	C	NA	NA	NA	NA
12 000	G	1m28s797ms	9627	3 028 626	3 217 619
	C	NA	NA	NA	NA

Table 4. G/C ratios of the measurements in Table 3. The columns marked with an asterisk contain the ratios

Length	Processing time*	No. of matches*	ScoreG*	ScoreC*
1000	9.0×10^{-3}	1.00	1.00	0.98
2000	6.4×10^{-3}	1.00	1.00	0.98
3000	4.7×10^{-3}	1.00	1.00	0.97
4000	3.3×10^{-3}	1.00	1.00	0 97
5000	2.8×10^{-3}	1.01	1 02	0.99
6000	2.1×10^{-3}	0 99	0 99	0.96
7000	1.8×10^{-3}	0.99	1.00	0.97
8000	1.6×10^{-3}	0.99	0.99	0.97
9000	1.4×10^{-3}	0.99	0.99	0.96
Average	3.7×10^{-3}	1.00	1.00	0.97

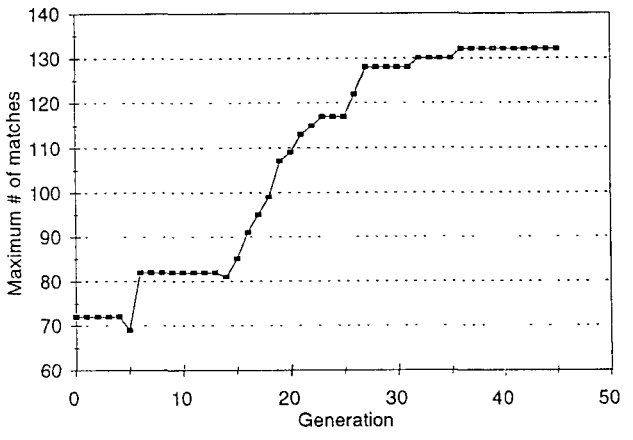

Fig. 4. Maximum numbers of matches in the generations.

The genetic algorithm has been used to align two, three,..., and 10 of the mRNA sequences. For a given number of sequences, the algorithm was applied to subsequences of different lengths. Figure 5 illustrates the relationship between computing time and sequence length. Each curve is for a given number of sequences. By comparing the curves with the complexity model of $Q \log Q$ or $mn \log(mn)$ of the section 'Algorithm complexity', it can be observed that the model fits well with the experimental results.

Results of aligning other sequence sets

In addition to the protein and the mRNA data sets, the genetic algorithm has been applied to many other data sets and satisfactory results have been obtained.

In this section, the results of using the genetic algorithm to align 11 other data sets are presented and compared with those by CLUSTALW. The sequence data sets are from the ftp site of the European Bioinformatics Institute. IDs of the sequences are listed in the Appendix. Of the 11 sequence sets, four are DNA and seven are RNA. The minimum average length is 212 and the maximum is 4582. Table 5 lists the processing time and the quality measurements. Table 6 gives the ratios of the measurements. With the 11 data sets, the genetic algorithm also produced alignments of very similar quality to CLUSTALW. Its processing time is approximately two orders lower. The averages of the ratios in the number of matches is 0.98. The average of the ratios in ScoreG and ScoreC are 0.98 and 1.00, respectively, and the average of the two is 0.99.

An important character of the genetic algorithm can again be observed by examining this group of experimental results. The processing time of the genetic algorithm is not simply proportional to the number and the length of the sequences aligned. Instead, it is heavily dependent on the sizes of match blocks and the number of match blocks in a given length of sequences. Higher efficiency is closely related to the existence of longer match blocks and thus a smaller number of match blocks. The processing time increases proportionally with the increase in the sequence lengths and the number of sequences only in situations where the sizes and the number of match blocks in a given length of the sequences are similar (as in the cases of the protein and the mRNA data sets).

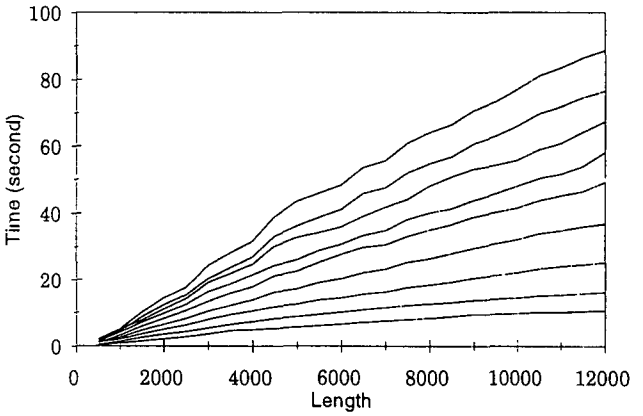

Fig. 5. Processing time versus sequence length (for the genetic algorithm method). The curves from the bottom to top are for two, three,..., and 10 sequence alignments, respectively.

Table 5. Time and quality measurements of the alignments of other DNA and RNA data sets by the two programs

Data set	No. of sequences	Average length	Methods	Processing time	No. of matches	ScoreG	ScoreC
S1	10	211	G	0s215ms	198	56 491	71 847
			C	27s113ms	197	56 491	69 010
S2	5	1780	G	6s194ms	1553	101 507	120 763
			C	13m8s776ms	1611	103 018	115 785
S3	4	2433	G	0s983ms	2303	85 615	104 867
			C	18m28s150ms	2305	85 701	104 149
S4	8	1437	G	17s943ms	804	196 746	191 880
			C	15m38s568ms	901	210 317	226 516
S5	8	1680	G	10s685ms	1420	269 432	291 406
			C	21m5s898ms	1439	269 092	303 390
S6	4	4582	G	0s784ms	4332	159 532	180 620
			C	1hr4m58s781ms	4409	160 860	182 475
S7	5	1093	G	4s394ms	894	60 153	75 516
			C	5m12s948ms	905	60 719	73 280
S8	8	1071	G	3s782ms	999	176 665	227 985
			C	8m35s650ms	999	176 665	216 359
S9	6	1456	G	8s553ms	1046	116 221	139 681
			C	11m21s416ms	1118	120 534	142 237
S10	7	1448	G	6s265ms	1179	170 552	211 327
			C	13m22s264ms	1193	172 756	206 195
S11	5	1092	G	4s281ms	840	57 682	68 951
			C	5m3s838ms	855	60 100	71 361

Table 6. G/C ratios of the measurements in Table 5. The columns marked with an asterisk contain the ratios

Data set	No. of sequences	Average length	Processing time*	No of matches*	ScoreG*	ScoreC*
S1	10	211	7.92×10^{-3}	1.00	1.00	1.04
S2	5	1780	7.85×10^{-3}	0.96	0.99	1.04
S3	4	2433	8.87×10^{-4}	1.00	1.00	1.01
S4	8	1437	1.91×10^{-2}	0.89	0.94	0.85
S5	8	1680	8.44×10^{-3}	0.99	1.00	0.96
S6	4	4582	2.01×10^{-4}	0.98	0.99	0.99
S7	5	1093	1.40×10^{-2}	0.99	0.99	1.03
S8	8	1071	7.33×10^{-3}	1.00	1.00	1.05
S9	6	1456	1.26×10^{-2}	0.94	0.96	0.98
S10	7	1448	8.03×10^{-3}	0.99	0.99	1.02
S11	5	1092	1.41×10^{-2}	0.98	0.96	0.97
Average			9.13×10^{-3}	0.98	0.98	1.00

Compare the results of S4 and S5. Both data sets have eight sequences each, and S4 has an average length of 1437, while S5 has 1680. The processing time for the genetic algorithm to align S4 is ~18 s and the time for S5 is ~11 s. Different from CLUSTALW, the genetic algorithm spent a longer time on the shorter sequence data set. Compare the results of S9 and S10. Both data sets have sequences of approximately the same length. S9 has six sequences and S10 has seven. The processing time for the genetic algorithm to align S9 is ~9 s and the time for S10 is ~6 s. Again, different from CLUSTALW, the genetic algorithm spent a longer time on the data set with the smaller number of sequences. A similar phenomenon is also observed in comparing the results of S7 and S8.

Examine the data set pairs again. The ratios of the number of matches over the sequence length may shed some light on the relationship between the string length and efficiency. For instance, S4 has a ratio of 56.0% and S5 has a ratio of 84.5%. Thus, it is more likely that there exist more longer match blocks in the alignment of set S5 than S4, and therefore the genetic algorithm could spend less time aligning S5. Similar situations can also be found in the alignments of S7 and S8, as well as of S9 and S10.

Discussion

The experimental results indicate that the major strength of the genetic algorithm is its high efficiency in computing time. This strength is especially significant when aligning the long sequences. For the same sequences, while a conventional method, for example, one based on pairwise dynamic programming, may require hours to align, the genetic algorithm may produce results of similar quality within a few minutes (even seconds).

At the present stage, we do not expect the algorithm to be superior to the existing multiple molecular sequence alignment approaches in every aspect. However, the initial experiments have shown that merely with a very simple fitness function, a genetic algorithm could yield alignments of good quality. Being able to identify match columns in alignments at high speed, the genetic algorithm approach has good potential which enables us to incorporate other techniques or other criteria for processing the intervals between match columns to achieve higher quality efficiently.

The additional advantages of the genetic algorithm include simultaneousness, automated processing, easy implementation and a simple data structure. The algorithm does not need any extra interactions, e.g. grouping sequences, selecting subsequences, or shuffling alignments. No pre-processing, such as creating a tree or calculating the distances, is required. Using the algorithm does not need any specific knowledge about species origins, sequence structures, and so on. The implementation involves coding the three simple operations only, and requires a one-dimensional array of strings.

The alignments generated by the genetic algorithm have correctly identified the majority of the one-to-one correspondence between the sequences aligned, although it is not guaranteed that the alignments are optimal. Such alignments are acceptable for many applications. With the strength of high efficiency, genetic algorithms are very promising to become useful tools for multiple sequence alignment in applications which do not strictly require the optimal alignments.

Appendix

An evaluation matrix of CLUSTALW

	a	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*k*	*l*	*m*	*n*	*p*	*q*	*r*	*s*	*t*	*v*	*w*	*y*
a	5																			
c	−1	12																		
d	−2	−3	7																	
e	−1	−3	2	6																
f	−2	−2	−4	−3	8															
g	0	−3	−1	−2	−3	7														
h	−2	−3	0	0	−2	−2	10													
i	−1	−3	−4	−3	0	−4	−3	5												
k	−1	−3	0	1	−3	−2	−1	−3	5											
l	−1	−2	−3	−2	1	−3	−2	2	−3	5										
m	−1	−2	−3	−2	0	−2	0	2	−1	2	6									
n	−1	−2	2	0	−2	0	1	−2	0	−3	−2	6								
p	−1	−4	−1	0	−3	−2	−2	−2	−1	−3	−2	−2	9							
q	−1	−3	0	2	−4	−2	1	−2	1	−2	0	0	−1	6						
r	−2	−3	−1	0	−2	−2	0	−3	3	−2	−1	0	−2	1	7					
s	1	−1	0	0	−2	0	−1	−2	−1	−3	−2	1	−1	0	−1	4				
t	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	0	−1	−1	−1	2	5			
v	0	−1	−3	−3	0	−3	−3	3	−2	1	1	−3	−3	−3	−2	−1	0	5		
w	−2	−5	−4	−3	1	−2	−3	−2	−2	−2	−2	−4	−3	−2	−2	−4	−3	−3	15	
y	−2	−3	−2	−2	3	−3	2	0	−1	0	0	−2	−3	−1	−1	−2	−1	−1	3	8

(27)

Sequence IDs of the 11 data sets

S1: (DNA)					
HCV2L1A10	HCV2L3A5	HCV2L3A7	HCV2L3A9	HCV2L3B1	HCV2L3B2
HCV2L3C1	HCV2L3C8	HCV2L3D4	HCV2L3E6		
S2: (DNA)					
HS06674	HS06675	HS06676	HS06677	HS06679	
S3: (RNA)					
HS04816	HS04817	HS04818	HS04824		
S4: (DNA)					
HI1U16764	HI1U16766	HI1U16768	HI1U16770	HI1U16772	HI1U16774
HI1U16776	HI1U16778				
S5: (DNA)					
HI1U16765	HI1U16767	HI1U16769	HI1U16771	HI1U16773	HI1U16775
HI1U16777	HI1U16779				
S6: (RNA)					
BTPHOSA	BTPHOSB	BTPHOSC	BTPHOSD		
S7: (RNA)					
PH35624	PH35625	PH35626	PH35627	PH35628	
S8: (RNA)					
PT10537	PT10538	PT10539	PT10540	PT10541	PT10542
PT10543	PT10544				
S9: (RNA)					
PP59651	PP59652	PP59653	PP59654	PP59655	PP59656
S10: (RNA)					
FC07667	FC07668	FC07669	FC07670	FC07672	FC07673
FC07674					
S11: (RNA)					
MNMHDRBA	MNMHDRBB	MNMHDRBC	MNMHDRBD	MNMHDRBE	

References

Ankenbrandt,C.A. (1991) An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. In Rawlins,G.J.E. (ed.), *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, pp. 53–68.

Bethke, A.D. (1981) Genetic algorithms as function optimizer. Ph.D Thesis, University of Michigan.

Buckles, B.P., Petry, F.E. and Kuester, R.L. (1990) Schema survival rates and heuristic search in genetic algorithms. In *Proceedings of Tools for Artificial Intelligence*. IEE Computer Society Press, Los Alamitos, CA, pp. 322–327.

Chan,S.C. (1990) A hierarchical sequence synthesis procedure. PhD Thesis, University of Waterloo, Waterloo, Ontario, Canada.

Chan,S.C., Wong,A.K.C. and Chiu,D.K.Y. (1992) A survey of multiple sequence comparison methods. *Bull. Math. Biol* , **54**, 563–598.

Dandekar,T. and Argos,P. (1992) Potential of genetic algorithms in protein folding and protein engineering simulations. *Protein Eng.*, **5**, 637–545.

De Jong,K. (1988) Learning with genetic algorithms: an overview. *Machine Learning 3*. Kluwer Academic, Hingham, MA, pp. 121–138.

Feng,D.F. and Doolittle,R.F. (1987) Progressive sequence alignment as prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, **25**, 351–360.

Feng,D.F., Johnson,M.S. and Doolittle,R.F. (1985) Aligning amino acid sequences: comparison of commonly used methods. *J. Mol. Evol.*, **21**, 112–125.

Goldberg,D.E. (1987) Simple genetic algorithms and the minimal, deceptive problem. In Davis,L. (ed.), *Genetic Algorithms and Simulated Annealing*. Pitman, London, pp. 74–88.

Goldberg,D.E. (1989a) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York.

Goldberg,D.E. (1989b) Sizing populations for serial and parallel algorithms. In *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York.

Gotoh,O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705–708.

Gotoh,O. (1990) Consistency of optimal sequence alignments. *Bull. Math. Biol.*, **52**, 509–525.

Grefenstette,J.J. (1986) Optimization of control parameters for genetic algorithms. *IEEE Trans. System Man Cybernet.*, **16**, 122–128.

Grefenstette,J.J. and Fitzpatrick,J.M. (1985) Genetic search with approximate function evaluations. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 112–120.

Gultyaev,A.P., Batenbury,F.H.D.V. and Pleij,C.W.A. (1995) The influence of a metastable structure in plasmid primer RNA on antisense RNA binding kinetics. *Nucleic Acids Res.*, **23**, 3718–3725.

Gusfield,D. (1993) Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.*, **55**, 141–154.

Hirschberg,D.S. (1975) A linear space algorithm for computing longest common subsequences. *Commun. Assoc. Comput. Mach* , **18**, 341–343.

Holland,J.H. (1975) *Adoption in Natural and Artificial Systems*. University of Michgan Press, Ann Arbor, MI.

Holland,J.H. (1986) Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski *et al.* (eds), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, San Mateo, CA, Vol. 2, pp. 593–623.

Holland,J.H. (1987) Genetic algorithms and classifier systems: foundations and future directions. In *Genetic Algorithms and their Applications: Proceeding of the Second International Conference on Genetic Algorithms*, pp. 82–89.

Jeanteur,D., Lakey,J.H. and Pattus,F. (1991) The bacterial porin superfamily: sequence alignment and structure prediction. *Mol. Microbiol.*, **5**, 2153–2164.

Michalewicz,Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Miller,W. (1993) Building multiple alignment from pairwise alignments. *Comput. Applic. Biosci.*, **9**, 169–176.

Myers,E.W. and Miller,W. (1988) Optimal alignments in linear space. *Comput. Applic. Biosci.*, **4**, 11–17.

Roytberg,M. (1992) A search for common patterns in many sequences. *Comput. Applic. Biosci.*, **8**, 57–64.

Sankoff,D. (1972) Matching sequence under deletion-insertion constraints. *Proc. Natl Acad. Sci. USA*, **64**, 4–6.

Sankoff,D., Cedergren,R.J. and McKay,W. (1982) A strategy for sequence phylogeny research. *Nucleic Acids Res.*, **10**, 421–431.

Spears,W.M. and De Jong,K.A. (1990) Using genetic algorithms for supervised concept learning. In *Proceedings of Tools for Artificial Intelligence*. IEEE Computer Society Press, Los Alamitos, CA, pp. 335–341.

Thompson,J.D., Higgins,D.G. and Gibson,T.J. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Unger,R. and Moult,J. (1993) Genetic algorithms for protein folding simulations. *J. Mol. Biol.*, **231**, 75–81.

van Batenbury,F.H.D., Gultyaev,A.P. and Pleij,C.W.A. (1995) An APL-programmed genetic algorithm for the prediction of RNA secondary structure. *J. Theor. Biol.*, **174**, 269–280.

Vihinen,M., Euranto,A., Luostarinen,P. and Nevalainen,O. (1992) MULTICOMP: A program package for multiple sequence comparison. *Comput. Applic. Biosci.*, **8**, 35–38.

Vingron,M. and Argos,P. (1991) Motif recognition and alignment for many sequences by comparison of dot-matrices. *J. Mol. Biol.*, **218**, 33–43.

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

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