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 $O(m_1 m_2)$ time and $O(m_1 m_2)$ 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; Gotro, 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-
Genetic algorithms have been used as effective tools for pro-

A search process starts with a population of strings. In a

C.Zhang and A.K.C.Wong
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:

$$n_H(t + 1) = n_H(t) \cdot \left\{ f(H) \cdot \left( 1 - p^i \cdot \delta(H) \cdot \frac{1}{t-1} \right) \right\}$$

where $n_H(t)$ is the number of instances of $H$ in generation $t$, $f(H)$ 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 have usually very long and numerous possible alignments may be inefficient in dealing with such a problem 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 an alignment of the $n$ sequences, denoted as $X^1\#X^2\#\ldots\#X^n$, is:

$$X^1 = x_{11}^1 x_{12}^1 \ldots x_{1m_1}^1$$
$$X^2 = x_{21}^2 x_{22}^2 \ldots x_{2m_2}^2$$
$$\ldots$$
$$X^n = x_{n1}^n x_{n2}^n \ldots x_{nm_n}^n$$

where $x_{ik}^l \in A$, $\forall k$ indicates that the subunit belongs to the $k$th sequence, subscript $i$ indicates that the $i$th subunit in the sequence, $1 \leq i \leq m_k$, $m_k$ is the length of the $k$th sequence, and $1 \leq k \leq n$. $m_n$ is used to denote the average length.

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

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

$$X^1\#X^2\#\ldots\#X^n = x_{11}^l \ 1 \ x_{12}^l \ 1 \ \ldots \ x_{1m_1}^l \ 1$$
$$\ 1 \ 1 \ 1 \ \ldots \ 1$$
$$\ldots$$
$$\ 1 \ 1 \ 1 \ \ldots \ 1$$
$$x_{n1}^l \ 1 \ x_{n2}^l \ 1 \ \ldots \ x_{nm_n}^l$$

where $x_{ik}^l \in A'$, $\vert$ indicates a match or a mismatch, and $m \geq \max(m_1, m_2, \ldots, m_n)$.

An alignment has $m$ columns and $n$ rows. It satisfies the following conditions. (i) The subunits included in the $i$th row must be subunits in the $i$th sequence ($1 \leq i \leq n$). (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 $\phi$.

An alignment may also be denoted as a set: $X^1 # X^2 # ... # X^n = (x_{11}^1, x_{12}^1, ..., x_{m_1}^1, x_{21}^2, x_{22}^2, ..., x_{m_2}^2, ..., x_{n1}^n, x_{n2}^n, ..., x_{nm_n}^n)$, where $(x_{ij}^i, x_{ij}^j, ..., x_{ij}^n)$ may include matches and mismatches.

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

$$X^1 # X^2 # ... # X^n = (x_{11}^1, x_{12}^1, ..., x_{1m_1}^1, x_{21}^2, x_{22}^2, ..., x_{2m_2}^2, ..., x_{n1}^n, x_{n2}^n, ..., x_{nm_n}^n)$$

where $x_{ij}^k \in \mathcal{A}$, $\parallel$ 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 $i$th row must be subunits in the $i$th sequence $(1 \leq i \leq n)$. (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 $\phi$. The latter represents possible mismatches. By comparing the two definitions, we note that a pre-alignment allows columns of entire $\phi$ 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. $C_1 \ A_1 \ C_1 \ C_1 \ G_1 \ T_1 \ C_1$
2. $G_1 \ A_2 \ A_2 \ T_2 \ C_2$
3. $G_1 \ A_2 \ C_1 \ T_1 \ G_2 \ C_1$

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

- $G_1 \ A_1 \ \phi \ \phi \ \phi \ C_1$
- $G_1 \ A_2 \ \phi \ \phi \ \phi \ C_1$
- $G_1 \ A_1 \ \phi \ \phi \ \phi \ C_1$
- $G_1 \ A_2 \ \phi \ \phi \ \phi \ C_1$
- $G_1 \ A_1 \ \phi \ \phi \ \phi \ C_1$
- $G_1 \ A_2 \ \phi \ \phi \ \phi \ C_1$

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

- $G_1 \ A_1 \ \phi \ \phi \ T_1 \ C_1$
- $G_1 \ A_1 \ \phi \ \phi \ T_2 \ C_1$
- $G_1 \ A_1 \ \phi \ \phi \ T_3 \ C_1$

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

- $G_1 \ A_1 \ C_1 \ G_1 \ T_1 \ \phi \ \phi \ C_1$
- $G_1 \ A_1 \ C_1 \ \phi \ T_1 \ \phi \ \phi \ C_1$
- $G_1 \ A_1 \ C_1 \ \phi \ T_1 \ \phi \ C_1$

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 $\Omega$, 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 $\Omega$. The pre-alignment in equation (8) corresponds to point

$$\omega = \begin{pmatrix} u_{11} & u_{12} & ... & u_{1m_1} \\ u_{21} & u_{22} & ... & u_{2m_2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & ... & u_{nm_n} \end{pmatrix}$$

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

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 5 & 6 \\ 1 & 1 & 0 & 0 & 4 & 5 \\ 1 & 1 & 0 & 0 & 4 & 6 \end{pmatrix}$$

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 $\Omega$. 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 subunits with each from an input sequence. For example, the match tuple for


given cutting positions if the two new strings satisfy condition (ii) of the example. 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).

Formally, a match block of length $r$ is

\[
\begin{align*}
U_{1j} & U_{1j+1} \ldots U_{1j+r-2} U_{1j+r-1} \\
U_{2j} & U_{2j+1} \ldots U_{2j+r-2} U_{2j+r-1} \\
\vdots & \vdots \ldots \vdots \vdots \\
U_{n_j} & U_{n_j+1} \ldots U_{n_j+r-2} U_{n_j+r-1}
\end{align*}
\]

such that

\[
\begin{align*}
u_{k_j} + 1 &= u_{k_j + 1}, u_{k_j + 1} + 1 = u_{k_j + 2}, \ldots, u_{k_j + r - 2} + 1 = u_{k_j + r - 1} \\
&\quad \text{and } \left(u_{k_j + 1}, u_{k_j + 2} - 1, \ldots, u_{k_j + r - 1} + 1, u_{k_j + r} - 1 + 1, \ldots, u_{k_j + r} + 1\right)^T \text{ are not match tuples, for } k = 1, 2, \ldots, n.
\end{align*}
\]

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 $\omega$ is a string of match blocks, i.e.:

\[A = a_1 a_2 \ldots a_l\]

where $\omega_i$ is the $i$th match block in $\omega$ and $1 \leq i \leq m_1$.

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
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 \( m \) and \( n \), and the number is usually <1/100 of the product. For an alignment task, we choose the population size \( Q \) as:

\[
Q = mn/100
\]  

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 \( B_2 \):

\[
B_1 = \begin{pmatrix}
    u_{1j} & u_{1j+1} & \ldots & u_{1n} \\
    u_{2j} & u_{2j+1} & \ldots & u_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    u_{nj} & u_{nj+1} & \ldots & u_{nn}
\end{pmatrix}
\]

\[
B_2 = \begin{pmatrix}
    u_{1j} & u_{1j+1} & \ldots & u_{1n} \\
    u_{2j} & u_{2j+1} & \ldots & u_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    u_{nj} & u_{nj+1} & \ldots & u_{nn}
\end{pmatrix}
\]

where \( u_{ij} \leq u_{ij}, u_{ij} < u_{ij}, \) and \( u_{ij} \leq u_{ij} (i = 1, \ldots, n) \).

The average left distance of \( B_2 \) is:

\[
\text{ALD}_{B_2} = \frac{1}{n} \sum_{i=1}^{n} (u_{ij} - u_{ij})
\]  

(15)

The average right distance of \( B_1 \) is:

\[
\text{ARD}_{B_1} = \text{ALD}_{B_2}
\]  

(16)

The maximum left distance of \( B_2 \) is:

\[
\text{MLD}_{B_2} = \max_{i}(u_{ij} - u_{ij})
\]  

(17)

The maximum right distance of \( B_1 \) is:

\[
\text{MRD}_{B_1} = \text{MLD}_{B_2}
\]  

(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:

\[
D = \frac{((\text{MLD} - \text{ALD}) + (\text{MRD} - \text{ARD}))}{L}
\]  

(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 \( (v_{1j}, v_{2j}, \ldots, v_{nj}) \) is a match tuple

\[
\text{Form a new match block which includes } (v_{1j}, v_{2j}, \ldots, v_{nj});
\]

\[
\text{Add } (v_{1j} + j, v_{2j} + j, \ldots, v_{nj} + j) \text{ to the match block;}
\]

EndIf

EndWhile

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 \( X^1, X^2, \ldots, X^n \) which have lengths \( m_1, m_2, \ldots, m_n \), respectively. The output is pre-alignment \( X^1 \# X^2 \# \ldots \# X^n \).

\[
\text{procedure PRE-ALIGN (Input: } X^1, X^2, \ldots, X^n; \text{ Output: } X^1 \# X^2 \# \ldots \# X^n \text{ )}
\]

begin
1. Find match blocks from \( X^1, X^2, \ldots, X^n \).
2. Create genetic strings from the match blocks (\( t \) – number of match blocks, \( j \) – number of strings):
   \[
   j = 0
   \]

end
For \( i = 1 \) through \( I \) do
  For \( j = 1 \) through \( J \) do
    If (\( A_1 \) satisfies conditions (ii) and (iii) in (8) after inserting \( B_j \) into it)
      Insert \( B_j \) into \( A_1 \);
      Quit the inner loop;
    EndIf
  EndFor
If (\( B_1 \) is not inserted)
  \( j = J + 1; \)
  Create \( A_J \) to contain \( B_j \);
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 \( G^0 \) 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 \( A_i' (1 \leq i \leq Q) \) 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 \( A_j \);
     Perform crossover on \( A_i \) and \( A_j \) to create two new strings;
     Replace \( A_i \) and \( A_j \) with the new strings;
   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 \( m_t/20 \) in the best string
   Call PRE-ALIGN to align the substrings.
10. Produce \( X^1 \# X^2 \# \ldots \# X^n \) 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 \( \varphi \), 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 \( X^j \) which has the maximum number of such subunits (\( 1 \leq j \leq n \)).
3. Fill row \( i \) of the gap of the pre-alignment with the subunits of \( X^j \).
4. For \( j = 1 \) through \( n \) and \( j \neq i \)
   Place the missing subunits \( x_1, x_j, x_{j+1}, \ldots, x_n \) at the lower end of the gap in row \( j \);
   For \( k = s \) down to 1.
   Move \( x_{j-k} \) to the column of \( x_{i,j} \) in the free space, such that \( F(x_{i,j}, x_{i,j}) \) is the maximum for all the subunits of \( X^j \) 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 \( \varphi \), 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 \( \mathcal{A} = \{a, g, c, r\} \) 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.
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:

$$O(Q \log Q)$$

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:

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

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

$$Q \log Q = (m_n/100) \log(m_n/100)$$

When using a dynamic programming method to align two sequences of length $m_1$ and $m_2$, the computing time is $O(m_1m_2)$. When the two sequences have similar lengths, we use $m_n$, the average length, to approximate them. We can thus estimate the time as $O(m_n^2)$. 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:

$$O(m_n^2(n-1))$$

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

$$\frac{m_n^2(n-1)}{(m_n/100) \log(m_n/100)} > \frac{m_n}{\log(m_n/100)} > 1$$

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 $Q = m_n/100$, the total memory space required for the array is $l_{max}m_n/100$. Recall that the length of a string is the number of match blocks. In most cases, we have $l_{max} \ll m_n$ 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

<table>
<thead>
<tr>
<th>No. of sequences</th>
<th>Average length</th>
<th>Methods</th>
<th>Processing time</th>
<th>No. of matches</th>
<th>ScoreG</th>
<th>ScoreC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>335</td>
<td>G</td>
<td>0s312ms</td>
<td>208</td>
<td>1505</td>
<td>1184</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>8s524ms</td>
<td>211</td>
<td>1687</td>
<td>1312</td>
</tr>
<tr>
<td>3</td>
<td>333</td>
<td>G</td>
<td>0s499ms</td>
<td>191</td>
<td>5048</td>
<td>4149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>14s257ms</td>
<td>189</td>
<td>5224</td>
<td>4320</td>
</tr>
<tr>
<td>4</td>
<td>336</td>
<td>G</td>
<td>0s764ms</td>
<td>187</td>
<td>10279</td>
<td>8785</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>21s860ms</td>
<td>187</td>
<td>10654</td>
<td>9207</td>
</tr>
<tr>
<td>5</td>
<td>338</td>
<td>G</td>
<td>0s876ms</td>
<td>153</td>
<td>15539</td>
<td>11817</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>29s818ms</td>
<td>155</td>
<td>16936</td>
<td>14176</td>
</tr>
<tr>
<td>6</td>
<td>341</td>
<td>G</td>
<td>1s120ms</td>
<td>148</td>
<td>23425</td>
<td>16980</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>39s12ms</td>
<td>151</td>
<td>25081</td>
<td>21625</td>
</tr>
<tr>
<td>7</td>
<td>341</td>
<td>G</td>
<td>1s344ms</td>
<td>139</td>
<td>32625</td>
<td>23148</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>46s125ms</td>
<td>145</td>
<td>34836</td>
<td>28014</td>
</tr>
<tr>
<td>8</td>
<td>341</td>
<td>G</td>
<td>1s390ms</td>
<td>139</td>
<td>43708</td>
<td>31346</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>53s663ms</td>
<td>143</td>
<td>46457</td>
<td>37414</td>
</tr>
</tbody>
</table>

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:

\[ (x_{1_{a_1}}, x_{2_{a_2}}, ..., x_{n_{a_n}}) \in X^1 \# X^2 \# \ldots \# X^n \]

such that

\[ x_{1_{a_1}} = x_{2_{a_2}} = \ldots = x_{n_{a_n}} \]

'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:

\[ \text{ScoreG} = \sum_{i=1}^{m} \sum_{j=k}^{n} F(x_{i_{a_i}}, \cdot, x_{j_{a_j}}) \]  \hspace{1cm} (26) \]

where \( m \) is the length of the alignment, \( 1 \leq j \leq n \), \( 1 \leq k \leq n \), \( j < k \), \( n \geq 2 \), 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 matrices 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.

<table>
<thead>
<tr>
<th>No. of sequences</th>
<th>Processing time*</th>
<th>No. of matches*</th>
<th>ScoreG*</th>
<th>ScoreC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.7 x 10^2</td>
<td>0.99</td>
<td>0.89</td>
<td>0.90</td>
</tr>
<tr>
<td>3</td>
<td>3.5 x 10^2</td>
<td>1.01</td>
<td>0.97</td>
<td>0.96</td>
</tr>
<tr>
<td>4</td>
<td>3.5 x 10^2</td>
<td>1.00</td>
<td>0.97</td>
<td>0.95</td>
</tr>
<tr>
<td>5</td>
<td>2.9 x 10^2</td>
<td>0.99</td>
<td>0.92</td>
<td>0.83</td>
</tr>
<tr>
<td>6</td>
<td>2.9 x 10^2</td>
<td>0.98</td>
<td>0.93</td>
<td>0.79</td>
</tr>
<tr>
<td>7</td>
<td>2.9 x 10^2</td>
<td>0.96</td>
<td>0.94</td>
<td>0.83</td>
</tr>
<tr>
<td>8</td>
<td>2.6 x 10^2</td>
<td>0.97</td>
<td>0.94</td>
<td>0.84</td>
</tr>
<tr>
<td>Average</td>
<td>3.1 x 10^2</td>
<td>0.99</td>
<td>0.94</td>
<td>0.87</td>
</tr>
</tbody>
</table>

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 1 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 x 10^-3 to 1.4 x 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.

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 1 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 x 10^-3 to 1.4 x 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.
Genetic algorithm for multiple sequence alignment

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

<table>
<thead>
<tr>
<th>Length</th>
<th>Methods</th>
<th>Processing time</th>
<th>No. of matches</th>
<th>ScoreG</th>
<th>ScoreC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>G</td>
<td>5s092ms</td>
<td>797</td>
<td>257 766</td>
<td>284 173</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>9m24s352ms</td>
<td>797</td>
<td>257 813</td>
<td>288 591</td>
</tr>
<tr>
<td>2000</td>
<td>G</td>
<td>14s461ms</td>
<td>1504</td>
<td>501 230</td>
<td>545 930</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>37m50s329ms</td>
<td>1508</td>
<td>501 539</td>
<td>556 687</td>
</tr>
<tr>
<td>3000</td>
<td>G</td>
<td>24s276ms</td>
<td>2319</td>
<td>758 391</td>
<td>807 544</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1hr25m46s520ms</td>
<td>2322</td>
<td>759 854</td>
<td>828 861</td>
</tr>
<tr>
<td>4000</td>
<td>G</td>
<td>31s552ms</td>
<td>3130</td>
<td>1 012 239</td>
<td>1 086 277</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>2hr41m30s783ms</td>
<td>3143</td>
<td>1 015 712</td>
<td>1 116 731</td>
</tr>
<tr>
<td>5000</td>
<td>G</td>
<td>43s679ms</td>
<td>3922</td>
<td>1 286 592</td>
<td>1 372 633</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>4hr17m53s542ms</td>
<td>3877</td>
<td>1 265 871</td>
<td>1 390 669</td>
</tr>
<tr>
<td>6000</td>
<td>G</td>
<td>48s344ms</td>
<td>4700</td>
<td>1 512 500</td>
<td>1 600 349</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>6hr25m16s070ms</td>
<td>4732</td>
<td>1 522 469</td>
<td>1 659 179</td>
</tr>
<tr>
<td>7000</td>
<td>G</td>
<td>55s757ms</td>
<td>5544</td>
<td>1 776 488</td>
<td>1 896 154</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>8hr41m23s699ms</td>
<td>5587</td>
<td>1 784 108</td>
<td>1 956 777</td>
</tr>
<tr>
<td>8000</td>
<td>G</td>
<td>1hr04s230ms</td>
<td>6328</td>
<td>2 030 002</td>
<td>2 164 507</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1hr04m55s431ms</td>
<td>6383</td>
<td>2 040 507</td>
<td>2 237 351</td>
</tr>
<tr>
<td>9000</td>
<td>G</td>
<td>1hr10s844ms</td>
<td>7205</td>
<td>2 285 279</td>
<td>2 421 534</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1hr03m13s902ms</td>
<td>7266</td>
<td>2 300 673</td>
<td>2 512 553</td>
</tr>
<tr>
<td>10 000</td>
<td>G</td>
<td>1hr17s227ms</td>
<td>8038</td>
<td>2 547 223</td>
<td>2 715 173</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>11 000</td>
<td>G</td>
<td>1hr23s472ms</td>
<td>8906</td>
<td>2 806 860</td>
<td>2 990 517</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>12 000</td>
<td>G</td>
<td>1hr28s797ms</td>
<td>9627</td>
<td>3 028 626</td>
<td>3 217 619</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Length</th>
<th>Processing time*</th>
<th>No. of matches*</th>
<th>ScoreG*</th>
<th>ScoreC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>9.0 x 10^-3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>2000</td>
<td>6.4 x 10^-3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>3000</td>
<td>4.7 x 10^-3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>4000</td>
<td>3.3 x 10^-3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>5000</td>
<td>2.8 x 10^-3</td>
<td>1.01</td>
<td>1.02</td>
<td>0.99</td>
</tr>
<tr>
<td>6000</td>
<td>2.1 x 10^-3</td>
<td>0.99</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>7000</td>
<td>1.8 x 10^-3</td>
<td>0.99</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>8000</td>
<td>1.6 x 10^-3</td>
<td>0.99</td>
<td>0.99</td>
<td>0.97</td>
</tr>
<tr>
<td>9000</td>
<td>1.4 x 10^-3</td>
<td>0.99</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>Average</td>
<td>3.7 x 10^-3</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
</tr>
</tbody>
</table>

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).

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

<table>
<thead>
<tr>
<th>Data set</th>
<th>No. of sequences</th>
<th>Average length</th>
<th>Methods</th>
<th>Processing time</th>
<th>No. of matches</th>
<th>ScoreG</th>
<th>ScoreC</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>10</td>
<td>211</td>
<td>G</td>
<td>0s215ms</td>
<td>198</td>
<td>56491</td>
<td>71847</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>27s113ms</td>
<td>197</td>
<td>56491</td>
<td>69010</td>
</tr>
<tr>
<td>S2</td>
<td>5</td>
<td>1780</td>
<td>G</td>
<td>6s194ms</td>
<td>1553</td>
<td>101507</td>
<td>120763</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>13ms877ms</td>
<td>1611</td>
<td>103018</td>
<td>115785</td>
</tr>
<tr>
<td>S3</td>
<td>4</td>
<td>2433</td>
<td>G</td>
<td>0s983ms</td>
<td>2303</td>
<td>85615</td>
<td>104867</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>18ms28150ms</td>
<td>2305</td>
<td>85701</td>
<td>104149</td>
</tr>
<tr>
<td>S4</td>
<td>8</td>
<td>1437</td>
<td>G</td>
<td>17s943ms</td>
<td>804</td>
<td>196746</td>
<td>191880</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>15ms38568ms</td>
<td>901</td>
<td>210317</td>
<td>226516</td>
</tr>
<tr>
<td>S5</td>
<td>8</td>
<td>1680</td>
<td>G</td>
<td>10s685ms</td>
<td>1420</td>
<td>269432</td>
<td>29406</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>21ms5898ms</td>
<td>1439</td>
<td>269092</td>
<td>303390</td>
</tr>
<tr>
<td>S6</td>
<td>4</td>
<td>4582</td>
<td>G</td>
<td>0s784ms</td>
<td>4332</td>
<td>159532</td>
<td>180620</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>1hr4ms58781ms</td>
<td>4409</td>
<td>160860</td>
<td>182475</td>
</tr>
<tr>
<td>S7</td>
<td>5</td>
<td>1093</td>
<td>G</td>
<td>4s394ms</td>
<td>894</td>
<td>60153</td>
<td>75516</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>5ms12948ms</td>
<td>905</td>
<td>60719</td>
<td>73280</td>
</tr>
<tr>
<td>S8</td>
<td>8</td>
<td>1071</td>
<td>G</td>
<td>3s782ms</td>
<td>999</td>
<td>176665</td>
<td>227985</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>8ms35650ms</td>
<td>999</td>
<td>176665</td>
<td>216359</td>
</tr>
<tr>
<td>S9</td>
<td>6</td>
<td>1456</td>
<td>G</td>
<td>8s553ms</td>
<td>1046</td>
<td>116221</td>
<td>139681</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>11ms29146ms</td>
<td>1118</td>
<td>120534</td>
<td>142237</td>
</tr>
<tr>
<td>S10</td>
<td>7</td>
<td>1448</td>
<td>G</td>
<td>6s265ms</td>
<td>1179</td>
<td>170552</td>
<td>211327</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>13ms22264ms</td>
<td>1193</td>
<td>172756</td>
<td>206195</td>
</tr>
<tr>
<td>S11</td>
<td>5</td>
<td>1092</td>
<td>G</td>
<td>4s281ms</td>
<td>840</td>
<td>57682</td>
<td>68951</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>5ms3838ms</td>
<td>855</td>
<td>60100</td>
<td>71361</td>
</tr>
</tbody>
</table>
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

\[
\begin{pmatrix}
an & c & d & e & f & g & h & i & k & l & m & n & p & q & r & s & t & v & w \\
5 & -1 & 12 & -2 & -3 & 7 & -1 & -3 & 2 & 6 & -2 & -2 & -4 & -1 & 1 & 8 & 0 & -3 & -1 & 2 & -3 & 7 \\
-2 & 0 & 0 & -2 & -2 & 10 & -1 & -3 & 0 & -4 & 5 & -1 & -3 & 0 & 1 & -3 & 2 & -1 & 3 & 5 \\
-1 & -2 & -3 & -2 & 0 & -2 & 0 & 0 & 2 & 1 & 2 & 6 & -1 & -2 & -3 & -2 & 0 & 2 & -1 & 2 & 6 \\
-1 & 2 & 0 & -2 & 0 & 1 & -2 & 0 & -3 & 2 & 6 & -1 & -2 & 0 & 1 & -2 & 0 & 2 & -1 & 2 & 6 \\
-1 & -4 & -1 & 0 & -3 & -2 & -2 & -1 & -3 & -2 & -2 & 9 & -1 & -3 & 0 & 2 & -3 & 0 & 2 & -3 & 0 \\
-2 & 2 & 0 & -2 & 0 & 3 & 3 & 2 & -1 & 0 & -2 & 1 & 7 & -2 & -3 & -1 & 1 & 0 & -3 & 2 & 1 & 7 \\
1 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & -1 & -3 & 2 & 1 & 0 & 0 & 1 & 0 & -1 & -3 & 2 & 1 & 7 \\
0 & -1 & -3 & -3 & 0 & -3 & 3 & 2 & 0 & 1 & -3 & -3 & 0 & 0 & -1 & -1 & 0 & -2 & -3 & 0 & 0 & -1 & -1 \\
y & -2 & -3 & -2 & 0 & -1 & 0 & 0 & 2 & 3 & -1 & 3 & 1 & 2 & -2 & -1 & -1 & 3 & 8 & 1 & 1 & 3 & 8
\end{pmatrix}
\]

(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
S6: (RNA)
BTPHOSA BTPHOSB BTPHOSC BTPHOSD
S7: (RNA)
PH35624 PH35625 PH35626 PH35627 PH35628
S8: (RNA)
PT10537 PT10538 PT10539 PT10540 PT10541 PT10542
PT10543
S9: (RNA)
PP59651 PP59652 PP59653 PP59654 PP59655 PP59656
S10: (RNA)
FC07667 FC07668 FC07669 FC07670 FC07672 FC07673
FC07674
S11: (RNA)
MNMHDRBA NMNHDRCB NMNHDRCB NMNHDRCB NMNHDRCBE

580
Genetic algorithm for multiple sequence alignment

References


