Using Genetic Algorithms to Solve Multiple Sequence Alignments

Jorng-Tzong Horng
Dept. of CSIE
National Central Univ.
Taiwan
horng@db.csie.ncu.edu.tw

Ching-Mei Lin
Dept. of CSIE
National Central Univ.
Taiwan
gmlin@db.csie.ncu.edu.tw

Baw-Jhiune Liu
Dept. of CSIE
Yuan-Ze Univ.
Taiwan
bjliu@cs.yzu.edu.tw

Cheng-Yen Kao
Dept. of CSIE
National Taiwan Univ.
Taiwan
cykao@csie.ntu.edu.tw

Abstract

Multiple sequence alignment is an important tool in molecular sequence analysis. This paper presents genetic algorithms to solve the problems of multiple sequence alignments. Several data sets are tested and the experimental results are compared with those run from the the tools on the web. We find our approach could obtain good performance in the data sets with high similarity and long sequences.

1 INTRODUCTION

Multiple sequence alignment is an important tool in molecular sequence analysis. Molecular sequences are composed of a number of elements; for instance, the DNA sequence is composed of four kinds of elements called bases, denoted as A, T, C, and G respectively.

In the evolutionary process, molecular sequences may have several mutations, such as insertion, deletion and substitution. These mutation events result in the diversified difference between sequences.

Multiple sequence alignment can help compare the structure-relationship between sequences by simultaneously aligning multiple sequences such that one base in a sequence corresponds to a space or bases in the other sequences. The alignment of multiple sequences is a 2-dimensional matrix. The total number of rows is equal to the number of sequences to be aligned. The number of columns is the length of alignment. The different base symbols or space symbols in a column of an alignment indicate that several mutation events occur.

Figure 1 shows an alignment composed of four DNA sequences which are *ATCCGCTTAC*, *CTCCTAG*, *ATCCGCTA*, and *TCTCCGTAC*, and a symbol - is introduced to represent a space. The insertion mutations occur in the ninth column composed of three space symbols and one base symbol T. The base symbols in the tenth column are identical and mean no mutation occurred. A score measures the performance of the alignment. Lower mutation obtains higher scores. The best alignment is the one with the highest score. The problem of multiple sequence alignment is an optimization problem searching for the best alignment from large, complex search space. Multiple sequence alignment also can help derive the function of genetic genes and investigate the evolutionary history between organisms. Furthermore, an efficient tool of multiple alignment is needed to help sequence analysis to the exponential growth rate of the amount of molecular sequences.

```
-ATCCGCTTAC
-CT-C-CT-AG
A-TCCGCT-A-
TCTCCG-T-AC
```

Figure 1: An Example of an Alignment

Multiple molecular sequence alignment is characterized by very high computational complexity. Needeman and Wunsch (1970) first used dynamic programming in the comparison of two sequences. This method also has been extended directly to the comparison of three sequences using a three-dimensional matrix (Jue et al., 1980) reduced by Murata et al.(1985) with $O(n^3)$ computational complexity, where n is the longest length of sequences to be aligned. When the dynamic programming method is used for simultaneous multiple sequence alignment, the computational complexity is $O(n^k)$, where k is the number of sequences. Hence, many heuristic methods attempt to find good alignments that are not necessarily optimal within reasonable time. For more details about multiple sequence alignment, the reader may refer Chan et al. (1992).

Stochastic optimization method is one method used to solve multiple sequence alignment such as simulated annealing (Aart et al. 1987), Gibbs sampling (Lawrence et al. 1993) or genetic algorithms (Isokawa et al. 1996; Wayama et al. 1995; Notredame et al. 1996 , 1997; Zhang et al. 1997; and Chellapilla et al. 1999). Isokawa et al. (1996) and Wayama et al. (1995) applied simple genetic algorithms (Goldberg, 1989) with bit matrices. Zhang and Wong(1997) developed a method combining the techniques of genetic algorithms and pairwise dynamic programming. Notredame et al. (1997) used a genetic algorithm for aligning two homologous RNA sequences through their secondary structure.

In this paper, we develop a method through the advantage of genetic algorithm to solve the problem of multiple sequence alignment. The remainder of this paper is organized as follows. In Section 2, we present our genetic algorithm. Section 3 introduces the implementation environment used for this research. Next, Section 4 shows the experimental results of seven data sets which are used to test the performance of our approach. Discussion and conclusions are finally made in Section 5.

2 METHOD

Genetic algorithms (Goldberg, 1989; Whitley, 1994) are stochastic approaches based on the concept of biological evolution and biological genetics. Genetic algorithms operate on chromosome-like data structures that encode possible solutions of the problems, and apply crossover and mutation operators to generate new chromosomes in a search space. Then, based on the principle of survival-of-the-fittest, chromosomes with good performance are selected though selection operator. Genetic algorithms (Jone, 1988) provide efficient and robust search and are suitable for problems with large, complex, and poorly understood search spaces.

In this section, we describe the main components used about crossover, mutation, and fitness in our genetic algorithm.

2.1 Chromosomes

We define a chromosome as multiple number-strings with fixing lengths represented for sequences, including spaces, in an alignment. Each number in a number-string is unique and corresponded to a position of space in an alignment, and the string length is exactly the total number of spaces of sequence in an alignment. A chromosome X of an alignment whose length is L composed of sequences $S_1, S_2, \cdots, S_k$, denoted as $X_1\#X_2\#\cdots\#X_k$, can be represented by

$$\begin{aligned} & X_1\#X_2\#\cdots\#X_k \\ = \; & (x_{1,1}, x_{1,2}, \cdots, x_{1,m_1})(x_{2,1}, x_{2,2}, \cdots, x_{2,m_2})\cdots \\ & (x_{k,1}, x_{k,2}, \cdots, x_{k,m_k}), \end{aligned}$$

where X_i is a number-string represented for the i-th sequence in an alignment; the symbol $\#$ is represented for cancatenation; $x_{i,j}$ is a number indicating a space position of an alignment at the i-th row and j-th column; $x_{i,j} < x_{i,j+1}$; m_i is the length of the i-th number-string; $\forall i$, $l_i + m_i$ is equal to L, if l_i is the length of the i-th sequence; and $1 \le i \le k$ and $\forall i$, $1 \le j \le m_i$. We use **space ratio**, denoted as r_{sp}, to determine the L value. If the longest length of sequence to be aligned is l_{max}, then $L = l_{max} \times (1 + r_{sp})$. The alignments have no full space columns; but we permit this kind of alignments to express the alignments whose lengths are less then the length L of alignments expressed by chromosomes. For example, if an alignment converted from a chromosome has three full space columns, the actual length of the alignment is $L - 3$. In Figure 1 , the $l_{\max}$ is 10. If we let r_{sp} be 0.2, then L is 12. A chromosome $(0,11)(0,3,5,8,11)(1,8,10,11)(6,8,11)$ is one of the encoding versions of the alignment in Figure 1.

Isokawa et al. (1996) and Wayama et al. (1995) encoded chromosomes as a bit matrix consisting of 0 and 1. In the bit matrix, the position of 1 corresponded to a space, and the position of 0 corresponded to a base symbol. The concept of our approach of encoding chromosomes is like the bit matrix. But we only record the space positions.

2.2 System Process Flow

The system process flow is depicted in Figure 2. The symbol |P| represents the size of the population. The concept of the system process flow bases on the architecture of the simple genetic algorithm (Goldberg, 1989). The first population is generated at random. The numbers in the number-string X_i of chromosome X are generated by randomly picking m_i unique and less than L numbers and then sorting these numbers by increasing. The elitism method (Goldberg, 1989) is taken to select the chromosomes of next generation from the offspring and the original chromosomes. The process of crossover and mutation is repeated until the termination conditions are satisfied. The termination conditions are as follows:

```
MAP_GA()
{   generate the initial population P
    let n be multiplied population size by crossover rate
    WHILE( not satisfy the termination condition)
    {   FOR i = 1 TO n DO
        {   select two chromosomes X and Y
              from population at random
            let X' = X and Y' = Y
            Mutation(X')
            Mutation(Y')
            Crossover(X', Y')
            add X' and Y' into mating pool
        }
    }
    select the best top |P| chromosomes to replace
      the originial population
}
```

Figure 2: System Process Flow

- The number of generations exceeded the maximum number of generations permitted, denoted as g_{max}.
- The best fitness does not improve over 50 generations.

2.3 Fitness

To evaluate their fitness, the chromosomes must be converted to the alignment form to be applied sum-of-pairs function (Joao Setubal and Joao Meidanis, 1997). The sum-of-pairs function is defined as the sum of scores of all pairs of symbols in the column. A pair of symbols might get zero score or one of the three kinds of scores, namely, match score, mismatch score and space score. If two base symbols are the same, then the fitness receives a match score; otherwise the fitness obtains a mismatch score. If one of the two symbols is a space symbol and the other is a base symbol, then the fitness receives a space score. If both two symbols are space symbols, then a zero score is given. The mismatch and space scores punish alignments; however, match score rewards alignments. The fitness values of chromosomes are recomputed in mutation and crossover process.

2.4 Crossover

The principle of crossover (Pal and Wang, 1996) is exchanging the information of chromosomes to produce offspring. The chromosomes with better performance might be produced through preserving good structures of parent chromosomes. In the crossover process, two parent chromosomes, denoted as X and Y, are randomly selected and are used to produce two child chromosomes, called best child and random child respectively. Then, cutting points are randomly selected in

$X = (1,4,5,7,8,9,10,15,16,18)(0,3,6,7,10,14,16,17,20)$
$\quad (3,6,8,10,16,20,21)$
$Y = (2,4,5,6,8,9,11,14,17,20)(3,5,6,8,11,13,15,17,20)$
$\quad (3,7,9,11,15,17,18)$

$A_{0,6} = (1,4,5)(0,3)(3)$
$B_{0,6} = (2,4,5)(3,5)(3)$

$A_{6,12} = (7,8,9,10)(6,7,10)(6,8,10)$
$B_{6,12} = (6,8,9,11)(6,8,11)(7,9,11)$

$A_{12,22} = (15,16,18)(14,16,17,20)(16,20,21)$
$B_{12,22} = (14,17,20)(13,15,17,20)(15,17,18)$

Figure 3: Crossover Block Example

parent chromosomes. The blocks among the cutting points are called crossover blocks. The child chromosomes are composed of multiple crossover blocks. Every crossover block, in the best child, is made by duplicating the whole corresponding parent crossover block with good performance. The crossover blocks of random child is made by applying an crossover block operator selected at random to the crossover blocks of parent chromosomes.

The definition of **crossover block** is given below. Let the parent chromosomes X and Y be represented by

$$X_1\#X_2\#\cdots\#X_k = (x_{1,1},x_{1,2},\cdots,x_{1,m_1})\cdots(x_{k,1},x_{k,2},\cdots x_{k,m_k}) \text{ and}$$
$$Y_1\#Y_2\#\cdots\#Y_k = (y_{1,1},y_{1,2},\cdots,y_{1,m_1})\cdots(y_{k,1},y_{k,2},\cdots y_{k,m_k}) \text{ respectively.}$$

$$\text{If } \forall i = 1\ldots k, \left\{ \begin{array}{c} \left(\begin{array}{c} \exists a_i, x_{i,a_i} < p \le x_{i,a_i+1} \text{ and} \\ y_{i,a_i} < p \le y_{i,a_i+1} \end{array} \right) \\ \text{and} \left(\begin{array}{c} \exists b_i, x_{i,b_i} < q < x_{i,b_i+1} \\ \text{and } y_{i,b_i} < q < y_{i,b_i+1} \end{array} \right) \end{array} \right\},$$

we called $(x_{1,a_1+1},\cdots,x_{1,b_1})\cdots(x_{k,a_k+1},\cdots,x_{k,b_k})$ and $(y_{1,a_1+1},\cdots,y_{1,b_1})\cdots(y_{k,a_k+1},\cdots,y_{k,b_k})$ as the crossover blocks $A_{p,q}$ and $B_{p,q}$ for X and Y respectively. The number of crossover blocks are randomly chosen in a given range determined by the longest length of the alignments which chromosomes can express. For example, the crossover blocks depicted in Figure 3 have three crossover blocks respectively in each parent chromosomes X and Y, namely $A_{0,6}$, $A_{6,12}$, $A_{12,22}$, $B_{0,6}$, $B_{6,12}$, and $B_{12,22}$.

We use three kinds of crossover block operators in our approach, namely, *CombineBest*, *GoodPosCombine* and *OnePointCombine*.

The *CombineBest* operator makes a new crossover block by duplicating the whole parent crossover block with good performance. The *OnePointCombine* operator would like to recombine the space positions from the parent chromosomes. The purpose of the *GoodPosCombine* operator is to preserve the combination

OnePointCombine(two crossover blocks $A_{p,q}$ of X and $B_{p,q}$ of Y)
{ **FOR** each number-string X_i **DO**
  { pick a number n in the range from $a_i + 1$ to $b_i - 1$ at random
    IF $x_{i,n} < y_{i,n+1}$ **THEN**
      make a new number-string
        $(x_{i,a_i+1}, x_{i,a_i+2}, \cdots, x_{i,n}, y_{i,n+1}, \cdots, y_{i,b_i})$
    ELSE
      make a new number-string
        $(y_{i,a_i+1}, y_{i,a_i+2}, \cdots, y_{i,n}, x_{i,n+1}, \cdots, x_{i,b_i})$
  }
  new crossover block is composed of these new number-strings
}

Figure 4: The Flow of OnePointCombine Crossover Block Operator Operator

GoodPosCombine(two crossover blocks $A_{p,q}$ of X and $B_{p,q}$ of Y)
{ **FOR** $i = 1$ **TO** k **DO**
  { let $s = (x_{i,a_i+1}, x_{i,a_i+2}, \cdots, x_{i,b_i})$
    let $t = (y_{i,a_i+1}, y_{i,a_i+2}, \cdots, y_{i,b_i})$
    let C be a number set, and $\forall c \in C$, c commonly appears in s and t
    let D be a number set, and $\forall d \in D$, d only belongs to s or t
    let n be a length of the number-string
    make a new number-string by duplicated the numbers in C and randomly selected $|s| - |C|$ numbers from D
  }
  new crossover block is composed of these new number-strings
}

Figure 5: The Flow of the GoodPosCombine Crossover Block Operator

of good space positions and alter the combination of bad space positions. The space positions which both parents have are thought to be the good space positions; otherwise, the space positions which only one parent have are thought as the bad space positions. The new combination of bad space positions is selected from the bad space positions of two parents.

The flows of *OnePointCombine* and *GoodPosCombine* are shown in Figure 4 and Figure 5, respectively. In Figure 5, the symbol $|C|$ is represented for the number of elements of C. Figure 6 shows an example for crossover operators: *OnePointCombine* and *GoodPosCombine*. Figure 6(a) shows a new crossover block that combines two parent crossover blocks $B_{12,22}$ by *OnePointCombine*. The bold and italic numbers in each new number-string are provided from X crossover block and the others are provided from Y. Figure 6(b) shows a new crossover block that combines two parent crossover blocks by *GoodPosCombine*. The bold and italic numbers in each new-number string are the numbers that both parents have; the others are randomly selected from the corresponding number-strings of two parents.

$B_{12,22}$ $X = (15, 17, 18, 19, 21)(14, 16, 17, 20)(16, 19, 21)$
$Y = (14, 17, 18, 20, 21)(13, 17, 18, 20)(17, 18, 20)$
(a)
$(\mathbf{15}, \mathbf{17}, 18, 20, 21)(\mathbf{14}, \mathbf{16}, 18, 20)(\mathbf{16}, 18, 20)$
(b)
$(14, \mathbf{17}, \mathbf{18}, 19, \mathbf{21})(14, \mathbf{17}, 18, \mathbf{20})(16, 18, 20)$

Figure 6: Example of Crossover Operator

2.5 Mutation

There are four kinds of mutation operators in our approach, namely, *MergeSpace*, *MoveSpaceCol*, *BreakSpaceCol* and *MoveRowSpace*. In the mutation process, the input chromosomes are applied m times to mutation operators selected at random. The m value depends on the longest length of number-string in chromosomes. The purpose of the *MergeSpace* operator is to merge two or three spaces together. The *MoveSpaceCol* operator is aimed to move one or two continuous space columns to continuous symbol columns having no space. The *BreakSpaceCol* operator changes a space column to a symbol column, and the spaces in that space column seek new positions again.

The flows of the *MergeSpace*, *MoveSpaceCol*, and *BreakSpaceCol* operators are presented in Figure 7, Figure 8, and Figure 9 respectively. Figure 10 shows an example of mutation operators. Figure 10(a) shows a chromosome mutated from the input chromosome Y by the *MergeSpace* mutation operator. The space positions 7, 9 and 10 in third number-string are selected to shift and merge together, then new space position 13, 14 and 15 are generated. The chromosome in Figure 10(b) is mutated though the *MoveSpaceCol* mutation operator. Space positions 8, 9 and 10 are selected at random and move to the right-hand side. The new space positions 11, 12 and 13 which are not in input chromosome Y are chosen. Then the numbers 8, 9 and 10 in each number-string are replaced with 11, 12 and 13 respectively. The chromosome depicted in Figure 10(c) is mutated from the *BreakSpaceCol* mutation operator. The number 10 is randomly chosen from the number-strings in chromosome Y, then the number 10 is replaced in each number-string including this number to a random number 13, 16 and 15 respectively.

The *MoveRowSpace* mutation operator rearranges the space positions of some sequences in a given column range according to the summary information of the sequences in the alignment. The *MoveRowSpace* operator is based on the basic two-sequence dynamic programming (Setubal and Meidanis, 1997) and has some modification in the dynamic table. We use the example in Figure 11 to explain. Figure 11(a) shows an input chromosome X. First, a sub-alignment with short length is extracted from the long length alignment converted by chromosome X. A sub-alignment depicted in Figure 11(b) begins with the twenty-first column and ends with the forty-fifth column. Second, a target sequence is produced by deleting the spaces from a sequence including spaces randomly selected in the sub-alignment. Then, a template sequence is made in which symbols are the symbols with highest occurring frequency rate in each column of sub-alignment. In Figure 11(c), $s2$ is the target sequence produced from the first sequence in the sub-alignment and $s1$ is the template sequence. Third, the target and template sequence are aligned by two-sequence dynamic programming under the condition that there is no new space to insert into the template sequence. In the process of the two-sequence alignment, the spaces in the template sequence are thought of as a kind of base symbols. After the aligning process, new space positions in target sequence is made (see Figure 11(d)), and then is translated to a new sub-number-string sorting by increasing (see Figure 11(e)). Finally, this new sub-number-string replaces the corresponding old sub-number-string.

The basic two-sequence dynamic programming needs to calculate a 2-dimensional matrix M. If the lengths of template sequence s and target sequence t are p and q respectively, M is of the size $p \times q$. The method of computing dynamic table is modified generally. The entry values stored in M are defined as

$$\begin{aligned} M_{0,0} &= 0 \\ M_{0,j} &= -\infty \\ M_{i,0} &= \sum_{h=1}^{j} P(s_i, -) \\ M_{i,j} &= \max \left\{ \begin{array}{l} M_{i-1,j} + P(s_i, -) \\ M_{i-1,j-1} + P(s_i, t_j) \end{array} \right. \end{aligned}$$

where $M_{i,j}$ is the entry at the i-th row and j-th column $(1 \le i \le p, 1 \le j \le q)$; the function P calculates the score of the two input symbols; s_i is the i-th symbol in the template sequence s; t_j is the j-th symbol in the target sequence t. After creating the matrix M, an two-sequence alignment can be formed by tracing back from $M_{p,q}$. The *MoveRowSpace* operator randomly picks some sequences to rearrange their space positions by the method described above and the template sequence is the same in each alignment process.

3 ALGORITHM IMPLEMENTATION

The algorithm is implemented using the Microsoft Visual C++. The machine used for this research is a

MergeSpace(a chromosome X)
{ randomly pick a number-string $X_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,m_i})$ in X
randomly pick numbers $x_{i,g}, x_{i,g+1}$ or $x_{i,g}, x_{i,g+1}, x_{i,g+2}$
randomly pick numbers $h, h+1 \notin X_i$ or $h, h+1, h+2 \notin X_i$
replace $x_{i,g}, x_{i,g+1}$ or $x_{i,g}, x_{i,g+1}, x_{i,g+2}$ to
$h, h+1$ or $h, h+1, h+2$ respectively
sort the numbers in X_i by increasing
}

Figure 7: The Flow of MergeSpace Operator

MoveSpaceCol(a chromosome X)
{ randomly pick number $h \in X$ or $h, h+1 \in X$
randomly pick number $t \notin X$ or $t, t+1 \notin X$
replace number h or $h, h+1$ to t or $t, t+1$
respectively for each number-string in X
sort each number-string by increasing
}

Figure 8: The Flow of MoveSpaceCol Operator

BreakSpaceCol(a chromosome X)
{ randomly pick a number $h \in X$
d is a number could be 0 or 1
FOR each number-string X_i **DO**
{ **IF** $d = 0$
THEN r is a random number and $r > h$
ELSE r is a random number and $r < h$
IF $\exists h \in X_i$ **THEN** replace h to r
sort X_i by increasing
}
}

Figure 9: The Flow of BreakSpaceCol Operator

$Y = (2, 4, 5, 6, 8, 9, 10, 14, 17, 20)(3, 5, 6, 8, 10, 14, 15, 17, 20)\ (3, 7, 9, 10, 16, 17, 18)$
(a)
$Y = (2, 4, 5, 6, 8, 9, 10, 14, 17, 20)(3, 5, 6, 8, 10, 14, 15, 17, 20)\ (3, \mathbf{13}, \mathbf{14}, \mathbf{15}, 16, 17, 18)$
(b)
$Y = (2, 4, 5, 6, \mathbf{11}, \mathbf{12}, \mathbf{13}, 14, 17, 20)(3, 5, 6, \mathbf{11}, \mathbf{13}, 14, 15, 17, 20)(3, 7, \mathbf{12}, \mathbf{13}, 16, 17, 18)$
(c)
$Y = (2, 4, 5, 6, 8, 9, \mathbf{13}, 14, 17, 20)(3, 5, 6, 8, 14, 15, \mathbf{16}, 17, 20)(3, 7, 9, \mathbf{15}, 16, 17, 18)$

Figure 10: Example of Mutation Operator

(a)
$(\cdots 23,25,33,38,39,40,41,44\cdots)(\cdots 33\cdots)(\cdots 23,25,26,33\cdots)$
$(\cdots 22,23,25,26,33\cdots)(\cdots 23,25,33,38,39,40,41\cdots)$
$(\cdots 23,25,26,33\cdots)$
(b)
```
GGT-T-AAGTTTA-A ATT----TT-AGGGG
GGTCGCAGGTTAA-G TTTAAATTTTAGGGG
CCA-T--GGTTAA-G TTTAAATTTTAGGGG
AG--T--GGCCAT-G GTTAAGATTAAATTT
GGT-T-AAGTTTA-A ATT----TTAGGTGG
CCA-T--GGTTAA-G TTTAAATTTTAGGGG
```
(c)
s_1: `GGT-T--GGTTAA-GTTTAAATTTTAGGGG`
s_2: `GGTTAAGTTTAAATTTTAGGGG`
(d)
s_1: `GGT-T--GGTTAA-GTTTAAATTTTAGGGG`
s_2: `GGT-T------AA-GTTTAAATTTTAGGGG`
(e)
$(23, 25, 26, 27, 28, 29, 30, 33)$
(f)
$(\cdots 23,25,26,27,28,29,30,33\cdots)(\cdots 33\cdots)(\cdots 23,25,26,33\cdots)$
$(\cdots 22,23,25,26,33\cdots)(\cdots 23,25,33,38,39,40,41\cdots)$
$(\cdots 23,25,26,33\cdots)$

Figure 11: An Example of the MoveRowSpace Operator

personal computer with an Intel Pentium III processor rated at 500MHz. The main memory is 512 megabytes. The operating system is Microsoft Windows NT 4.

4 EXPERIMENTAL RESULTS

Our genetic algorithm is applied to seven sets of molecular sequences. These data sets differed with respect to their length, number, or similarity. The experimental results of multiple sequence alignment are compared with the tool of ClustalW (Thompson et al. 1994) which is the most commonly used available. Owing to the different scoring system, we only compared the numbers of match columns of alignments between our algorithm and the ClustalW.

The average lengths of sequences in the first four data sets were shorter than the average lengths in the last three data sets. The sequences in each data set had the similar length. All the data sets we used can be acquired from GenBank. The keywords of the sequences used in experiment are listed in Appendix. Data set 1 was composed of ten sequences from Zhang and Wong (1997). The average length of data set 1 was 212 nucleotides. Data set 2, data set 3 and data set 4 were from Chellapilla and Fogel (1999). Data set 2 was composed of eight sequences of 16S rRNA and the sequences in Data set 2 had the equal length that was 457 nucleotides. Data set 3 and data set 4 were composed of twenty-one sequences extracted from the histone H3 gene and of which average length were 122 and 333 nucleotides. Data set 5, data set 6 and data set 7 were extracted from human immunodeficiency virus. Data set 5 was composed of six sequences of which average length were about two thousands nucleotides; data set 6 was composed of eight sequences of which average length were about three thousands nucleotides. Data set 7 was composed of nine sequences of which average length were about eight thousands nucleotides.

Table 1 shows the sequence data in each data set. The column 'Data Set' identifies the number of data set. The second column is the number of sequences. The third column is the average, shortest, and longest length of sequences. The genetic algorithm has been applied to each data set for continuous ten times. Table 2 shows the experimental average result. The column 'Avg Score' and 'Avg M. C.' denote the average scores and the numbers of match columns of the alignments. The column 'Avg Gen#' represente the average number of generations in ten continuous trials. Table 3 shows the best results in this ten continuous trials. The score and number of match columns of the best result are recorded in column 'Best Score' and 'Best M. C.', respectively. The column 'ClustalW M. C.' describe the numbers of match columns in alignments resulted from ClustalW. Our genetic algorithm had good performance in data set 1, 2, 3 and 5 composed of high similar sequences. The alignments in data set 1, 2 and 3 had the same numbers of match columns with the ones resulted from the ClustalW. The alignments in data set 5 had better quality than the ones resulted from the ClustalW.

Next, we describe the parameters used in the experiment. The space ratio r_{sp} was 0.2. Chellapilla and Fogel (1999) observed that solutions to common alignment problems rarely contained more than 20 percent gaps. But when the sequences to be aligned has a great similarity, our algorithm could be more efficient, yielding the r_{sp} value. If the sequences were less similarity, the value of r_{sp} was too small to find the optimal alignment. The match, mismatch, and space scores were $+2$, -1, and -2 respectively. The penalty of insertion or deletion mutation is more heavy than the substitution mutation. The probabilities of *OnePointCombine*, *CombineBest*, and *GoodPosCombine* crossover operator were 0.55, 0.15 and 0.3 respectively. The probabilities of *MergeSpace*, *MoveSpace*, *BreakSpaceCol* and *MoveRowCol* mutation operator were 0.2, 0.2, 0.3 and 0.3 respectively. The crossover rate was 1. The population size of the first three data sets was two hundreds, and the population size of the last four data sets was three hundreds.

Data Set	Seq #	Avg. Length (Min,Max)
1	10	211(211,212)
2	8	457(457,457)
3	21	122(122,122)
4	21	333(324,346)
5	6	2685(2677,2692)
6	8	3330(3311,3370)
7	9	8810(8834,8785)

Table 1: Data Set

Data Set	Avg. Score	Avg. M. C.	Avg. Gen#
1	18576	198	188.6
2	25312.7	448.9	265.2
3	47363.4	109	197.2
4	76180.6	83	600
5	79375.8	2639.6	180.8
6	132488	1901.1	600
7	507211.8	5917.3	600

Table 2: Average Results of the Continuous Ten Trials

Data Set	Best Score	Best M. C.	ClustalW M. C.
1	18612	198	198
2	25343	449	449
3	47775	109	109
4	79291	100	107
5	79473	2641	2638
6	142815	2124	2352
7	521687	6220	6315

Table 3: Best Result in the Continuous Ten Trials

5 CONCLUSION

Multiple sequence alignment is an important problem in molecular biology and a need-efficient tool to align multiple and long sequences. Our algorithm has good performance and efficiency in the data set of high similar sequences as well as low-similar sequences. Our method has the quality that is close to the alignments of the ClustalW. Our approach could quickly and globally find the similar regions in alignments even though the average length of sequences is long.

The scoring method used in our approach is simple. In the future, we will revise our scoring method to make the alignment score more realistic, improve our algorithm such that also could have better performance in lower-similar sequences, and experiment with the additional data sets which having longer sequences and large number of sequences.

References

Aart, E.H.L. and van Laarhoven, P.J.M. (1987). *Simulated Annealing: a Review of Theory and Applications.* Kluwer Academic Publishers, Amsterdam.

Chan, S. c., Wong, A. K. C. and Chiu, D. K. Y. (1992). A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology.* 54: 563-598.

Chellapilla, Kumar and Fogel, Gary B. (1999). Multiple sequence alignment using evolutionary programming. *Congress on Evolutionary Computation.* 445-452.

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

Isokawa, M., Wayama, M. and Shimizu, T. (1996). Multiple sequence alignment using a genetic algorithm. *Genome Informatics 7*: 176-177.

Jong, K. De (1988). Learning with genetic algorithms: An overview. In *Machine Learning 3*, 121-138. Hingham, MA: Kluwer.

Jue, R. A., Woodbury, N. W. and Doolittle, R. F. (1980). Sequence homologies among E. coli ribosomal proteins: evidence for evolutionary related groupings and internal duplications. *Journal of Molecular Evolution.* 15: 129-148.

Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F. and Wootton, J.C. (1993). Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science.* 262:208-214.

Murata, M., Richardson, J. S. and Sussman, J. L.

(1985). Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Science U. S. A.* 82: 3073-3077.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology.* 42: 245-161.

Notredame, Cedric and Higgins, Desmond G. (1996). SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research.* 24(8): 1515-1524.

Notredame, Cedric , O'Brien, Emmet A. and Higgins, Desmond G. (1997). RAGA: RNA sequence alignment by genetic algorithm. *Nucleic Acids Research.* 25(22): 4570-4580.

Pal ,Sankar K. ,Wang ,Paul P. (1996). *Genetic algorithms for pattern recognition.* Boca Raton : CRC Press.

Setubal, Joao and Meidanis, Joao (1997). Sequence Comparison and Database Search. In *Introduction To Computational Molecular Biology,* 47-103. PWS.

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 Research.* 22(22):4673-4680.

Wayama, M., Takahashi, K. and Shimizu, T. (1995). An approach to amino acid sequence alignment using a genetic algorithm. *Genome Informatics 6*: 122-123.

Whitley, Darrell (1994). A Genetic Algorithm Tutorial. *Statistics and Computing.* 4:65-85.

Zhang, Ching and Wong, Andrew K. C. (1997). Toward efficient multiple molecular sequence alignment: A system of genetic algorithm and dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics-part B: Cybernetics.* 27(6): 918-932.

Appendix

Data Set 1: (Zhange and Wong, 1997)

HCV2L1A10	HCV2L3A5	HCV2L3A7
HCV2L3A9	HCV2L3B1	HCV2L3B2
HCV2L3C1	HCV2L3C8	HCV2L3D4
HCV2L3E6		

Data Set 2: (Chellapilla and Fogel, 1999)

AF095268	AF095267	AF095266
AB023287	AB023286	AB023285
AB023284	AB023283	AB023279
AB023278	AB023276	

Data Set 3: (Chellapilla and Fogel, 1999)

TPAHISIN	TNIHISIN	TNHISIN
TMIHISIN	TMHISIN	TLHISIN
THHISIN	TFHISIN	TEHISIN
TCUHISIN	TCHISIN	TCAHISIN
TBHISIN	TAUHISIN	TAHISIN
TTHISIN	TSHISIN	TRHISIN
TPYHISIN	TPIHISIN	TPHISIN

The first 122 symbols from each of the above sequences were used for alignment.

Data Set 4: (Chellapilla and Fogel, 1999)

The sequences were extracted at 123rd symbol and the lengths of each sequences in data set were 329, 328, 332, 337, 344, 336, 334, 325, 335, 342, 334, 333, 335, 330, 333, 324, 334, 334, 333, 348 respectively.

Data Set 5:

U12035	U12034	U12033
U12032	U12031	U12030

Data Set 6:

L22951	AF069673	AF071473
AF069672	AF067156	AF004885
AF069670	U51190	

The sequences were extracted at 1rd, 4950rd, 4960rd, 4952rd, 5110rd, 8219rd, 4965rd, and 5128rd symbols and end with 3346, 8266, 8305, 8276, 8430, 8540, 8298,and 8479 respectively.

Data Set 7:

AF069673	AF069670	AF071474
AF004885	AF107771	U51190
M62320	U54771	AF069671

The sequences were extracted at 1rd, 3rd, 1rd, 260rd, 192rd, 166rd, 263rd, 361rd, and 28rd symbols and end with 8785, 8816, 8813, 9058, 8984, 8999, 9065, 9172, and 8859 respectively.