Genetic Algorithm Approaches for Efficient Multiple Molecular Sequence Alignment

by

Ching Zhang

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Systems Design Engineering

Waterloo, Ontario, Canada, 1998

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Multiple biomolecular sequence alignment is among the most important and challenging tasks in computational biology. Current approaches are characterized by great complexity in computational time. The complexity has limited the use of the approaches in many practical applications.

In this research, new approaches based on a genetic algorithm have been developed for multiple biomolecular sequence alignment. Their major strengths are very high efficiency and good alignment quality. Experiments using real data sets have shown that the average computing time of these approaches is one to three orders of magnitude lower than that of a most widely used program while the qualities are very similar.

The key component of these approaches is an enhanced genetic algorithm. Genetic algorithms are a set of stochastic algorithms for efficient and robust search. The basic idea of this approach is the conversion of multiple sequence alignment into a search problem. The conversion enables us to apply a genetic algorithm for efficient identification of matches between multiple sequences.

Three methods, two of them based on dynamic programming, have been developed to handle mismatches. The combination of the genetic algorithm and these methods may produce high quality alignments in an efficient manner.

In this thesis, the theoretical fundamentals of the approaches are discussed, the

procedures of the enhanced genetic algorithm as well as the three methods are presented and analyzed, and the experimental results are described and compared with the results obtained by using a most widely used multiple molecular sequence alignment program.

ACKNOWLEDGEMENTS

I would like to thank Dr. Andrew K. C. Wong, my supervisor, for his invaluable advice during the whole process of this project. I would also like to thank him for his encouragement and financial support.

I would like to thank Drs. T. Carey, M. Kamel, T. Jiang, M. Li, and D. Stashuk, the members on my advisory and examination committees, for their advice and suggestions.

To my parents and my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem of Multiple Sequence Alignment

A biomolecule, such as DNA (deoxyribonucleic acid), RNA (ribonucleic acid) or protein, is made up of a long chain of subunits – deoxyribonucleotides, ribonucleotides, or amino acids, respectively (Figure 1.1 and Figure 1.2). The analysis of molecular sequences is essential in discovering molecular structures, characteristic motifs, feature patterns and understanding the biochemical functions which are associated with them. With the development of rapid DNA sequencing technology, large volumes of molecular data are at our disposal. Developing efficient computational analysis techniques is in high demand for us to extract more valuable information from the data.

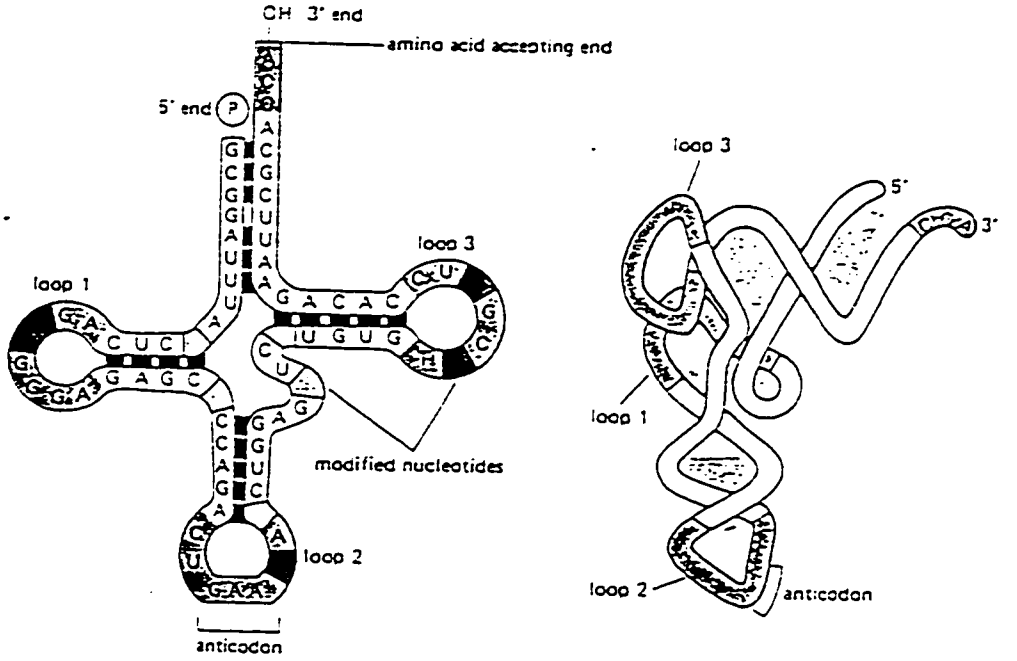


Figure 1.1: Phenylalanine tRNA of yeast. On the left, the nucleotides of $\{A, U, G, C\}$ are arranged to show the base-pairing that forms internal helical regions in the tRNA molecule. It is also known as the secondary structure of the tRNA. On the right, the folded molecule of the tRNA is illustrated (Albert, el at., 1983).

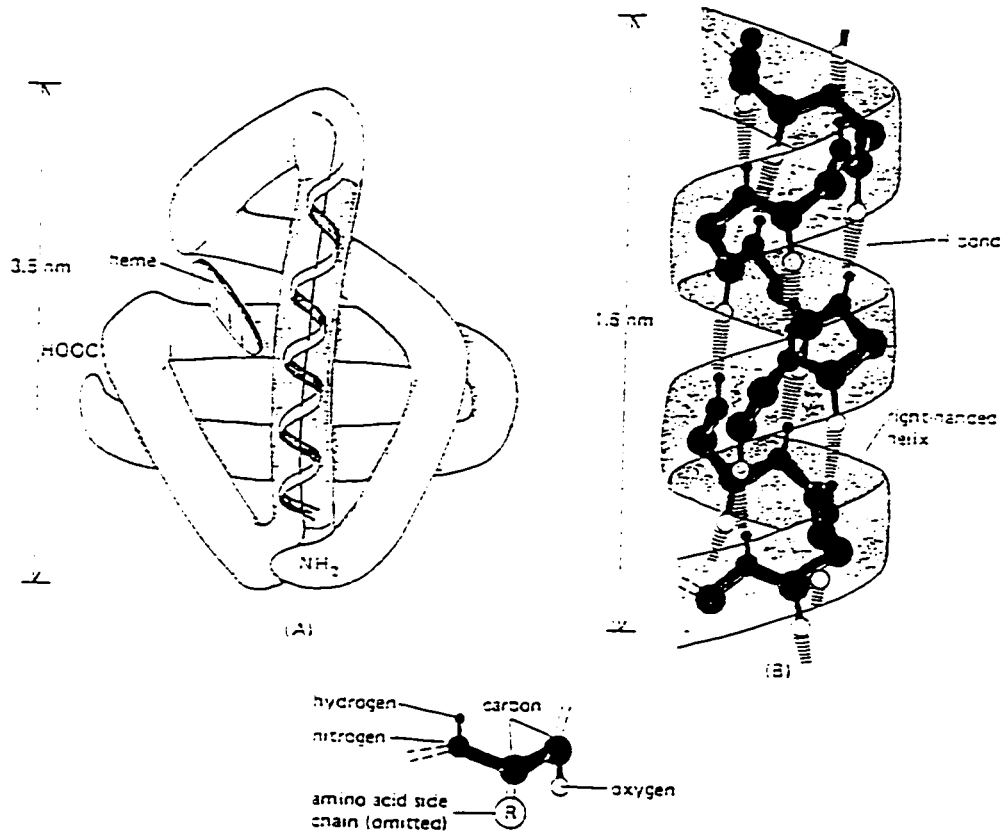


Figure 1.2: At the bottom, the chemical composition of an amino acid molecule. On the right, the chain connection of amino acids which forms a specific type of secondary structure of protein – α-helix. On the left, the oxygen-carrying protein – myoglobin which is 153 amino acid long, with one region of α-helix (Albert, el at., 1983).

Multiple molecular sequence alignment is a most valuable step in molecular sequence analysis. It involves determining the similarity or dissimilarity among sequences. In the output of sequence alignment, one-to-one correspondences between/ among subunits of the input sequences are identified. It is an important approach towards pattern matching and recognition in biological data (Lander, et. al., 1991). Two of the most general purposes of multiple molecular sequences alignment are: finding highly conserved subregions or embedded patterns of a set of biological sequences; and inferring the evolutionary history of a set of taxa from their associated biological sequences (Gusfield, 1993). Therefore, many tasks such as protein motifs matching, structural and functional subdomain recognition, and DNA gene region identification, are heavily dependent on the output of this step.

Multiple molecular sequence alignment is characterized by great computational complexity. In a practical application, the number of possible alignments is huge and therefore an exhaustive (or directed) enumeration is almost impossible. As estimated by Laquer (1981) and Griggs et. al. (1989) respectively, the total number of possible alignments is 10^{767} for two sequences of 1000 subunits and 10^{1755} for three sequences of 1000 subunits. The efficient alignment algorithms required should be able to handle possible insertions, deletions, substitutions, as well as matches.

The traditional approaches to the multiple sequence alignment problem are challenged by high computational costs. The dynamic programming method developed by Sankoff (1972) has been accepted as an effective method for two-sequence align-

ment. It guarantees to generate the *optimal* alignment in aligning two sequences. For two sequences of lengths m_1 and m_2, a straightforward implementation of this method needs $O(m_1 m_2)$ time and $O(m_1 m_2)$ space when single insertions/deletions are considered: or $O(m_1 m_2^2 + m_1^2 m_2)$ when multiple insertions/deletions are considered (Waterman, 1984). 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 time complexity is theoretically $O(m^n)$, where n is the number of sequences and m is the length of the sequences.

To align multiple sequences with reasonable computer resources, a number of methods have been developed. Some representatives will be reviewed in the next chapter. Many of the methods use heuristics to find good alignments which are not necessarily optimal. According to a comprehensive survey conducted by Chan et al. (1992), the 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; and (v) the template approaches.

Of the more recent methods, a large group combines heuristic techniques with pairwise dynamic programming alignment process. 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. Gotoh, 1990; Vingron and Argos, 1991; Roytberg, 1992; Vihinen et al., 1992; Feng and Doolittle, 1987; Thompson, et al., 1994; Chan, 1990; Wang et al. 1996). 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). Results of these methods may have deviations from the optimal and most of the methods are still very costly. While aligning n sequences, they usually require at least $(n-1)$ pairwise alignment processes. The costs are still high especially for long sequences and have prevented the methods from wider use. More efficient methods are highly needed.

1.2 An Overview of the Research

In this research, new approaches have been developed for efficient *simultaneous* multiple sequence alignment [1]. These approaches are much faster than most of the previous approaches and they are able to achieve good solutions.

The core part of these approaches is a genetic algorithm. Genetic algorithms are a set of stochastic algorithms for efficient and robust search. Unlike the majority of the conventional search algorithms, a genetic algorithm starts with a population of points in the search space instead of a single one. In each step of search, it generates a new, and usually better generation of points. Searching towards the better points may improve efficiency, and searching with a population may decrease the chances 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).

In this research, the task of molecular sequence alignment is decomposed into two steps. In the first step, some matches between sequence subunits are identified and organized into *match blocks*. In the second step, subsequences between the match blocks are processed, and matches and mismatches (i.e. insertions, deletions, and substitutions) between match blocks are handled. The matches identified in the first step are represented in a structure called *pre-alignment*[2].

[1] A simultaneous multiple sequence alignment is one in which all the sequences are processed at the same time, instead of in a pairwise manner.

The identification of match blocks is converted into a search problem. A *pre-alignment space*[2] is defined which contains all the possible pre-alignments. An enhanced genetic algorithm, which has different characteristics from standard ones, is designed to search the space for an optimal or a near-optimal pre-alignment. The algorithm starts with a population of pre-alignments. In each step of search, it produces a new, usually a better, generation of pre-alignments. The pre-alignment with the highest fitness value is selected from the population after the process terminates.

In the second step, an alignment is produced from the best pre-alignment selected. This step involves handling insertions, deletions, and substitutions as well as matches between the match blocks identified in the pre-alignment. Different methods are used to handle the mismatches in order to produce a high quality, compact alignment. The methods include a simple shift-up method, a pairwise dynamic programming method and a sequence synthesis method.

The major strength of the approach is its high efficiency and good alignment quality. Theoretical analysis has shown that it has lower time complexity than current techniques based on pairwise dynamic programming. Experiments using real data sets have shown that the actual computing time of this approach is one to three orders lower than that of a most widely used approach while the qualities are very similar.

[2]The definitions of pre-alignment and pre-alignment space are given in Section 4.2.

1.3 Thesis Synopsis

This thesis is organized into eight chapters. The next chapter (Chapter 2) is the literature review. The sequence alignment techniques reviewed include dynamic programming for optimal two sequence alignment, pair-wise dynamic programming-based heuristic approaches for multiple sequence alignment, and many others. Then genetic algorithms are briefly described and their applications, especially those in biomolecular sequence analysis, are examined.

In Chapter 3, the technology of genetic algorithms is described in a detailed and formal manner. The issues include the fundamentals of genetic algorithms, the search procedure, the law of the growth of "good" strings, algorithm complexity analysis, and population size selection.

In Chapters 4, 5, 6, and 7, the core part of this research is presented. Chapter 4 presents the fundamental idea underlying the design of an enhanced genetic algorithm for multiple sequence alignment. The algorithm for multiple sequence alignment is given as a procedure. The selection of important algorithm parameters is discussed. The survival ability of different string elements is studied. Then the time and space complexities of the algorithm are analyzed.

Chapter 5 presents the initial experimental results, in which the pre-alignments are obtained by applying the genetic algorithm to real data sets. The results are then compared with those obtained by CLUSTALW, a most widely used multiple sequence

alignment program (Thompson et al. 1994).

In Chapter 6, a simple shift-up method is combined with the genetic algorithm for processing the subsequences between match blocks. The alignments by this approach are presented and compared with those by CLUSTALW. This chapter also gives a description of the system implementations.

Chapter 7 describes two approaches integrating the genetic algorithm and dynamic programming. In one approach, dynamic programming is used for pairwise alignment and in the other, for sequence synthesis. The approaches combine the strengths of the two techniques. They are aimed at achieving even better alignments efficiently. Following the examination of the complexity of the combined approaches is the presentation and analysis of the experimental results. The quality improvement obtained by the two approaches is then discussed.

In Chapter 8, the major strength of these approaches is summarized. This last chapter ends with a proposal of some future research topics.

Chapter 2

Background and Literature Review

In this chapter, the theoretical background of dynamic programming and the existing heuristic techniques for multiple molecular sequence alignment are reviewed. The development and applications of genetic algorithms as well as the applications of genetic algorithms in biosciences are examined.

2.1 Dynamic Programming

Dynamic programming methods are bottom-up optimization approaches by avoiding redundant calculation. Sankoff (1972) and Sankoff and Sellers (1973) developed dynamic programming methods for minimum distance alignment of two sequences with single deletions/insertions. These algorithms were extended to handle multiple insertions/deletions by Waterman et al. (1976). A more practical method for two sequence optimal alignment with linear space complexity was later developed by Hirschberg (1975), and Myers and Millers (1988). Gaps in alignments were often assigned a penalty which is a linear function of the number of inserted or deleted subunits (Waterman, 1984). These methods have been very successful for two-sequence alignment. Also, they have provided a base for building heuristic methods for multiple sequence alignment.

2.1.1 The distance

In applying dynamic programming algorithms to the comparison of molecular sequences, distance measurement is an important issue. Levenshtein (1966) developed an early definition of the minimum distance between two sequences as the minimum number of insertions, deletions and substitutions required to change one into another.

As noted earlier, a sequence is a chain of *subunits*. In sequence analysis the subunits are represented by *characters* depending on the different types of sequences.

Assume the sequences are represented as:

$$\begin{aligned} \mathbf{a} &= a_1 a_2 a_3 ... a_m, \\ \mathbf{b} &= b_1 b_2 b_3 ... b_n \end{aligned} \tag{2.1}$$

where the a's and b's are characters of the sequences, and, m and n are the lengths of the sequences.

The *minimum distance*, $D(\mathbf{a}, \mathbf{b})$, can thus be defined as the smallest possible weighted sum of insertions, deletions and substitutions which transform sequence **a** into sequence **b**.

The following **definition of distance** is commonly used in dynamic programming:

- $D(\mathbf{a}, \mathbf{b})$ is the minimum distance between sequences **a** and **b**, which is defined in terms of $d(a, b)$.
- $d(a, b)$ is the weighted cost value for the following operations on characters a and b.
  - $d(a_i, b_j)$ represents the cost of substituting b_j for a_i, where $a_i \in \mathbf{a}$, $b_j \in \mathbf{b}$ and $a_i \neq b_j$.
  - $d(a_i, \phi)$ represents the cost of deleting a_i from **a**, where ϕ is the null character.
  - $d(\phi, b_j)$ represents the cost of inserting b_j into sequence **a**
  - $d(a_i, b_j) = 0$ if and only if $a_i = b_j$.

2.1.2 Optimal two sequences alignment

Sellers (1974a, 1974b) presented several algorithms as theorems for sequence pattern recognition. The first theorem resulted in an algorithm for finding the best alignment between two sequences which had the smallest distance (or cost). Sellers' Best Alignment Theorem is based on the algorithms developed by Sankoff (1972) and Needleman and Wunsch (1970) but using a metric distance and the function in Equation (2.2) (Tyler, 1991).

The algorithms of dynamic programming for two-sequence alignment developed by Sellers (1974a, 1974b) and later modified by Waterman (1984) can be expressed as:

The Algorithm of Best Alignment

Considering single deletions/insertions, the defined initial distance function d is:

$$d(a_i, b_j) = \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{if } a_i \neq b_j \end{cases} \tag{2.2}$$

and let

$$D_{0,0} = 0,$$

$$D_{0,j} = \sum_{k=1}^{j} d(\phi, b_k),$$

and

$$D_{i,0} = \sum_{k=0}^{i} d(a_k, \phi).$$

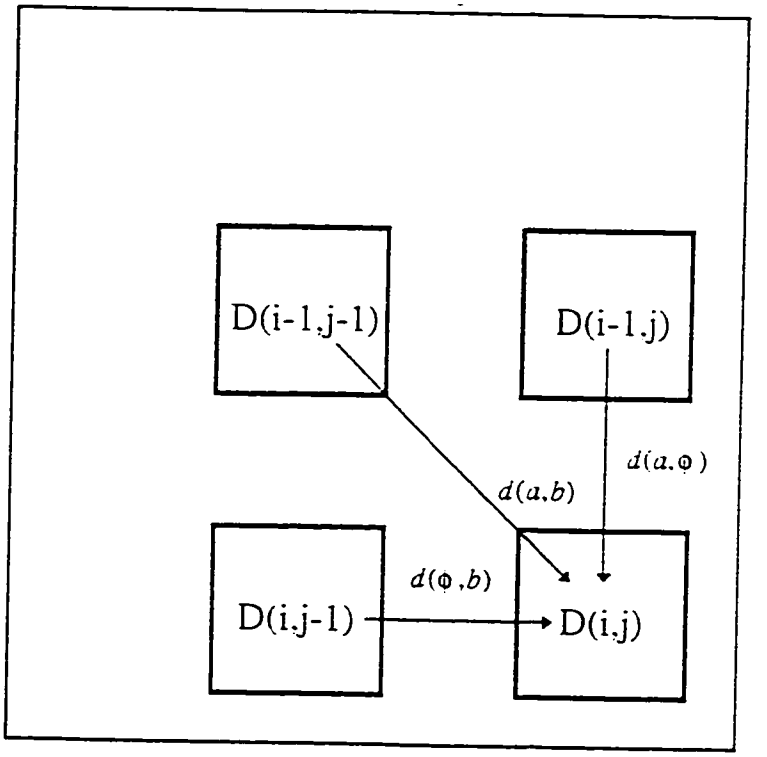


Figure 2.1: Scheme for computing the forward distance matrix.

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + d(a_i, \phi) \\ D_{i-1,j-1} + d(a_i, b_j) \\ D_{i,j-1} + d(\phi, b_j) \end{cases} \tag{2.3}$$

where the subscripts of D indicate sequence lengths counted from the beginnings and the subscripts of a and b are positions of characters.

The scheme for computing the forward distance matrix is shown in Figure 2.1. The computational complexity of the algorithm is $O(nm)$.

Algorithm W-S-B (Waterman, 1984) is an extension of the above basic method

which has taken into account multiple deletions/insertions.

The Algorithm of W-S-B:

Let x_k be the weight chosen for a multiple-deletion/insertion interval of k characters long, $k \geq 1$.

Define the states to be:

$$D_{i,0} = x_i,$$

$$D_{0,j} = x_j,$$

$$D_{0,0} = 0;$$

and

$$D_{i,j} = D(a_1 a_2 ... a_i, b_1 b_2 ... b_j).$$

Then

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + d(a_i, b_j) \\ \min_{k \geq 1}\{D_{i,j-k} + x_k\} \\ \min_{k \geq 1}\{D_{i-k,j} + x_k\} \end{cases} \tag{2.4}$$

□

The computational complexity of the algorithm for multiple deletions/insertions is $O(nm^2 + n^2 m)$ (Waterman, 1984).

Gotoh (1982) developed an improved version of dynamic programming which can handle multiple-deletions/insertions in $O(nm)$ time. In practice, the space requirement of $O(nm)$ often limited the applicability of the above methods. Various kinds of strategies were reported which might reduce the space consumption of dynamic

programming by constant factors (Taylor, 1984; Watanable et al., 1985; Altschul and Erichson, 1986; Gotoh, 1986, 1987).

2.1.3 Optimal alignment with linear space

As early as 1975, Hirschberg (1975) demonstrated in his paper how to produce an optimal conversion or alignment in $O(m_2)$ space, when m_2 is the shorter sequence length. This method had not been widely noticed until Myers and Miller developed an algorithm based on it (Myers and Miller, 1988).

Myers and Miller (1988) applied Hirschberg's technique to a concave gap penalty algorithm that subsumes Gotoh's algorithm (1982) as a special case. In their algorithm, gap cost is defined as: $\text{gap}(k) = g + hk$ for the cost of a k-symbol indel (i.e., insertion or deletion). Informally, opening up a gap costs g and each symbol in the gap costs h.

The algorithm

Let $\mathbf{a}_i$ denote the i-symbol prefix $a_1a_2...a_i$ of $\mathbf{a}$ and let $\mathbf{b}_j$ denote $b_1b_2...b_j$ of $\mathbf{b}$.

Define:

$C(i,j) =$ minimum cost of a conversion of $\mathbf{a}_i$ to $\mathbf{b}_j$,

$D(i,j) =$ minimum cost of a conversion of $\mathbf{a}_i$ to $\mathbf{b}_j$ that deletes a_i, $(i > 0)$.

$I(i,j) =$ minimum cost of a conversion of $\mathbf{a}_i$ to $\mathbf{b}_j$ that inserts b_j. $(j > 0)$.

The value of $C(i,j)$ satisfies the recurrence relations:

$$C(i,j) = \begin{cases} \min\{D(i,j), I(i,j), C(i-1,j-1) + w(a_i, b_j)\} & \text{if } i > 0 \text{ and } j > 0 \\ \text{gap}(j) & \text{if } i = 0 \text{ and } j > 0 \\ \text{gap}(i) & \text{if } i > 0 \text{ and } j = 0 \\ 0 & \text{if } i = 0 \text{ and } j = 0 \end{cases} \tag{2.5}$$

where $w(a_i, b_j)$ is the cost of replacing a_i by b_j.

The boundary conditions of D are carefully defined:

- For $j > 0$, an optimal conversion of $\mathbf{a}_0$ to $\mathbf{b}_j$ is to insert all j symbols, so $C(0,j) = \text{gap}(j)$.

- For $j > 0$, define $D(0,j) = C(0,j) + g$.

  $D(i,j)$ is then obtained:

$$D(i,j) = \begin{cases} \min\{D(i-1,j), C(i-1,j-1) + g\} + h & \text{if } i > 0 \text{ and } j > 0 \\ C(0,j) + g & \text{if } i = 0 \text{ and } j > 0 \end{cases} \tag{2.6}$$

□

- For I, similarly:

  $I(i,0) = C(i,0) + g$, $i > 0$, and ignore $I(0,j)$ for $j \leq 0$.

  Then,

$$I(i,j) = \begin{cases} \min\{I(i,j-1), C(i,j-1)+g\}+h & \text{if } i>0 \text{ and } j>0 \\ C(i,0)+g & \text{if } i=0 \text{ and } j>0 \end{cases} \tag{2.7}$$

□

It can be observed from Equations (2.5), (2.6) and (2.7) that the values in the ith rows of C and D depend only on the values in rows i and $i-1$ of C, D and I, while values in the ith row of I depend only on values in rows i of C and I. With two vectors it suffices: if CC and DD contain the $(i-1)$st rows of C and D, then the ith rows may be computed by overwriting values in the $(i-1)$st rows in a left-to-right sweep with the aid of three scalars, e, c and s:

$$CC(k) = \begin{cases} C(i,k) & \text{if } k<j \\ C(i-1,k) & \text{if } k \geq j \end{cases} \tag{2.8}$$

$$DD(k) = \begin{cases} D(i,k) & \text{if } k<j \\ D(i-1,k) & \text{if } k \geq j \end{cases} \tag{2.9}$$

$$e = I(i,j-1) \tag{2.10}$$

$$c = C(i,j-1) \tag{2.11}$$

$$s = C(i-1,j-1) \tag{2.12}$$

where i and j are the row and column numbers of the elements in C and D which are being processed.

□

When **a** and **b** are of lengths m_1 and m_2, C and D are of the size $m_1 \times m_2$. Equations (2.5), (2.6) and (2.7) indicate that C, D and I are computed in a left-to-right sweep: the m_2 elements in the first row, then the m_2 elements in the second row, ... CC and DD store the latest N elements, some of which may be in the current row and the others may be in the previous row, processed in C and D respectively.

The algorithms described in the two sections are still so far the most popular methods for two molecular sequence alignment. Although the situation for alignment of more than two sequences is quite different, most of the heuristic approaches for multiple sequence comparison use the above methods in a pairwise manner.

2.2 Heuristic Approaches for Multiple Sequence Alignment

To align multiple sequences with reasonable computer resources, a number of methods have been developed. Many of the methods are heuristic ones which attempt to find good alignments that are not necessarily optimal. Some representatives are reviewed in this section.

2.2.1 Regional approaches

(a) A subsequence-based approach:

The *subsequence-based approach* developed by Johnson and Doolittle (1986) compares three or more protein sequences by progressively aligning a group of selected segments (subsequences). In this process, only a small subset of all the possible segments from each sequence is compared, and minimum information is retained for the trace-back of the alignment. Starting at the amino terminal of the sequences, a local window is established to limit the number of elements in the segment comparison. The results of the comparison are used to align one subunit of each sequence. The window is then moved forward so that more subsequent segments are compared and the subunits following are aligned. This process continues until the carboxy terminal is reached. The time complexity of this approach is proportional to $O[n(m-W)W^{n-1}]$

(Johnson and Doolittle. 1986), where W is the window size. n the number of sequences and m the sequence length. This method is not practical for aligning a large number of sequences and is not guaranteed to be globally optimal (Johnson and Doolittle. 1986: Waterman. 1984).

(b) A repeated finding approach:

Martinez (1983) suggested a *repeated finding approach* for multiple sequence alignment. In this approach. if a region occurs in all n sequences of length m then locating the region in the n sequences can be done in a computational time of $O(mn)$. The alignment of two sequences using this method presents a list of regions which can be extended for the alignment of multiple sequences. The implemented program called MALIGN can also find near optimal alignments and provide a means for randomizing the given sequences for testing the statistical significance of an alignment (Martinez. 1983). The implementation of an optimization algorithm in this program allows it to find all the alignments which lie within a specified distance of the optimal alignment. The limitations of this method include: (1) it is unable to account for the unaligned portions between successive regions; and, (2) there is no allowance for reduced commonality.

2.2.2 Tree based approaches

In this type of approach, a tree is used to represent the taxonomical or phyletic relation among the sequences (Sankoff, 1975; Sankoff and Cedergren, 1983; Waterman and Perlwitz, 1984; Feng and Doolittle, 1987; Gusfield, 1993).

(a) A binary tree based approach:

If the sequences are related by a *binary tree*, the heuristic method by Waterman and Perlwitz (1984) is able to align n sequences of length m in computational time of $O(nm^2)$ and memory storage of $O(m^2 + mn)$. This method begins with the construction of an average sequence out of two original sequences that are related by a node of the tree. A weight based on the number of the original sequences is assigned to the constructed sequence. The sequence construction process continues until the root of the tree is reached. An overall alignment is then obtained by aligning each of the original sequences with this final weighted average sequence.

(b) Phyletic tree based approaches:

More efficient alignment of multiple sequences can be achieved when a phyletic tree relating the sequences is available. In this type of approach, phyletic trees are used to represent hypothesised ancestral relation among the sequences and the construction of such a tree presupposes alignment of the sequences. Hogeweg and Hesper (1984) suggested that the alignment of sets of sequences and the construction of phyletic trees could not be treated separately. Their points were: (1) the concept of "good

alignment" seemed meaningless without reference to ancestry; (2) the criteria for sequence alignment and tree construction were closely related, with both being based in some ways on maximal matching. They proposed an integrated method that might generate both an alignment of a set of sequences and a phyletic tree. In their method, an iterated process is used: a putative tree is used to align the sequences and the alignment obtained is used to adjust the tree.

(c) The progressive alignment method:

The *progressive alignment method* by Feng and Doolittle (1987) essentially first finds a minimum spanning tree, or in other words, first does a single-link clustering, based on edit distances. The algorithms to build the minimum spanning trees are called "myopic algorithms" because each successive decision about which link to include in the tree is made without considering the implication of that choice on possible future choices (Feng and Doolittle, 1987; Gusfield, 1993). Once a hypothetical phylogenetic tree has been established, multiple sequence alignment processed by progressively aligning groups of sequences according to the branching order in the tree.

Some widely used multiple sequence alignment programs are developed based on the progressive method, for instance CLUSTALW (Thompson et al., 1994) and CLUSTAL4 (Higgins and Sharp, 1989).

(d) A center star method:

This method was developed by Gusfield (1993) and proved as an efficient multiple

sequence alignment method with a guaranteed error bound. In this method, for a given set of k sequences $\mathcal{X}$, the center star is defined to be a star tree of k nodes. The center node X_c is the center sequence with minimized value of $\sum_{J \neq c} D(X_c, X_J)$, where D is a distance. Each of the $k-1$ nodes is for a distinct sequence in $\mathcal{X} - X_c$. Then, a multiple alignment of the string set $\mathcal{X}$ is derived from and consistent with the center star.

This method is effective when applying to multiple sequence alignment and the deviation from the optimal is less than a factor of two (Gusfield, 1993).

2.2.3 Methods of consensus

The *methods of consensus* have been studied in social sciences for a long time. It was first implemented for the purpose of multiple sequence alignment by Waterman et al (1984). And later, it was developed for practical use in this field. This approach simulates the process of "by hand" alignment of a group of related sequences. It uses a controlled iterative alignment procedure that weights the key features of a protein family and thus forces the alignment of conserved subunits. The procedure basically determines the *consensus sequences* that incorporate the characteristic features of the related sequences (Patthy, 1987). Patthy's consensus sequence method mainly includes four steps:

(1) Sequences are compared pairwise and the closely related sequences are grouped

on the basis of similarity scores.

(2) In each group, alignments of sequences are surveyed to identify the subunits conserved in most of the pairwise alignments and to determine the location of the gaps. From these data, a tentative consensus sequence is deduced for each group. The real sequences of the group are then aligned with the conserved regions. From the resulted multiple alignment, a corrected group-consensus sequence is determined.

(3) By aligning the consensus sequences of the various groups, a unified consensus sequence which characterizes the majority of sequences is produced.

(4) The sequences are aligned with the unified consensus sequence and the resultant multiple alignment is used to deduce a corrected consensus sequence.

2.2.4 The random graph-based hierarchical sequence synthesis approach

The random graph-based hierarchical sequence synthesis approach by Chan (Chan, 1990; Chan and Wong, 1991) has successfully used the random graph approach (You, 1983; Wong and You, 1985; Wong et al, 1990) for aligning multiple molecular sequences. The algorithm of *Hierarchical Sequence Synthesis Procedure* synthesizes a set of attributed graphs which represent the ensemble of sequences and adopts the dynamic programming method for multisequences alignment.

According to Chan, the random graph synthesis approach returns the grand syn-

thesis of an ensemble of patterns that were finite-length sequences. It applies a hierarchical clustering of the sequences in such a way that each sequence forms a group of its own at the lowest level and the groups are then successively fused into larger ones and finally into a single one at the highest level through a sorting strategy that determines how the hierarchy of groups is constructed. At each level of the hierarchy, the synthesis of a group of sequences is obtained by identifying the one-to-one correspondence among the subunits of the sequence such that the probability of the subunits occurring at the corresponding site of the sequences can be inferred. The one-to-one correspondences form the alignment of the sequences as well as the primary structure of the synthesis, while the probability of the symbols' occurrence at the corresponding sites of the sequences form the context of the synthesis. This procedure has been applied to the problems of constructing phylogenetic tree, and of analyzing structure and function of molecular sequences.

2.2.5 The dot matrix method

The dot matrix method (Maizel and Lenk, 1981; and Novotny, 1982) was an important method of multiple sequence alignment in the early days, which was described as "just look at it". It was a widely used visual method that generally utilizes computers. In this method, a matrix $M = (m_{ij})$ is formed where $m_{ij} = 0$ if the ith element of sequence **a** is unequal to the jth element of sequence **b** and $m_{ij} = 1$ otherwise. Runs

of exact matches show up as diagonals of 1s. The dot matrix method is useful in locating regions of high match between two DNA sequences. It is natural to ask for the probability distribution of the longest match between two sequences. This distribution allows the analyst to locate the significant matches.

2.3 Genetic Algorithms and Applications

In the following, some of the most significant or the most recent research achievements in the theory and applications of genetic algorithms are reviewed. The technical details of genetic algorithms are discussed in Chapter 3.

2.3.1 Genetic algorithms

Genetic algorithms originated from the studies of cellular automata, conducted by John Holland and his colleagues at the University of Michigan. Holland's book, published in 1975, is generally acknowledged as the beginning of genetic algorithm research.

Since the pioneering work of Holland, research has been conducted to develop genetic algorithms in both theory and applications. Important work includes that by Bethke (1981), Booker (1985, 1987), Grefenstette and Fitzpatrick (1985), Grefenstette (1986), Holland (1987) and Goldberg (1987). In recent years, Grefenstette and Baker (1989) addressed genetic search behavior from the viewpoint of implicit parallelism. Goldberg (1989b) observed that choosing a suitable population size was a key decision faced by all users of genetic algorithms and he proposed an approach for calculating suitable population sizes. Buckles and his colleagues (1990) examined schema survival rates in genetic algorithms and proposed a crossover rule that took advantage of the knowledge of survival rates.

2.3.2 Applications of genetic algorithms

Genetic algorithms have been applied to machine learning, image analysis, neural networks, control, intelligent systems, and many other fields. Some of the applications are reviewed in this section and the applications in bioscience will be reviewed in greater details in the following section.

Intelligent systems and machine learning

Holland (1986) applied genetic algorithms to build rule-based intelligent systems. According to Hollland, the major tasks for machine learning algorithms are apportionment of credit and rule discovery. The latter depends on the discovery of good "building blocks" for generating plausible rules. Genetic algorithms, using fitnesses, may offer effective ways of discovering good building blocks.

Genetic algorithms were applied to a symbolic learning task which was supervised concept learning from examples (Spears and De Jong, 1990). In the learning task, a genetic algorithm concept learner was developed that could learn a concept from a set of positive and negative examples. The learner was then run in a batch-incremental concept learner. Concepts were represented as subsets of points in an n-dimensional feature space which was defined apriori and for which all the legal values of the features were known. While the concept learning program was presented with both a description of the feature space and a set of correctly classified examples of the concepts, it was expected to generate a reasonably accurate description of the (un-

known) concepts. By using an *incremental model* to evaluate the performance of the concept learning algorithms. better performance was achieved on the target concepts of varying complexity.

De Jong (1988) suggested that genetic algorithms could be used as an effective learning approach. According to him. genetic algorithms might provide a set of efficient domain-independent search heuristics which were a significant improvement over traditional "weak methods" without the need for incorporating highly domain-specific knowledge.

Control systems and robotics

Genetic algorithms have been applied to optimize complex control systems. A major task in optimizing a complex system is to tune various parameters of control algorithms for efficiency. Grefenstette (1986) used genetic algorithms to tune control parameters for optimizing a wide variety of complex systems.

Schultz and his colleagues (Schultz et al. 1993) used a genetic algorithm based machine learning technique to evaluate autonomous-vehicle software controllers. In this technique, a set of simulated fault scenarios was applied to a controller. and a genetic algorithm searched for significant combinations of faults.

Davidor (1990) developed a genetic algorithm based approach for robotic control. In robotic control, the intrinsic characteristics of trajectory generation are often too complex for many conventional optimization algorithms. The genetic algorithm

developed by Davidor was capable of specifying near optimum trajectories.

Neural networks

Genetic algorithms have been used to reinforce learning with multilayer neural networks. The research by Whitley and his colleagues (Whitley et al. 1991) indicated that a genetic hill-climber, with a strong reliance on mutation rather than hyperplane sampling, might produce good performance for neural network weight optimization. Janson and Frenzel (1993) applied genetic algorithms in training product unit neural networks. Product unit neural networks are useful tools because they can handle higher order combinations of inputs. Zhang (1994a) used genetic algorithms in training image processing neural networks to avoid local extrema and to speed up training processes.

Other applications

Ankenbrandt and his colleagues (1990) developed a model of genetic algorithms with semantic nets in which the relationships between concepts was depicted as semantic nets. This model was used to identify the oceanic features in remote sensing imagery.

In 1991, Salay and Wong (Salay 1991; Salay and Wong, 1991) applied genetic algorithm techniques to an NP-complete combinatoric optimization problem known as the *largest common subgraph problem* (LCS) for unattributed graphs. In this approach, the LCS problem was first recast into a form suitable for a genetic algorithm

method. Then several genetic algorithms were developed that incorporated different degrees of heuristic information about LCS. The algorithms were tested using a large test suite of randomly generated problems with known solutions. The performance was then compared and the optimal genetic algorithms among them were identified.

2.4 Genetic Algorithms in Biosciences

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, 1993a) and predicting RNA secondary structures (Gultyaev, Batenburg and Pleij, 1995; Batenburg, Gultyaev and Pleij, 1995). In molecular sequence analysis, a genetic algorithm was developed for two sequence alignment (Zhang, 1994b), which was the initial stage of the research reported in this thesis. This work was the first time a genetic algorithm was successfully applied to molecular sequence alignment.

2.4.1 A genetic algorithm for RNA secondary structure simulation

A conventional approach for predicting RNA secondary structures is the minimal-energy approach (Jaeger et al., 1989; Jacobson and Zuker, 1993). The problems with this approach are that the user is given a number of alternative solutions without a guideline on how to choose the best, and that some essential features of RNA folding are not taken into account. Another commonly used approach is the stepwise simulation of RNA folding (Abrahams et al., 1990; Gultyaev, 1991). In this approach, the RNA double helical stems are used as the primary unit of folding. A weakness of

this approach is that it considers the folding process as an irreversible pathway where new stems are added step by step until the final structure is reached. Once a stem is formed there is no way back. However, some of the stems should be formed only temporarily.

A genetic algorithm was developed which might overcome the shortcomings of the conventional approaches (Batenburg et al., 1995). The genetic algorithm serves not only as a method for finding a particular solution, but also a convenient tool for investigating the possible models of folding pathways. The genetic algorithm developed by Batenburg et al. consisted of the following basic steps:

1. Compute all the possible stems which were determined by base pairing in the RNA structure.

2. Generate a population of several possible solutions randomly. Each solution was a "mask" of the numbers "0" or "1". A "0" indicated that the stem was not included in the solution and a "1" indicated the stem was chosen.

3. Compute the "fitness" of each solution. The fitness was calculated by summing the lengths of all stems in the solution, resulting in a number which represented the total number of pairs.

4. Perform reproduction by randomly choosing strings to form a new generation of the same size out of the previous generation. The random selection heavily favored the fittest solutions.

5. Perform mutation by randomly changing string elements, i.e., some 1s into 0s and vice versa.

6. Perform crossover by randomly organizing the solutions into pairs, randomly choosing crossover positions, and then the bit-configurations between two crossover positions were interchanged between the pairs.

7. Repeat all the steps from 3 through 7 for a number of iterations. The program stopped after the maximum fitness did not change for a certain number of iterations.

To obtain better results, improvements were made to the above basic genetic algorithm:

- Fitness criterion modification:

  A new criterion is used that is the sum of stem stacking energies. Such an improvement may steer the search to a better direction of the prediction.

- Incompatibility modification:

  Instead of choosing a random stem out of each set of incompatibles, the "best" one was chosen. The best one was defined as the stem which attributed most to the fitness. In the runs which used stem-length as fitness, this was the longest stem. In the runs with stacking-energy, this was the stem with the highest energy value.

- The growth modification:

  In this modification the genetic algorithm was initially restricted to a small part of the RNA sequence. After each iteration the size of this part was increased. This resulted in the creation of intermediate stems which would compete with the alterative stems later as soon as they emerged in due course of the simulated "growth" process.

Results demonstrated the possibility to simulate RNA folding process by a genetic algorithm and the flexibility of the algorithm allowed for introduction of different models of folding kinetics.

2.4.2 A genetic algorithm for protein folding simulation

Computing the functional conformation of a protein molecule from the amino acid sequence is very difficult. The major problems are (1) the contributions to the free energy that stabilize the folded conformation are poorly understood (Dill, 1990); and (2) the space of possible conformations is too large and complex to search (Levinthal, 1968).

Unger and Moult (1993a) designed a genetic algorithm to effectively address the second problem and to find the functional protein conformations. This application of genetic algorithms was regarded as an extension of the widely used Monte Carlo (MC) methods (Metropolis, et al., 1953).

The simplified model:

A simple model was used for protein folding simulation which still captured the essence of the important components of protein folding (Unger and Moult, 1993b). In this model, the linear sequence was composed of "amino-acids" of only two types: hydrophobic and hydrophilic. This sequence was "folded" on a two-dimensional square lattice on which at each point the chain could turn 90 degrees left or right, or continue ahead. The energy function was simple: -1 for each direct contact of non-bonded hydrophobic-hydrophobic amino acids.

The low energy conformations were compact with a hydrophobic core: the hydrophobic-hydrophobic iterations were rewarded and the hydrophobic residues tend to be on the inside of a low energy structure, while the hydrophilic residues were forced to the surface. Each residue could participate in only two contacts at most, but three contacts for each terminal residue.

The genetic algorithm for protein folding simulation

1. The process started with N extended linear sequence structures which contained amino acids of the two types (hydrophbic and hydrophilic). In each generation, each structure was subject to a number of mutation steps.

2. Each mutation step was the same as a single MC step and was subject to similar acceptance criteria as in an MC process.

3. At the end of this MC stage the crossover operation was performed.

The probability for a structure to be selected for crossover, $p(S_i)$, was proportional to its energy value E_i:

$$p(S_i) = \frac{E_i}{\sum_{j=1}^{N} E_j} \tag{2.13}$$

The lower energy conformations had a better chance of being selected. For a pair of selected structures, a random point was chosen along the sequence. The N-terminal portion of the first structure was connected to the C-terminal portion of the second structure.

4. Once a valid structure S_k was created, its energy E_k was evaluated and compared with the averaged energy $E_{ij} = (E_i + E_j)/2$ of its "parents". The structure was accepted if $E_k \leq E_{ij}$, or if the energy would be increased based on the decision:

$$Rnd < \exp\left[\frac{E_k - E_{ij}}{c_k}\right]$$

where Rnd was a random number between 0 and 1, and c_k was linearly decreased during the simulation to achieve convergence (i.e., $c_k = \alpha c_{k-1}$, and α was a constant smaller than but close to 1).

5. This crossover operation was repeated until $N - 1$ newly accepted hybrid structures had been constructed to constitute the population of the next generation. The lowest energy conformation in each generation was directly replicated to the next generation.

The genetic algorithm was not significantly more costly per step than the regular Monte Carlo method: most of the genetic algorithm steps were mutations which were the same as the regular MC steps, and a crossover was not much more expensive.

The problem of protein folding, at least on a lattice, is a member of the class of NP-complete problems, and therefore there exists no general search algorithm that can guarantee to find the global free energy minimum for real proteins (Unger and Moult, 1993a). The real folding process may thus end up in a functional conformation that is not the global minimum of free energy. The genetic algorithms may be to mimic the folding pathway rather than conducting a hopeless brute force search for the global minimum.

2.5 Summary

Dynamic programming can generate the optimal solution when aligning two sequences. A linear space dynamic programming method reduces space complexity.

The heuristic approaches for multiple sequence alignment include the regional approaches, the tree-based approaches, the consensus approaches, the random graph based approach, and others.

Genetic algorithms were initially developed in 1970's. Genetic algorithms have been used in machine learning, control systems, robotics, neural networks, and so on.

In bioscience, genetic algorithms have been used for RNA secondary structure simulation, protein folding simulation, as well as other studies.

Chapter 3.

An Overview of Genetic Algorithms

A genetic algorithm is the core part of the multiple sequence alignment approaches developed in this research. In this chapter, the fundamentals of genetic algorithms, the important concepts, the operators, and the computational complexity are discussed. Attention is also given to an important issue in genetic algorithms, i.e. population size selection.

3.1 Fundamentals of Genetic Algorithms

Genetic algorithms are a class of stochastic algorithms for efficient and robust search. These algorithms are developed in a way to imitate biological evolutionary process and genetic operations on chromosomes. They are designed to satisfy the four basic evolutionary conditions: (1) the ability for an individual to reproduce itself; (2) the existence of a self-reproducing population; (3) the existence of variety among the individuals; and (4) the survival ability associated with this variety.

In a search process, a genetic algorithm starts with a population of points (states) in the problem space. In each step of search, good points survive while poor points are eliminated. A new generation of points is generated out of good points. The new generation is usually better than the old one in terms of search objectives. The search process continues until a termination condition is satisfied.

Compared with the enumerative and random search methods, such as width-first search and blind search, genetic algorithms are more efficient. Generating "better" points avoids searching the points which are "worse" than the current ones. Compared with the calculus-based methods, such as hill climbing, genetic algorithms are more robust. Searching simultaneously with a population of points, instead of a single one, minimizes the chances of falling into local extrema.

The major references included in this chapter are: Goldberg (1989a, 1989b), Rawlins (1991) and Buckles and Petry (1992).

3.2 Components and Operators of a Genetic Algorithm

The major components of a genetic algorithm include a string representation of solution points in the problem space, an objective (or fitness) function and three operators: reproduction, crossover, and mutation.

The *objective function* can be represented as

$$f : \Omega \longrightarrow R \tag{3.1}$$

where Ω is the problem space and R is a set of real numbers. The function is used to evaluate the fitness of the points in Ω. In practical use, it is of the form

$$f(\omega) = r \tag{3.2}$$

where $\omega \in \Omega$ and $r \in R$.

For applying a genetic algorithm, each point (state) ω in the problem space is represented as a string A:

$$A = a_1 a_2 ... a_l \tag{3.3}$$

where a_i $(1 \leq i \leq l)$ are *string elements* and l is the length of strings. The elements are taken from a domain $\mathcal{D}$ which is a collection of symbols, i.e. $a_i \in \mathcal{D}$. A one-to-one *encoding function* g is needed to convert the points into strings and its inverse is needed to convert the strings back to points:

$$A = g(\omega) \tag{3.4}$$

and

$$\omega = g^{-1}(A). \tag{3.5}$$

Each search step is composed of three genetic operations:

(a) Reproduction.

(b) Crossover, and

(c) Mutation.

The order of applying crossover and mutation may vary with different applications. The three operators are discussed below. The term *population* here refers to the set of strings on which the three operations are performed.

Reproduction

Given a population of strings (which represent points), the operation of *reproduction* duplicates the strings of high fitness values. The reproduction is conducted according to the rule that for a string, the higher its fitness value, the higher the probability that it may be duplicated. After an operation of reproduction, the strings of high fitness values may be duplicated one or many times, whereas the strings of low fitness values may have a smaller chance to be duplicated. For string A_i, the number of *times* to duplicate it is calculated as

$$\text{round}(Q \cdot pr_i) \tag{3.6}$$

where Q is the population size which is defined as the total number of strings in the

population and pr_i is the probability for string A_i to be duplicated:

$$pr_i = \frac{f(g^{-1}(A_i))}{\sum_{j=1}^{Q} f(g^{-1}(A_j))} \tag{3.7}$$

where f is an objective (or fitness) function, $\sum_{j=1}^{Q} f(g^{-1}(A_j))$ is the total fitness of the population, and g is a string encoding function defined in (3.4) and (3.5). The duplicated strings are placed in a *mating pool* for creating a new generation of strings by a crossover operation.

Crossover

The operation of *crossover* is conducted on the strings in the mating pool. A crossover operation creates two new individual strings out of two strings in the pool. Two major steps are included in this operation:

1. String pairs are selected at random from the mating pool. A crossover rate p^c is used to determine the number of pairs to mate.

2. For each string pair to be mated, a number k is randomly selected within the range between 1 and $l-1$. Each string is cut into two substrings (head and tail) at the location between the kth and $k+1$th elements. Then two new strings are created by exchanging the two strings' heads (or tails).

The new offsprings created from the above operation inherit certain information from each of their parents, although they are usually different from their parent strings

and different from each other.

Mutation

A *mutation* operation is the alternation of the value of a string element. In a mutation process, a string is randomly selected from the mating pool and a *mutation position* is selected between position 1 and position l. The element value at the mutation position is changed. The frequency of the mutation operation is controlled by a *mutation probability* p^m. For an application, usually a very small value of p^m, for example, 0.1%, of the total number of string elements in the generation, is used. In each step of search, the number of string elements which should be alternated is determined by multiplying the predefined p^m and the number of elements in the population. A mutation position can be selected randomly or based on a selection condition. Mutation is especially needed when a small modification on a string may greatly improve the fitness of the strings generated.

The process of creating a new generation is one in which the three operators are applied sequentially. As mentioned before, in different applications, the order may be different.

When a certain termination criterion is satisfied, a genetic algorithm stops producing new generations. A termination condition may be: (1) a predefined "maximum" number of generations has been reached; (2) the population has converged to a single

string; or, (3) the population has certain characteristics, for example, the maximum fitness value remains unchanged for a certain number of generations.

3.3 The Law of String Growth

In this section, the growth of good strings is examined in detail. The discussion enables us to understand what kind of strings may survive and what kind may die in the creation of new generations of strings. This knowledge is useful in designing a genetic algorithm.

An important concept in genetic algorithms is *schema* (plural, *schemata*). A schema is a similarity template describing a subset of strings with similar elements at certain string positions. Schemata are represented by the symbols in D plus symbol $*$ which represents *don't care*:

$$\mathcal{D} \cup \{*\}. \tag{3.8}$$

The strings described by a schema are called its *instances*. In a schema, the positions where the elements are symbols in $\mathcal{D}$ are called *fixed positions*.

For example, if $\mathcal{D}$ is the set of binary digits, i.e.

$$\mathrm{D} = \{0, 1\} \tag{3.9}$$

schemata on $\mathcal{D}$ are represented by using 0, 1, and $*$. "1 1 1 1 $*$" is a schema on $\mathcal{D}$ which describes all the strings in which elements at the first four positions are 1's and the element at the fifth position may be 0 or 1. In other words, strings "11111" and "11110" are instances of the schema. Positions 1, 2, 3 and 4 are fixed positions in the above schema but position 5 is not.

Schemata have two important properties: *order* and *defining length*. The order of

a 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 the last fixed position. For example, the order of the above sample schema is 4, and the defining length is 3. The part between the first and the last fixed position is called the *defining segment.*

As discussed in the previous section, a search step of a genetic algorithm consists of the three operations of reproduction, crossover and mutation. Let us consider the individual and then the combined effects of these operations on instances of a schema.

Reproduction

It is assumed that at a given step τ (number of generations) there are m instances of a particular schema H contained within the population. We write $m = m(H, \tau)$. From Equations (3.6) and (3.7), we have the reproductive schema growth equation for schema H:

$$m(H, \tau + 1) = m(H, \tau) \cdot Q \cdot \frac{f(H)}{\sum_{j=1}^{Q} f(g^{-1}(A_j))} \tag{3.10}$$

where $f(H)$ is the average fitness of the instances of H. If we write the average fitness of the population as

$$\bar{f} = \frac{\sum_{j=1}^{Q} f(g^{-1}(A_j))}{Q} \tag{3.11}$$

we may rewrite Equation (3.10) as

$$m(H, \tau + 1) = m(H, \tau) \cdot \frac{f(H)}{\bar{f}}. \tag{3.12}$$

Equation (3.12) indicates that when reproduction is considered alone, the growth rate of the instances of a particular schema is proportional to the ratio of the average fitness of the schema to the average fitness of the entire population. Schemata with fitness values above the population average will contribute an increasing number of instances in the new generation, while schemata with fitness values below the population average will contribute a decreasing number of instances. $f(H)/\bar{f}$ may be used to form the reproduction probability for H, which is denoted as p_H^r.

$$p_H^r = \begin{cases} 1 & \text{if } f(H)/\bar{f} \geq 1 \\ f(H)/\bar{f} & \text{otherwise} \end{cases} \tag{3.13}$$

When $f(H) = k\bar{f}$ ($k > 0$), i.e., schema H has k times the average fitness, an instance of H will have k times the likelihood of being selected as a parent for the subsequent mating.

Assuming in the process of search, a particular schema H remains above the average fitness an amount $c\bar{f}$ where c is a positive constant. Equation (3.12) can be rewritten as

$$m(H, \tau + 1) = m(H, \tau)\frac{(\bar{f} + c\bar{f})}{\bar{f}} = (1 + c) \cdot m(H, \tau). \tag{3.14}$$

From Equation (3.14), we obtain the growth equation in terms of $m(H, 0)$, and the number of instances at $\tau = 0$:

$$m(H, \tau) = m(H, 0) \cdot (1 + c)^{\tau}. \tag{3.15}$$

Equation (3.15) indicates that reproduction allows schemata with fitness values above the population average to increase exponentially. It can be shown that schemata with fitness values below the population average decrease exponentially.

Crossover

In a crossover operation, a random number is selected for cutting the strings. When a crossover operation is performed on a schema, the schema is said to have been destroyed if the cutting occurs within the defining section, otherwise the schema is said to survive. The probability for a particular schema H to be destroyed is proportional to the ratio of its defining length to the schema length

$$p_H^d = \frac{\delta(H)}{(l-1)}. \tag{3.16}$$

Thus the survival probability is

$$p_H^s = 1 - \frac{\delta(H)}{(l-1)}. \tag{3.17}$$

If crossover is conducted by random choice, say with probability p^c at a particular mating, the survival probability may be given as

$$p_H^s = 1 - p^c \cdot \frac{\delta(H)}{(l-1)} \geq 1 - \frac{\delta(H)}{(l-1)}. \tag{3.18}$$

Thus when the effect of crossover is considered alone, we have

$$m(H, \tau+1) = m(H, \tau) \cdot p_H^s \geq m(H, \tau) \cdot [1 - \frac{\delta(H)}{(l-1)}]. \tag{3.19}$$

Mutation

Mutation is the alteration of a single element on a string with probability p^m. In order for a schema to survive, all of its positions must themselves survive. Therefore, since a single position survives with probability $(1-p^m)$, and since each of the mutations is statistically independent, a particular schema survives when each of the $o(H)$ fixed positions within the schema survives. Multiplying the survival probability by itself $o(H)$ times, we have the probability of surviving mutation, $(1-p^m)^{o(H)}$. For small values of p^m ($p^m \ll 1$), the schema survival probability may be approximated by $(1-p^m o(H))$.

The combined effects

The combined effect of reproduction and crossover can be expressed in the following equation, assuming that reproduction and crossover are independent.

$$\begin{aligned} m(H,\tau+1) &= m(H,\tau)\cdot\left\{\frac{f(H)}{\bar{f}}\cdot\left[1-p^c\cdot\frac{\delta(H)}{l-1}\right]\right\} \\ &\geq m(H,\tau)\cdot\left\{\frac{f(H)}{\bar{f}}\cdot\left[1-\frac{\delta(H)}{l-1}\right]\right\}. \end{aligned} \tag{3.20}$$

The expected number of instances of a particular schema in the next generation under reproduction, crossover, and mutation can be given by

$$\begin{aligned} m(H,\tau+1) &\approx m(H,\tau)\cdot\left\{\frac{f(H)}{\bar{f}}\cdot\left[1-p^c\cdot\frac{\delta(H)}{l-1}\right]\cdot[1-p^m o(H)]\right\} \\ &\geq m(H,\tau)\cdot\left\{\frac{f(H)}{\bar{f}}\cdot\left[1-p^c\cdot\frac{\delta(H)}{l-1}-p^m\cdot o(H)\right]\right\} \end{aligned} \tag{3.21}$$

where p^m is the mutation probability.

Equation (3.21) 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.

3.4 Complexity of Genetic Algorithms

Binary genetic algorithms:

In a binary genetic algorithm, a string element may take the value 0 or 1. Let τ denote the number of generations and P_i represent the proportion of string elements set to the value 1 at time $\tau = i$, for a particular element position j; and P_0 represent P at $\tau = 0$. Let f_1 represent the fitness of all strings with element value 1 at a particular position j; and f_0 represent the fitness of all strings with element value 0 at position j. Let r represent the fitness ratio, i.e. $r = f_1/f_0$ and assume that r is constant over time.

From Equation (3.7), we obtain:

$$P_{\tau-1} = \frac{f_1 \cdot P_\tau}{f_1 \cdot P_\tau + [1 - P_\tau] \cdot f_0} \tag{3.22}$$

or

$$P_{\tau-1} = \frac{r \cdot P_{\tau+0}}{1 + (r-1)P_{\tau+0}} \tag{3.23}$$

Using the induction of Rawlins (1991), we then have:

$$P_\tau = \frac{r^\tau \cdot P_0}{r^\tau \cdot P_0 + (1 - P_0)} \tag{3.24}$$

Nonbinary genetic algorithms:

In a non-binary algorithm, the number of possible element values is more than two. Let the fitness ratio r denote the fitness of those strings with elements set to a specified

element value at a particular element position over the fitness of the remaining strings.

Let $P_{i,a}$ represent the proportion of elements set to value a, at time $\tau = i$ and a particular element position j. Let P_0 represent P at $\tau = 0$, and f_a represent the average fitness of all the strings with element value a at a particular position j. Subscript notation is used to indicate the fitness of the strings with other legal elements at position j.

Then, similarly from Equation (3.7) we have:

$$P_{\tau-1,a} = \frac{f_a \cdot P_{\tau,a}}{f_a \cdot P_{\tau,a} + f_b \cdot P_{\tau,b} + f_c \cdot P_{\tau,c} + \ldots} \tag{3.25}$$

Note that the sum of P over all possible element values in $(a, b, \ldots)$ at position j must be equal to one. That is:

$$1 = P_{\tau,a} + P_{\tau,b} + P_{\tau,c} + \ldots \tag{3.26}$$

And therefore:

$$1 - P_{\tau,a} = P_{\tau,b} + P_{\tau,c} + \ldots \tag{3.27}$$

Given that the fitness function is non-negative, and P is defined in the interval $[0, 1]$, there exists an f whose value is in the interval $[f_{min}, f_{max}]$, such that Equation (3.28) holds. f_{min} and f_{max} are the minimum and maximum values of $f_b, f_c, \ldots$, respectively.

$$f[P_{\tau,b} + P_{\tau,c} + \ldots] = f_b P_{\tau,b} + f_c P_{\tau,c} + \ldots \tag{3.28}$$

Let r represent the fitness ratio in the non-binary case. i.e., $r = f_a/f$. Assume that r is constant over time. Combining Equation (3.25) and Equation (3.28) we have:

$$P_{\tau+1,a} = \frac{f_a \cdot P_{\tau,a}}{f_a \cdot P_{\tau,a} + f \cdot [P_{\tau,b} + P_{\tau,c} + ...]}. \tag{3.29}$$

Further we have:

$$P_{\tau+1,a} = \frac{f_a \cdot P_{\tau,a}}{f_a \cdot P_{\tau,a} + f \cdot [1 - P_{\tau,a}]}. \tag{3.30}$$

And then:

$$P_{\tau+1,a} = \frac{r \cdot P_{\tau,a}}{1 + (r-1) \cdot P_{\tau,a}}. \tag{3.31}$$

The recurrence solution above can then be solved by a linear differential equation approach (Ankenbrandt, 1991). A result similar to Equation (3.24) is then obtained as:

$$P_{\tau,a} = \frac{r^\tau \cdot P_{0,a}}{r^\tau P_{0,a} + (1 - P_{0,a})}. \tag{3.32}$$

The time to convergence for binary and non-binary genetic algorithms

Let τ_c be the value of τ when the system reaches convergence. At $\tau = \tau_c$, some element position j was the last one to converge. So if τ_c is solved for the slowest

individual element, we obtain the convergence time τ_c for the entire system. For simplicity, the element value a is dropped from P. Equation (3.32) then becomes:

$$P_\tau = \frac{r^\tau \cdot P_0}{r^\tau P_0 + (1 - P_0)}. \tag{3.33}$$

Let P_f be the value of P at convergence. We then have

$$P_f = \frac{r^{\tau_c} \cdot P_0}{r^{\tau_c} P_0 + (1 - P_0)} \tag{3.34}$$

and then

$$r^{\tau_c} = \frac{P_f(1 - P_0)}{P_0(1 - P_f)}. \tag{3.35}$$

τ_c can be obtained as:

$$\tau_c = \frac{\log[\frac{P_f(1-P_0)}{P_0(1-P_f)}]}{\log r}. \tag{3.36}$$

Probabilistic worst case analysis

Let m represent the length of the strings, and Q represent the size of the population. Assume no mutation is conducted and at least one string in the population has the element in consideration set to the value a, if the population is to converge to a. The minimum value for P_0 is $1/Q$, which represents the worst case.

Let convergence be defined with a tolerance of γ, where the population is said to have converged to a for this element position when $P_f = 1 - \gamma$. An estimate for γ can be $1/Q$. So that,

$$\tau_c = \frac{\log[\frac{(1-\gamma)(1-1/Q)}{(1/Q(1-(1-\gamma))}]}{\log r} \tag{3.37}$$

and further.

$$\tau_c = \frac{\log(Q-1)^2}{\log r}. \tag{3.38}$$

Equation (3.38) estimates the number of generations for a population to converge in the worst case.

In a generation, the processing time is proportional to the number of strings, i.e., the population size Q: $t \propto Q$. From Equation (3.38), we can obtain $\tau_c \propto \log Q$. We can then state the worst case complexity of the entire genetic algorithm as

$$O(Q \log Q). \tag{3.39}$$

3.5 Selection of Suitable Population Sizes

Choosing suitable population sizes for a genetic algorithm is a fundamental issue for the algorithm designer (Goldberg, 1989b). A genetic algorithm generally performs poorly with a very small population because the population provides an insufficient sample size for most hyperplanes and thus, the risk of falling into a local maximum becomes rather high. Since a large population is more likely to have representatives from a large number of hyperplanes, the genetic algorithm can perform a more informed search. Further, a large population discourages premature convergence to sub-optimal solutions. However, if it is too large, the computation might be very costly since it requires more evaluations per generation. It will result in long processing time for significant quality improvement (Goldberg, 1989b).

Expected number of schemata and the basic schema function

In selecting suitable population sizes, one first needs to understand the relationship between the expected number of schemata contained in a population and the population size. In counting the expected number of unique schemata in a population, consider the probability of having a particular schema of order i (the number of fixed position) i in a population of size Q when bit positions are equally likely. The probability of a single match may be calculated as:

$$p(\text{single matches of an order i schema}) = \left(\frac{1}{2}\right)^i. \tag{3.40}$$

The chance of having no matches of a single order i schema in a population of size Q, or simply the product of the Q failure probabilities, may be calculated as:

$$p(\text{no matches of an order i schema}) = \left[1 - (\frac{1}{2})^i\right]^Q. \tag{3.41}$$

The probability of one or more successes may then be calculated as:

$$p(\text{at least one success}) = 1 - \left[1 - (\frac{1}{2})^i\right]^Q. \tag{3.42}$$

Over i fixed positions there are 2^i such schemata and over a string of length l there are $\begin{pmatrix} l \\ i \end{pmatrix}$ sets of fixed positions. Thus, the *expected number of schemata* in a population of size Q over the strings of length l may be calculated as the following sum:

$$S(Q, l) = \sum_{i=0}^{l} \begin{pmatrix} l \\ i \end{pmatrix} 2^i \{1 - [1 - (\frac{1}{2})^i]^Q\}. \tag{3.43}$$

This function is the *schema function*. For population size $Q = 1$, the schema function always has a value of 2^l, because a single string is itself a representative of 2^l different schemata.

Applying the binomial theorem, Equation (3.43) becomes

$$S(Q, l) = 3^l - \sum_{i=0}^{l} \begin{pmatrix} l \\ i \end{pmatrix} 2^i [1 - (\frac{1}{2})^i]^Q. \tag{3.44}$$

Applying the binomial theorem again to Equation (3.44) and expanding the term $[1 - (\frac{1}{2})^i]^Q$ produce

$$S(Q,l) = 3^l - \sum_{i=0}^{l} \binom{l}{i} 2^i \sum_{j=0}^{Q} i \binom{Q}{j} (-1)^j (\frac{1}{2})^{ij}. \tag{3.45}$$

Reversing the order of summation and again applying the binomial theorem, one may place the schema function in a Q-form (as opposed to the previous l-form):

$$S(Q,l) = 3^l - \sum_{j=0}^{Q} \binom{Q}{j} (-1)^j \left[1 + (\frac{1}{2})^{j-1}\right]^l. \tag{3.46}$$

For small Q values, the schema function may be well approximated by the first two terms of this series:

$$S(Q,l) \approx 3^l - 3^l + Q2^l = Q2^l. \tag{3.47}$$

This corresponds to the well-known bound on the number of schemata contained in a population of a given size and length (Goldberg, 1989a; Holland, 1975).

Equation (3.46) and (3.47) indicate that the number of schemata contained in a population of strings is a function of the population size. The larger the population size, the more the schemata which may be contained, and the larger the probability that good strings may be contained. Therefore, for a search task, a large population may increase the probability of finding a near-optimal solution. However, there is a tradeoff factor to be considered here, that is, the computing time.

Computing time on serial and parallel machines

As Equation (3.39) indicates, the computing time required for a population to converge is dependent on two factors: The number of generations to convergence and the computing time per generation.

As we have obtained from Equation (3.38), the number of generation is proportional to $\log Q$, i.e.

$$\tau_c \propto \log Q. \tag{3.48}$$

Or, for a given task,

$$\tau_c = \eta' \log Q \tag{3.49}$$

where $\eta' > 0$ is a constant.

According to Goldberg (1989a), the computing time per generation can be estimated as

$$t = \eta'' Q^{1-\beta} \tag{3.50}$$

where β is the degree of parallelism and $\eta'' > 0$ is a constant. $\beta = 0$ for a serial machine and $\beta = 1$ for an ideal parallel machine. Therefore, the total computing time for a serial machine is

$$\begin{aligned} \Delta t &= t\tau_c \\ &= \eta'' Q \eta' \log Q \\ &= \eta Q \log Q. \end{aligned} \tag{3.51}$$

where $\eta = \eta'\eta''$.

The total computing time for an ideal parallel machine is

$$\begin{aligned} \Delta t &= t\tau_c \\ &= \eta''\eta' \log Q \\ &= \eta \log Q. \end{aligned} \tag{3.52}$$

Equation (3.51) suggests that on a serial machine Δt is proportional to $Q \log Q$. Increasing Q will greatly increase the total computing time. Therefore, for a serial implementation of a genetic algorithm we may have to choose relatively small population sizes.

Equation (3.52) suggests that on a highly parallel machine Δt is proportional to $\log Q$. Increasing Q does not have a considerable impact to the total computing time compared with that of a serial machine, and therefore we can select relatively large population sizes for a parallel implementation of a genetic algorithm.

3.6 Summary

The major components of a genetic algorithm include a string representation of solution points in the problem space, an objective (or fitness) function and three operators: reproduction, crossover, and mutation.

A genetic algorithm searches with a population of solution points. In a search step, reproduction, mutation, and crossover are performed on the strings in the population. The operations enable good strings to be duplicated and combined into better ones.

The time complexity of a genetic algorithm is a function of the population size. Studies have indicated that for a serial implementation of a genetic algorithm, the population size could be relatively small, while for a parallel implementation of a genetic algorithm, the population size should be relatively large.

Chapter 4

The Genetic Algorithm for Multiple Sequence Alignment

In this chapter, the genetic algorithm designed for multiple molecular sequence alignment is described. First, the basic idea of using a genetic algorithm for multiple sequence alignment is discussed. Then, the definitions and the representations used in expressing the algorithm are presented. The algorithm is formulated as a procedure. The survival probability of different kinds of string elements is examined. Finally, the complexity of the algorithm is analyzed.

4.1 Using a Genetic Algorithm for Sequence Analysis

The task of multiple sequence alignment is to look for the optimal or the near-optimal alignments. In our approaches, multiple sequence alignment is divided into two steps: identification of matches and identification of mismatches as well as the missing matches. The matches identified in the first step are represented in a *pre-alignment.*

All the possible pre-alignments can be considered to form a *pre-alignment space* Ω which contains a set of points $\{\omega\}$, representing the possible pre-alignments. Each point is associated with a fitness value which measures the goodness of the corresponding pre-alignment. The task of identification of matches can thus be viewed as searching for an optimal or near-optimal point in the space.

In practical situations, the sequences to be aligned, such as DNA and protein sequences, are usually very long. There may be numerous possible pre-alignments. The existing search methods, including hill climbing, may be too inefficient to deal with such problems. By converting sequence alignment problem into a search problem, a genetic algorithm can be used to generate near-optimal pre-alignments. Being able to move in a "good" direction, a genetic algorithm is more efficient than many conventional search methods. In addition, a genetic algorithm-based method may minimize the chances of falling into local extrema.

The key issues in designing a genetic algorithm for multiple sequence alignment

include the development of a suitable representation scheme of pre-alignments and a search procedure which allows good pre-alignments to survive and to grow during the search process.

4.2 Basic Definitions and Representations

For the problem of multiple sequence alignment, a straightforward implementation of a conventional genetic algorithm, which includes binary string elements, equal length genetic chromosomes (strings), and standard operations, may not be sufficient for achieving high efficiency and good quality. The genetic algorithm designed in this research is enhanced in the representation of problem states and the search procedure. To describe the genetic algorithm, necessary definitions are given in this section.

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 $\mathcal{D}$. The problem of multiple sequence alignment can be formally stated in the following.

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, $1 \leq i \leq m_k$, m_k is the length of the kth sequence, $1 \leq k \leq n$, and n is the number of sequences.

To take into account deletions, insertions and substitutions, we introduce o to represent "blank" elements. Thus we have $\mathcal{A}' = \mathcal{A} \cup \{\phi\}$, and we use x_0^k for o. i.e.,

$x_0^k = \phi$.

Definition 1: An **alignment** of n sequences, denoted as $X^1\#X^2\#...\#X^n$, is

$$X^1\#X^2\#...\#X^n = \begin{array}{cccc} x^1_{u_{1,1}} & x^1_{u_{1,2}} & \cdots & x^1_{u_{1,m}} \\ | & | & \cdots & | \\ x^2_{u_{2,1}} & x^2_{u_{2,2}} & \cdots & x^2_{u_{2,m}} \\ | & | & \cdots & | \\ \cdots & \cdots & \cdots & \cdots \\ | & | & \cdots & | \\ x^n_{u_{n,1}} & x^n_{u_{n,2}} & \cdots & x^n_{u_{n,m}} \end{array} \tag{4.1}$$

where (a) $x^k_{u_{k,i}} \in \mathcal{A}'$;
(b) " | " indicates a match, deletion, insertion, or substitution;
(c) $0 \leq u_{k,i} \leq m_k$;
(d) $u_{k,i} \neq u_{k,j}$, if $i \neq j$;
(e) $u_{k,i} < u_{k,j}$, if $i < j$ and $u_{k,i} \neq 0$ and $u_{k,j} \neq 0$;
(f) there is not an i such that $x^k_{u_{k,i}} = \phi$, $\forall k$;
(g) all the $x^k_{u_{k,i}}$ of the kth input sequence are in the kth row of $X^1\#X^2\#...\#X^n$;
(h) all the subunits in the kth row of $X^1\#X^2\#...\#X^n$ are from the kth input sequence;
(i) $1 \leq k \leq n$, $1 \leq i, j \leq m$; and
(j) $m \geq \max(m_1, m_2, ..., m_n)$.

□

The alignment in (4.1) may also be denoted as a set:

$$X^1\#X^2\#\cdots\#X^n = \{(x^1_{u_{1,i}}, x^2_{u_{2,i}}, \cdots, x^n_{u_{n,i}})\}_{i=1}^m \tag{4.2}$$

where $(x^1_{u_{1,i}}, x^2_{u_{2,i}}, \cdots, x^n_{u_{n,i}})$ corresponds to the ith column of the alignment in (4.1), which may include matches, insertions, deletions, or substitutions between the n subunits, $x^k_{u_{k,1}}, x^k_{u_{k,2}}, \cdots, x^k_{u_{k,m}}$ correspond to the kth row of the alignment, and conditions (a) through (j) of (4.1) are satisfied by the set.

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

$$X^1\#'X^2\#'...\#'X^n = \begin{array}{cccc} x^1_{u_{1,1}} & x^1_{u_{1,2}} & \cdots & x^1_{u_{1,m1}} \\ \| & \| & \cdots & \| \\ x^2_{u_{2,1}} & x^2_{u_{2,2}} & \cdots & x^2_{u_{2,m1}} \\ \| & \| & \cdots & \| \\ \cdots & \cdots & \cdots & \cdots \\ \| & \| & \cdots & \| \\ x^n_{u_{n,1}} & x^n_{u_{n,2}} & \cdots & x^n_{u_{n,m1}} \end{array} \qquad (4.3)$$

where (a) $x^k_{u_{k,i}} \in \mathcal{A}'$;
(b) "$\|$" indicates a match;
(c) $0 \leq u_{k,i} \leq m_k$;
(d) $u_{k,i} \neq u_{k,j}$, if $i \neq j$;
(e) $u_{k,i} < u_{k,j}$, if $i < j$, and $u_{k,i} \neq 0$, $u_{k,j} \neq 0$;
(f) all the subunits in the kth row of $X^1\#'X^2\#'...\#'X^n$ are from the kth input sequence;
(g) $1 \leq k \leq n$, $1 \leq i, j \leq m_1$; and
(h) m_1 is the length of X^1.

□

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 alignments. On the basis of a pre-alignment, by discovering insertions, deletions, substitutions, as well as some matches, an alignment can be constructed. A pre-alignment has m_1 columns, each of which corresponds to a position on X^1. A column may be n matched subunits from the n sequences. A column may be entirely ϕ's which may later be filled with deletions, insertions, substitutions or matches.

The difference between a *pre-alignment* and an *alignment* can be clarified by using

the following example:

(a) Three sequences to be aligned:

1. $G_1^1 A_2^1 C_3^1 G_4^1 T_5^1 C_6^1$
2. $G_1^2 A_2^2 A_3^2 T_4^2 C_5^2$
3. $G_1^3 A_2^3 C_3^3 T_4^3 G_5^3 C_6^3$

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

$$\begin{array}{cccccc} G_1^1 & A_2^1 & \phi & \phi & \phi & C_6^1 \\ \| & \| & \| & \| & \| & \| \\ G_1^2 & A_2^2 & \phi & \phi & \phi & C_5^2 \\ \| & \| & \| & \| & \| & \| \\ G_1^3 & A_2^3 & \phi & \phi & \phi & C_3^3 \end{array} \qquad \begin{array}{cccccc} \phi & \phi & \phi & \phi & T_5^1 & C_6^1 \\ \| & \| & \| & \| & \| & \| \\ \phi & \phi & \phi & \phi & T_4^2 & C_5^2 \\ \| & \| & \| & \| & \| & \| \\ \phi & \phi & \phi & \phi & T_4^3 & C_6^4 \end{array}$$

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

$$\begin{array}{cccccc} G_1^1 & A_2^1 & \phi & \phi & T_5^1 & C_6^1 \\ \| & \| & \| & \| & \| & \| \\ G_1^2 & A_2^2 & \phi & \phi & T_4^2 & C_5^2 \\ \| & \| & \| & \| & \| & \| \\ G_1^3 & A_2^3 & \phi & \phi & T_4^3 & C_6^3 \end{array}$$

(d) An alignment generated from the pre-alignment in (c) by discovering a substitution and two insertions

$$\begin{array}{ccccccc} G_1^1 & A_2^1 & C_3^1 & G_4^1 & T_5^1 & \phi & C_6^1 \\ | & | & | & | & | & | & | \\ G_1^2 & A_2^2 & A_3^2 & \phi & T_4^2 & \phi & C_5^2 \\ | & | & | & | & | & | & | \\ G_1^3 & A_2^3 & C_3^3 & \phi & T_4^3 & G_5^3 & C_6^3 \end{array}$$

□

Definition 3: 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 a dimension in

the space. A pre-alignment corresponds to a **point** in Ω. The pre-alignment in (4.3) corresponds to point

$$\omega = \begin{array}{cccc} \frown & \frown & & \frown \\ u_{1,1} & u_{1,2} & \cdots & u_{1,m_1} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m_1} \\ \cdots & \cdots & \cdots & \cdots \\ u_{n,1} & u_{n,2} & \cdots & u_{n,m_1} \\ \smile & \smile & & \smile \end{array} \tag{4.4}$$

□

The dimension of Ω can also be equal to the length of any other sequence. Without losing generality, we use m_1. The ith column of (4.4) is used as the ith coordinate of ω. The pre-alignment in (c) of the above example corresponds to the following point in Ω:

$$\begin{array}{cccccc} \frown & \frown & \frown & \frown & \frown & \frown \\ 1 & 2 & 0 & 0 & 5 & 6 \\ 1 & 2 & 0 & 0 & 4 & 5 \\ 1 & 2 & 0 & 0 & 4 & 6 \\ \smile & \smile & \smile & \smile & \smile & \smile \end{array}$$

To generate pre-alignments from multiple sequences, we first need to find *match tuples.*

Definition 4: An integer tuple $T = (v_{1,i}, v_{2,i}, ..., v_{n,i})$ is a **match tuple** if

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

where $1 \leq i \leq m_1$.

□

Definition 5: A **match tuple series** is a list of transposed match tuples

$$S = \begin{array}{cccc} \overbrace{\phantom{v_{1,1}}} & \overbrace{\phantom{v_{1,2}}} & & \overbrace{\phantom{v_{1,q}}} \\ v_{1,1} & v_{1,2} & \cdots & v_{1,q} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ v_{n,1} & v_{n,2} & \cdots & v_{n,q} \\ \underbrace{\phantom{v_{1,1}}} & \underbrace{\phantom{v_{1,2}}} & & \underbrace{\phantom{v_{1,q}}} \end{array} \tag{4.5}$$

where $1 \leq q \leq m_1$, and $v_{k,i} < v_{k,j}$ if $i < j$, $\forall k$.

□

A match tuple series can be transformed into the pre-alignment space Ω as a point (i.e. a pre-alignment).

Definition 6: The transformation of a match tuple series S into a point $\omega \in \Omega$ is

$$S = \begin{array}{cccc} \overbrace{\phantom{v_{1,1}}} & \overbrace{\phantom{v_{1,2}}} & & \overbrace{\phantom{v_{1,q}}} \\ v_{1,1} & v_{1,2} & \cdots & v_{1,q} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ v_{n,1} & v_{n,2} & \cdots & v_{n,q} \\ \underbrace{\phantom{v_{1,1}}} & \underbrace{\phantom{v_{1,2}}} & & \underbrace{\phantom{v_{1,q}}} \end{array} \rightarrow \begin{array}{cccc} \overbrace{\phantom{u_{1,1}}} & \overbrace{\phantom{u_{1,2}}} & & \overbrace{\phantom{u_{1,m_1}}} \\ u_{1,1} & u_{1,2} & \cdots & u_{1,m_1} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m_1} \\ \cdots & \cdots & \cdots & \cdots \\ u_{n,1} & u_{n,2} & \cdots & u_{n,m_1} \\ \underbrace{\phantom{u_{1,1}}} & \underbrace{\phantom{u_{1,2}}} & & \underbrace{\phantom{u_{1,m_1}}} \end{array} = \omega \tag{4.6}$$

where

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

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

□

To use a genetic algorithm for sequence alignment, the points are encoded into *genetic strings*. In encoding a point into a genetic string, match tuples are grouped into *match blocks*.

Definition 7: A **match block** consists of the maximum number of *successive* match tuples

$$\begin{array}{cccccc} \frown & \frown & & \frown & \frown \\ v_{1,r} & v_{1,r+1} & \cdots & v_{1,s-1} & v_{1,s} \\ v_{2,r} & v_{2,r+1} & \cdots & v_{2,s-1} & v_{2,s} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ v_{n,r} & v_{n,r+1} & \cdots & v_{n,s-1} & v_{n,s} \\ \smile & \smile & & \smile & \smile \end{array} \qquad (4.7)$$

such that

$$v_{k,r} + 1 = v_{k,r+1}, \quad v_{k,r+1} + 1 = v_{k,r+2}, \quad \ldots, \quad v_{k,s-1} + 1 = v_{k,s}$$

for $k = 1, 2, \ldots, n$, and $1 \leq r, s \leq m_1$.

□

Definition 8: The **length of a match block** B, denoted as $L(B)$, is the number of match tuples in the match block.

□

To be uniform, isolated single match tuples are also represented as match blocks. A match block is indecomposable in the search process. Thus the order and defining length of a schema should be calculated in terms of match blocks. For a point composed of a fixed number of tuples (columns), the more tuples of successive values, the less the number of match blocks. It thus would be of lower order and shorter defining length. Usually, such a point can be encoded as a short genetic string. Recall the discussion in the section of "The Law of String Growth" (Section 3.3), a lower order and a shorter defining length may increase the probability for a string to survive.

Therefore, the representation method defined in this research favours short genetic string and it facilitates the evolution process of the designed genetic algorithm.

Definition 9: A **genetic string** is an encoded version of ω

$$A = g(\omega) = a_1 a_2 ... a_l$$

where a_i is an encoded version of the ith match block in ω and $1 \leq l \leq m_1$.

□

A genetic string represents a pre-alignment. In the rest of this thesis, we may use "genetic string" and "pre-alignment" interchangeably when confusion is not likely.

In a genetic string, the columns of entire ϕ's are the intervals between match blocks.

Definition 10: In a genetic string, the **interval** between two successive match blocks B_i and B_{i+1}, denoted as I_i, is the n subsequences between the two blocks.

□

Definition 11: Let B_i and B_{i+1} be two successive match blocks in a genetic string:

$$B_i = \begin{array}{cccc} \overbrace{} & \overbrace{} & & \overbrace{} \\ v_{1,r} & v_{1,r-1} & \cdots & v_{1,s} \\ v_{2,r} & v_{2,r+2} & \cdots & v_{2,s} \\ \cdots & \cdots & \cdots & \cdots \\ v_{n,r} & v_{n,r+1} & \cdots & v_{n,s} \\ \underbrace{} & \underbrace{} & & \underbrace{} \end{array} \qquad B_{i+1} = \begin{array}{cccc} \overbrace{} & \overbrace{} & & \overbrace{} \\ v_{1,t} & v_{1,t+1} & \cdots & v_{1,w} \\ v_{2,t} & v_{2,t+2} & \cdots & v_{2,w} \\ \cdots & \cdots & \cdots & \cdots \\ v_{n,t} & v_{n,t+1} & \cdots & v_{n,w} \\ \underbrace{} & \underbrace{} & & \underbrace{} \end{array}$$

where $r \leq s$, $s < t$ and $t \leq w$ and let I_i be the interval between the two blocks. The **interval length** of the jth $(1 \leq j \leq n)$ subsequence in I_i, denoted as $L_j(I_i)$, is

defined as

$$L_j(I_i) = v_{j,t} - v_{j,s}. \tag{4.8}$$

□

In a mutation operation the following lengths are required in selecting mutation positions.

Definition 12: Let B_i and B_{i+1} be two successive match blocks.

The **average left interval length** of B_{i+1} is

$$\begin{aligned} \mathrm{ALIL}(B_{i+1}) &= [\sum_{j=1}^{n} L_j(I_i)]/n \\ &= [\sum_{j=1}^{n} (v_{j,t} - v_{j,s})]/n. \end{aligned} \tag{4.9}$$

The **average right interval length** of B_i is

$$\mathrm{ARIL}(B_i) = \mathrm{ALIL}(B_{i+1}). \tag{4.10}$$

The **maximum left interval length** of B_{i+1} is

$$\begin{aligned} \mathrm{MLIL}(B_{i+1}) &= \max_j \{L_j(I_i)\} \\ &= \max_j \{v_{j,t} - v_{j,s}\}. \end{aligned} \tag{4.11}$$

The **maximum right interval length** of B_i is

$$\mathrm{MRIL}(B_i) = \mathrm{MLIL}(B_{i+1}). \tag{4.12}$$

□

When applying crossover to two strings, we must check if the strings are *matable*.

Definition 13: Two strings are **matable** at given cutting positions if each of the two new strings formed satisfies conditions (d) and (e) in (4.3).

□

4.3 Algorithm Parameters

Determining appropriate population sizes and mutation positions are important issues in application of genetic algorithms. In the following, the two issues are discussed. Apart from that, this section also presents the conditions for terminating the search process and the fitness function.

Population size

Population size largely affects a genetic algorithm in both the ultimate performance and the computational costs. As Goldberg (1989b) demonstrated (included in Section 3.5), relatively small population sizes are appropriate for serial implementations of genetic algorithms and relatively large sizes are appropriate for parallel implementations.

Since our genetic algorithm is coded as a serial program, we use relatively small population sizes. In an alignment task, the population size is determined when choosing a number of match tuple series to form the first generation. Of the genetic strings generated, there exists a small group of strings with high fitness values (higher than 1/10 of the maximum value) while the rest had lower values. As described by Equation (3.7), the strings of very low fitness values have small probabilities to survive. Therefore, we use only the highly fitted strings to form the first generation. Discarding the strings of low fitness values will have no significant effects on the population's

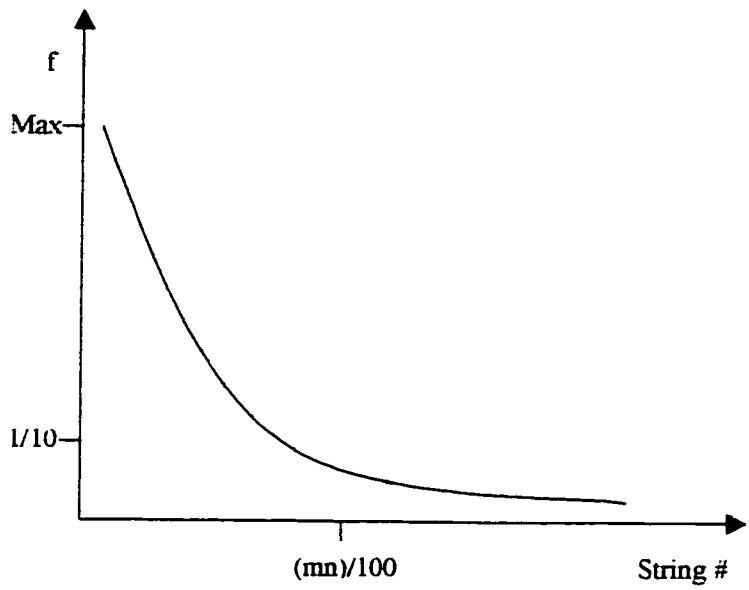


Figure 4.1: The relationship between sequence fitness value and string number.

representativeness and the genetic algorithm performance. It was observed in many experiments conducted in this research that the number of highly fitted strings is proportional to the product of $(m_a n)$, where m_a is the average sequence length and n the number of sequences. The number is usually smaller than 1/100 of the product. Figure 4.1 illustrates the observation via a simplified sketch of the relationship between the fitness and the string number, in which the strings are ordered in a descending order by their fitness values and each string is numbered according to its position in the order.

Therefore for an alignment task, the population size Q is determined as

$$Q = m_a n/100. \tag{4.13}$$

The first generation thus formed includes the strings with fitness values greater than, equal to or slightly lower than 1/10 of the maximum fitness value.

Mutation rate

The mutation operation plays a very significant role in the genetic algorithm which increases the variability of the population. It is unlike the common belief that mutation is not so important because the mutation rate is small.

A mutation rate p^m is used to determine the number of strings to be mutated. In the genetic algorithm, the mutation rate is chosen to be 0.1 %. The rounded product of the rate and the total number of match blocks determines the number of strings to mutate. We change one element value in each of the strings randomly selected. If the product is less than 1, then one string is selected.

Selection of a mutation position

In the genetic algorithm, for a string selected for mutation, assuming the string has more than one element, the operation removes the element (match block) B, which has the largest D value in the string, where

$$\mathrm{D} = [(\mathrm{MLIL(B)} - \mathrm{ALIL(B)}) + (\mathrm{MRIL(B)} - \mathrm{ARIL(B)})]/[L(B)]. \qquad (4.14)$$

$L(B)$ is defined in Definition 8 and MLIL, ALIL, MRIL and ARIL are defined in (4.9), (4.10), (4.11) and (4.12) of Definition 12, respectively.

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 4.2 illustrates such an element.

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

Figure 4.2: A "harmful" element, which is marked by "@@".

Crossover rate

The crossover rate controls the frequency with which the crossover operation is applied. In a newly generated population, $round(p^c \times Q)$ strings undergo crossover. The higher the crossover rate, the more quickly new strings are introduced into the population. If the crossover rate is too high, high-performance strings may be discarded faster than reproduction can produce improvement. If the crossover rate is too low, the search may stagnate due to the lower exploration rate. In the designed algorithm, the crossover rate is chosen as 20 %.

Termination condition

The termination condition used in this research is to stop the search process if the population has been converged to a string or there is no change to the maximum number of matches for ten successive generations.

Fitness values

The fitness value of a string is equal to the number of match tuples in the corre-

sponding pre-alignment.

4.4 The Genetic Algorithm for Multiple Sequence Pre-Alignment

The algorithm for generating alignments is composed of two steps: the first one for pre-alignment generation and the second one for alignment construction. The genetic algorithm is used in the first step which is the most important and difficult. The second step is relatively straightforward, which involves identifying insertions, deletions, substitutions, and missing matches between match blocks (i.e., intervals).

The procedure for the first step is included in this chapter and the procedure for the second step is described in the following two chapters. Before describing the first procedure, the step for match blocks creation is discussed.

4.4.1 Matching

Finding match blocks is a very important first step in the algorithm. Conceptually, match tuples are first identified and then match blocks are formed by grouping them. In implementing the algorithm, the match blocks are directly created to improve efficiency. The following is the procedure for building a match block when a match tuple is found:

procedure MATCH (**Input:** $X^1, X^2, ..., X^n$; **Output:** $B_1, B_2, ... B_i, ...$)

If $((v_{1,i}, v_{2,i}, ..., v_{n,i})$ is a match tuple)
Form a new match block which includes $(v_{1,i}, v_{2,i}, ... v_{n,i})$;
$j = 1$;
While $((v_{1,i} + j, v_{2,i} + j, ... v_{n,i} + j)$ is a match tuple)
Add $(v_{1,i} + j, v_{2,i} + j, ... v_{n,i} + j)$ to the match block;
$j = j + 1$;
EndWhile
EndIf.

end procedure

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

4.4.2 Pre-alignment construction

The genetic algorithm for pre-alignment construction is given as the following procedure. The sequences to be aligned are $X^1, X^2, ...$ and X^n which have lengths $m_1, m_2, ...$ and m_n respectively. The output is pre-alignment $X^1\#'X^2\#'...\#'X^n$.

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 $X^1, X^2, \cdots$, and X^n using procedure MATCH.

2. Create genetic strings from the match blocks (K – number of match blocks. J – number of genetic strings):

```
J = 0
For i = 1 through K do
    For j = 1 through J do
        If (A_j satisfies conditions (d) and (e) in (4.3) 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^τ where τ denotes generation number ($\tau = 0, 1, \ldots$).

```
For i = 1 through Q do
    Calculate the fitness value for A_i ∈ G^τ;
    Determine the number of duplicates for A_i using Equation (3.7);
    If (the number is greater than or equal to 1)
            Duplicate A_i and place the duplicate(s) into matepool M^τ;
    EndIf
EndFor.
```

The matepool is $M^\tau = \{A'_1, A'_2, \ldots, A'_Q\}$ where A'_i ($1 \leq i \leq Q$) is duplicated from a string in G^τ.

5. Apply mutation to M^τ:

   Calculate the number of element(s) to be mutated:
   Randomly select strings for mutation:
   In each selected string, determine the mutation position using (4.9)-(4.12) and (4.14):
   Remove the element at the position.

6. Apply crossover to M^τ to generate $G^{\tau+1}$:

   While (There are strings in M^τ to mate) do
      Randomly select A'_i and A'_j from M^τ:
      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
   $G^{\tau+1} = M^\tau$.

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

8. Select the best string from the population.

9. If there exist intervals longer than $m_a/20$ in the best string
      Call PRE-ALIGN to allign such subsequences in the intervals:
   EndIf.

10. Produce $X^1\#'X^2\#'...\#'X^n$ from the best string.

end procedure

4.5 Survival Probability of Match Blocks

The first step of procedure PRE-ALIGN creates a large number of match blocks. They are elements of pre-alignments (genetic strings). Of the match blocks, some are building blocks of near optimal pre-alignments while some are spurious blocks which may lead to poor pre-alignments. In this section, the probability for a match block to survive is discussed for an understanding of what kind of match blocks may most probably be included in the output of the procedure.

Generally, a pre-alignment A can be expressed as

$$A = B_1 B_2 \cdots B_l$$

where B_i $(1 \leq i \leq l)$ is the ith match block, and l is the length of A. There is an interval between every two adjacent match blocks. Let $f(A)$ denote the fitness value of pre-alignment A, which is defined as

$$f(A) = \sum_{i=1}^{l} L(B_i)$$

(see Section 4.3).

Lemma 4.5.1: The probability for match block B_k to be contained in A_i, a genetic string in the first generation, can be

$$p^1_{B_k} = \begin{cases} 1 & \text{if } f(A_i)/f_{\max} > 0.1 \\ (f(A_i)/f_{\max}) \times 10 & \text{otherwise} \end{cases}$$

where $f_{\max}$ is the maximum fitness value of all the genetic strings created.

□

Lemma 4.5.2: The probability for B_k to survive in a reproduction can be

$$p^2_{B_k} = \begin{cases} 1 & \text{if } Q \cdot [f(A_i)/\sum_{j=1}^{Q} f(A_j)] \geq 1 \\ Q \cdot [f(A_i)/\sum_{j=1}^{Q} f(A_j)] & \text{otherwise} \end{cases}$$

where Q is the population size and A_i is the string containing B_k.

□

Lemma 4.5.3: Let B_k be the kth match block in a genetic string ($1 \leq k \leq l$). The probability for B_k to survive in a mutation is

$$p^3_{B_k} = \begin{cases} 1 & \text{if } D(B_k) < Thr \\ 1 - p^m & \text{otherwise} \end{cases}$$

where p^m is the mutation rate, D is defined in (4.13), and Thr is a threshold used to select string elements to mutate.

□

Lemma 4.5.4: The probability for B_k to survive in a crossover is

$$p^4_{B_k} = 1.$$

□

A match block is a decomposable element in a genetic string and therefore it is not destroyed by a crossover operation.

Theorem: The probability for B_k to survive the whole process of search is

$$p^s_{B_k} = p^1_{B_k} \cdot (p^2_{B_k} \cdot p^3_{B_k} \cdot p^4_{B_k})^{\tau_c}$$

where τ_c is the number of generations when the search converges.

Proof: Since survivals in reproduction, mutation and crossover are independent events, the probability for B_k to survive in a generation is thus $p^2_{B_k} \cdot p^3_{B_k} \cdot p^4_{B_k}$, and in τ_c generations is $(p^2_{B_k} \cdot p^3_{B_k} \cdot p^4_{B_k})^{\tau_c}$.

□

Let B_k be the kth match block in A_i, and I_{k-1} and I_k be the intervals before and after the block. It can be observed that $p^1_{B_k}$ and $p^2_{B_k}$ are proportional to $f(A_i)$ and $p^3_{B_k}$ favors blocks with a small D. $L(B_k)$ contributes to $f(A_i)$. $D(B_k)$ is a function of $L(B_k)$, $L_j(I_{k-1})$ and $L_j(I_k)$ $(j = 1, 2, \cdots n)$. The discussion in the section of "Basic Definitions and Representations" (Section 4.2) suggests that $D(B_k) \propto 1/L(B_k)$ and $D(B_k) \propto L_j(I_{k-1}) + L_j(I_k)$.

In most alignment tasks, match blocks of large lengths and small D values may form good pre-alignments. In the algorithm, they have higher probability to survive. Spurious match block may have small lengths, and may have large $L_j(I_{k-1})$ and $L_j(I_k)$. They have lower probability to survive.

4.6 Algorithm Complexity

In this section, the complexity of the genetic algorithm for sequence alignment is analyzed in terms of computing time and memory space. The algorithm is compared with a pairwise dynamic programming method which is the basis of many existing approaches for multiple sequence alignment.

The time complexity of the genetic algorithm can be estimated using the model developed by Ankenbrandt (1991). As discussed in the section on "Complexity of Genetic Algorithms" (Section 3.4), 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 $Q > Q_0$. It will be seen in Chapter 5 that Equation (3.39) fits well with the experimental results.

For a particular task, the total time required is affected by the fitness ratio. As discussed on Section 3.4, the time would be proportional to

$$\frac{Q \log Q}{\log r} \tag{4.15}$$

r is the fitness ratio which is defined as

$$r = \frac{\bar{f}'}{\bar{f}''} \tag{4.16}$$

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.

When using the genetic algorithm for sequence alignment, strings in a population have different fitness values. It is assumed that the fitness values are in the range

$[f_{min}, f_{max}]$ and A_{max} is the string with the value f_{max}. Let $\bar{f}'$ be the fitness value of A_{max} which is f_{max} and $\bar{f}''$ be the average fitness value of all the other strings. By using Equations (3.39) and (4.16), we can estimate the time for a search process to converge to A_{max}. Since f_{max} is always larger than $\bar{f}''$, we have

$$r = \frac{\bar{f}'}{\bar{f}''} > 1. \tag{4.17}$$

Let r take a very conservative value, say 1.1. By substituting the r value and $Q = m_a n/100$, Equation (4.15) becomes

$$\frac{(m_a n/100)\log(m_a n/100)}{\log 1.1}. \tag{4.18}$$

The designed genetic algorithm includes recursive procedure calls. As described in Section 4.4.2 that procedure PRE-ALIGN is recursively called when the length of an interval is longer than $m_a/20$ (Step 9 of the procedure).

In general cases, when the intervals have an average length m_a/k, $(1 \leq k \leq 20)$, the times of calling PRE-ALIGN for the intervals are less than or equal to k. Thus the computing time for a genetic algorithm should be less than or equal to

$$\begin{aligned}
& \{(m_a n/100)\log(m_a n/100) + k \cdot [(m_a/k)n/100]\log[(m_a/k)n/100]\}/\log 1.1 \\
\approx\ & 25 \cdot (m_a n/100) \cdot \{\log(m_a n/100) + \log[(m_a/k)n/100]\} \\
=\ & 25 \cdot (m_a n/100) \cdot \log[m_a^2 n^2/(k \cdot 100^2)] \\
=\ & (m_a n/4) \cdot 2 \cdot \log[m_a n/(k^{1/2} \cdot 100)]
\end{aligned}$$

$$= (n/2) \cdot m_a \cdot \log[m_a n/(100k^{1/2})] \tag{4.19}$$

Lemma 4.6.1: The time complexity of the genetic algorithm plus the k times recursive calls of the procedure is smaller than or equal to

$$(n/2) \cdot m_a \cdot \log[m_a n/(100k^{1/2})] \tag{4.20}$$

□

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, a pairwise dynamic programming method 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

Lemma 4.6.2: The time complexity for a pairwise dynamic programming is

$$O[m_a^2(n-1)]. \tag{4.21}$$

□

Theorem: The genetic algorithm has smaller time complexity than a pairwise dynamic programming method.

Proof: For $n \geq 2$, and $1 \leq k \leq 20$, and $n/(100k^{1/2}) < m_a$ we have

$$\begin{aligned} & \frac{(n/2) \cdot m_a \cdot \log[m_a n/(100k^{1/2})]}{m_a^2(n-1)} \\ = & \frac{(n/2) \cdot \log[m_a n/(100k^{1/2})]}{m_a(n-1)} \\ < & \frac{\frac{n}{2(n-1)} \cdot \log(m_a^2)}{m_a} \\ = & \frac{\frac{n}{(n-1)} \cdot \log(m_a)}{m_a} \\ < & 1. \end{aligned} \tag{4.22}$$

□

In the proof, we use the fact that $\frac{n}{(n-1)} \cdot \log(m_a) < m_a$ holds for $n \geq 2$.

Equation (4.22) also suggests that the longer the sequences, the greater the difference; and the larger the number of sequences, the greater the difference.

The major data structure of the genetic algorithm is a two dimensional array used to store both 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 $l_{\max} m_a n/100$.

4.7 Summary

By decomposing the task of multiple sequence alignment into steps of identifying matches and handling mismatches, and by converting the first step into a search problem, a genetic algorithm can be designed for simultaneous multiple sequence alignment.

The designed genetic algorithm is an enhanced one with additional functionality aimed at high efficiency and better alingment quality. This algorithm is characterized by the numeric and index gene representation, variable genetic string lengths, more informed genetic operations, a genetic repairing strategy and a fine-tuning mechanism.

The algorithm enables good match blocks to have higher survival probability and to serve as the building blocks of a near optimal pre-alignment. Compared with a pairwise dynamic programming-based approach, the algorithm has a smaller time complexity. It is also more efficient in terms of space complexity.

Chapter 5

Experiments with the Genetic Algorithm

In this chapter, the performance of the genetic algorithm is examined in detail by presenting and analyzing the initial experimental results which are *pre-alignments* produced by the genetic algorithm. The processing time and the identified matches in a protein data set, an mRNA data set, and 11 other DNA or RNA data sets are analyzed and compared with those obtained by using CLUSTALW, a most widely used program for multiple molecular sequence alingment (Thompson et al., 1994).

The machine used for this research is an IBM RISC 6000 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 genetic algorithm is implemented as a C program.

5.1 Results of Pre-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 about 12.000 subunits long each) will be discussed later.

The genetic algorithm was used to pre-align from two to eight of the protein sequences. The processing time and the number of matches in the results are listed in Table 5.1. CLUSTALW was applied to the same data for comparison. The time and the number of matches of its results are also listed. In this table, as well as in the following discussion, we use "G†" to indicate the genetic algorithm, "C" to indicate CLUSTALW, and a "†" to indicate "for/in pre-alignment only".

The "processing time" listed on the tables is elapsed 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 finding match blocks and generating pre-alignments. For CLUSTALW, this includes the time for pairwise alignment, multiple alignment and quality evaluation.

The "number of matches" for the genetic algorithm is the number of tuples of

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

# of seqn.	Average length	Methods	Processing time	# of matches
2	335	G†	0s375ms†	194†
		C	8s524ms	211
3	333	G†	0s490ms†	168†
		C	14s257ms	189
4	336	G†	0s734ms†	163†
		C	21s860ms	187
5	338	G†	0s872ms†	128†
		C	29s881ms	155
6	341	G†	1s044ms†	123†
		C	39s012ms	151
7	341	G†	1s299ms†	110†
		C	46s125ms	145
8	341	G†	1s352ms†	110†
		C	53s663ms	143

Table 5.1: Processing time and numbers of matches of the pre-/alignments of the protein data set by the two programs.

such that

$$x^1_{u_{1,i}} = x^2_{u_{2,i}} = \cdots = x^n_{u_{n,i}}.$$

And, the "number of matches" for CLUSTALW is the number of tuples of

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

such that

$$x^1_{u_{1,i}} = x^2_{u_{2,i}} = \cdots = x^n_{u_{n,i}}.$$

Table 5.2 lists ratios of the measurements of the genetic algorithm over those of CLUSTALW from Table 5.1. The following facts can be observed from the results of the two programs.

The computing time required by the genetic algorithm is about two orders of magnitude 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.

The number of matches identified by the genetic algorithm is no less than 76% of the number by CLUSTALW. On average, the genetic algorithm spent less than 4% of the time required by CLUSTALW to identify 84% of the matches by CLUSTALW.

When using the genetic algorithm to process all the eight sequences, 49 genetic strings were created and 24 of them were chosen to form the first generation. The

# of seqn.	Processing time*	# of matches*
2	4.36×10^{-2}	0.92
3	3.44×10^{-2}	0.89
4	3.35×10^{-2}	0.87
5	2.92×10^{-2}	0.83
6	2.68×10^{-2}	0.82
7	2.82×10^{-2}	0.76
8	2.52×10^{-2}	0.77
Average	3.16×10^{-2}	0.84

Table 5.2: $G^{\dagger}/C$ ratios of the measurements in Table 5.1. The columns marked with an "$\star$" contain the ratios.

search process stopped at the 46th generation. Figure 5.1 illustrates the maximum number of matches in the generations, which may help understand the search towards the final result. The maximum number of matches in a generation is the number of matches in the string which is the most fitted.

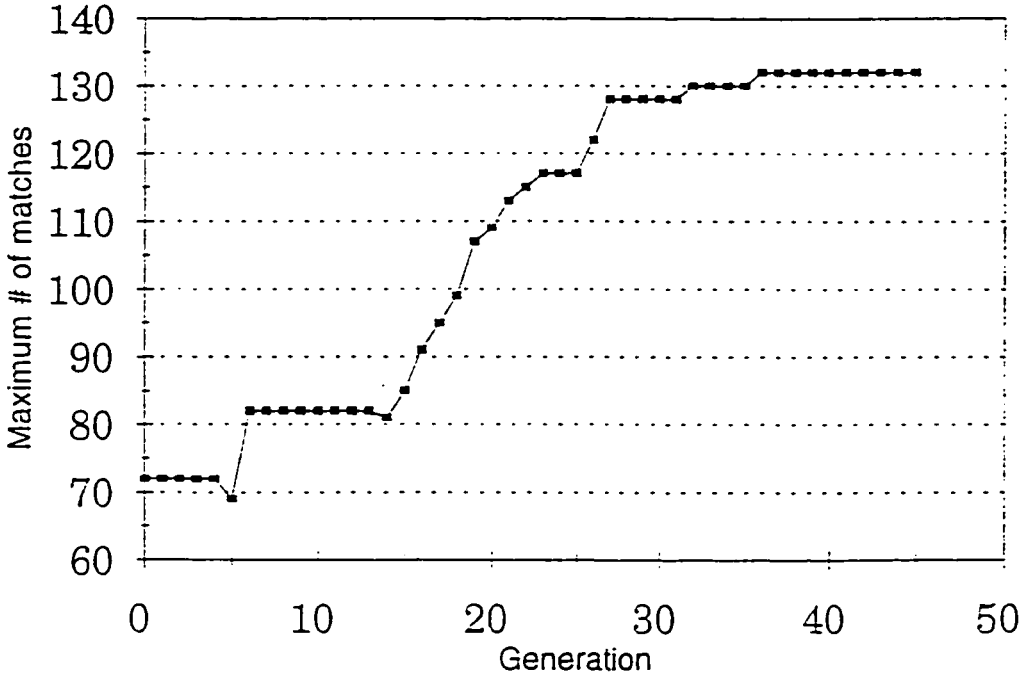


Figure 5.1: Maximum numbers of matches in generations.

The match blocks identified in the pre-alignment are shown in Figure 6.2 of the next chapter (marked by letter "M"). It can be noticed that there are no long match blocks in the pre-alignment generated by the genetic algorithm on this data set. The longest match block is only seven subunits long. In aligning such kind of sequences, 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 the genetic algorithm, since short match blocks usually lead to a large number of long

strings. Searching with long genetic strings requires more time. Still. the genetic algorithm is still much more efficient and achieves reasonably good results. As will be seen in the following two sections. when sequences have long match blocks. the genetic algorithm is much more superior to CLUSTALW in terms of efficiency.

5.2 Results of Pre-aligning an mRNA Sequence Set

The mRNA data set has ten sequences, each of which has a length of about 12,000 subunits. In this section, the results of pre-aligning the ten mRNA sequences of *different lengths* by the two programs are presented and compared in terms of the processing time and the number of matches. The ratios of the measurements are also presented. Finally, the time required by the genetic algorithm to process *different numbers* of sequences with lengths up to 12,000 subunits is illustrated in a figure representation.

Table 5.3 lists the processing time and the numbers of matches of the results obtained by applying the two programs to the ten sequences. Table 5.4 gives the ratios of the measurements. The results were obtained by processing subsequences of different lengths. For example, length "1000" indicates the first 1,000 subunits of the ten sequences. CLUSTALW was not able to align sequences longer than 9,000 subunits when applied to this data set.

The genetic algorithm produced better pre-alignments in dealing with this group of data than the protein data set. The average processing time required by the genetic algorithm to produce the pre-alignments is two to three orders of magnitude lower than the time required by CLUSTALW, while the number of matches identified are on average 95% of those identified by CLUSTALW. As the length of the ten sequences

Length	Methods	Processing time	# of matches
1000	G†	5s310ms†	762†
	C	9m24s352ms	797
2000	G†	14s018ms†	1427†
	C	37m50s329ms	1508
3000	G†	21s898ms†	2194†
	C	1hr25m46s520ms	2322
4000	G†	28s094ms†	2991†
	C	2hr41m30s783ms	3143
5000	G†	39s746ms†	3738†
	C	4hr17m53s542ms	3877
6000	G†	44s319ms†	4490†
	C	6hr25m16s070ms	4732
7000	G†	50s956ms†	5306†
	C	8hr41m23s699ms	5587

Table 5.3: Processing time and numbers of matches of the pre-/alignments of the mRNA data set by the two programs.

Length	Methods	Processing time	# of matches
8000	G†	58s753ms†	6026†
	C	11hr04m55s431ms	6383
9000	G†	1m06s051ms†	6909†
	C	14hr03m13s902ms	7266
10000	G†	1m11s150ms†	7711†
	C	NA	NA
11000	G†	1m18s152ms†	8565†
	C	NA	NA
12000	G†	1m22s395ms†	9253†
	C	NA	NA

Table 5.3: (Continued) Processing time and numbers of matches of the pre-/alignments of the mRNA data set by the two programs.

Lenght	Processing time*	# of matches*
1000	9.41×10^{-3}	0.96
2000	6.17×10^{-3}	0.95
3000	4.26×10^{-3}	0.94
4000	2.90×10^{-3}	0.95
5000	2.57×10^{-3}	0.96
6000	1.92×10^{-3}	0.95
7000	1.63×10^{-3}	0.95
8000	1.47×10^{-3}	0.94
9000	1.31×10^{-3}	0.95
Average	3.52×10^{-3}	0.95

Table 5.4: $G^{\dagger}/C$ ratios of the measurements in Table 5.3. The columns marked with an "$\star$" contain the ratios.

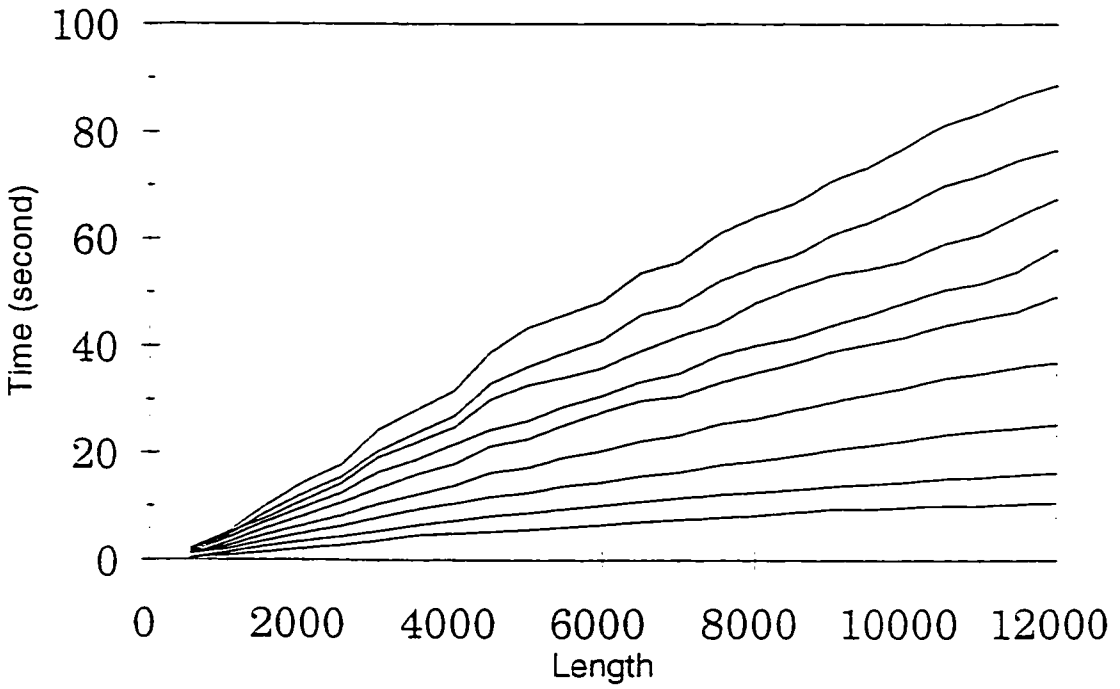


Figure 5.2: Processing time vs. sequence length. The curves from the bottom to top are for two, three, ..., and ten sequence alignments, respectively.

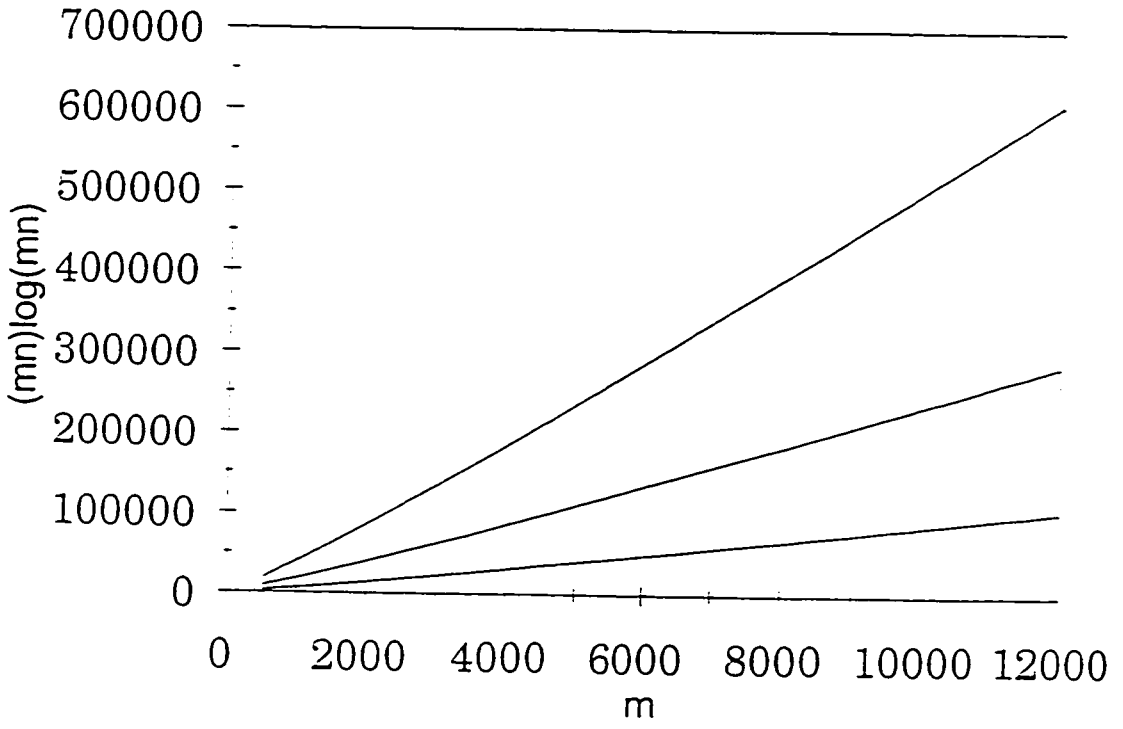


Figure 5.3: Curves of the complexity model: $mn\log(mn)$. The curves from the bottom to top are for $n = 2$, 3 and 10, respectively.

increases, the processing time required by the genetic algorithm increases. However, the difference in processing time between the two programs, too, increases. This is also indicated by the decrease of the ratio in processing time: when the length increases from 1,000 to 9,000, the ratios in processing time monotonously decreased from 9.4×10^{-3} to 1.3×10^{-3}.

The genetic algorithm has been used to pre-align two, three, ... and ten of the mRNA sequences. For a given number of sequences, the algorithm was applied to subsequences of different lengths. Figure 5.2 illustrates the relationship between the computing time and the sequence length. Each curve is for a given number of sequences. Figure 5.3 approximates the complexity model of $Q \log Q$. It shows three curves of $mn \log(mn)$ for sequence number $n = 2$, 3 and 10, respectively. By comparing the two sets of curves in Figure 5.2 and 5.3, it can be observed that the complexity model fits well with the experimental results.

5.3 Results of Pre-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 achieved satisfactory results.

In the following, the results of using the genetic algorithm to pre-align 11 data sets are presented and compared with the alignments by CLUSTALW. The sequence data sets are from the European Bioinformatics Institute. IDs of the sequences are

listed in Appendix II. Of the 11 sequence sets. four are DNA and seven are RNA. The minimum average length is 212 and the maximum is 4582.

Table 5.5 lists the processing time and the numbers of matches. Table 5.6 gives the measurement ratios. For the 11 data sets, the processing time of the genetic algorithm is about two orders of magnitude lower than that of CLUSTALW on average and the matches identified in the pre-alignments is 88% of the matches identified by CLUSTALW.

Data set	# of sequences	Average length	Methods	Processing time	# of matches
S1	10	211	G†	0s223ms†	184†
			C	27s113ms	197
S2	5	1780	G†	6s219ms†	1449†
			C	13m08s776ms	1611
S3	4	2433	G†	1s002ms†	2303†
			C	18m28s150ms	2305
S4	8	1437	G†	19s389ms†	576†
			C	15m38s568ms	901
S5	8	1680	G†	11s180ms†	1251†
			C	21m05s898ms	1439
S6	4	4582	G†	0s792ms†	4225†
			C	1hr4m58s781ms	4409
S7	5	1093	G†	4s274ms†	764†
			C	5m12s948ms	905

Table 5.5: Processing time and numbers of matches of the pre-/alignments of other DNA and RNA data sets by the two programs. Of the data sets, S1, S2, S4, and S5 are DNA while the rest are RNA.

Data set	# of sequences	Average length	Methods	Processing time	# of matches
S8	8	1071	G†	3s851ms†	971†
			C	8m35s650ms	999
S9	6	1456	G†	8s765ms†	849†
			C	11m21s416ms	1118
S10	7	1448	G†	6s432ms†	1091†
			C	13m22s264ms	1193
S11	5	1092	G†	4s384ms†	753†
			C	5m3s838ms	855

Table 5.5: (Continued) Processing time and numbers of matches of the pre-/alignments of other DNA or RNA data sets by the two programs. Of the data sets, S1, S2, S4, and S5 are DNA while the rest are RNA.

An important character of the genetic algorithm can be observed by examining this group of experimental results. The processing time for the genetic algorithm to produce the pre-alignments is not simply proportional to the number and the length of the sequences aligned. Instead, it depends heavily upon the size of match blocks and the number of match blocks in a given unit length of sequences. Higher efficiency is closely related with the existence of longer match blocks and thus smaller number of match blocks. The processing time proportionally increases with the increase of

Data set	# of sequences	Average length	Processing time*	# of matches*
S1	10	211	8.23×10^{-3}	0.93
S2	5	1780	7.88×10^{-3}	0.90
S3	4	2433	9.04×10^{-4}	1.00
S4	8	1437	1.96×10^{-2}	0.64
S5	8	1680	8.83×10^{-3}	0.87
S6	4	4582	2.03×10^{-4}	0.96
S7	5	1093	1.36×10^{-2}	0.84
S8	8	1071	7.47×10^{-3}	0.97
S9	6	1456	1.28×10^{-2}	0.76
S10	7	1448	8.02×10^{-3}	0.91
S11	5	1092	1.44×10^{-2}	0.88
Average			9.26×10^{-3}	0.88

Table 5.6: $G^{\dagger}/C$ ratios of the measurements in Table 5.5. The columns marked with an "$\star$" contain the ratios.

the sequence length and the number of sequences only in the situations that 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).

Compare the results of S4 and S5. Both data sets have eight sequences each, and S4 has an average length of 1437 and S5 has 1680. The processing time for the genetic algorithm to pre-align S4 is about 19 seconds and the time for S5 is about 11 seconds. Differing from CLUSTALW, the genetic algorithm spent longer time on the shorter sequence data set. Compare the results of S9 and S10. Both data sets have sequences of approximately the same lengths. S9 has six sequences and S10 has seven. The processing time for the genetic algorithm to align S9 is about nine seconds and the time for S10 is about six seconds. Again, differing from CLUSTALW, the genetic algorithm spent longer time on the data set of a smaller number of sequences. 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 the efficiency. For instance, S4 has a ratio of 40.1% and S5 has 74.5%. 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 to align S5. Similar situations can also be found in the alignments of S7 and S8, as well as those of S9 and S10.

5.4 Summary

The initial experimental results indicate that the genetic algorithm is able to produce very good pre-alignments which are bases for generating final multiple sequence alignments. The numbers of the identified matches are comparable to those achieved by a most widely used program - CLUSTALW: Overall, the genetic algorithm is able to generate pre-alignments which identify 84% to 95% of the matches identified by CLUSTALW while the processing time is two to three orders of magnitude lower than that of CLUSTALW. The major strength of the genetic algorithm is its high efficiency in computing time. This strength is especially significant when aligning the long sequences.

More detailed discovery regarding the performance of the genetic algorithm includes:

- Within initial generations, the maximum numbers of matches increase with the number of generations.
- The processing time of the genetic algorithm proportionally increases with the sequence length and the number of sequences in the circumstances that the sizes and the number of match blocks in a given unit length of the sequences are similar.
- The existence of long match blocks largely reduce processing time.

Chapter 6

Constructing Multiple Sequence Alignments

A multiple sequence alignment can be constructed by handling subsequences between match blocks in a pre-alignment. This chapter describes a simple shift-up method (Zhang and Wong, 1997a) which is designed to construct alignments from pre-alignments. The alignments obtained by combining the genetic algorithm with the shift-up method are presented and compared with those produced by CLUSTALW.

6.1 The Shift-up Method

A pre-alignment is a series of match blocks with an interval between every two adjacent blocks. An interval represents subsequences missing from the pre-alignment. In constructing an alignment from a pre-alignment, we align the missing subsequences in each interval and use the alignment to fill the interval. As previously mentioned, we may use different methods to align the subsequences which may contain mismatches as well as matches. In this chapter, we use the simple shift-up method to process the intervals. Later, two other methods, including a pairwise dynamic programming method and a sequence synthesis method, will be presented in Chapter 7.

The shift-up method:

For every two adjacent match blocks:

Find the missing subunits from the input sequences which should be
    in the interval;
Find sequence X^i which has the maximum number of such subunits ($1 \leq i \leq n$);
Fill row i of the interval with the subunits of X^i;
For $j = 1$ through n and $j \neq i$
    Place the s missing subunits $x^j_{u_{j,1}}, x^j_{u_{j,2}}, ..., x^j_{u_{j,s}}$ at the lower end of
        row j in the interval;
    For $k = s$ down to 1
        Move $x^j_{u_{j,k}}$ up to the column of $x^i_{v_{i,l}}$ in the free space, such that
            $F(x^j_{u_{j,k}}, x^i_{v_{i,l}})$ is the maximum for all the subunits
            of X^i corresponding to the free space;
    EndFor
EndFor

The following are the evaluation functions F used in this procedure for protein, DNA and RNA. The matrix in (6.1) 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 "$\star$" in the matrix represents a negative integer and "-1" is used for an $\star$ in this research for gap penalty. The scores, except those for o, are from the SG (Structure-Genetic) Matrix (Feng at. el., 1985). According to Feng, this scoring system has taken into account the structural 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 (6.2), 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).

$$
\begin{array}{c|cccccccccccccccccccc}
 & o & c & s & t & p & a & g & n & d & e & q & h & r & k & m & i & l & v & f & y & w \\
o & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & \star \\
c & & 6 & 4 & 2 & 2 & 2 & 3 & 2 & 1 & 0 & 1 & 2 & 2 & 0 & 2 & 2 & 2 & 2 & 3 & 3 & 3 \\
s & & & 6 & 5 & 4 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 2 & 2 & 2 & 3 & 3 & 2 \\
t & & & & 6 & 4 & 5 & 2 & 4 & 2 & 3 & 3 & 2 & 3 & 4 & 3 & 3 & 2 & 3 & 1 & 2 & 1 \\
p & & & & & 6 & 5 & 3 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 3 & 3 & 2 & 2 & 2 \\
a & & & & & & 6 & 5 & 3 & 4 & 4 & 3 & 2 & 2 & 3 & 2 & 2 & 2 & 5 & 2 & 2 & 2 \\
g & & & & & & & 6 & 3 & 4 & 4 & 2 & 1 & 3 & 2 & 1 & 2 & 2 & 4 & 1 & 2 & 3 \\
n & & & & & & & & 6 & 5 & 3 & 3 & 4 & 2 & 4 & 1 & 2 & 1 & 2 & 1 & 3 & 0 \\
d & & & & & & & & & 6 & 5 & 4 & 3 & 2 & 3 & 0 & 1 & 1 & 3 & 1 & 2 & 0 \\
e & & & & & & & & & & 6 & 4 & 2 & 2 & 4 & 1 & 1 & 1 & 4 & 0 & 1 & 1 \\
q & & & & & & & & & & & 6 & 4 & 3 & 4 & 2 & 1 & 2 & 2 & 1 & 2 & 1 \\
h & & & & & & & & & & & & 6 & 4 & 3 & 1 & 1 & 3 & 1 & 2 & 3 & 1 \\
r & & & & & & & & & & & & & 6 & 5 & 2 & 2 & 2 & 2 & 1 & 1 & 2 \\
k & & & & & & & & & & & & & & 6 & 2 & 2 & 2 & 3 & 0 & 1 & 1 \\
m & & & & & & & & & & & & & & & 6 & 4 & 5 & 4 & 2 & 2 & 3 \\
i & & & & & & & & & & & & & & & & 6 & 5 & 5 & 4 & 3 & 2 \\
l & & & & & & & & & & & & & & & & & 6 & 5 & 4 & 3 & 4 \\
v & & & & & & & & & & & & & & & & & & 6 & 4 & 3 & 3 \\
f & & & & & & & & & & & & & & & & & & & 6 & 5 & 3 \\
y & & & & & & & & & & & & & & & & & & & & 6 & 3 \\
w & & & & & & & & & & & & & & & & & & & & & 6
\end{array}
\tag{6.1}
$$

$$
\begin{array}{c|ccccc}
 & o & a & g & c & t(u) \\
o & 0 & 0 & 0 & 0 & 0 \\
a & & 6 & 1 & 1 & 1 \\
g & & & 6 & 1 & 1 \\
c & & & & 6 & 1 \\
t(u) & & & & & 6
\end{array}
\tag{6.2}
$$

6.2 System Implementation

The approach combining the genetic algorithm and the shift-up method has been implemented as a system, which consists of two major modules, one for the genetic algorithm and the other for the shift-up method.

Figure 6.1 illustrates the processing flow in the system.

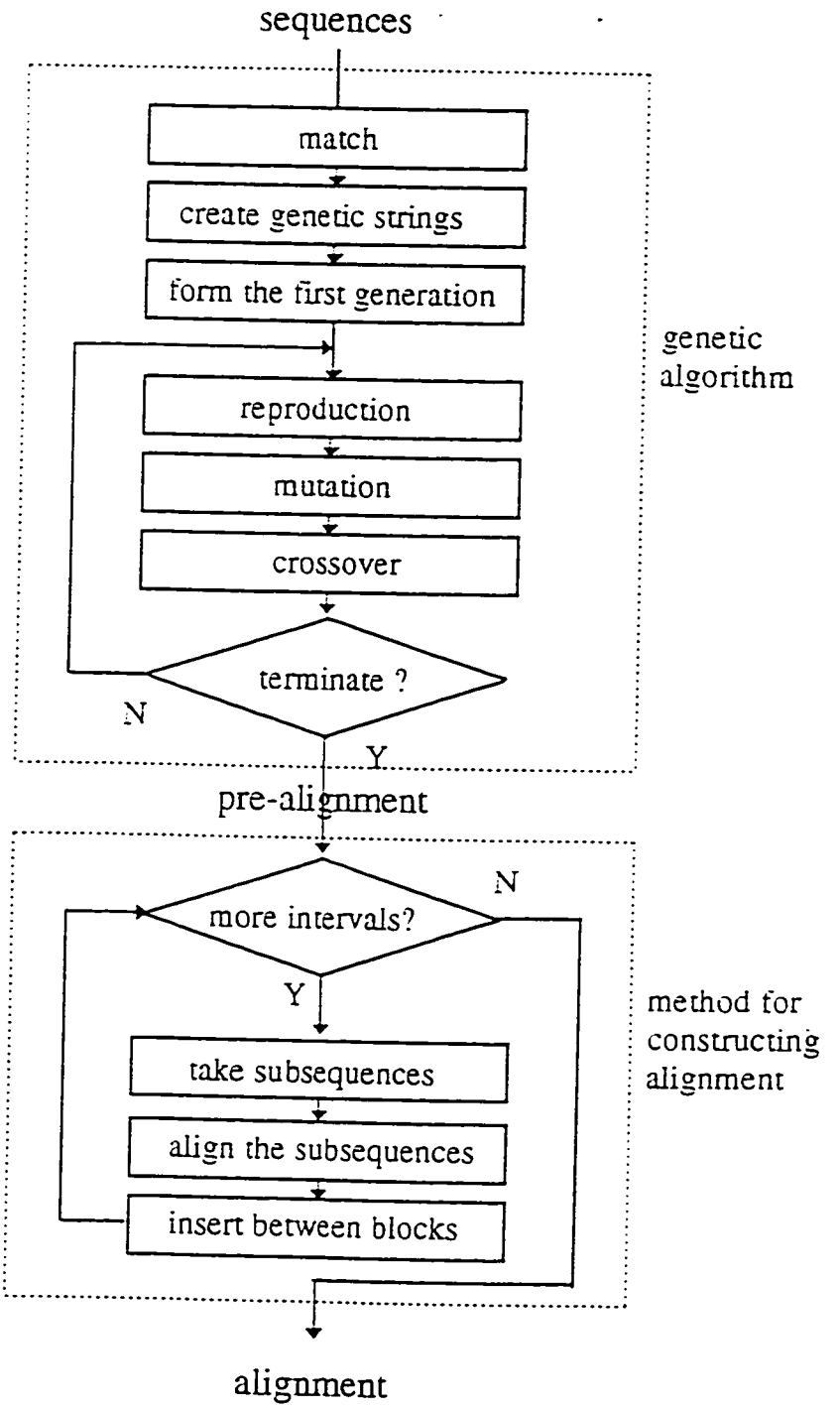


Figure 6.1: System processing flow.

6.3 Three More Quality Evaluation Parameters

In order to evaluate alignment quality in a more accurate and comprehensive way, three more quality measurements are conducted for the purpose of comparison. they are two types of score measurements. ScoreG and ScoreC, and an entropy measurement.

"ScoreG" and "ScoreC" are score measurements of alignments. The two programs, i.e., the genetic algorithm plus the shift-up procedure and CLUSTALW, use different scoring systems in aligning sequences. For a "fair" comparison, the alignments are also evaluated by using the scoring system of the other. (The time for the "mutual" evaluation is not included in the processing time).

"ScoreG" is calculated using the following formula:

$$\text{ScoreG} = \sum_{i=1}^{m} \sum_{j,k}^{n} F(x^{j}_{u_{j,i}}, x^{k}_{u_{k,i}}) \tag{6.3}$$

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 are defined in the matrices of (6.1) and (6.2) respectively.

"ScoreC" is calculated by using the evaluation method of CLUSTALW. The evaluation function for calculating values of "ScoreC" is represented by the matrix in Appendix I. Gap penalties of this scoring system is not included in the matrix. Details of the gap definition and the gap penalty can be found in an article by Thompson

et. al. (1994). When using CLUSTALW, we chose 10 as gap penalty.

"Entropy" is a measurement of diversity. It is calculated using the following formula:

$$\text{Entropy} = -\sum_{i=1}^{m}\sum_{j=1}^{N} p_{i,j} \log p_{i,j} \tag{6.4}$$

where m is the length of the alignment. N is the total number of characters in domain $\mathcal{A}$, and $p_{i,j}$ is the probability of the jth character in the ith column of the alignment. Entropy is calculated out of $p_{i,j} \neq 0$. A lower entropy value indicates less diversity and higher compactness in the alignment which are desirable alignment qualities.

6.4 Experimental Results

6.4.1 Results of aligning the protein sequence set

In the following discussion, we use "G" to refer to the program implementing the genetic algorithm plus the shift-up procedure and "C" to indicate CLUSTALW.

Table 6.1 lists the time and the quality measurements of the results obtained by applying the two programs to the protein data set and Table 6.2 lists the ratios of the measurements in Table 6.1. Note that in this and the next chapters, the "processing time" for a genetic algorithm based approach includes the time required by the genetic algorithm to produce the pre-alignments and the time by a mismatch handling method to construct the alignment. The following facts can be observed from the results.

The processing time required by the "G" program is about two orders of magnitude 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: when the sequence number increases from two to eight, the ratio of the processing time decreases from 3.7×10^{-2} to 2.6×10^{-2}.

The number of matches identified by the "G" program is no less than 96% of those by CLUSTALW. On average, the number of matches identified by the "G" program is 99% of that by CLUSTALW. The numbers increase by 15% from the pre-alignments.

# of sequences	Average length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
2	335	G	0s312ms	208	1505	1184	49.97
		C	8s524ms	211	1687	1312	39.74
3	333	G	0s499ms	191	5048	4149	55.52
		C	14s257ms	189	5224	4320	49.19
4	336	G	0s764ms	187	10279	8785	66.91
		C	21s860ms	187	10654	9207	50.98
5	338	G	0s876ms	153	15539	11817	107.44
		C	29s881ms	155	16936	14176	91.30
6	341	G	1s120ms	148	23425	16980	123.83
		C	39s012ms	151	25081	21625	109.72
7	341	G	1s344ms	139	32625	23148	134.31
		C	46s125ms	145	34836	28014	121.93
8	341	G	1s390ms	139	43708	31346	137.77
		C	53s663ms	143	46457	37414	125.35

Table 6.1: Processing time and quality measurements of the alignments of the protein data set by the two programs.

(The pre-alignments identified 84% of what is identified by CLUSTALW on average, as shown in Table 5.2.)

The scores of the alignments by the "G" program are only 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 may be some biases when the alignments generated by one program are evaluated by the original scoring system. Thus the actual performance difference between the two programs should be better evaluated by the average of the two ratio averages, which is about 91% in this case.

# of sequences	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
2	3.66×10^{-2}	0.99	0.89	0.90	1.26
3	3.50×10^{-2}	1.01	0.97	0.96	1.13
4	3.49×10^{-2}	1.00	0.97	0.95	1.31
5	2.93×10^{-2}	0.99	0.92	0.83	1.18
6	2.87×10^{-2}	0.98	0.93	0.79	1.13
7	2.91×10^{-2}	0.96	0.94	0.83	1.22
8	2.59×10^{-2}	0.97	0.94	0.84	1.10
Average	3.14×10^{-2}	0.99	0.94	0.87	1.19

Table 6.2: G/C ratios of the measurements in Table 6.1. The columns marked with an "$\star$" contain the ratios.

Figure 6.2 and Figure 6.3 illustrate the eight protein sequences aligned by the two programs respectively. The alignments are subdivided into successive segments, with each containing 60 subunits of the first sequence. An "M" indicates a match identified in a match block of the pre-alignment, and an "*" indicates a match identified by the shift-up method. It can be observed that both programs have correctly identified most of the matched subunits and discovered the overall relationships between the sequences. In constructing this alignment, the shift-up approach contributes 20% of the total number of the matches identified. The alignment generated by the "G" program is only slightly less compact than that by CLUSTALW. The former is 399 columns long and the latter is 395 columns long.

This fact can also be measured by their entropy values. The values of the two alignments generated by the "G" program and CLUSTALW are 137.77 and 125.35 respectively. On average, the alignments by the "G" program have entropy values 19% higher than those by CLUSTALW as shown in Table 6.2.

6.4.2 Results of aligning the mRNA sequence set

Table 6.3 lists the processing time and the quality measurements of the alignments of the ten mRNA sequences. Table 6.4 gives the ratios of the measurements. The alignments are again obtained by aligning subsequences of different lengths.

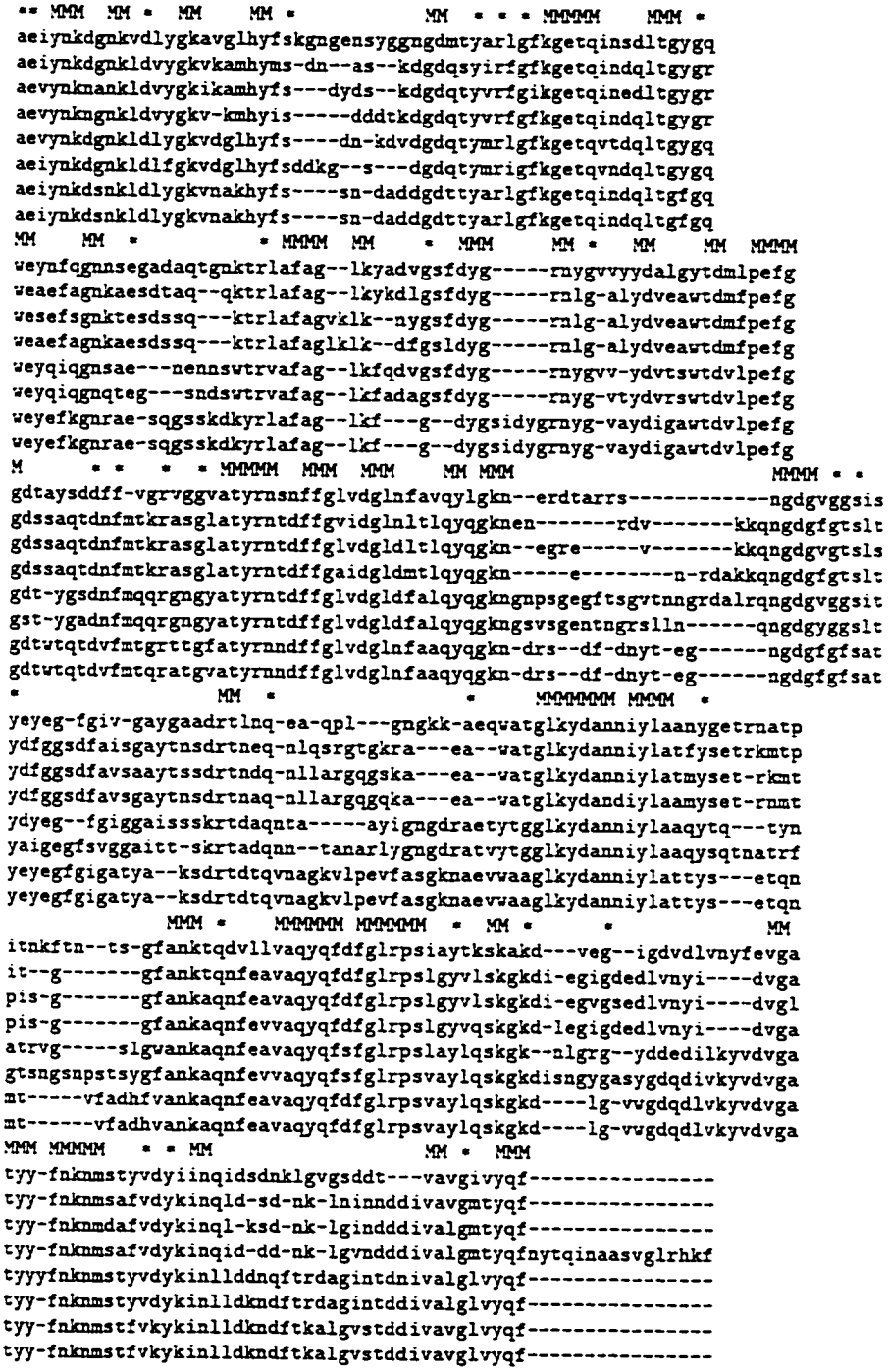


Figure 6.2: Alignment of the eight protein sequences by the "G" program.

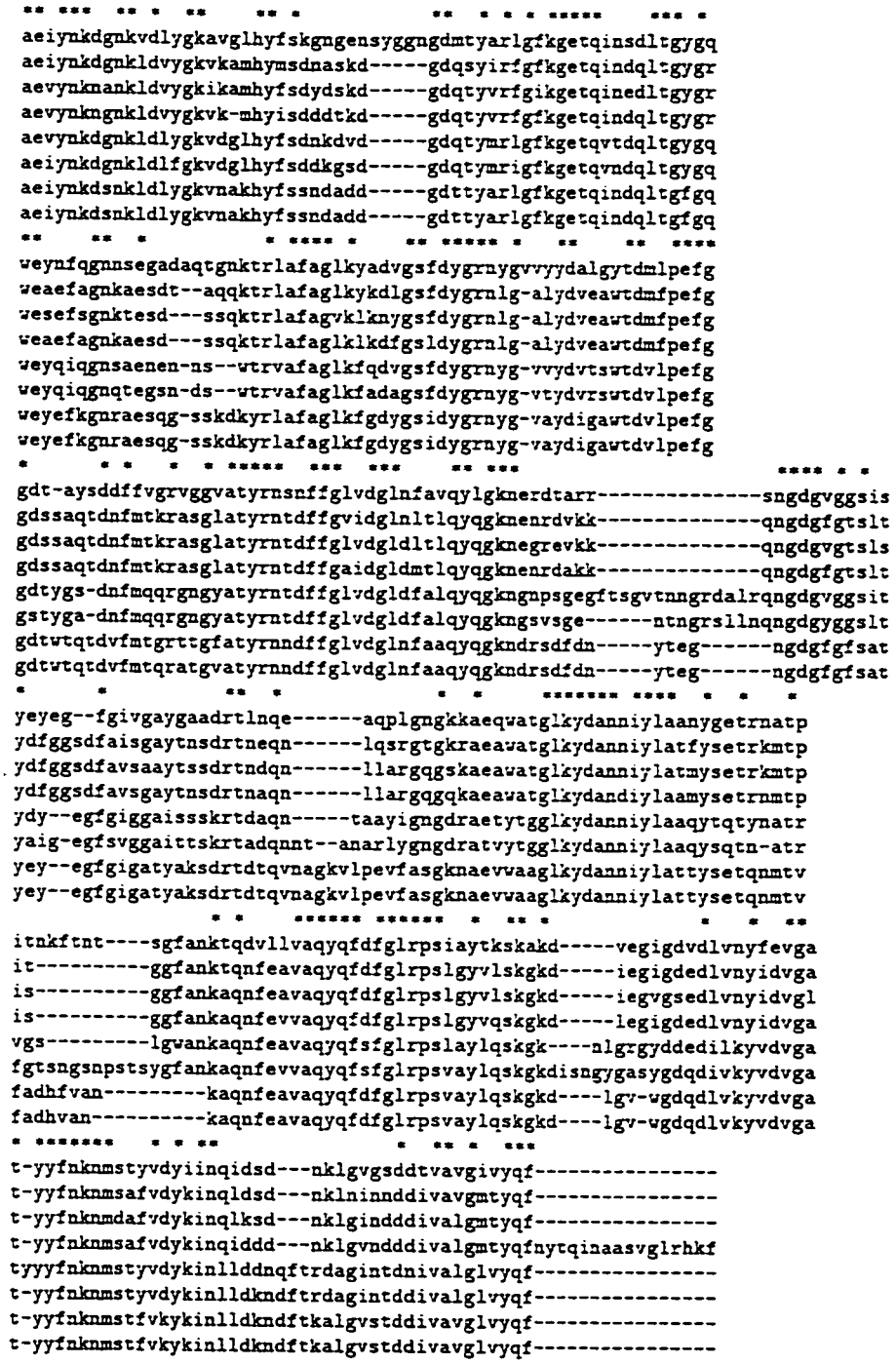


Figure 6.3: Alignment of the protein sequences by CLUSTALW.

Length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
1000	G	5s092ms	797	257766	284173	44.89
	C	9m24s352ms	797	257813	288591	43.82
2000	G	14s461ms	1504	501230	545930	238.16
	C	37m50s329ms	1508	501539	556687	231.86
3000	G	24s276ms	2319	758391	807544	280.54
	C	1hr25m46s520ms	2322	759854	828861	271.94
4000	G	31s552ms	3130	1012239	1086277	387.00
	C	2hr41m30s783ms	3143	1015712	1116731	350.66
5000	G	43s679ms	3922	1286592	1372633	534.41
	C	4hr17m53s542ms	3877	1265871	1390669	464.08
6000	G	48s344ms	4700	1512500	1600349	604.86
	C	6hr25m16s070ms	4732	1522469	1659179	508.13
7000	G	55s757ms	5544	1776488	1896154	678.81
	C	8hr41m23s699ms	5587	1784108	1956777	569.71

Table 6.3: Processing time and quality measurements of the alignments of the mRNA data set by the two programs.

Length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
8000	G	1m04s230ms	6328	2030002	2164507	773.66
	C	11hr04m55s431ms	6383	2040507	2237351	649.06
9000	G	1m10s844ms	7205	2285279	2421534	848.53
	C	14hr03m13s902ms	7266	2300673	2512553	680.06
10000	G	1m17s227ms	8038	2547223	2715173	889.98
	C	NA	NA	NA	NA	NA
11000	G	1m23s472ms	8906	2806860	2990517	972.01
	C	NA	NA	NA	NA	NA
12000	G	1m28s797ms	9627	3028626	3217619	1104.01
	C	NA	NA	NA	NA	NA

Table 6.3: (Continued) Processing time and quality measurements of the alignments of the mRNA data set by the two programs.

For this data set, the "G" program produces alignments of very similar quality as CLUSTALW in terms of the number of matches and the scores. The results are even better than the alignments of the protein data set when using the results of CLUSTALW as a reference. The "G" program requires much less processing time than CLUSTALW. For instance in aligning the 10 sequences of length 9,000, while CLUSTALW spent more than 14 hours to produce the results of 61 more (or 0.85%

Lenght	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
1000	9.02×10^{-3}	1.00	1.00	0.98	1.02
2000	6.37×10^{-3}	1.00	1.00	0.98	1.03
3000	4.72×10^{-3}	1.00	1.00	0.97	1.03
4000	3.26×10^{-3}	1.00	1.00	0.97	1.10
5000	2.82×10^{-3}	1.01	1.02	0.99	1.15
6000	2.09×10^{-3}	0.99	0.99	0.96	1.19
7000	1.78×10^{-3}	0.99	1.00	0.97	1.19
8000	1.61×10^{-3}	0.99	0.99	0.97	1.19
9000	1.40×10^{-3}	0.99	0.99	0.96	1.25
Average	3.67×10^{-3}	1.00	1.00	0.97	1.13

Table 6.4: G/C ratios of the measurements in Table 6.3. The columns marked with an "$\star$" contain the ratios.

more) matches, the "G" program spent only one minute and 11 seconds. In analyzing this set of results, the following observations are made:

The average processing time required by the "G" program to align the 10 sequences of different lengths is two to three orders of magnitude lower than the time by CLUSTALW. As the length of the 10 sequences increases, the processing time required by the "G" program increases. 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 1,000 to 9,000, the ratio in processing time decreases from 9.0×10^{-3} to 1.4×10^{-3}.

The average ratios of the number of matches is 1.00. A considerable number of matches are identified by the shift-up method. For instance, when aligning the 10 sequences of length 10,000, the total number of matches identified by the "G" program is 8,038, of which the shift-up method contributed 327 matches accounting for 4% of the total (7711 matches are identified in the pre-alignment as shown in Table 5.3.).

The averages of the ratios in ScoreG and ScoreC are 1.01 and 0.98 respectively, and the average of these two is 1.00. Overall, on this data set, the "G" program is able to generate alignments of similar quality in terms of the number of matches and the score within about 1/1000 of the time required by CLUSTALW.

The average of the ratios in entropy values is 1.13. The higher entropy values of the alignments by the "G" program may partially due to the fact that they are less compact. For example, the alignment of the subsequences of length 9,000 by the "G"

program contains 9453 columns while the alignment by CLUSTALW contains 9320.

6.4.3 Results of aligning the 11 other sequence sets

Table 6.5 lists the processing time and the quality measurements for the alignments of the 11 other DNA and RNA data sets (four DNA and seven RNA data sets). Table 6.6 gives the ratios of the measurements. For the 11 data sets, the "G" program also has very good performance compared with CLUSTALW.

Its processing time is approximately two orders of magnitude lower. The average of the ratios in the numbers of matches is 0.98. The average of the ratios in ScoreG and ScoreC is 0.98 and 1.00 respectively, and the average of the two is approximately 0.99.

As with the protein data set and the mRNA data set, the "G" program produces less compact alignments than CLUSTALW. The entropy values of the alignments by the "G" program are higher, and their average is higher than that of CLUSTALW by 34%.

Data set	# of sequ.	Aver. length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
S1	10	211	G	0s215ms	198	56491	71847	2.27
			C	27s113ms	197	56491	69010	2.27
S2	5	1780	G	6s194ms	1553	101507	120763	99.72
			C	13m8s776ms	1611	103018	115785	43.28
S3	4	2433	G	0s983ms	2303	85615	104867	68.37
			C	18m28s150ms	2305	85701	104149	64.86
S4	8	1437	G	17s943ms	804	196746	191880	300.11
			C	15m38s568ms	901	210317	226516	204.88
S5	8	1680	G	10s685ms	1420	269432	291406	81.12
			C	21m5s898ms	1439	269092	303390	46.02
S6	4	4582	G	0s784ms	4332	159532	180620	138.47
			C	1hr4m58s781ms	4409	160860	182475	149.91
S7	5	1093	G	4s394ms	894	60153	75516	79.26
			C	5m12s948ms	905	60719	73280	53.90

Table 6.5: Processing time and quality measurements of the alignments of other DNA and RNA data sets by the two programs. Of the data sets, S1, S2, S4, and S5 are DNA while the rest are RNA.

Data set	# of sequ.	Aver. length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
S8	8	1071	G	3s782ms	999	176665	227985	15.65
			C	8m35s650ms	999	176665	216359	15.65
S9	6	1456	G	8s553ms	1046	116221	139681	218.62
			C	11m21s416ms	1118	120534	142237	86.97
S10	7	1448	G	6s265ms	1179	170552	211327	119.26
			C	13m22s264ms	1193	172756	206195	85.07
S11	5	1092	G	4s281ms	840	57682	68951	139.56
			C	5m3s838ms	855	60100	71361	90.83

Table 6.5: (Continued) Processing time and quality measurements of the alignments of other DNA and RNA data sets by the two programs. Of the data sets, S1, S2, S4, and S5 are DNA while the rest are RNA.

The fact that the genetic algorithm's performance is affected by the existence of long match blocks can be observed again in this group of results.

Data set	# of sequ.	Aver. length	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
S1	10	211	7.92×10^{-3}	1.00	1.00	1.04	1.00
S2	5	1780	7.85×10^{-3}	0.96	0.99	1.04	1.30
S3	4	2433	8.87×10^{-4}	1.00	1.00	1.01	1.05
S4	8	1437	1.91×10^{-2}	0.89	0.94	0.85	1.46
S5	8	1680	8.44×10^{-3}	0.99	1.00	0.96	1.76
S6	4	4582	2.01×10^{-4}	0.98	0.99	0.99	0.92
S7	5	1093	1.40×10^{-2}	0.99	0.99	1.03	1.47
S8	8	1071	7.33×10^{-3}	1.00	1.00	1.05	1.00
S9	6	1456	1.26×10^{-2}	0.94	0.96	0.98	2.51
S10	7	1448	8.03×10^{-3}	0.99	0.99	1.02	1.40
S11	5	1092	1.41×10^{-2}	0.98	0.96	0.97	1.54
Average			9.13×10^{-3}	0.98	0.98	1.00	1.34

Table 6.6: G/C ratios of the measurements in Table 6.5. The columns marked with an "$\star$" contain the ratios.

6.5 Summary

Alignments can be constructed from pre-alignments by processing subsequences in intervals. The program combining the genetic algorithm and the shift-up method may produce multiple sequence alignments of very good quality efficiently. The total processing time is about 1/1000 of that required by CLUSTALW. The shift-up method is very effective for processing the subsequences. It has the strength of simplicity and high efficiency. It can produce the shortest alignment from a given pre-alignment.

Chapter 7

Combined Approaches of Genetic Algorithm and Dynamic Programming

For better processing intervals between match blocks, two different methods based on dynamic programming have been combined with the genetic algorithm. In one method, dynamic programming is used for pairwise alignment; and in the other, it is used for sequence synthesis.

When sequence lengths are short, a dynamic programming method can achieve good alignments with reasonable computing resources. With the strength of handling mismatches, as well as matches, in a systematic way, it might be a complement to the genetic algorithm. Integrating the strengths of the two methods, a combined

approach may achieve both high efficiency and improved alignment quality.

In this chapter, the two combined approaches of the genetic algorithm and the dynamic programming methods are described in detail. The experimental results are presented and the improvements are analyzed.

7.1 The Pairwise Dynamic Programming Method

This section presents a pairwise dynamic programming method which is based on the two-sequence alignment dynamic programming. It belongs to the methods which align sequence pairs in an arbitrary order. The method is used along with the genetic algorithm to produce multiple sequence alignments (Zhang and Wong, 1997b).

For each pair of successive match blocks, the pairwise dynamic programming method is applied to the n subsequences in the interval. To align n' $(2 < n' \leq n)$ subquences, the method first produces an alignment of the first $n'-1$ subsequences, then uses the two-sequence dynamic programming method to align the $(n'-1)$th and the n'th subsequences, and finally produces an alignment of the n' sequences by combining the two alignments. After aligning the n sequences, the alignment is inserted between the two blocks.

The two-sequence dynamic programming aligns two sequences to maximize the score which is defined later. The central data structure of the method is a two dimensional matrix M. (As mentioned before, the data structure of a dynamic programming for two sequences can be one dimensional arrays. For easy implementation, we use a two dimensional array.) For sequences $X^1 = x_1^1 x_2^1 ... x_{m_1}^1$ and $X^2 = x_1^2 x_2^2 ... x_{m_2}^2$, M is of the size $m_1 \times m_2$. The element values stored in M are defined as the following:

$$M_{0,0} = 0.$$

$$M_{0,j} = \sum_{k=1}^{j} F(\phi, x_k^2),$$

$$M_{i,0} = \sum_{k=0}^{i} F(x_k^1, \phi),$$

$$M_{i,j} = \max \begin{cases} M_{i-1,j} + F(x_i^1, \phi) \\ M_{i-1,j-1} + F(x_i^1, x_j^2) \\ M_{i,j-1} + F(\phi, x_j^2) \end{cases} \tag{7.1}$$

where $M_{i,j}$ is the element at the ith row and jth column and function $F(x_i^1, x_j^2)$ is the score between x_i^1 and x_j^2 $(0 < i \leq m_1,\ 0 < j \leq m_2)$. The score systems for protein, DNA and RNA are defined in Equations (6.1) and (6.2).

After creating the M matrix, an alignment can be formed by tracing back from M_{m_1,m_2}. The next matrix element to be traced from $M_{i,j}$ $(1 \leq i \leq m_1,\ 1 \leq j \leq m_2)$ is

$$\begin{cases} M_{i-1,j} & \text{if } [M_{i-1,j} + F(x_i^1, \phi)] \text{ is the maximum.} \\ M_{i-1,j-1} & \text{if } [M_{i-1,j-1} + F(x_i^1, x_j^2)] \text{ is the maximum.} \\ M_{i,j-1} & \text{if } [M_{i,j-1} + F(\phi, x_j^2)] \text{ is the maximum.} \end{cases} \tag{7.2}$$

The result of the tracing is a string of index pairs. A pair indicates that the two indexed subunits should be aligned. The index pair recorded into the string when

tracing from $M_{i,j}$ is determined by the next element:

$$\begin{cases} (i, \phi) & \text{if } M_{i-1,j} \text{ is the next.} \\ (i, j) & \text{if } M_{i-1,j-1} \text{ is the next.} \\ (\phi, j) & \text{if } M_{i,j-1} \text{ is the next.} \end{cases} \tag{7.3}$$

Pair (i, ϕ) indicates that the ith subunit of sequence X^1, i.e. x_i^1, is an insertion. (i, j) indicates that the ith subunit of sequence X^1, i.e. x_i^1, should be aligned to the jth subunit of sequence X^2, i.e. x_j^2, and, (ϕ, j) indicates that the jth subunit of sequence X^2 is an insertion,

The procedure PAIRALIGN was designed for generating an alignment from a pre-alignment, in which L is the number of match blocks in the pre-alignment. To simplify the procedure, two empty blocks, the 0th and $(L+1)$th blocks, are added to the head and end of the pre-alignment.

procedure PAIRALIGN (**Input:** $X^1\#'X^2\#'...\#'X^n$: **Output:** $X^1\#X^2\#...\#X^n$)
begin

For $i = 1$ through $L + 1$ do
Take the n subsequences $X_s^1, X_s^2, ..., X_s^n$
between the $(i-1)$th and the ith match blocks:
For $j = 2$ through n do
Align the $(j-1)$th and the jth sequences to have $X_s^{j-1}\#X_s^j$;
If $j > 2$
Combine $X_s^1\#X_s^2\#\cdots\#X_s^{j-1}$ with $X_s^{j-1}\#X_s^j$
to produce $X_s^1\#X_s^2\#...\#X_s^j$;
EndIf
EndFor
Place the alignment between the $(i-1)$th and ith match blocks:
EndFor.

end procedure

The method for combining two alignments is illustrated in the following.

Let

$$X_s^1 \# X_s^2 \# \cdots \# X_s^{j-1} = \{(x^1_{u_{1,l}}, x^2_{u_{2,l}}, \cdots, x^{j-1}_{u_{j-1,l}})\}_{l=1}^{m'}$$

and

$$X_s^{j-1} \# X_s^j = \{(x^{j-1}_{u'_{j-1,k}}, x^j_{u'_{j,k}})\}_{k=1}^{m''}.$$

We have

$$(x^1_{u_{1,l}}, x^2_{u_{2,l}}, \cdots, x^{j-1}_{u_{j-1,l}}, x^j_{u'_{j,k}}) \in X_s^1 \# X_s^2 \# \cdots \# X_s^j$$

if and only if

$$(x^1_{u_{1,l}}, x^2_{u_{2,l}}, \cdots, x^{j-1}_{u_{j-1,l}}) \in X_s^1 \# X_s^2 \# \cdots \# X_s^{j-1}, (x^{j-1}_{u'_{j-1,k}}, x^j_{u'_{j,k}}) \in X_s^{j-1} \# X_s^j,$$

$$\text{and } u_{j-1,l} = u'_{j-1,k}.$$

7.2 The Sequence Synthesis Method

The sequence synthesis method is also based on the two-sequence dynamic programming and the random graph technique (Wong et. al., 1990; Chan, 1990). Similar to the pairwise dynamic programming method, for each pair of successive match blocks, it is applied to the n subsequences in the interval, and then, the alignments of the n subquences are inserted between the two blocks. Differing from the pairwise approach, in aligning $2 < n' \leq n$ subquences, the alignment of the first $n' - 1$ subsequences is treated as a synthesized sequence and is aligned with the n'th subsequence. An additional combination step is not required. Another major difference is that this approach is based on minimization of entropy (Zhang and Wong, 1998).

Before introducing this approach, the calculation of entropy and the concepts of sequence synthesis are described as the following which is in accordance with You and Chan's work (You, 1983; Wong and You, 1985; Chan, 1990):

Let $\mathcal{A}' = \{\phi\} \cup \mathcal{A}$ be a character set with a null character ϕ and let $r^s = c_1 c_2 ... c_m$ be the synthesis representation of the alignment $X^1 \# X^2 \# \cdots \# X^n$, where m is the

length of the alignment and

$$c_i = \begin{bmatrix} a_o, p_{oi} \\ a_1, p_{1i} \\ . \\ . \\ . \\ a_N, p_{Ni} \end{bmatrix}$$

is for a column of the alignment. c_i satisfies:

- $a_j \in \mathcal{A}'$, for $0 \leq j \leq N$ where N is the number of characters in $\mathcal{A}$:
- p_{ji} is the probability of the occurrences of character a_j at c_i;
- $\sum_{j=0}^{N} p_{ji} = 1.0$. for $1 \leq i \leq m$:
- The "column" entropy of c_i is:

$$H(c_i) = -\sum_{j=0}^{N} p_{ji} \log p_{ji} \tag{7.4}$$

- The entropy of the whole alignment is:

$$\begin{aligned} H(r^s) &= -\sum_{i=1}^{m} \sum_{j=0}^{N} p_{ji} \log p_{ji} \\ &= \sum_{i=1}^{m} H(c_i) \end{aligned} \tag{7.5}$$

- Let c_i be for a column in an alignment on $\mathcal{A}'$ and c_j be for a column in another

alignment on $\mathcal{A}'$. The synthesis of c_i and c_j is defined as:

$$c_k = \begin{bmatrix} a_o, q_1 p_{oi} + q_2 p_{oj} \\ a_1, q_1 p_{1i} + q_2 p_{1j} \\ \cdot \\ \cdot \\ \cdot \\ a_N, q_1 p_{Ni} + q_2 p_{Nj} \end{bmatrix}$$

where $q_1 = f_1/(f_1 + f_2)$, $q_2 = f_2/(f_1 + f_2)$, and, f_1 and f_2 are the numbers of sequences in the two alignments from which c_i and c_j are taken.

The sequence synthesis method aligns sequences to minimize the the entropy value. In aligning the synthesis of the first $(n' - 1)$ sequences with the n'th sequence, we express the column entropy as $H(c_i, x_j^{n'})$ where c_i is for the ith column of the synthesis of the first $(n' - 1)$ sequences and $x_j^{n'}$ is the jth subunits of the n'th sequence. To calculate $H(c_i, x_j^{n'})$, we first add $x_j^{n'}$ to c_i and then use Equation (7.4).

The central data structure of the method is also a two dimensional matrix M. For synthesis $r^s = c_1 c_2 ... c_m$, and $X^{n'} = x_1^{n'} x_2^{n'} ... x_{m_{n'}}^{n'}$, M is of the size $m \times m_{n'}$. The element values stored in M are defined as the following:

$$M_{0,0} = 0,$$

$$M_{0,j} = \sum_{k=1}^{j} H(\phi, x_k^{n'}),$$

$$M_{i,0} = \sum_{k=0}^{i} H(c_k, \phi),$$

$$M_{i,j} = \min \begin{cases} M_{i-1,j} + H(c_i, \phi) \\ M_{i-1,j-1} + H(c_i, x_j^{n'}) \\ M_{i,j-1} + H(\phi, x_j^{n'}) \end{cases} \quad (7.6)$$

where $M_{i,j}$ is the element at the ith row and jth column ($1 \leq i \leq m$, $1 \leq j \leq m_{n'}$).

After creating the M matrix, an alignment can be formed by tracing back from $M_{m,m_{n'}}$. The next matrix element to be traced from $M_{i,j}$ ($1 \leq i \leq m$, $1 \leq j \leq m_{n'}$) is

$$\begin{cases} M_{i-1,j} & \text{if } [M_{i-1,j} + H(c_i, \phi)] \text{ is the minimum.} \\ M_{i-1,j-1} & \text{if } [M_{i-1,j-1} + H(c_i, x_j^{n'})] \text{ is the minimum.} \\ M_{i,j-1} & \text{if } [M_{i,j-1} + H(\phi, x_j^{n'})] \text{ is the minimum.} \end{cases} \quad (7.7)$$

The result of the tracing is a string of index pairs. A pair indicates that the indexed column and subunits should be aligned. The index pair recorded into the string when tracing from $M_{i,j}$ is determined by the next element:

$$\begin{cases} (i, \phi) & \text{if } M_{i-1,j} \text{ is the next.} \\ (i, j) & \text{if } M_{i-1,j-1} \text{ is the next.} \\ (\phi, j) & \text{if } M_{i,j-1} \text{ is the next.} \end{cases} \quad (7.8)$$

Pair (i, ϕ) indicates that the ith column of the synthesized sequence corresponds to a deletion in the n'th sequence, (i, j) indicates that the ith column of the synthesized sequence should be aligned to the jth subunit of the n'th sequence, and (ϕ, j) indicates that the jth subunit of the n' sequence is an insertion,

7.3 Complexity Analysis of the Combined Approaches

In this section, the complexity of the combined methods is analyzed in terms of computing time. The complexity is also compared with a "pure" pairwise dynamic programming method.

The computing time required by the combined method to align a set of sequences has two components: the time required by the genetic algorithm and the time required by a dynamic programming procedure.

As discussed in Section 4.3, when population size is determined as $Q = m_a n/100$ where m_a is the average length of the sequence set and n is the number of sequences, we have the complexity of the genetic algorithm:

$$\frac{n}{2} m_a \log[m_a n/(100k^{1/2})] \tag{7.9}$$

where k is the number of intervals for recursive application of the genetic algorithm.

In aligning two sequences of average length m_{avg}, the computing time for a dynamic programming method is estimated as $O(m_{avg}^2)$. When using a dynamic programming method to align n sequences of average length m_{avg}, both the pairwise and synthesis methods involve $(n-1)$ times of two-sequence alignment and the computing time is

$$O[m_{avg}^2(n-1)]. \tag{7.10}$$

Recall the description in the chapter of "The Genetic Algorithm for Multiple Sequence Alignment", that any interval would be re-processed by the genetic algorithm if its average length is longer than $m_a/20$. We thus assume the average length of the intervals, denoted as m_s, is not longer than $m_a/20$. The number of intervals to be re-processed is less than m_a/m_s. By substituting $m_{avg} = m_s$ into (7.10) and multiplying the expression by m_a/m_s, we have the computing time required by a dynamic programming method to align all the intervals:

$$(m_a/m_s)m_s^2(n-1) = m_a m_s(n-1). \tag{7.11}$$

Therefore the total time required by the combined approaches is of the order of:

$$\frac{n}{2} m_a \log[m_a n/(100k^{1/2})] + m_a m_s(n-1). \tag{7.12}$$

When a pure pairwise dynamic programming method is applied to the same data set (the whole sequences), the computing time can be estimated as $O[m_a^2(n-1)]$.

The combined approaches have smaller time complexity than a pure dynamic programming: For $2 \leq n \ll m_a$, $n < 100k^{1/2}$, $1 \leq k \leq 20$, $m_s \leq m_a/20$ and $m_a \geq 10$, we have

$$\begin{aligned}
& \frac{\frac{n}{2} m_a \log[m_a n/(100k^{1/2})] + m_a m_s(n-1)}{m_a^2(n-1)} \\
= & \frac{\frac{n}{2} \log[m_a n/(100k^{1/2})] + m_s(n-1)}{m_a(n-1)} \\
< & \frac{(n-1)\log[m_a n/(100k^{1/2})] + m_s(n-1)}{m_a(n-1)} \\
= & \frac{\log[m_a n/(100k^{1/2})] + m_s}{m_a}
\end{aligned}$$

$$< \frac{\log(m_a) + m_s}{m_a}$$

$$< 1. \tag{7.13}$$

Equation (7.13) indicates that the ratio may be smaller for longer sequences.

7.4 Experimental Results

The two combined methods of the genetic algorithm and the dynamic programming have been applied to the molecular sequence sets. In this section, the results of aligning the sequences are presented, analyzed, and also compared with the results obtained using CLUSTALW.

7.4.1 Results of aligning the data sets

Listed in Table 7.1 are the processing time and the four quality measurements of the alignments of the eight protein sequences (bacterial porin). In this table, as well as in the following discussion, we use "G+D" to denote the combined method of the pairwise dynamic programming and the genetic algorithm, "G+S" the combined method of the sequence synthesis and the genetic algorithm, and "C" CLUSTALW.

Table 7.2 lists the ratios of the measurements in Table 7.1. The following facts can be observed for the protein data set. The time required by the combined approaches is approximately one order of magnitude lower than the time required by CLUSTALW. The "G+D" approach is slightly faster than the "G+S" approach. In most cases, the combined approaches have larger numbers of matches. The exception is the alignments of two sequences for which the dynamic programming-based CLUSTALW may produce the optimal result.

"G+D" and "G+S" have achieved even better alignment quality: The average

# of sequences	Average length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
2	335	G+D	1s496ms	208	1514	1130	49.97
		G+S	1s476ms	209	1506	1167	49.67
		C	8s524ms	211	1687	1312	39.74
3	333	G+D	1s962ms	193	5073	4169	55.45
		G+S	2s757ms	192	5056	4186	55.25
		C	14s257ms	189	5224	4320	49.19
4	336	G+D	2s773ms	189	10381	8868	66.46
		G+S	3s728ms	188	10355	8989	66.15
		C	21s860ms	187	10654	9207	50.99
5	338	G+D	3s524ms	159	16605	13396	100.61
		G+S	5s782ms	160	16339	13197	98.94
		C	29s881ms	155	16936	14176	91.30
6	341	G+D	3s963ms	152	24715	19185	116.70
		G+S	7s244ms	153	24258	18517	113.76
		C	39s012ms	151	25081	21625	109.72
7	341	G+D	4s591ms	145	34703	26406	128.93
		G+S	9s679ms	145	33949	25143	124.20
		C	46s125ms	145	34836	28014	121.93
8	341	G+D	4s965ms	145	46455	35642	132.31
		G+S	11s279ms	145	45432	34128	127.57
		C	53s663ms	143	46457	37414	125.35

Table 7.1: Processing time and quality measurements for the alignments of the protein data set by the three programs.

ratio of the number of matches is 1.01 for both of the combined methods. The average ratio of ScoreG is 0.97 for "G+D" and 0.96 for "G+S" and the average ratio of ScoreC is 0.93 for "G+D" and 0.92 for "G+S". The entropy values are higher than those of CLUSTALW which is 1.42 of "G+D" and 1.39 of "G+S"

Figure 7.1 and Figure 7.2 illustrate the eight protein sequences aligned by the two combined methods respectively. Again, the alignments are subdivided into successive segments, with each containing 60 subunits of the first sequence. It can be observed that both methods have correctly identified most of the matched subunits and discovered the overall relationships among the sequences. There are 403 columns in the result of "G+D" and 401 columns in that of "G+S", while there are 395 columns in that of CLUSTALW. Less compactness generated by the two combined methods is the partial reason of higher entropy values.

# of sequences	Ratio type	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
2	G+D/C	1.76×10^{-1}	0.99	0.90	0.86	1.26
	G+S/C	1.73×10^{-1}	0.99	0.89	0.89	1.25
3	G+D/C	1.38×10^{-1}	1.02	0.97	0.97	1.20
	G+S/C	1.93×10^{-1}	1.02	0.97	0.97	1.20
4	G+D/C	1.27×10^{-1}	1.01	0.97	0.96	1.45
	G+S/C	1.71×10^{-1}	1.01	0.97	0.98	1.44
5	G+D/C	1.18×10^{-1}	1.03	0.98	0.95	1.51
	G+S/C	1.94×10^{-1}	1.03	0.97	0.93	1.48
6	G+D/C	1.02×10^{-1}	1.01	0.99	0.89	1.51
	G+S/C	1.86×10^{-1}	1.01	0.97	0.86	1.47
7	G+D/C	9.95×10^{-2}	1.00	1.00	0.94	1.51
	G+S/C	2.10×10^{-1}	1.00	0.98	0.90	1.45
8	G+D/C	9.25×10^{-2}	1.01	1.00	0.95	1.51
	G+S/C	2.10×10^{-1}	1.01	0.98	0.91	1.45
Average	G+D/C	1.22×10^{-1}	1.01	0.97	0.93	1.42
	G+S/C	1.91×10^{-1}	1.01	0.96	0.92	1.39

Table 7.2: "G+D"/"C" and "G+S"/"C" ratios of the measurements in Table 7.1. The columns marked with an "$\star$" contain the ratios.

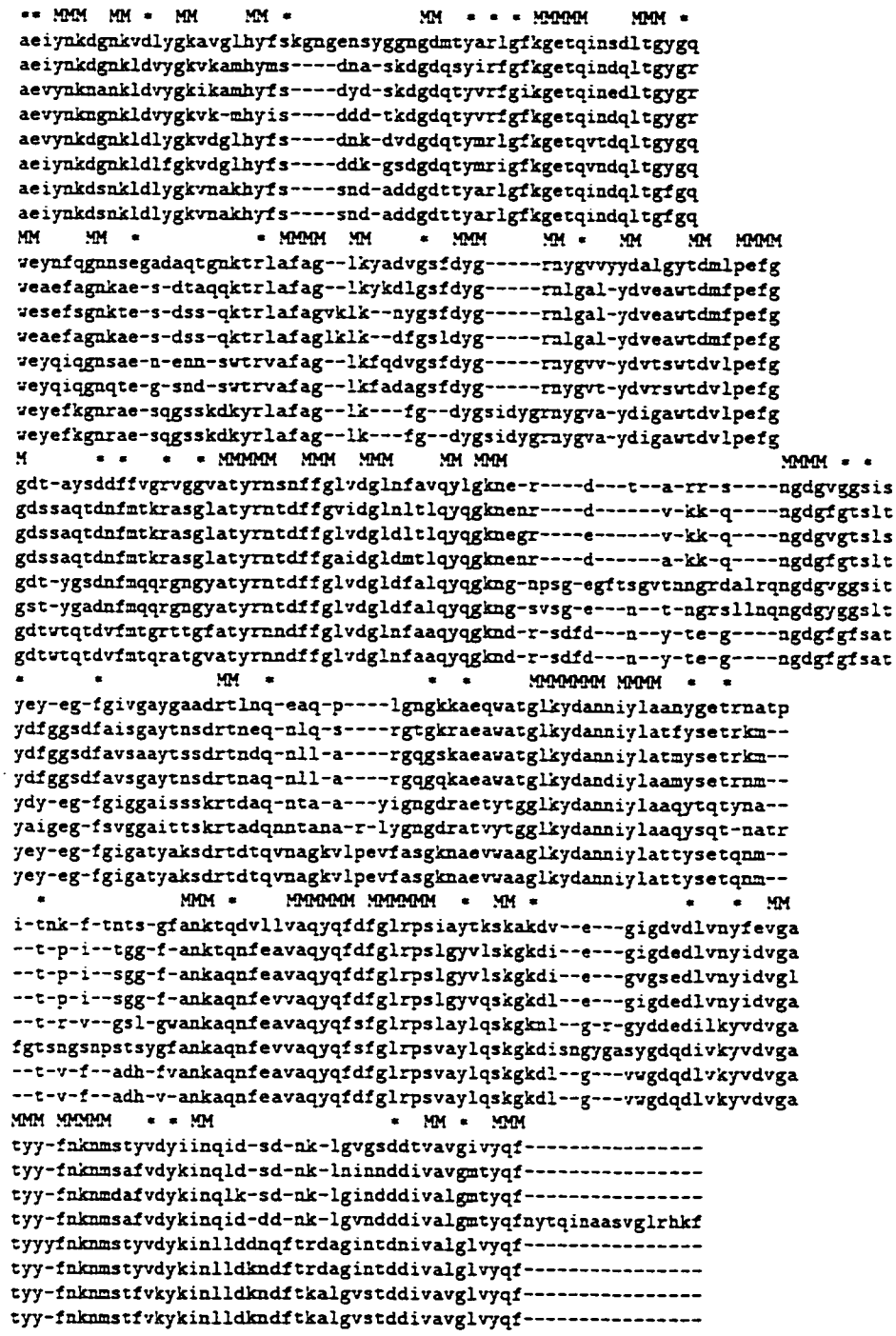


Figure 7.1: Alignment of the eight protein sequences by the combined method of "G+D". An "M" indicates a match tuple found by the genetic algorithm and an "*" indicates a match tuple found by the pairwise dynamic programming.

Figure 7.2: Alignment of the eight protein sequences by the combined method of "G+S". An "M" indicates a match tuple found by the genetic algorithm and an "*" indicates a match tuple by the synthesis method.

7.4.2 Results of aligning the mRNA data set

As shown in Table 7.3, in aligning the mRNA data set of ten sequences the alignments by the combined methods have more matches, higher ScoreG and slightly lower ScoreC, which imply better overall alignment quality. The processing time required by the combined approaches are approximately two orders of magnitude lower than that by CLUSTALW.

It is observed from Table 7.4.2 that the ratio differences between the time required by the combined methods and CLUSTALW increases as the sequence length increases, while the ratios of the four quality measurements remain almost unchanged. For the "G+D" approach, the ratio of the processing time is 2.29×10^{-2} at length 1000 and 2.09×10^{-3} at 9000. For the "G+S" approach, the ratio of the processing time is 5.20×10^{-2} at length 1000 and 6.79×10^{-3} at 9000.

Both of the combined approaches can identify more matches in their results than CLUSTALW. The ratios of the average number of matches are 1.04 and 1.02 respectively. The results of "G+D" have the same ScoreG values (indicated by an average ratio of 1.00 in Table 7.4), while the results of "G+S" have slightly higher ScoreG values (indicated by an average ratio of 1.01 in Table 7.4). The results of both the combined approaches have lower ScoreC values. The ratios of the average ScoreC values are 0.93 and 0.98 respectively. CLUSTALW can handle mismatches better and produce more compact alignments. This fact is indicated by the average ratios

Length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
1000	G+D	10s177ms	833	257334	273267	107.09
	G+S	23s125ms	805	258188	284212	46.44
	C	9m24s352ms	797	257813	288591	43.82
2000	G+D	24s934ms	1561	499024	520565	370.13
	G+S	1m41s043ms	1518	502639	544566	239.44
	C	37m50s329ms	1508	501539	556687	231.86
3000	G+D	38s567ms	2388	755222	769839	459.21
	G+S	2m15s304ms	2343	762089	805767	278.33
	C	1hr25m46s520ms	2322	759854	828861	271.94
4000	G+D	50s608ms	3222	1003359	1031086	677.51
	G+S	2m22s581ms	3165	1016043	1086503	393.71
	C	2hr41m30s783ms	3143	1015712	1116731	350.66
5000	G+D	1m14s812ms	4047	1276086	1305614	903.35
	G+S	4m22s814ms	3979	1293107	1375231	538.42
	C	4hr17m53s542ms	3877	1265871	1390669	464.08
6000	G+D	1m18s221ms	4819	1502280	1521794	1040.78
	G+S	4m28s969ms	4756	1524194	1605188	588.13
	C	6hr25m16s070ms	4732	1522469	1659179	508.13

Table 7.3: Processing time and quality measurements for the alignments of the mRNA data set by the three programs.

Length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
7000	G+D	1m32s342ms	5684	1761652	1805763	1164.11
	G+S	4m40s999ms	5605	1783838	1898197	648.42
	C	8hr41m23s699ms	5587	1784108	1956777	569.71
8000	G+D	1m40s605ms	6496	2014351	2062907	1315.56
	G+S	6m16s697ms	6410	2040591	2169808	773.85
	C	11hr04m55s431ms	6383	2040507	2237351	649.06
9000	G+D	1m45s946ms	7374	2271833	2317453	1421.85
	G+S	5m39s437ms	7295	2299458	2431209	836.54
	C	14hr03m13s902ms	7266	2300673	2512553	680.06
10000	G+D	1m57s829ms	8239	2530484	2598895	1516.16
	G+S	5m59s612ms	8142	2559454	2720041	894.24
	C	NA	NA	NA	NA	NA
11000	G+D	2m09s509ms	9125	2787907	2865356	1651.58
	G+S	6m02s201ms	9017	2819916	2998565	979.16
	C	NA	NA	NA	NA	NA
12000	G+D	2m18s773ms	9879	3008421	3082497	1822.23
	G+S	6m42s825ms	9763	3042945	3225112	1106.97
	C	NA	NA	NA	NA	NA

Table 7.3: (Continued) Processing time and quality measurements for the alignments of the DNA data set by the three programs. "NA" - not available.

Length	Ratio type	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
1000	G+D/C	2.29×10^{-2}	1.05	1.00	0.95	2.76
	G+S/C	5.20×10^{-2}	1.01	1.00	0.99	1.19
2000	G+D/C	1.10×10^{-2}	1.04	1.00	0.94	2.66
	G+S/C	4.45×10^{-2}	1.01	1.00	0.98	1.72
3000	G+D/C	7.49×10^{-3}	1.04	1.00	0.94	2.66
	G+S/C	2.63×10^{-2}	1.01	1.00	0.98	1.72
4000	G+D/C	5.22×10^{-3}	1.03	0.99	0.92	2.98
	G+S/C	1.47×10^{-2}	1.01	1.00	0.97	1.73
5000	G+D/C	4.84×10^{-3}	1.05	1.01	0.94	2.99
	G+S/C	1.69×10^{-2}	1.03	1.02	0.99	1.77
6000	G+D/C	3.38×10^{-3}	1.04	1.00	0.93	2.99
	G+S/C	1.16×10^{-2}	1.03	1.02	0.98	1.68
7000	G+D/C	2.95×10^{-3}	1.04	1.00	0.93	3.09
	G+S/C	8.98×10^{-3}	1.02	1.01	0.98	1.80
8000	G+D/C	2.52×10^{-3}	1.03	1.00	0.93	3.07
	G+S/C	9.44×10^{-3}	1.02	1.01	0.98	1.79
9000	G+D/C	2.09×10^{-3}	1.03	1.00	0.93	3.05
	G+S/C	6.79×10^{-3}	1.02	1.01	0.98	1.76
Average	G+D/C	6.91×10^{-3}	1.04	1.00	0.93	2.91
	G+S/C	2.12×10^{-2}	1.02	1.01	0.98	1.68

Table 7.4: G+D/C and G+S/C ratios of the measurements in Table 7.3. The columns marked with an "$\star$" contain the ratios.

of the entropy values (2.91 and 1.68).

7.4.3 Results of aligning the 11 other sequence sets

In addition to the protein and the mRNA data sets, the combined methods have also been applied to the 11 other data sets. The experimental measurements and their ratios are listed in Table 7.5 and 7.6 respectively. For this group of data sets the alignments by the two combined approaches are slightly better on average in terms of the number of matches, ScoreG and ScoreC. The processing time required by the two combined approaches is approximately two orders of magnitude lower than that by CLUSTALW. "G+D" is about one time faster than "G+S". The average "G+D"/"C" ratios are all 1.01 for the three quality measurements and the average "G+S"/"C" ratios are 1.01,1.01 and 1.03 respectively. The alignments by CLUSTALW have lower entropy values.

Data set	# of sequ.	Aver. length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
S1	10	211	G+D	1s141ms	198	56447	71571	3.34
			G+S	1s789ms	198	56491	71847	2.27
			C	27s113ms	197	56491	69010	2.27
S2	5	1780	G+D	11s721ms	1566	101719	120943	101.79
			G+S	15s946ms	1564	101757	121174	98.30
			C	13m8s776ms	1611	103018	115785	43.28
S3	4	2433	G+D	2s032ms	2303	85615	104867	68.38
			G+S	12s088ms	2303	85615	104867	68.38
			C	18m28s150ms	2305	85701	104149	64.86
S4	8	1437	G+D	38s378ms	909	205335	186646	405.43
			G+S	3m48s348ms	898	210355	204021	243.46
			C	15m38s568ms	901	210317	226516	204.88
S5	8	1680	G+D	21s858ms	1427	269237	288583	94.27
			G+S	30s683ms	1420	269432	291406	81.12
			C	21m5s898ms	1439	269092	303390	46.02
S6	4	4582	G+D	8s674ms	4426	161364	181712	117.40
			G+S	1m25s887ms	4420	161224	181928	114.09
			C	1hr4m58s781ms	4409	160860	182475	149.91

Table 7.5: Processing time and quality measurements for the alignments of the 11 data sets by the three programs. Of the data sets, S1, S2, S4 and S5 are DNA while the rest are RNA.

Data set	# of sequ.	Aver. length	Methods	Processing time	# of matches	ScoreG	ScoreC	Entropy
S7	5	1093	G+D	8s566ms	905	60158	73732	97.95
			G+S	11s994ms	898	60266	75769	78.25
			C	5m12s948ms	905	60719	73280	53.90
S8	8	1071	G+D	6s775ms	1002	176490	225802	26.25
			G+S	8s713ms	999	176665	227985	15.65
			C	8m35s650ms	999	176665	216359	15.65
S9	6	1456	G+D	15s549ms	1066	115751	135494	258.26
			G+S	29s195ms	1063	116632	140112	216.81
			C	11m21s416ms	1118	120534	142237	86.97
S10	7	1448	G+D	13s165ms	1182	171236	208936	141.72
			G+S	35s368ms	1128	171741	212366	114.17
			C	13m22s264ms	1193	172756	206195	85.06
S11	5	1092	G+D	8s473ms	847	59497	70381	124.11
			G+S	25s828ms	846	59555	71533	115.12
			C	5m3s838ms	855	60100	71361	90.83

Table 7.5: (Continued) Processing time and quality measurements for the alignments of the 11 data sets by the three programs. Of the data sets, S1, S2, S4 and S5 are DNA while the rest are RNA.

Data set	# of sequ.	Aver. length	Ratio type	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
S1	10	211	G÷D/C	4.21×10^{-2}	1.01	1.00	1.04	1.47
			G÷S/C	6.60×10^{-2}	1.01	1.00	1.04	1.00
S2	5	1780	G÷D/C	1.48×10^{-2}	0.97	0.99	1.05	2.35
			G÷S/C	2.02×10^{-2}	0.97	0.99	1.05	2.27
S3	4	2433	G÷D/C	9.06×10^{-4}	1.00	1.00	1.01	1.05
			G÷S/C	1.83×10^{-3}	1.00	1.00	1.01	1.05
S4	8	1437	G÷D/C	4.09×10^{-2}	1.02	0.98	0.83	1.98
			G÷S/C	2.43×10^{-1}	1.01	1.00	1.00	1.19
S5	8	1680	G÷D/C	1.72×10^{-2}	1.00	1.01	0.96	2.05
			G÷S/C	2.42×10^{-2}	1.00	1.01	0.97	1.76
S6	4	4582	G÷D/C	2.20×10^{-3}	1.00	1.00	1.00	0.78
			G÷S/C	2.20×10^{-2}	1.00	1.00	1.00	0.76
S7	5	1093	G÷D/C	2.74×10^{-2}	1.04	1.02	1.04	1.82
			G÷S/C	3.83×10^{-2}	1.04	1.03	1.07	1.45
S8	8	1071	G÷D/C	1.31×10^{-2}	1.06	1.05	1.10	1.68
			G÷S/C	1.69×10^{-2}	1.06	1.05	1.11	1.00

Table 7.6: G÷D/C and G÷S/C ratios of the measurements in Table 7.5. The columns marked with an "⋆" contain the ratios.

Data set	# of sequ.	Aver. length	Ratio type	Processing time*	# of matches*	ScoreG*	ScoreC*	Entropy*
S9	6	1456	G+D/C	2.28×10^{-2}	0.97	0.98	0.97	2.97
			G+S/C	4.28×10^{-2}	0.97	0.99	1.00	2.49
S10	7	1448	G+D/C	1.64×10^{-2}	1.01	1.01	1.03	1.67
			G+S/C	4.40×10^{-2}	1.01	1.01	1.05	1.34
S11	5	1092	G+D/C	2.79×10^{-2}	1.05	1.04	1.04	1.37
			G+S/C	8.50×10^{-2}	1.05	1.04	1.06	1.27
Average			G+D/C	2.04×10^{-2}	1.01	1.01	1.01	1.74
			G+S/C	5.48×10^{-2}	1.01	1.01	1.03	1.42

Table 7.6: (Continued) G+D/C and G+S/C ratios of the measurements in Table 7.5. The columns marked with an "$\star$" contain the ratios.

7.5 Improvement from the "G" Program

A. Local improvement by the two combined approaches

In this section, we will focus on the improvement achieved by the two combined approaches. First, we examine and compare the results obtained by applying the two approaches as well as the "G" program to an interval segment of the data set S4. This comparison enables us to have an insight into the performance of the two approaches. and to understand the local improvement achieved by them.

Figures 7.3, 7.4 and 7.5 illustrate the corresponding segments taken from the alignments of S4 which are produced by "G+D", "G+S" and "G" respectively. The last interval of these segments is taken for a close examination.

Table 7.7 lists the quality measurements of this interval aligned by the three approaches.

Approach	# of mactches	ScoreG	ScoreC	Length	Entropy
G	74	24551	22944	179	24.82
G+D	107	25887	21320	201	37.81
G+S	102	25966	24958	179	18.19

Table 7.7: Quality measurements of the last interval in Figures 7.3, 7.4 and 7.5.

Examine the alignments of this interval. "G+D" achieves the highest number of matches: 33 more matches than the "G" program. However, the alignment is longer and the entropy is higher. "G+S" achieved the highest values of ScoreC and ScoreG.

Its alignment has the same length as the one by "G" which is 179 subunits long and its entropy value is the lowest.

Of the interval alignment by "G+S", the number of "−"s is three (3), which is the same as that of the alignment by "G". Of the alignment by "G+D", the number of "−"s is 162. This fact suggests that the alignment by "G+S" is similarly compact as the alignment by "G", but the alignment by "G+D" is not. The larger number of "−"s make the alignment by "G+D" less compact and more diversified.

	# of mactches	ScoreG	ScoreC	Entropy
[(G+D)-G]/G	+0.4459	+0.0544	−0.0708	+0.5234
[(G+S)-G]/G	+0.3784	+0.0576	+0.0878	−0.2671

Table 7.8: Ratios of the quality measurements in Table 7.7.

The data listed in Table 7.8 may provide a clearer picture about the improvement made by the two combined approaches. In the table, a plus sign (+) indicates that the value increases and a minus sign (−) indicates that the value decreases. The alignment by "G+D" has the largest increase in the number of matches which is about 44.6%. However, it also has the largest increase in the entropy value which is about 52.3%. The alignment by "G+S" has the largest increases in the score values and a decrease in the entropy value, which are 5.8% and 8.8% for ScoreG and ScoreC respectively, and 26.7% for entropy.

It will be seen in the following discussion of global improvement that the qual-

ity improvement in this interval is typical in the quality improvement achieved by applying the two combined approaches to all the test data sets.

Figure 7.3: Alignment of the eight DNA sequences of S4 by the combined method of G+D. An "M" indicates a match tuple found by the genetic algorithm and an "*" indicates a match tuple found by the pairwise dynamic programming.

```
 MMMMMMM **** * * * MMMMMMMMM * * MMMMMM MMMMMM MMMMMMMMMMMMMMMMMMMMM MMMMM
gattctaaagtgtaatgataaaaagttcaatggaacaggaccatgtacaaatgtcagcacagtacaatgtacacatggaattaggccagt
gattctaaagtgtagagaaaacaagttcaatggaaaaggaccatgtaaagatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaagtgtaaagataagacgttcaatggaacaggaccatgtacaaatgtcagtacagtacaatgtacacatggaattaggccagt
gattctaaagtgtaatgataagaagttcaatggaacaggaccatgtagcaatgtcagcacagtacaatgtacacatggaattaggccagt
aatttctaaagtgtaatgatacaaagttcaatggatcaggaccatgtaccaatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaaatgtaatgaaaagaagttcaatggatcaggaccatgtaaaaatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaagtgtaatgataagaagttcaatggaagtgaaacatgtaaaaatgtcagcacagtacaatgtacacatggaattaggccagt
gattctaaagtgtaataataagacgttcaatggaaaaggaccatgtacaaatgtcagcacagtacaatgtacacatggaattaggccagt

 * MMMM * * MMMMM MMMMMMMMM * MMMMMM * ** ** **** MMMMM *** ** * * *** * * ** MMM
agtatcaactcaactgctgtt---aaatggcagtctagcagaaaaagaaataataattagatctgaaaatttcacaaacaatgccaaaaccat
agtatcaactcaactgctgtt---aaatggcagtctagcagaagaagaggtagtaattagatctgtcaatttctcggacaatgctaaaatcat
agtatcaacccaactgctgtt---aaatggcagcctagcagaagaagaggtaataattagatctgaaaatctcacggacaatactaaaaccat
ggcatcaactcaactactgtt---aaatggcagtctagcagaagaggaggtagtaattagatctcaaaatttcacaaacaatgctaaaatcat
agtatcaactc-a--gctgttgttaaatggcagtcaagcagaagaagaggtagtaattagatctgaaaatttcacaaacaatgctaaaaccat
agtgtcaactcaactgctgtt---aaatggcagtctggcagaagaagagatagtaatcagatctgaaaatttcacgaacaatgctaaaaccat
agtatcaacgcaactgctgtt---aaatggcagtctagcagaagaggaggtagtaattagatctgaaaatttcacaaataatgctaaaaccat
agtatcaacccaaccgctgtt---aaatggcagtctagcagaagaagaggtagtaattagatctgccaatttctcggacaatgctagaaccat

MMMM MMMMMMMMM * ** *** ***** **** ****** ***** ***** * * * *** * * **  ** **
aatagtacagctgaatgattctgtagtaattaattgtacaagacccaacaacaatacaagaaaaagtatacatatggggccaggaagggc
aatagtacagctgaatgaatctgtagaaattaattgtacaagacccaacaacaatacaagacaaggtatacatataggtccagggagagc
aataatacagctgaatgaatctgtagaaattaactgtacaagacccaacaataatacaagaaaaggtataggtataggaccagggagagt
aatagtacagctgaacgagtctgtagaaattaattgtacaagacccaacaacaatacaagaagaagtatacatatgggattcgggagagc
aatagtacagctgaacgaatctgtagtaattaattgtataagaccaaacaacaatacaagaaaaggtatatatataggaccagggagagc
aatagtacagctgaatcaatctatagagattaattgtacaagacccaacaacaatacaagaaaaaggataactatgggaccagggagagt
aatagtacagctgaaagaagctgtagaaattaattgtacaagacccaacaacaatacaagaagaggtatacatttaggactagggagaag
aatagtacagctgaatgaatctgtagaaattaattgtacaagacccaacaactatacagcaaaaaggataagtataggaccagggagagc

* *  *** * *  * ******* * * *  * * *** **** * *** * * ** ****** * ******  ** *
attttatgcaacaggagaaataataggagatatacgacaggcatattgtgacattagtagaacaaaatggaagaacactttaatgcagat
aatttatacaacaggaagaacaataggagatataagacaagcacattgtaacattagtaaagcaaaatggaagaacactttaaaacaggt
attttatgcaacaggaggaataataggagacataagacaagcacattgtaacattagtagagtaaaatggaataacactttaggacagat
attgtatacaaacggagaaataataggaaatataagacaagcacattgtaacattagtggaacaaaatggaataacactttaaaacaggt
attttatacaacagaaaaaataataggaaatataagaaaagcatattgcacccttaataaaacaatatggaataatactttagaaaagat
actctatacaacaggacaaataataggagatataaaaaaagcatattgtaacattagtagagcaaaatggaatgatactttaagaaaggt
attttacacaacag-ac--ataataggagatacaagacaagcacattgtaacattagtacagcaaaatggaataacactttaaaacaggt
atttcgtacaaaaggaagaataataggagatataagacaagcacattgtaccattaataaagcaaaatggaacgaaactttaggaaagat

**  * * ** MMMMMMMMMM
agcaaaaaaactaagagaacaattt
agctaaaaaattacgagaacaattt
agctaaaaaattaagagaacaattt
agttaagaaattaagagaacaattt
agttaacaaattaagagaacaattt
agctataaaattaagagaacaattt
agttgaaagattaagagaacaattt
agttacaaaattaagagaacaattt
```

Figure 7.4: Alignment of the eight DNA sequences of S4 by the combined method of G+S. An "M" indicates a match tuple found by the genetic algorithm and an "*" indicates a match tuple found by the synthesis method.

```
 MMMMMMMM ****   * *  * MMMMMMMMMM   * * MMMMMM   MMMMMMM MMMMMMMMMMMMMMMMMMMMMMMM MMMMMM
gattctaaagtgtaatgataaaaagttcaatggaacaggaccatgtacaaatgtcagcacagtacaatgtacacatggaattaggccagt
gattctaaagtgtagagaaaacaagttcaatggaaaaggaccatgtaaagatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaagtgtaaagataagacgttcaatggaacaggaccatgtacaaatgtcagtacagtacaatgtacacatggaattaggccagt
gattctaaagtgtaatgataagaagttcaatggaacaggaccatgtagcaatgtcagcacagtacaatgtacacatggaattaggccagt
aattctaaagtgtaatgatacaaagttcaatggatcaggaccatgtaccaatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaaatgtaatgaaaagaagttcaatggatcaggaccatgtaaaaatgtcagcacagtacaatgtacacatggaattaagccagt
gattctaaagtgtaatgataagaagttcaatggaagtgaaacatgtaaaaatgtcagcacagtacaatgtacacatggaattaggccagt
gattctaaagtgtaataataagacgttcaatggaaaaggaccatgtacaaatgtcagcacagtacaatgtacacatggaattaggccagt

 *  MMMM * *   MMMMM   MMMMMMMM * MMMMM * **  ** **** MMMMMM   *** ** *  * *** * * ** MM
agtatcaactcaactgctgtt---aaatggcagtctagcagaaaaagaaataataattagatctgaaaatttcacaaacaatgccaaaaccat
agtatcaactcaactgctgtt---aaatggcagtctagcagaagaagaggtagtaattagatctgtcaatttctcggacaatgctaaaatcat
agtatcaacccaactgctgtt---aaatggcagcctagcagaagaagaggtaataattagatctgaaaatctcacggacaatactaaaaccat
ggcatcaactcaactactgtt---aaatggcagtctagcagaagaggaggtagtaattagatctcaaaatttcacaaacaatgctaaaatcat
agtatcaactc-a--gctgttgttaaatggcagtcaagcagaagaagaggtagtaattagatctgaaaatttcacaaacaatgctaaaaccat
agtgtcaactcaactgctgtt---aaatggcagtctggcagaagaagagatagtaatcagatctgaaaatttcacgaacaatgctaaaaccat
agtatcaacgcaactgctgtt---aaatggcagtctagcagaagaggaggtagtaattagatctgaaaatttcacaaataatgctaaaaccat
agtatcaacccaaccgctgtt---aaatggcagtctagcagaagaagaggtagtaattagatctgccaatttctcggacaatgctagaaccat


MMMM MMMMMMMMM  *  ** ***  ***** **** ****** *****  *****  *  * * ***  * * **    ** **
aatagtacagctgaatgattctgtagtaattaattgtacaagacccaacaacaatacaagaaaaagtatacatatggggccaggaagggc
aatagtacagctgaatgaatctgtagaaattaattgtacaagacccaacaacaatacaagacaaggtatacatataggtccagggagagc
aataatacagctgaatgaatctgtagaaattaactgtacaagacccaacaataatacaagaaaaggtataggtataggaccagggagagt
aatagtacagctgaacgagtctgtagaaattaattgtacaagacccaacaacaatacaagaagaagtatacatatgggattcgggagagc
aatagtacagctgaacgaatctgtagtaattaattgtataagaccaaacaacaatacaagaaaaggtatatatataggaccagggagagc
aatagtacagctgaatcaatctatagagattaattgtacaagacccaacaacaatacaagaaaaaggataactatgggaccagggagagt
aatagtacagctgaaagaagctgtagaaattaattgtacaagacccaacaacaatacaagaagaggtatacatttaggactagggagaag
aatagtacagctgaatgaatctgtagaaattaattgtacaagacccaacaactatacagcaaaaaggataagtataggaccagggagagc

* *    ***  *   ** * **    * *  *       *       **   * *    **        *     *
attttatgcaacaggagaaataataggagatatacgacaggcatattgtgacattagtagaacaaaatggaagaacactttaatgcagat
aattttatacaacaggaagaacaataggagatataagacaagcacattgtaacattagtaaagcaaaatggaagaacactttaaaacaggt
attttatgcaacaggaggaataataggagacataagacaagcacattgtaacattagtagagtaaaatggaataacactttaggacagat
attgtatacaaacggagaaataataggaaatataagacaagcacattgtaacattagtggaacaaaatggaataacactttaaaacaggt
attttatacaacagaaaaaataataggaaatataagaaaagcatattgcacccttaataaaacaatatggaataatactttagaaaagat
actctatacaacaggacaaataataggagatataaaaaaagcatattgtaacattagtagagcaaaatggaatgatactttaagaaaggt
attttacacaacagacataataggagatacaagacaagcacattgtaacattagtacagcaaaatggaataacactttaaaacaggtagt
atttcgtacaaaaggaagaataataggagatataagacaagcacattgtaccattaataaagcaaaatggaacgaaactttaggaaagat

 *  *  * * ** MMMMMMMMMM
agcaaaaaaactaagagaacaattt
agctaaaaaattacgagaacaattt
agctaaaaaattaagagaacaattt
agttaagaaattaagagaacaattt
agttaacaaattaagagaacaattt
agctataaaattaagagaacaattt
tg--a-aagattaagagaacaattt
agttacaaaattaagagaacaattt
```

Figure 7.5: Alignment of the eight DNA sequences of S4 by the "G" program. An "M" indicates a match tuple found by the genetic algorithm and an "*" indicates a match tuple found by the shift-up method.

B. Global improvement by the two combined approaches

For a data set, the three approaches align the intervals in three different ways. The alignments of the entire data set by the three approach are thus different in the intervals. The quality of an entire alignment partially depends on the quality of each interval alignment. Therefore, the global improvement achieved by a combined approach is characterized by its local improvement. A comparison between Tables 7.8 and 7.9 may verify this point.

The data in Table 7.9 are calculated from Tables 6.1 and 7.1, Tables 6.3 and 7.3, and Tables 6.5 and 7.5 for the study of the global improvement. First, the increases/decreases of the quality measurements are calculated. Then the ratios of the increases/decreases to the measurements of the alignments of "G" are computed. The data in the table are the average and the standard deviation for each data set.

The performance of the two combined approaches can be analyzed from the table:

- The major improvement made by the "G+D" approach is in the number of matches. In most cases, "G+D" is able to achieve the highest number of matches. The increase in the number of matches ranges from 2% to 3%. However, its alignments have poor entropy values.

- The major improvement made by the "G+S" approach is the better overall quality. Compare with "G" program, the alignments by "G+S" have higher number of matches, higher valies of ScoreG and ScoreC, and lower values of

entropy. Compare the three approaches. in most cases. "G+S" has the highest values of ScoreG and ScoreC, and the best entropy values.

Data set			# of matches*	ScoreG*	ScoreC*	Entropy*
Protein	[(G+D)-G]/G	average	+0.0248	+0.0389	+0.0721	−0.0300
		standard dev.	0.0150	0.0273	0.0593	0.0600
	[(G+S)-G]/G	average	+0.0259	+0.0251	+0.0571	−0.0473
		standard dev.	0.0179	0.0189	0.0398	0.0946
mRNA	[(G+D)-G]/G	average	+0.0292	−0.0063	−0.0453	+0.7405
		standard dev.	0.0047	0.0128	0.0907	0.1100
	[(G+S)-G]/G	average	+0.0120	+0.0046	+0.0012	−0.0013
		standard dev.	0.0015	0.0009	0.0012	0.0158
Other	[(G+D)-G]/G	average	+0.0192	+0.0080	−0.0029	+0.1837
		standard dev.	0.0208	0.0104	0.0160	0.1342
	[(G+S)-G]/G	average	+0.0117	+0.0116	+0.0111	−0.0562
		standard dev.	0.0194	0.0143	0.0143	0.1123

Table 7.9: [(G+D)-G]/G and [(G+S)-G]/G ratios of the measurements in Chapters 6 and 7. The columns marked with an "⋆" contain the average and the standard deviation of the ratios.

7.6 Summary

The two combined approaches have integrated the strengths from both the genetic algorithm and the dynamic programming: The genetic algorithm enables the approaches to rapidly identify match blocks and the dynamic programming enables them to cope with mismatches in a near-optimal manner. Combining the two advanced techniques, these approaches are able to deal with multiple, long and complex sequence data efficiently. These strengths make the combined approaches a promising tool to be applied to molecular sequence data in practical applications.

The following can be summarized from the experimental results:

- **Compared with CLUSTALW**, the two combined approaches achieved very comparable alignment quality: The number of matches of the two approaches are higher, the score values are similar and the entropy values are higher. The processing time was only 1/100 to 1/10 of that by CLUSTALW. For longer sequences, the computation is out of the feasible bounds of CLUSTALW.

- **Compared with the "G" program**, the combined approaches can achieve better alignment quality: "G+D" may produce alignments of the highest number of matches and "G+S" may produce better alignments than "G" in terms of all the four quality measurements. The processing time required by the two combined methods is no more than six times of that by the "G" program.

- **Compare the two approaches.** Overall, "G+S" may produce better align-

ments than "G+D". It may achieve higher scores and lower entropy values in most cases, while the time required by the former is two to three times of that by the latter.

- The weakness of the combined approaches is less compactness in comparison with CLUSTALW, and longer processing time in comparison with the "G" program.

Chapter 8

Conclusion and Future Work

In this chapter, we conclude the thesis by summarizing the major strengths of the genetic algorithm-based approaches, and then propose some future projects for our ongoing research, in which a genetic algorithm may be a promising tool to address challenging tasks in computational biology.

8.1 The Conclusion

In this research, we have exploited a new path to multiple sequence alignment. Along this path, an alignment technique, with an enhanced genetic algorithm as its core component, has been developed which has the remarkable strength of high efficiency for simultaneous multiple sequence alignment. Such strength is especially significant when aligning long sequences.

Differing from the traditional alignment techniques, this new technique converts the multiple sequence alignment problem into a search problem so that a genetic algorithm can be used as an efficient search tool to achieve good solutions. Differing from the standard genetic algorithms, the genetic algorithm designed for pre-aligning sequences has enhanced characteristics including numeric and index gene representation, variable gnetic string lengths, and additional functionality, which are aimed at tackle the problem in a more efficient manner. The technique enables good match blocks to have higher survival probability and to serve as building blocks in favor of a pre-alignment of an overall better quality. More information has been incorporated into the search process including contextual information as well as block and interval lengths. Compared with an approach mainly based on pairwise dynamic programming, the technique has a much smaller time complexity. It is also efficient in terms of space complexity.

To the author's knowledge, this research was the first published one (Zhang, 1994;

Zhang and Wong, 1997a; Zhang and Wong, 1997b) in which genetic algorithms have been successfully applied to the challenging problem of multiple molecular sequence alignment and achieved high efficiency as well as good quality.

A major breakthrough of this technique is in the processing time. The experimental results have indicated that the processing time required by the technique is about two to three orders of magnitude lower than that by a widely used conventional method – CLUSTALW. Quite often, for the same sequences, while CLUSTALW may require hours to align, a genetic algorithm-based approach may produce results of similar quality within a few minutes (even seconds). Such high efficiency can hardly be achieved by the conventional methods.

Overall, the alignments generated by the genetic algorithm-based approaches have comparable quality as those generated by CLUSTALW in the four quality measurements with the average ratios in processing time range from 1.91×10^{-1} to 3.67×10^{-3}. The average ratios in the number of matches range from 0.98 to 1.04, the average ratios in the ScoreG values range from 0.94 to 1.01, the average ratios in the ScoreC values range from 0.87 to 1.03, and the average ratios in the entropy values range from 1.13 to 2.91.

The alignments generated by the genetic algorithm-based approaches can correctly identify 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, for example, gene bank retrieval, common pattern

or subsequence search, sequence phylogenetic relation identification, and many others. 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.

Each of the three genetic algorithm based approaches has its own strength: The "G" approach is the fastest one and can produce the most compact alignments; the "G+D" approach is able to identify the highest number of matches; and the "G−S" may produce alignments with the best overall quality. From the assessment of their relative strengths, we would have a better guide line to choose the most suitable approaches according to the requirements and the nature of applications.

The additional advantages of the genetic algorithm-based approaches of multiple sequence alignment include simultaneousness, automated processing, easy implementation, and simple data structures. These approaches do not need any extra interactions, for example, 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 in this thesis involves coding the three simple operations only, and requires a one dimensional array of strings.

With the above strengths, the genetic algorithm-based technique can be developed into a practical approach for efficient multiple sequence alignment. In addition

to computational biology, in many fields when information can be represented as sequences, such as speech recognition, language interpretation, and biomedical signal analysis, the technique developed in this research can be adopted as an effective tool for recognizing patterns, examining similarity, identifying common features, and so on. This technique will have great potential and excellent capabilities in those fields.

8.2 Further Improvements and Applications

8.2.1 Improving the combined genetic algorithm-synthesis approach

The combined genetic algorithm-synthesis approach may generate alignments with more matches and comparable ScoreG and ScoreC values with those produced by CLUSTALW. However, the alignments by the combined approach have higher entropy values in the comparison with those of CLUSTALW. This implies that the alignments have higher diversity and lower levels of compactness. This may be largely due to the way and the order in which subsequences in intervals are synthesized.

Currently, to align $n' > 2$ subsequences, $(n'-1)$ subsequences are first aligned, and then their synthesized subsequence is aligned with the n'th subsequence. The order for aligning the subsequences is arbitrary. It has been noted that when subsequences of similar patterns are aligned first, a synthesis method may produce good results, but when dissimilar subsequences are aligned first, it may produce poor results. The key to further quality improvement is the order for synthesizing the sequences.

It is expected that the alignment quality can be further improved by using a hierarchical approach (Chan, 1990; Chan and Wong, 1991; Wong, et al, 1996), in which instead of arbitrarily aligning subsequences, similarity between subsequences is first examined and then used to group subsequences into a similarity hierarchy. A

synthesized sequence is generated for each group of relatively similar subsequences and is aligned with the synthesized sequences of other groups. In this way the global entropy values of the generated alignment will be further reduced, and thus lower diversity and higher compactness can be achieved. The key to this improvement is to use an efficient method for measuring the similarity.

8.2.2 Pattern matching in sequence databases

A direct extension of the thesis project is the use of the genetic algorithm for pattern matching in sequence databases. With the development of rapid DNA sequencing technology, more than 1.6 billion base pairs of nucleotides are sequenced per year (Friedland and Kedes, 1985; Lander, et. al., 1991). The sequences are stored in various sequence databases. Such databases are likely to become the profound research tools of biologists over the next decades (Lander, et. al., 1991).

An important application of the sequence databases is matching target patterns against sequences in the databases (Lander, et. al., 1991; Sibbald and Argos, 1990). Since the current sequence databases are simply organized as sequential files and do not have indices on sequence structures, pattern matching has to be done in the way of sequential search (Kehoe, 1990; Seuchter and Skolnick, 1988; Urrows, 1992). This is a very time-consuming task. It involves performing approximate pattern matching on every sequence in the database, which may contain tens of thousands of sequences and each sequence may be of several thousand subunits. The current approaches of approximate matching (e.g. dynamic programming) are obviously inefficient.

With the strength of high efficiency, the genetic algorithm developed in this research can be modified for pattern matching in sequence databases. For each sequence in a database, it may determines the similarity with the target very quickly. The sequence(s) matches the target best can then be selected.

8.2.3 Modeling molecules

Construction of accurate three-dimensional models of organic/biological molecules is an important task of computational biology (Perczel and Csizmadia, 1995; Cohen et al. 1990). One of the simplest and most reliable ways is to use libraries of typical molecular fragments and data banks, such as the Cambridge X-ray Crystallographic Data Base which contains about 50,000 structures, and the Brookhaven Protein Data Bank (Bernstein et al. 1977; Humblet and Dunbar, 1993). A molecule is constructed by assembling pre-existing fragments, followed by successive adjustments of the current structure which allows the user's full control over building a reasonable starting conformation with the desired stereochemistry.

The currently used approaches involve initially drawing chemical structures as a two-dimensional sketch describing the atom types (element and hybridization) and connectivity, along with some other methods of specifying stereochemistry. While in principle these are simple and intuitive approaches, there is still a great challenge for robustness to convert the initial information into reasonable low energy conformations. Most of these approaches often become trapped quickly into poor local minima during the conversion of two dimensional sketches to three dimensional structures.

In view of this, genetic algorithms can provide a robust and efficient method for modeling molecules. A conversion of a two-dimensional sketch into a three-dimensional structure can be represented as a point in the solution space. Working

with a population of solution points. a genetic algorithm may. in an efficient manner. minimize the chances of having the solution being trapped into poor local energy minima.

8.2.4 Modeling drug-receptor interactions

Modeling molecular binding interactions is of great application significance in finding better enzyme inhibitors or more selective drugs. The major interactions involved in enzyme or drug-receptor binding are electrostatic (including hydrogen bonding), dispersion, and hydrophobic. Hydrophobic interactions usually provide the major driving force for binding (Cohen et al. 1990).

In the case of drug-receptor interactions, the conventional drug-receptor "docking" is typically conducted interactively using molecular surface displays to guide the fitting, based on hydrophobic or electrostatic potential color coding. In determining drug-receptor docking, the user follows a path in a sort of interactive energy minimization. However, due to a huge degree of freedom, the conventional energy minimization approaches are easily trapped into local minima and can give deceptive results (Cohen et al. 1990).

A genetic algorithm would provide a better approach in modeling this kind of interactions. By representing different enzyme or drug-receptor binding as solution points, and associating each solution with an energy level, a genetic algorithm may provide an efficient way to minimize the chances of being trapped into local minima and produce a solution of the minimal or a near-minimal energy.

Appendix I

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

Appendix II

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

Bibliography

[1] Abrahams, J. P., M. van de Berg, E. van Batenbeug, and C. Pleij. 1990. 'Prediction of RNA Secondary Structure, Including Pseudoknotting, by Computer simulation'. *Nucleic Acids Res.*, Vol. 18, pp. 3035-3044.

[2] Albert, B., D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson, 1983. *Molecular Biology of the Cell*, Garland Publishing, Inc.

[3] Altschul, S., and B. W. Erichson. 1986, 'Optimal Sequence Alignments Using Affine Gap Costs'. *Bull. Math. Biol.*, Vol. 48, pp. 606-616.

[4] Ankenbrandt, C. A., B. P. Buckles, and F. E. Petry, 1990, 'Scence Recognition Using Genetic Algorithms with Semantic Nets', *Pattern Recognition Letters 11*. Elseeevier North-Holland, New York, NY, pp.285-293.

[5] Ankenbrandt, C. A., 1991, 'An Extension to the Theory of Convergence and a Proof of the Time Complexity of Genetic Algorithms', in B. M. Spatz eds. *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, Inc.

[6] Auger, I. E., and C. E. Lawrence, 1989, 'Algorithms for the Optimal Identification of Segment Neighborhoods', *Bulletin of Mathematical Biology*, Vol. 51, No. 1, pp. 39-54.

[7] Barber, M. N., and B. W. Ninham, 1970, *Random and Restricted Walks*, Gordon and Breach, New York.

[8] van Batenbury, F. H. D., A. P. Gultyaev and C. W. A. Pleij, 1995, 'An APL-programmed Genetic Algorithm for Prediction of RNA Secodary Structure', *J. theor. Biol.*, Vol. 174, pp. 269-280.

[9] Bendiktsson, J. A., P. H. Swain and O. K. Ersoy, 1990, 'Neural Network Approachs Versus Statistical Methods in Classification of Multisource Remote Sensing Data', *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 28, pp. 540-552.

[10] Bethke, A. D., 1981, *Genetic Algorithms as Function Optimizer*, Ph.D. Thesis, University of Michigan.

[11] Booker, L. B., 1985, 'Improving the Performace of Genetic Algorithms in Classifier Systems', *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 80-92.

[12] Booker, L. B., 1987, 'Improving Search in Genetic Algorithms', in L. Davis eds. *Genetic Algorithms and Simulated Annealing*, London: Pitman., pp. 61-73.

[13] Buckles, B. P., and F. E. Petry, 1992, *Genetic Algorithms*, IEEE Computer Society Press, LosAlamitos, California.

[14] Buckles, B. P., F. E. Petry, and R. L. Kuester, 1990, 'Schema Survival Rates and Heuristic Search in Genetic Algorithms', *Proceedings of Tools for Artificial Intelligence*, IEEE computer Society Press, Los Alamitos, CA, pp. 322-327.

[15] Chan, S. C., 1990, *A Hierarchical Sequence Synthesis Procedure*, Ph. D. Thesis, University of Waterloo.

[16] Chan, S. C., and A. K. C. Wong, 1991, 'Synthesis and Recognition of Sequences', *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 12, pp. 1245-1255.

[17] Chan, S. C., A. K. C. Wong, and D. K. Y. Chiu, 1992, 'A Survey of Multiple Sequence Comparison Methods', *Bull. Math. Biol.*, Vol. 54, No. 4, pp. 563-598.

[18] Cohen, N. C., J. M. Blaney, C. Humblet, P. Gund, and D. C. Barry, 1990, 'Molecular Modeling Software and Methods for Medicinal Chemistry', *J. Med. Chem.*, Vol. 33, No. 3, pp. 883-895.

[19] Dandekar, T., and P. Argos, 1992, 'Potential of Genetic Algorithms in Protein Folding and Protein Engineering Simulations", *Protein Engineering*, Vol. 5, No. 7 pp. 637-545.

[20] Davidor, Y., 1990, 'Robot Programming with a Genetic Algorithm', *IEEE International Conference Procedings on Computer Systems and Software Engineering*, Tel-Aviv: Israel.

[21] Dayhoff, M. O., 1978, 'A Model of Evolutionary Change in Proteins. Matrices for Detecting Distant Relationship', in Dayhoff eds. *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 3.

[22] De Jong, K., 1988, 'Learning with Genetic Algorithms: an Overview', *Machine Learning 3* Kluwer Academic, Hingham, Mass. pp. 121-138.

[23] Dill, K. A., 1990, 'Dominant Forces in Protein Folding', *Biochemistry*, Vol. 29, pp. 7133-7155.

[24] Dorigo, M. and U. Schnepf, 1993, 'Genetics-Based Machine Learning and Behavior-Based Robotics: A New Synthesis', *IEEE Transaction on Systems, Man, Cybernetics* Vol. 23, No. 1, pp. 141-154.

[25] Feng, D. F., M. S. Johnson and R. F. Doolittle, 1985, 'Aligning Amino Acid Sequences: Comparison of Commonly Used Methods', *J. Mol. Evol.*, Vol. 21, pp. 112-125.

[26] Feng, D. F. and R. F. Doolittle, 1987, 'Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees', *J. Mol. Evol.*, Vol. 25, pp. 351-360.

[27] Friedland, P., and L. H. Kedes, 1985, 'Discovering the Secrets of DNA', *Communication of the ACM*, Vol. 28, pp. 1164-1186.

[28] Goldberg, D. E., 1987, 'Simple Genetic Algorithms and the Minimal, Deceptive Problem', in L. Davis eds. *Genetic Algorithms and Simulated Annealing*, London: Pitman. pp. 74-88.

[29] Goldberg, D. E., 1989a, *Genetic Algorithms in Search, Optimization, and Machine Learning*, New York, NY: Addison-Wesley Publishing Company, Inc..

[30] Goldberg, D. E., 1989b, 'Sizing Populations for Serial and Parallel Genetic Algorithms', *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 70-79.

[31] Gotoh, O., 1982, 'An Improved Algorithm for Matching Biological Sequences', *J. Mol. Biol.*, Vol. 162, pp. 705-708.

[32] Gotoh, O., 1986, 'Alignment of Three Biological Sequences with an Efficient Traceback Procedure', *J. Theor. Biol.*, Vol. 121, pp. 327-337.

[33] Gotoh, O., 1987, 'Pattern Matching Biological Sequences with Limited Storage', *CABIOS*, Vol. 3, pp. 17-20.

[34] Gotoh, O., 1990, 'Consistency of Optimal Sequence Alignments', *Bull. Math. Biol.*, Vol. 52, pp. 509-525.

[35] Grefenstette, J. J., and J. M. Fitzpatrick, 1985, 'Genetic Search with Approximate Function Evaluations', *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 112-120.

[36] Grefenstette, J. J., 1986, 'Optimization of Control Parameters for Genetic Algorithms', *IEEE Transactions on System, Man, and Cybernetics*, Vol. 16, No. 1, pp. 122-128.

[37] Grefenstette, J. J. and J. E. Baker, 1989, 'How Genetic Algorithms Work: A Critical Look at Implicit Parallelism', in Morgan Kaufmann ed. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, pp. 20-27.

[38] Griggs, J. R., P. J. Hanlon and M. S. Waterman, 1989, "On the Number of Alignments of k Sequences', *Asian J. Comb.*, 1989.

[39] Gultyaev, A. P., 1991, 'The Computer Simulation of RNA Folding Involving Pseudoknot Formation', *Nucleic Acids Res.*, Vol. 19, pp. 2489-2494.

[40] Gultyaev, A. P., F. H. D. V. Batenbury, and C. W. A. Pleij, 1995, 'The Influence of a Metastable Structure in Plasmid Primer RNA on Antisence RNA Binding Kinetics", *Nucleic Acids Res.*, Vol. 23, No. 18, pp. 3718-3725.

[41] Gusfield, D., 1993, 'Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds', *Bull. Math. Biol.*, Vol. 55, pp. 141-154.

[42] Guttman, A. J., B. W. Ninham, and C. J. Thompson, 1968, 'Determination of Critical Behavior in Lattice Statistics from Series Expansions", *Phys. Rev.*, Vol. 172, pp. 554-558.

[43] Heeb, D., 1949, *The Organization of Behavior*, Wiley, New York.

[44] Higgins, D. G. and P. M. Sharp, 1989, 'Fast and Sensitive Multiple Sequence Alignment on a Microcomputer', *CABIOS*, Vol. 5, No. 2, pp. 151-153.

[45] Himes. G. S. and M. Inigo, 1992, 'Automatic Target Recognition Using a Neocognitron', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4., No. 2. pp. 167-172.

[46] Hirschberg, D. S., 1975, 'A Linear Space Algorithm for Computing Longest Common Subsequences', *Commun. Assoc. Comput. Mach.*, Vol. 18, pp. 341-343.

[47] Hogeweg, P., and B. Hesper, 1984, 'The Alignment of Sets of Sequences and the Construction of Phyletic Trees: an Integrated Method', *J. Mol. Evol.*, Vol. 20. pp. 175-186.

[48] Holland, J. H., 1975, *Adoption in Natural and Artificial Systems*, University of Michgan Press, Ann Arbor, Mich.

[49] 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.

[50] Holland, J. H., 1987, 'Genetic Algorithms and Classifier Systems: Foundations and Future Sirections', *Genetic Algorithms and Their Applications: Proceding of the Second International Conference on Genetic Algorithms*. pp. 82-89.

[51] Holley, L. H., and M. Karplus, 1989. 'Protein Secondary Structure Prediction with a Neural Network', *Proc. Natl. Acad. Sci. USA*, Vol. 86. pp. 152-156.

[52] Humblet, C., and J. B. Dunbar, 1993, '3D Database Searching and Docking Strategies', *Annual Report in Medicinal Chemistry - 28*, Academic Press. Inc., pp. 275-284.

[53] Jacobson, A. B. and M. Zuker, 1993, 'Structureal Analysis by Energy Dot Plot of a Large mRNA' *J. Molec. Biol.*, Vol. 233, pp. 261-269.

[54] Jaeger, J. A., D. H. Turner and M. Zuker, 1989, 'Improved predictions of secondary structures for RNA', *Pro. Natn. Acad. Sci. U.S.A.*, Vol. 86, pp. 7706-7710.

[55] Jaeger, J. A., J. SantaLucia and M. Zuker, 1993, 'Determination of RNA Structure and Thermodynamics', *A. Rev. Biochem.* Vol. 62, pp. 255-287.

[56] Janson, D. J., and J. F. Frenzel, 1993, 'Training Product Unit Neural Networks with Genetic Algorithms', *IEEE Expert*, Vol. 8, No. 5, pp. 26-33.

[57] Jeanteur, D., J. H. Lakey and F. Pattus, 1991, 'The Bacterial Porin Superfamily: Sequence alignment and Structure Prediction', *Molecular Microbiology*, Vol. 5(9), pp. 2153-2164.

[58] Johnson, M. S., and R. F. Doolittle, 1986, 'A Method for the Simultaneous Alignment of Three or More Amino Acid Sequences', *J. Mol. Evil.*, Vol. 23, pp. 267-278.

[59] Kehol, K., 1990, 'Specialized Databases in Molecular Biology and Genetics: The Nucleic Acid and Protein Sequence Database', *Electronic Information Systems in Sci-Tech Libraries*, The Haworth Press Inc., pp. 99-125.

[60] Lander, E. S., R. Langridge and D. M. Saccocio, 1991, 'Computing in Molecular Biology: Mapping and Interpreting Biological Information', *IEEE Computer*, Vol. 14, No. 11, pp. 6-13.

[61] Laquer, H. T., 1981, 'Asymptotic Limits for a Two-Dimensional Recursion', *Stud. App., Math.*, Vol. 64, pp. 271.

[62] Levenshtein, V. I., 1966, 'Binary Codes Capable of Correcting Deletions, Insertions, and Reversals', *Cybernet. Control Theor.*, Vol. 10, pp. 707-710.

[63] Levinthal, C., 1968, 'Are there Pathways for Protein Folding?', *J. Chem. Phys.*, Vol. 65, pp. 44-45.

[64] Maizel, J. V., and R. P. Lenk, 1981, 'Enhanced Graphic Matrix Analysis of Nucleic Acid and Protein Sequences', *Proc. Natl. Acad. USA*, Vol. 16, pp. 7665-7669.

[65] Martinez, H. M., 1983, 'An Efficient Method for Finding Repeats in Molecular Sequences', *Nucleic Acids Res.*, Vol. 11, pp. 4629-4634.

[66] Martinez, H. M., 1984, 'An RNA folding rule', *Nucleic Acids Res.*, Vol. 12, pp. 323-334.

[67] Metropolis, N., A. W. Rosenbluth, M. N. Teller, A. H. and E. Teller, 1953, 'Equation of State Calculationa by Fast Computing Machines', *J. Chem. Phys.* Vol. 21, pp. 1087-1092.

[68] McGregor, M. J., T. P. Flores and M. J. E. Sternberg, 1989, 'Prediction of β-turns in Proteins Using Neural Networks', *Protein Engng.*, Vol. 2, pp. 521-526.

[69] Michalewicz, Z., 1992, 'Genetic Algorithms + Data Structures = Evolution Programs', *Springer-Verlag.*

[70] Miller, W., 1993, 'Building Multiple Alignment from Pairwise Alignments', *Comput. App.ic. Biosci.*, Vol. 9, pp. 169-176.

[71] Myers, E. W., and W. Miller, 1988, 'Optimal Aligments in Linear Space', *Comput. Applic. Biosci.*, Vol. 4, No. 1, pp. 11-17.

[72] Needleman, S. B. and C. D. Wunsch, 1970, 'A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins' *J. Mol. Biol.*, Vol. 48, pp. 444-453.

[73] NeuralWare, 'Neural Computing –NeuralWorks Professional II/PLUS and NeuralWorks Explorer', *NeuralWare, Inc.*.

[74] Panjukov, V. V., 1993, 'Finding Steady Alignments: Similarity and Distance', *CABIOS*, Vol. 9, No. 3, pp. 285-290.

[75] Patthy, L., 1987, 'Detecting Homology of Distantly Related Proteins with Consensus Sequences', *J. Mol. Biol.*, Vol. 198, pp. 567-577.

[76] Perczel, A., and I. G. Csizmadia, 1995, 'Searching for the Simplest Structural Units to Describe the Three-deimensional Structure of Proteins', *Taylor & Francis Ltd.*, pp. 128-168.

[77] Qian, N., and T. J. Sejnowski, 1988, 'Prediction the Secondary Structure of Globullar Proteins Using Neural Network Models', *J. Mol. Biol.*, Vol. 202, pp. 865-884.

[78] Rawlins, G. J. E., 1991, *Foundations of Genetic Algorithms*, Morgan Kaufmenn Publishers.

[79] Rohde, K., and P. Bork, 1993, 'A Fast, Sensitive Pattern-Matching Approach for Protein Sequences', *CABIOS*, Vol. 9, No. 2, pp. 183-189.

[80] Roytberg, M., 1992, 'A search for Common Patterns in Many Sequences', *Comput. Applic. Biosci.*, Vol. 8, pp. 57-64.

[81] Salay, R., 1991, *The Largest Common Subgraph Problem Using Genetic Algorithms*, Master Thesis, Department of Systems Design Engineering, University of Waterloo.

[82] Salay, R., and A. K. C. Wong, 1991, 'Genetic Algorithms for Solving the Largest Common Subgraph Problem', *4th UNB AI Symposiom*, Fredenricton, NB, Canada.

[83] Sankoff, D., 1972, 'Matching Sequence Under Deletion-Insertion Constraints', *Proc. natn. Acad. Sci. U.S.A.*, Vol.64, pp. 4-6.

[84] Sankoff, D., 1975, 'Minimum Mutation Trees of Sequence', *SIAM J. Appl. Math.*, Vol. 78, pp. 35-42.

[85] Sankoff, D., and P. H. Sellers, 1973, 'Shotcuts, Diversions, and Maximal Chains in Partially Ordered Sets', *Discrete Math.* Vol. 4, pp. 287-258.

[86] Sankoff, D. and R. J. Cedergren, 1983, 'Simultaneous Comparison of Three or More Sequences Related by a Tree', in D. Sankoff and J. B. Kruskal eds. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*, London: Addison-Wesley.

[87] Sasagawa, F. and K. Tajima, 1993, 'Prediction of Protein Secondary Structure by a Neural Network', *CABIOS* Vol. 9, pp. 147-152.

[88] Schalkoff, R., 1992, *Pattern Recognition: Statistical, Structural and Neural Approaches*, John Wiley and Sons, Inc., New York.

[89] Schultz, A. C., J. J. Grefenstette, and A. A De Jong, 1993, 'Test and Evaluation by Genetic Algorithms', *IEEE Expert*, Vol. 8, No. 5, pp. 9-14.

[90] Sellers, P. H., 1974a, 'An algorithm for the Distance Between Two Finite Sequences', *J. Combinational Theory*, Vol. 16A, pp. 253-258.

[91] Sellers, P. H., 1974b, 'On the theory and computation of evolutionary distances', *SIAMJ. Appl. Math.*, Vol. 26, No. 4, pp. 787.

[92] Seuchter, S. A., and M. H. Skolnich, 1988, 'HGDBMS: A Human Genetics Database Management System', *Computers and Biomedical Rearch*, Vol. 21, pp. 478-487.

[93] Sibbald, P. R., and P. Argos, 1990, 'Scrutineer: a computer program that flexibly seeks and describes motifs and profiles in protein sequence databases', *J. CABIOS.*, Vol. 6, No. 3, pp. 279-288.

[94] Simpson, P. K., 1989, 'Artificial Neural Systems', *Pergamon Press Inc.*.

[95] Spears, W. M., and K. A. De Jong, 1990, 'Using Genetic algorithms for Supervised Concept Learning', *Proceedings of Tools for Artificial Intelligence*, IEEE computer Society Press, Los Alamitos, CA, pp. 335-341.

[96] Stolorz, P., A. Lapedes and Y. Xia, 1992, 'Predicting Protein Secondary Structure Using Neural Net and Statistical Methods', *J. Mol. Biol.*, Vol. 225, pp. 363-377.

[97] Sweet, R. M., 1986, 'Evolutionary Similarity Among Peptide Segments Is a Basis for Prediction of Protein Folding", *Biopolymers*, Vol. 25, pp. 1565-1577.

[98] Tajima, K., and K. Matsuo, 1988, 'Genetic Information Analysis by Neural Networks', *Third Symposium on Biological and Physiological Engineering*, Nov., Osaka.

[99] Talor, P., 1984, 'A Fast Homology Program for Aliging Biological Sequences', *Nucleic Acids Research*, Vol. 12, pp. 447-455.

[100] Thompson, J. D., D. G. Higgins and T. J. Gibson, 1994, 'CLUSTAL W: Improving the Sensitivity of Pregressive Multiple Seqeunce Alignment Through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice', *Nucleic Acids Research*, Vol. 22, No. 22, pp. 4673-4680.

[101] Tyler, C. E., R. H. Martha and P. R. Krause, 1991, 'A Review of Algorithms for Molecular Sequence Comparison', *Computer and Biomedical Research*, Vol. 24, pp. 72-96.

[102] Unger, R., and J. Moult, 1993a, 'Genetic Algorithms for Protein Folding simulations', *J. of Molecular Biology*, Vol. 231, pp. 75-81.

[103] Unger, R., and J. Moult, 1993b. 'Finding the Lowest Free Energy Conformation of a Protein is a NP-hard Problem: Proof and Implications". *Bull. Math. Biol.* Vol. 55, No. 6, pp. 1183-1198.

[104] Urrows, H., and E., 1992. 'CD-Gene. GenBank, and Their Cousins', *CD-Rom Librarian.* pp. 29-34.

[105] Vingron, M., and P. Argos, 1991. 'Motif Recognition and Alignment for Many Sequences by Comparison of Dot-Matrices'. *J. Mol. Biol.*, Vol. 218, pp. 33-43.

[106] Vihinen, M., A. Euranto, P. Luostarinen and O. Nevalainen, 1992. 'MULTICOMP: a Program Package for Multiple Sequence Comparison', *Comput. Applic. Biosci.*, Vol. 8, pp. 35-38.

[107] Wang, L., T. Jiang and E. L. Lawler, 1996. App.oximation algorithms for tree alignment with a given phylogeny. *Algorithmica.* Vol. 16, pp. 302-315.

[108] Watanabe, K., Y. Urano, and T. Tamaoki. 1985, 'Optimal Alignments of Biological Sequences on a Microcoputer'. *CABIOS*, Vol. 1, pp. 83-87.

[109] Waterman, M. S., T. F. Smith, and W. A. Beyer, 1976. 'Some Biological Sequence Metrics'. *Adv. Math.*, Vol. 20, pp. 367-387.

[110] Waterman, M. S., 1984, 'General Methods of Sequence Comparison', *Bull. Math. Biol.*, Vol. 46, No. 4, pp. 473-500.

[111] Waterman, M. S., Galas and Arratia, 1984, 'Pattern Recognition in Several Sequences: Consensus and Alignment', *Bull. Math. Biol.*, Vol. 46, pp. 515-527.

[112] Waterman, M. S. and M. D., Perlwitz, 1984, 'Line Geometries for sequence Comparison', *Bull. Math. Biol.*, Vol. 46, pp. 567-577.

[113] Whitley, D., S. Domonic, and R. Das, 1991, 'Genetic Reinforcement Learning with Multilayer Neural Networks', *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Faufmann, San Mateo, CA, pp. 562-569.

[114] Wilbur, W. J., and D. J. Lipman, 1983, 'Rapid Similarity Searches of Nucleic Acid and Protein Data Banks', *Proc. Natl. Acad. Sci. USA*, Vol.80, pp. 726-730.

[115] Wong, A. K. C., and M. You, 1985, 'Entropy and Distance of Random Graphs with application to Structural Pattern Recognition', *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 7, pp. 599-609.

[116] Wong, A. K. C., J. Constant, and M. You, 1990, 'Random Graphs', *Syntactic and Structural Pattern Recognition - Fundamentals, Advances, and Applications*, World Scientific Publishing Company Pte. Ltd.

[117] Wong, A. K. C., S. C. Chan, and D. K. Y. Chiu, 1996, 'Pattern Detection in Biomolecules Using Synthesized Random Sequence', *Pattern Recognition*, Vol. 29, No. 9, pp. 1581-1586.

[118] You, M., 1983, *A Random Graph Approach to Pattern Recognition*, Ph. D. thesis, University of Waterloo.

[119] Zhang, C., 1994a, 'A genetic algorithm for training image classification neural networks', *Proceedings of IEEE Systems, Man, and Cybernetics'94*, pp. 2242-2247.

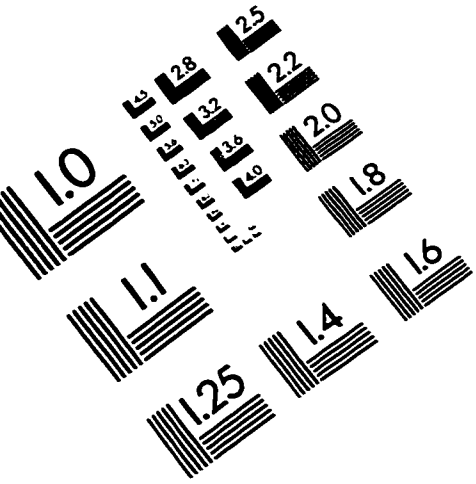
[120] Zhang, C., 1994b, 'A genetic algorithm approach for molecular sequence comparison', *Proceedings of IEEE Systems, Man, and Cybernetics'94*, pp. 1926-1931.

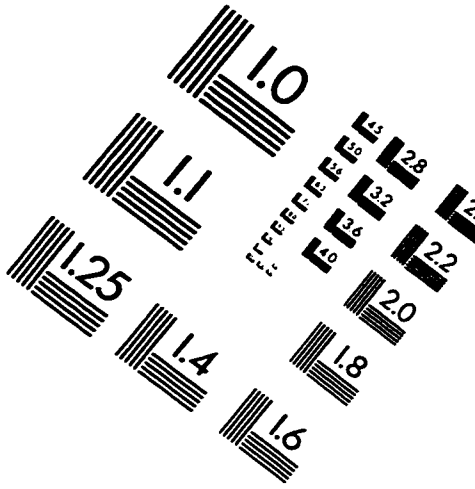
[121] Zhang, C., and A. K. C. Wong, 1997a, "A genetic algorithm for multiple molecular sequence alignment", *Computer Applications in Biosciences (CABIOS)*, Vol. 13, No. 6, pp. 565-581.

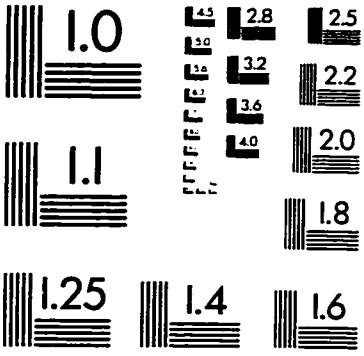
[122] Zhang, C., and A. K. C. Wong, 1997b, 'Towards Efficient Multiple Molecular Sequence Alignment: a System of Genetic Algorithm and Dynamic Programming', *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 26B, No. 6, pp. 918-932.

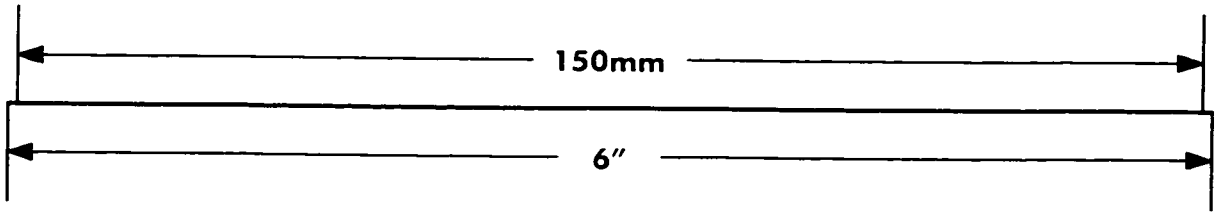
[123] Zhang, C., and A. K. C. Wong, 1998, 'An Efficient Technique of Genetic Algorithm and Sequence Synthesis for Multiple Molecular Sequence Alignment', submitted to *The Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB'98)*.

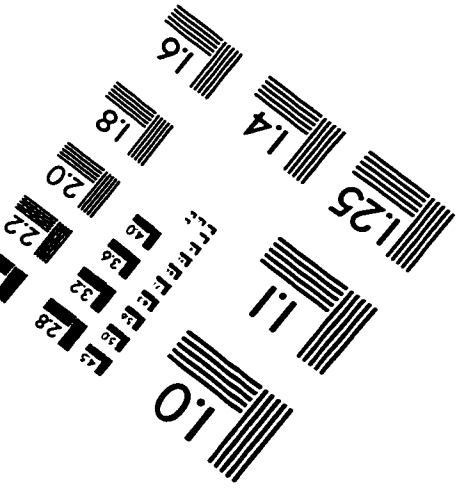
IMAGE EVALUATION TEST TARGET (QA-3)

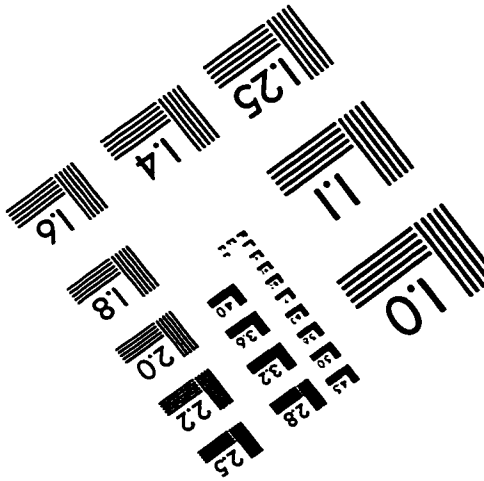












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989