

PAIRWISE AND MULTIPLE SEQUENCE
ALIGNMENT ALGORITHMS
ANALYSIS

By

XIAO HONG WANG

Bachelor of Science
Shandong Normal University
Jinan, China
1994

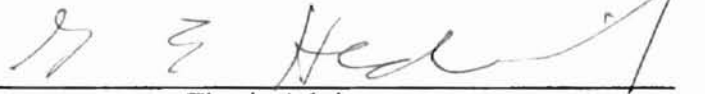
Master of Science
East China Normal University
Shanghai, China
1996

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE

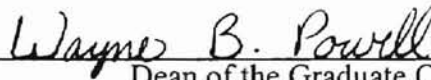

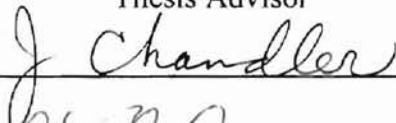
December, 1999

PAIRWISE AND MULTIPLE SEQUENCE
ALIGNMENT ALGORITHMS
ANALYSIS

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to thank my major advisor, Dr. George E. Hedrick for his intelligent, patient and constructive instruction. My sincere appreciation extended to my other advisory committee members, Dr. John P. Chandler and Dr. Douglas R. Heisterkamp for their helpful advisement and suggestions.

In addition, I wish to express my appreciation to all of my friends who gave me assistance and encouragement for this study.

I would also like to give my special appreciation to my husband Jun Li, for his precious suggestions to my research, and his love, understanding and support throughout this whole process. My special thanks also go to my parents for their love, support and encouragement.

TABLE OF CONTENTS

Chapter	Page
INTRODUCTION	1
BACKGROUND OF SEQUENCE ALIGNMENT	4
2.1 Definition of Sequences and Sequence Alignment.....	4
2.2 Key Issues in Sequence Alignment	5
2.2.1 Global and Local Alignment.....	6
2.2.2 Scoring Model.....	7
PAIRWISE SEQUENCE ALIGNMENT ALGORITHMS.....	12
3.1 Global Distance Alignment	12
3.2 Global Similarity Alignment	14
3.3 Local Similarity Alignment	14
3.4 Gotoh Algorithm.....	15
3.5 Myers and Miller Algorithm With Linear Gap Penalty.....	17
3.5.1 Linear Space Alignment.....	17
3.5.2 Linear Gap Penalty Function	18
3.5.3 Implementation of Myers-Miller Algorithm with Linear Gap Penalty.....	19
3.5.4 Result	21
MULTIPLE SEQUENCE ALIGNMENT ALGORITHMS	23
4.1 Overview of Multiple Sequence Alignment	23
4.1.1 Multidimensional Dynamic Programming.....	23
4.1.2 Tree or Hierarchical Method.....	25
4.1.3 Statistic Approach.....	27
4.2 Clustal W Analysis.....	29
4.2.1 Introduction.....	29
4.2.2 Algorithm Analysis.....	33
4.3 SAM (Sequence Alignment and Modeling System).....	37
4.3.1 Introduction.....	37
4.3.2 Linear HMM Architecture	39
4.3.3 Model Initialization.....	42
4.3.4 Model Estimation: the Baum-Welch Algorithm	43
4.3.5 Multiple Alignment: the Viterbi Algorithm.....	48
4.3.6 Time and Space Complexity of SAM	48
4.4 MSA (Multiple Sequence Alignment).....	49
4.4.1 Introduction.....	49
4.4.2 Setting the Bound and Regions – Stage I in MSA.....	50
4.4.3 Search the Optimal Alignment – Stage II in MSA	57
SUMMARY AND CONCLUSIONS	59
GLOSSARY	61
REFERENCES	63

LIST OF FIGURES

Figure	Page
1. Bioinformatics methods in sequence analysis.....	3
2. Comparison of global alignment and local alignment GLOSSARY	6
3. The PAM250 mutation matrix	8
4. Dynamic alignment matrix calculation	12
5. Dynamic alignment matrix for sequences ACACACTA and AGCACACA	13
6. Time comparison of two algorithms	22
7. A three-dimensional multiple alignment matrix for 3 protein sequences.....	24
8. To calculate each node, 2^N-1 positions need to be visited	24
9. Multiple alignment using progressive approach	26
10. Markov chain of DNA	27
11. Flow chart of Feng-Doolittle Algorithm.....	30
12. The basic progressive alignment procedure.....	32
13. SAM program outline	38
14. Linear HMM	39
15. Graphic view of pairwise alignment	50
16. Graphic view of multiple alignment	50
17. All 3 pairwise projections of the alignment	52

CHAPTER I

INTRODUCTION

Bioinformatics is a science that combines biology, statistical methods and computer science together³⁹. It is generated because of the development of advanced experimental technology in molecular biology area over the last decade, especially because of the *Human Genome Project* (HGP)¹². With the development, great quantity of data about genomic or protein sequence has been generated. New databases are needed to store this information and new methods are needed to analyze this information. Therefore the basic purpose of bioinformatics is developing computer databases and algorithms to store and to analyze biological data, and the ultimate goal of it is predicting the organization, structure and function of macromolecules such as DNA and proteins by analyzing their molecular sequences⁴². Because DNA is the molecule that carries genetic information and proteins are the essential components of all organs and chemical activities, the study of these molecules is essential for understanding human diseases and identifying of new molecular targets for drug discovery.

Sequence alignment (or comparison) is a very important and successful method to analyze molecular sequences and thus is one of the important areas in bioinformatics. It is the process of searching similarity in two or more DNA or protein sequences. It is often used to search a DNA or protein sequence database with a well-studied sequence provided by users. Sequence alignment is very helpful in the study of molecular evolution, protein structure and function prediction, PCR primer design, gene expression and so on². Some uses of sequence comparison methods are shown in Figure 1.

At present, many programs have been implemented to align sequences^{4,14,16,19,21,38}. Different results will be obtained by using different programs due to their different algorithms and strategies. Understanding how these methods were implemented is important for appropriate and efficient use of sequence alignment software. In addition, analyzing programs in use will be helpful for development of new methods. Based on pairwise sequence alignment, multiple sequence alignment is more useful and more powerful to find the homologous, conserved or functional regions in macromolecules³⁹.

The objective of this study is to analyze pairwise and multiple sequence alignment algorithms in use, especially focusing on three multiple sequence alignment methods^{38,21,24}. A linear space pairwise sequence alignment program, which is a base stone for multiple sequence alignment is implemented.

In the following chapters, Chapter 2 introduces basic definition of sequence alignment. Chapter 3 analyzes some optimal pairwise sequence alignment algorithms. Chapter 4 analyzes multiple sequence alignment. In the last chapter, a summary and conclusion is presented.

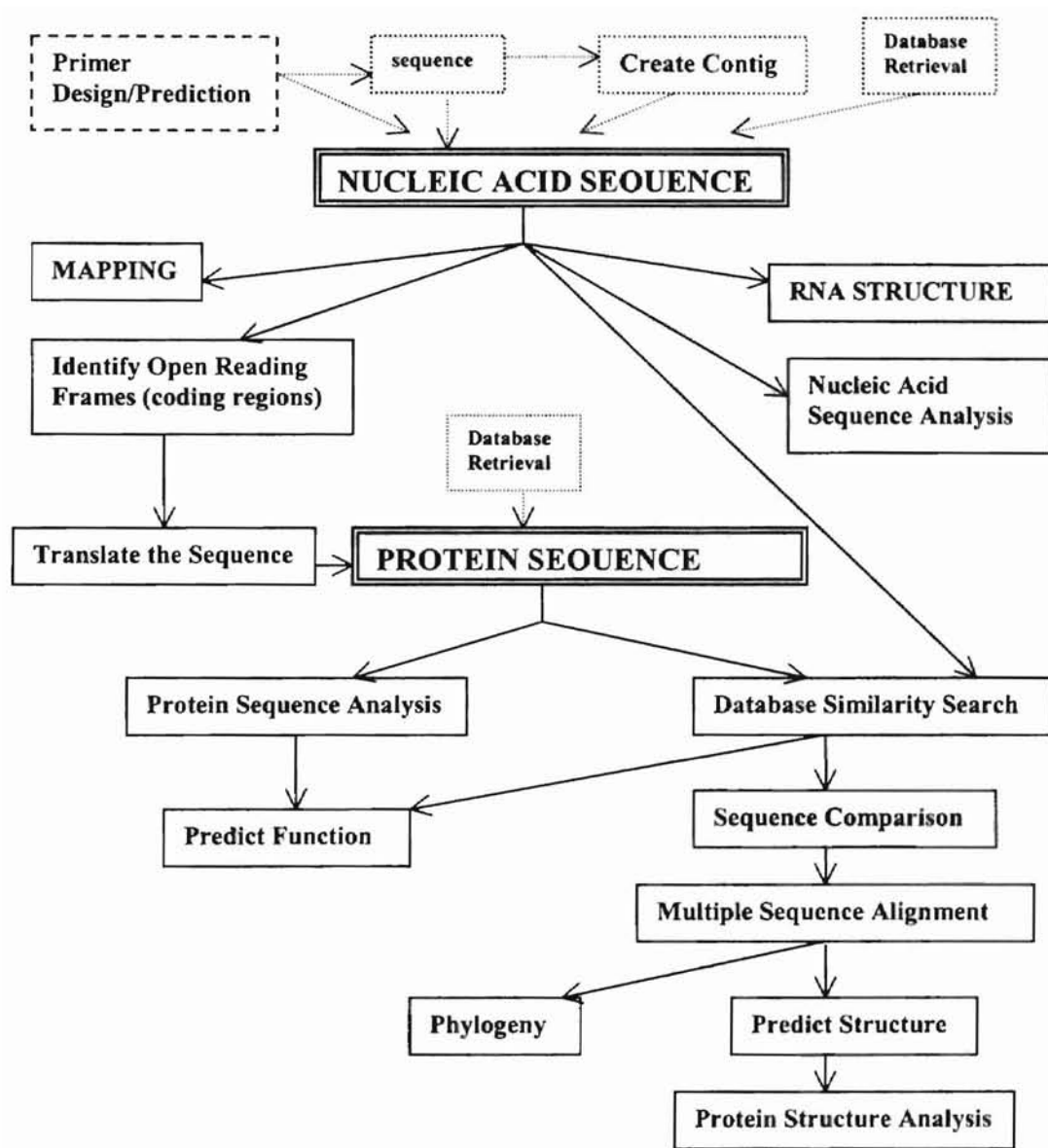


Figure 1. Bioinformatics methods in sequence analysis⁴³
 (modified from UCSC computational biology group)

CHAPTER II

BACKGROUND OF SEQUENCE ALIGNMENT

2.1 Definition of Sequences and Sequence Alignment

“A sequence is an ordered string of elements which are symbols or letters of an alphabet and is represented by simple concatenation of these elements”²⁶. In bioinformatics, sequences are always nucleic acid or protein sequences.

For computational purposes, a DNA molecule can be defined as a word over the four-letter alphabet of nucleotides: {A, C, G, T} where A, C, G, T are the abbreviations of four nucleotides.

In RNA, T is replaced by U, so an RNA molecule can be defined as a word over four-letter alphabet of ribonucleotides: {A, C, G, U} where U takes the place of T.

All proteins are built from 20 amino acids. Therefore, a protein molecule can be defined as a word over 20 amino acids

{A, R, D, N, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V}

Because it is not always the case that all positions in a sequence are known precisely, the symbol ‘-’ is included in the sequence alphabet.

“An alignment of the sequences S_1, \dots, S_k is another set of sequences S'_1, \dots, S'_k , such that each sequence S'_i is obtained from S_i by inserting the blank character ‘-’ in positions where some of the other sequences have a nonblank character”². $S_{i,j}$ means the character at i-th sequence at j-th position.

Therefore if the sequences are written as

$$\begin{aligned}
S_1 &= S_{1,1} \ S_{1,2} \ \dots \ S_{1,|S_1|} \\
S_2 &= S_{2,1} \ S_{2,2} \ \dots \ S_{2,|S_2|} \\
&\quad \dots \\
S_N &= S_{N,1} \ S_{N,2} \ \dots \ S_{N,|S_N|}
\end{aligned}$$

Then the multiple alignment will be written as

$$\begin{aligned}
S'_1 &= S'_{1,1} \ S'_{1,2} \ \dots \ S'_{1,|A|} \\
S'_2 &= S'_{2,1} \ S'_{2,2} \ \dots \ S'_{2,|A|} \\
&\quad \dots \\
S'_N &= S'_{N,1} \ S'_{N,2} \ \dots \ S'_{N,|A|}
\end{aligned}$$

Some of the S'_{ij} are gaps, if ignored, the rows become the original sequences. An alignment of N sequences is a rectangular of N rows and $|A|$ columns (A represents an alignment, $|A|$ represents the length of an alignment), each column contains at least one character different from "-".

In sequence alignment, (a, a) denotes a match or identity; $(a, -)$ denotes deletion of a in the second sequence or insertion of a in the first sequence; (a, b) denotes substitution of a by b where $a \neq b$.

2.2 Key Issues in Sequence Alignment

The key issues in multiple sequence alignment are ⁷:

- 1) What sorts of alignment should be considered;
- 2) How to choose an appropriate scoring system;
- 3) The algorithm used to find optimal (or good) scoring alignment;
- 4) The statistical methods used to evaluate the significance of an alignment.

Base on the number of sequences, sequence alignment can be divided into pairwise and multiple sequence alignment. Base on the scoring strategy, sequence alignment can be divided into global and local alignment.

2.2.1 Global and Local Alignment

There are two types of sequence alignments: global alignment and local alignment (Figure 2). In global alignment, an attempt is made to align the entire sequences, as many characters as possible. Global alignment is appropriated for sequences that are known to share similarity over their whole length³⁹. In local alignment, stretches of sequence with the highest density of matches (which are usually the most biologically significant) are given the highest priority, thus generating one or more islands of matches in the aligned sequences. Local alignment is appropriate when the sequences have isolated regions of similarity³⁹.

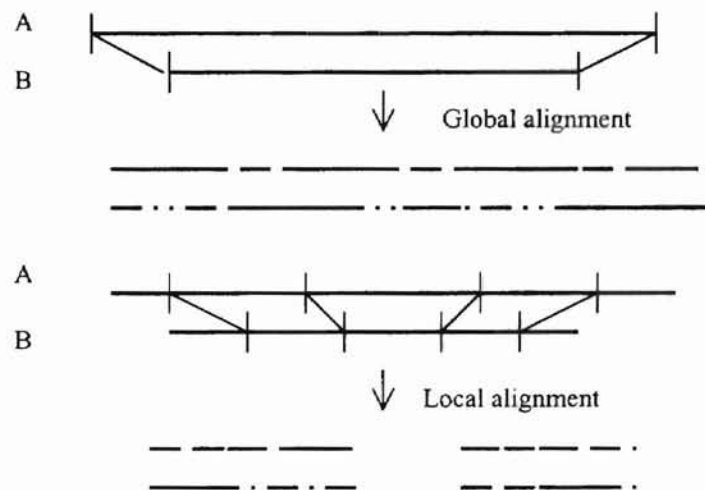


Figure 2. Comparison of global alignment and local alignment

2.2.2 Scoring Model

If two sequences of length 1000 are aligned randomly, there are about 10^{600} alignments³⁹. However, because insertion or deletion of nucleotides or amino acids is more difficult than substitution, and because some substitutions are more costly than others, a scoring model is used to evaluate sequence alignment^{39,5}. The scoring model includes the mutation scores for DNA or protein and the gap penalty scores for insertions or deletions (indels or gaps),

1. Mutation Matrix:

Mutation matrix is a matrix of mutation scores, which is the value of changing one DNA nucleotide or protein amino acid to another one. Algorithms to compare protein or DNA sequences rely on some mutation scoring.

DNA scoring matrices are rather simple, usually counting a match as one and mismatch as zero.

Amino acid scoring schemes are complex. They have to score each of the 210 possible pairs of amino acids (190 pairs of different amino acids + 20 pairs of identical amino acids). Most scoring schemes represent the 210 pairs of scores as 20×20 mutation matrix⁵. In the matrix, the amino acid pairs of identical amino acids have highest scores, the pairs with biological similarity have higher score, and the pairs with lowest similarity have lowest score.

A mutation scoring scheme should take into account some important features of multiple alignment. For example, scores should be position-specific. This is because different *residues* in a functional sequence are under different *selective pressures*, some

positions are more *conserved* than others. Mutations in a region of high conservation should ideally be penalized more than in a variable region.

Dayhoff PAM 250 matrix⁵ is one of the most used matrices (Figure 3). In the PAM 250 matrix, positive values represent evolutionarily conservative replacements; the amino acids were grouped basing on the physicochemical properties of the amino acids.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	12																				C
S	0	2																			S
T	-2	1	3																		T
P	-3	1	0	6																	P
A	-2	1	1	1	2																A
G	-3	1	0	-1	1	5															G
N	-4	1	0	-1	0	0	2														N
D	-5	0	0	-1	0	1	2	4													D
E	-5	0	0	-1	0	0	1	3	4												E
Q	-5	-1	-1	0	0	-1	1	2	2	4											Q
H	-3	-1	-1	0	-1	-2	2	1	1	3	6										H
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6									R
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5								K
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6							M
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5						I
L	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6					L
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4				V
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9			F
Y	0	-3	-3	-5	-3	-5	-1	-4	-4	-4	0	-4	-4	-1	-1	-2	7	10			Y
W	0 <td>-2</td> <td>-2</td> <td>-6</td> <td>-6</td> <td>-7</td> <td>-4</td> <td>-7</td> <td>-5</td> <td>-5</td> <td>-3</td> <td>-2</td> <td>-3</td> <td>-4</td> <td>-5</td> <td>-2</td> <td>6</td> <td>0</td> <td>12</td> <td></td> <td>W</td>	-2	-2	-6	-6	-7	-4	-7	-5	-5	-3	-2	-3	-4	-5	-2	6	0	12		W
C		S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Figure 3. the PAM250 mutation matrix⁵
(from David G., *et al* 1988)

A cost function is used to represent the scoring scheme. $\text{cost}(a, b)$: cost of substituting a b in the second sequence for an a in the first sequence; $\text{cost}(-, b)$: cost for columns where the first sequence has a gap and the second has a b ; $\text{cost}(a, -)$: cost for columns where the first sequence has an a and the second has a gap.

2. Gap Penalty

The *gap penalty* is usually a large negative value given to each gap introduced into an alignment, because introducing gap increases the uncertainty of an alignment. If gaps are introduced without a penalty or with low penalty, then eventually a high alignment score is achievable even in unrelated or random sequences. However, it is very difficult to test this subsequence for significance.

The gap penalty is position-dependent. Gaps are more likely to be found in parts of the sequence corresponding to loops in the three-dimensional structure. Gaps are unlikely to be found in the structural core of a protein because insertion of extra amino acids in the core may destroy the whole structure³⁹.

The gap penalty is also length-dependent³⁹. When gaps do occur, they are often longer than one residue.

3. Evaluation of Sequence Alignment

Two ways to evaluate sequence alignment are the similarity measure and the distance measure.

In similarity measure, defines the scoring function as $s(a,b)$, a higher value indicates greater similarity. The optimal alignment of two sequences S_i and S_j is the alignment with maximum overall cost:

$$Cost(S_i, S_j) = \max \sum_{1 \leq k \leq |A|} s(S'_{i,k}, S'_{j,k})$$

Where $|A|$ represents the length of an alignment.

In distance measure, the scoring function is defined as $d(a,b)$. The smaller the similarity, the larger the distance score. Thus an optimal alignment is the alignment with minimum cost:

$$\text{Cost}(S_i, S_j) = \min \sum_{1 \leq k \leq |A|} d(S'_{i,k}, S'_{j,k})$$

4. Sum-of-pairs Scoring Model

The sum-of-pairs cost⁷ is used to evaluate alignment, abbreviated to SP-cost. It evaluates the costs/weights of alignment column by column.

Using the SP-cost and the cost function $c(a,b)$, the total cost of the alignment for sequences (S_i, S_j) is:

$$\text{Cost}(S_i, S_j) = \sum_{1 \leq k \leq |A|} c(S'_{i,k}, S'_{j,k})$$

For example, if the cost function is defined as:

$$c(a, a) = 0, \quad c(a, b) = 1 \text{ for } a \neq b, \quad c(a, -) = c(-, b) = 1$$

then the cost of the alignment $\left\{ \begin{array}{l} \text{AGCACAC} - \text{A} \\ \text{A} - \text{CACACTA} \end{array} \right\}$ is:

$$c(\text{A}, \text{A}) + c(\text{G}, -) + c(\text{C}, \text{C}) + \dots + c(\text{A}, \text{A}) = 2$$

In a multiple alignment, the score of each column is computed by summing all the costs of all possible pairs of symbols by summing unit costs of the pairs (1,2), (1,3), ..., (2,3), (2,4), Therefore, the cost of an overall alignment of (S_1, \dots, S_n) is:

$$\text{Cost}(S_1, \dots, S_n) = \sum_{1 \leq k \leq |A|} \sum_{(i,j), 1 \leq i < j \leq n} c(S'_{i,k}, S'_{j,k})$$

CHAPTER III

PAIRWISE SEQUENCE ALIGNMENT ALGORITHMS

Pairwise sequence alignment is the process of aligning two sequences. It is the base of multiple sequence alignment.

Dynamic programming approach is usually used to find an optimal alignment between two sequences. Needleman and Wunsch³⁹ first applied dynamic programming on protein sequence comparison during the late 1960s. Since then a group of similar dynamic programming algorithms have been introduced that can calculate the optimal alignment with its score in the order of mn steps where m, n are the lengths of aligned sequences³. Use of any of the algorithm guarantees a mathematically optimal alignment, given the substitution matrix and gap penalties.

Let the two sequences of lengths m and n be $x = x_1 x_2 x_3 \dots x_m$ and $y = y_1 y_2 y_3 \dots y_n$. Let ‘-’ be the symbol for a single gap. At each aligned position, there are three possible events:

- 1) a substitution (x_i, y_i) or a match (x_i, y_i) depending on whether $x_i = y_i$,
- 2) a deletion $(x_i, -)$.
- 3) an insertion $(-, y_i)$.

As described earlier, the similarity measure defines the score function for each event as $s(a,b)$, and the distance measure defines the score as $d(a,b)$. Dynamic algorithms apply to either case.

3.1 Global Distance Alignment

Global distance alignment algorithm (from Waterman 1995)³⁹

input: $a, b, d(a, b)$
 output: $D_{i,j}, 0 \leq i \leq n, 0 \leq j \leq m$
 step 1: for $i \leftarrow 1$ to n
 step 2: for $j \leftarrow 1$ to m
 step 3:

$$D(i, j) = \min \left\{ \begin{array}{l} D(i-1, j-1) + d(x_i, y_j) \\ D(i, j-1) + d(x_i, -) \\ D(i-1, j) + d(-, y_j) \end{array} \right\}$$

D is a dynamic programming matrix. The value $D(i, j)$ is the score of the best alignment between the initial subsequence $x_{1...i}$ of x and the initial subsequence $y_{1...j}$ of y .

That is:

$$D(i, j) = \text{Cost}(x_1 \dots x_i \text{ and } y_1 \dots y_j)$$

Step 3 is applied recursively to fill the matrix of $D(i, j)$ values as in the following figure:

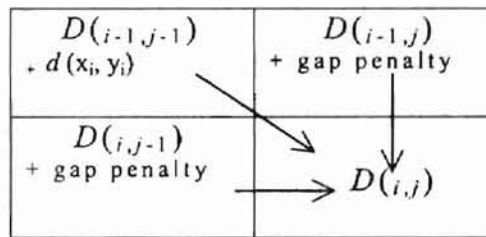


Figure 4. Dynamic alignment matrix calculation

If one of the prefixes is empty, i.e. either $i = 0$ or $j = 0$ or both, different schemes can be used to define the value. One possible scheme is define: $D(i, 0) = i \times d(x_i, -)$ and $D(0, j) = j \times d(-, y_j)$.

Let the sequence $x = x_1 x_2 x_3 \dots x_m$ and $y = y_1 y_2 y_3 \dots y_n$. Then the value in the final cell of the matrix $D(n, m)$ is the best score for the alignment of x and y . The final alignment is obtained by tracing back from the final value according to the choices that lead to it.

For example, using cost function (X) for alignment with $D(i, 0) = i$, $D(0, j) = j$, where (X) is the equation of number of cost defined on page 10, then using the dynamic programming the alignment for the two sequence: AGCACACA and ACACACTA is:

	-	A	C	A	C	A	C	T	A
-	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	2	3	4	5	6	7
C	3	2	1	2	3	3	4	5	6
A	4	3	2	1	2	2	3	4	5
C	5	4	3	2	1	2	2	3	4
A	6	5	4	3	2	1	2	3	3
C	7	6	5	4	3	2	1	2	3
A	8	7	6	5	4	3	2	2	2

Figure 5. Dynamic alignment matrix for sequences
ACACACTA and AGCACACA

The path in this diagram represents the optimal alignment. In which a diagonal line means substitution or match; a vertical line means a deletion in the first sequence (or an insertion in the second sequence); a horizontal line means an insertion in the first sequence (or a deletion in the second one). Thus, the result alignment is:

A - CACACTA
AGCACAC - A

3.2 Global Similarity Alignment

The algorithm used in global similarity alignment is very similar to that used in global distance alignment, except that step 3 is changed to:

$$S(i, j) = \max \left\{ \begin{array}{l} S(i-1, j-1) + s(x_i, y_j) \\ S(i, j-1) + s(x_i, -) \\ S(i-1, j) + s(-, y_j) \end{array} \right\}$$

where $S(i, j) = \text{Cost}(x_1 \dots x_i \text{ and } y_1 \dots y_j)$

In similarity measure, a match is rewarded by a positive score, that is, $s(a, b) > 0$ if a and b are similar. $s(a, b) < 0$ if a, b are not similar. $s(a, -) < 0$ and $s(-, a) < 0$ in particular (referred to as the gap penalty).

3.3 Local Similarity Alignment

There are cases where sequences share a similar region but are otherwise completely different. For example, suppose s and t be two proteins, which carry some functional related subunits. However, most part of s and t do not contribute to this function and may be very different. To handle this case, local multiple alignment algorithms have been developed. It is usually the most sensitive way to detect the similarity between two very highly diverged sequences that may have a shared evolutionary origin along their entire length⁴².

For protein sequences, the most commonly used local alignment algorithm allowing gaps is described by Smith and Waterman in the early 1980s³³. This algorithm is closely related to the algorithm for global similarity alignment except two differences.

1) An extra possibility is added to the equation, allowing $S(i,j)$ to take the value 0 if all other options have value less than 0. Thus all $S(i,j)$ now must have a value ≥ 0 .

$$S(i,j) = \max \left\{ \begin{array}{l} 0 \\ S(i-1, j-1) + s(x_i, y_j) \\ S(i, j-1) + s(x_i, -) \\ S(i-1, j) + s(-, y_j) \end{array} \right\}$$

2) The score for the best local alignment is simply the largest value of $S(i,j)$ which can be anywhere in the matrix. The corresponding alignment is obtained by tracing back from the cell of largest value.

The time and space of the algorithms introduced above are $O(mn)$, where n and m are the lengths of the sequences. The reason is that there are $(n+1) \times (m+1)$ numbers to store and each number costs a constant number of calculations to compute.

3.4 Gotoh Algorithm

The gap model considered in the early examples is just the simplest model, in which the gap penalty score is a simple multiplication of the length of gap. That is, if W_k be the gap weight for a gap of k bases, and W_1 be the single gap penalty, then $W_k = K \times W_1$

This type of scoring model is not good for biological sequences, since deletion or insertion of several adjacent characters in the biological sequence may be the result of one mutation event in sequence evolution³⁹. Since multiple deletions or insertions are not the sum of single deletion or insertion (indel), they should not be weighted by summing single indel weights. Instead, $W_k \leq K \times W_1$.

If multiplication of indels (insertion and deletion) in global distance alignment is allowed, the recursive step should be changed to:

$$D(i, j) = \min \left\{ \begin{array}{l} D(i-1, j-1) + d(x_i, y_j) \\ \min_{1 \leq k \leq j} [D(i, j-k) + W_k] \\ \min_{1 \leq k \leq i} [D(i-k, j) + W_k] \end{array} \right\}$$

Waterman⁴⁰ first introduced an algorithm using this equation. The time and space of this algorithm is $O(mn(n+m)) = O(n^3)$ if $m=n$, where m and n are the lengths of the two sequences under comparison, rather than $O(n^2)$ for the linear gap cost version.

Gotoh¹³ introduced an algorithm which solves this problem in $O(n^2)$ time if the gap penalty function is of this form: $W_k = \alpha + \beta(k-1)$. According to this, $W_1 = \alpha$ and $W_{k+1} = \beta + W_k$. Let $P_{i,j} = \min_{1 \leq k \leq j} [D(i, j-k) + W_k]$ and $Q_{i,j} = \min_{1 \leq k \leq j} [D(i-k, j) + W_k]$.

$P_{i,j}$ and $Q_{i,j}$ can be calculated in a single step according to the following recursion relations:

$$\begin{aligned} P_{i,j} &= \min_{1 \leq k \leq j} [D(i, j-k) + W_k] \\ &= \min \left\{ D(i, j-1) + w_1, \min_{2 \leq k \leq j} [D(i, j-k) + W_k] \right\} \\ &= \min \left\{ D(i, j-1) + w_1, \min_{1 \leq k \leq j-1} [D(i, j-1-k) + W_{k+1}] \right\} \\ &= \min \left\{ D(i, j-1) + w_1, \min_{1 \leq k \leq j-1} [D(i, j-1-k) + W_k + \beta] \right\} \\ &= \min \left\{ D(i, j-1) + w_1, \min_{1 \leq k \leq j-1} [D(i, j-1-k) + W_k] + \beta \right\} \\ &= \min \{ D(i, j-1) + \alpha, P_{i,j-1} + \beta \} \\ Q_{i,j} &= \min \{ D(i-1, j) + \alpha, Q_{i-1,j} + \beta \} \end{aligned}$$

3.5 Myers and Miller Algorithm with Linear Gap Penalty

3.5.1 Linear Space Alignment

Pairwise sequence alignment algorithms based on dynamic programming generally have computational time and space of $O(mn)$ where m and n are the lengths of the sequences being compared. Quadratic space is required because of the need of tracing the alignment through the matrix. If only the cost of an optimal alignment is needed, the space requirement is linear space. Quadratic memory usage may limit the use of the dynamic programming if the lengths of the two sequences are too long.

To reduce the space requirement, Myers and Miller²⁷ first introduced a linear space alignment algorithm into computation biology in 1988, hence this algorithm is called Myers-Miller algorithm in sequence analysis field. This algorithm uses a divide-and-conquer technique by computing a "mid-point" of an optimal alignment recursively using the forward and reverse cost-only dynamic programming algorithms. An optimal alignment can be obtained by recursively determining the optimal conversions on both sides of its mid-point.

Let sequences $x = x_1 x_2 x_3 \dots x_m$ and $y = y_1 y_2 y_3 \dots y_n$.

Let $u = \lfloor m/2 \rfloor$.

Myers-Miller algorithm first finds the optimal mid-point (u, j) where j is the row where the optimal alignment crosses the u column of the matrix. This mid-point (u, j) is found by combining the results of forward and reverse dynamic programming that passes at row u . Then the dynamic programming problem is split into parts, one part is from $(0, 0)$ to (u, j) and another part is from (u, j) to (m, n) . The optimal alignment for the whole matrix will be the concatenation of the optimal alignments from these two parts.

The forward algorithm computes the cost of the prefixes of the two sequences:

$$\text{Cost}(x_1, x_2, \dots, x_i, y_1, y_2, \dots, y_j)$$

Reverse algorithm is very similar to forward algorithm, with a little modification.

It computes the cost of the postfixes of the two sequences:

$$\text{Cost}(x_m, x_{m-1}, \dots, x_{i+1}, y_n, y_{n-1}, \dots, y_{j+1})$$

Both forward and backward algorithms are standard dynamic programming algorithms. Since each of them returns the cost of an alignment only, the space it required is linear space.

3.5.2 Linear Gap Penalty Function

In sequence evolution, when gap occurs, it is often of length more than one symbol. If treating an indel (insertion or deletion) of k consecutive symbols as an atomic operation, its cost W_k should be less than $W_1 \times k$.

Linear gap penalty function $W_k = \alpha + \beta \times k$ is usually used in sequence alignment algorithms³⁹. Here α is opening gap penalty (*gop*) and β is extension gap penalty (*gep*).

Opening gap penalty is a penalty for the initiation of a gap in sequence. Larger opening gap penalty will make it difficult to insert a gap and will decrease the length of the gap. Extension gap penalty is applied for increasing an already existing gap by one symbol. Increasing the extension gap penalty will decrease the length of the gaps.

Gotoh's algorithm introduced before shows how to deal with multiple insertions or deletions in quadratic time if the gap penalty function is a linear function. With that idea, when implementing Myers-Miller algorithm, the matrix $S(i, j)$ is calculated as:

$$S(i, j) = \max \left\{ \begin{array}{l} S(i-1, j-1) + d(x_i, y_i) \\ P(i, j) \\ Q(i, j) \end{array} \right\}$$

$$P(i, j) = \max \left\{ \begin{array}{l} S(i-1, j) + W_1 \\ P(i-1, j) + gep \end{array} \right\}$$

$$Q(i, j) = \max \left\{ \begin{array}{l} S(i, j-1) + W_1 \\ Q(i, j-1) + gep \end{array} \right\}$$

Only linear space and quadratic time are needed to compute the matrix.

3.5.3 Implementation of Myers-Miller Algorithm with Linear Gap Penalty

Assume the two sequences being compared:

$$A = a_1, a_2, \dots, a_m, B = b_1, b_2, \dots, b_n.$$

Define $C(i, j)$ be the forward cost: $\text{Cost}(a_1, a_2, \dots, a_i, b_1, b_2, \dots, b_j)$

Define $C^R(i, j)$ be the backward cost: $\text{Cost}(a_m, a_{m-1}, \dots, a_{i+1}, b_n, b_{n-1}, \dots, b_{j+1})$

Myers and Miller²⁷ showed that any conversion of A to B must either:

Type 1: Convert a_1, \dots, a_p to b_1, \dots, b_j and a_{x+1}, \dots, a_m to b_{j+1}, \dots, b_n for $p = x = m/2$ and some j .

Type 2: Convert a_1, \dots, a_p to b_1, \dots, b_j , delete $x-p$ symbols, and convert a_{x+1}, \dots, a_m to b_{j+1}, \dots, b_n for $p < m/2$, $x > m/2$ and some j .

Let (p, j, x) be a mid-point of type 1 or type 2. The cost of the mid-point is:

$$\text{Cost}(p, j, x) = \begin{cases} C(p, j) + C^R(x, j) & \text{if } p = x \text{ (type 1)} \\ C(p, j) + g(x-p) + C^R(x, j) & \text{if } p < x \text{ (type 2)} \end{cases}$$

To handle these two types of cost, two linear arrays are used in forward or reverse algorithm, one for type 1, one for type 2. Therefore in our implementation of Myers-Miller algorithm, four arrays are used:

CC: of length n , is the current column of a dynamic matrix in forward pass.

CD: of length n , is the current column ending with deletion in forward pass.

RC: of length n , is the current column of a dynamic matrix in reverse pass.

RD: of length n , is the current column ending with deletion in reverse pass.

To trace the alignment, an array ALIGN of length $n+m$, is used to store current operation such as insertion, deletion or institution.

The outline of the implementation is:

Algorithm forward pass :

Input: Astart, midA, Bstart, Blen, tB

Output: CC, CD

Algorithm reverse pass :

Input: Astart, Alen, Bstart, Blen, midA, tE

Output: RC, RD

Algorithm trace alignment :

Input: ALIGN

Output: Alignment

If $ALIGN[i] = 0$ substitution

If $k = ALIGN[i] > 0$ insertion of k symbols in sequence B

If $k = -ALIGN[i] > 0$ deletion of k symbols in sequence B

Algorithm Pair Align

Input: Astart, Alen, Bstart, Blen, tB, tE

Output: ALIGN and the alignment score

1. if $Blen \leq 0$ and $Alen \geq 0$
delete (Alen)
if $Alen \leq 0$
delete (Blen)
if $Alen = 1$
if $midB = 0$
delete(1); insert (Blen);
if $midB > 1$
insert (midB-1); replace (); insert (Blen-midB);
2. $midA \leftarrow Alen/2$
forward_pass (Astart, midA, Bstart, Blen, tB)
reverse_pass (Astart, Alen, Bstart, Blen, midA, tE)

3. *find_optimal_midpoint*
 $M \leftarrow \max_{0 \leq j \leq n} CC[j] + RC[j]$
 $N \leftarrow \max_{0 \leq j \leq n} CD[j] + RD[j] + gop$
 If $N > M$, type = 2
 Else type = 1
 MidB is the j that leads to the maximal cost

4. if type = 1
 Pair_Align (Astart, midA, Bstart, midB, tB, gop)
 Pair_Align (Astart+midA, Alen-midA, Bstart+midB,
 Blen-midB, gop, tB)

- If type = 2
 Pair_Align (Astart, midA-1, Bstart, midB, tB, 0.0)
 delete (2);
 Pair_Align (Astart+midA+1, Alen-midA-1, Bstart+midB,
 Blen-midB, 0.0, tE)

3.5.4 Result

Because the space used by this algorithm includes the four arrays CC, CD, RC, RD, each of them of length n , and the array ALIGN of length $n+m$ which is used to trace the alignment. Assuming n is equal to m , the total memory space used is $O(n)$.

This algorithm is a recursive algorithm. The first pass of *Pair_Align* is done in time proportional to: $2(\frac{m}{2} \times n) = mn$. After that, the problem divides into problems of size $m/2, j$, and $m/2, n-j$. Each of the problems takes time proportional to:

$$2(\frac{m}{4} \times j) + 2(\frac{m}{4} (n - j)) = \frac{1}{2} mn$$

Thus, the total running time of this algorithm is proportional to:

$$O(\sum_{k \geq 0} mn 2^{-k}) = O(2mn) = O(mn)$$

This is the same as the standard dynamic programming using quadratic space (Figure 5 and 6).

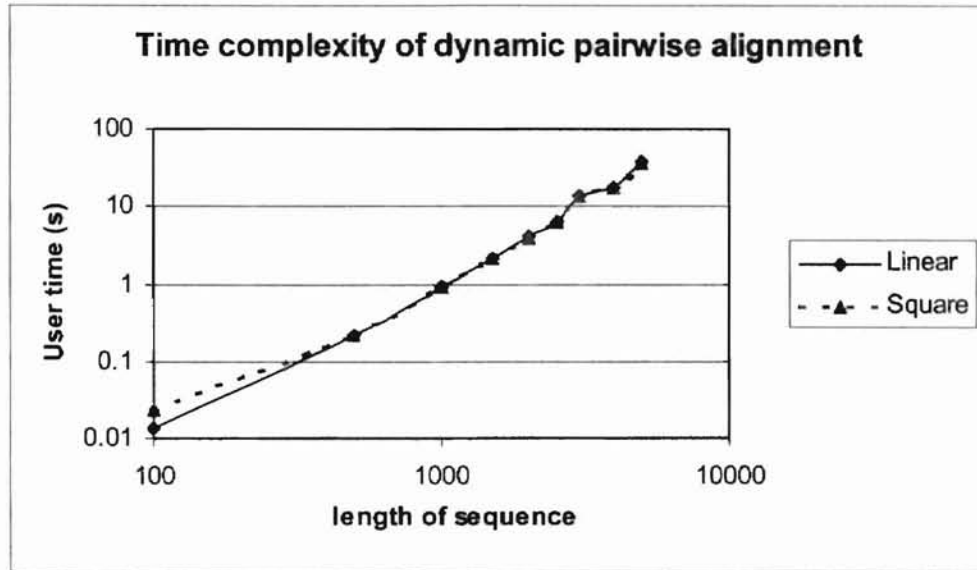


Figure 6. Time comparison of two algorithms

One algorithm is the Myers-Miller algorithm, which is linear space dynamic programming algorithm. The other algorithm is a standard dynamic programming algorithm using quadratic time.

User time here is the real time obtained under Solaris 7 system by using the TIME command.

CHAPTER IV

MULTIPLE SEQUENCE ALIGNMENT ALGORITHMS

Multiple sequence alignment is a process of aligning more than two sequences simultaneously, showing how the sequences are related to each other. In multiple alignment, amino acids or nucleotides diverging from common ancestor are aligned together in columns. Results of multiple sequence alignments are helpful in detecting distantly related proteins, in understanding protein function, and in predicting protein secondary structure.

Till 1987, biologists constructed high quality multiple sequence alignments by hand, using knowledge of protein evolution. It was very time-consuming and extensive knowledge in proteins was required. Nowadays, several programs are used to make multiple sequence alignment automatically^{16,19,23,38}. Dynamic programming approach, progressive method and statistical approach are commonly used in multiple sequence alignments.

4.1 Overview of Multiple Sequence Alignment Methods

4.1.1 Multidimensional Dynamic Programming

When using the multidimensional dynamic programming approach^{3,16,28}, n sequences are compared simultaneously by using an n -dimensional dynamic programming matrix. An example of aligning three sequences using this approach is given in Figure 7.

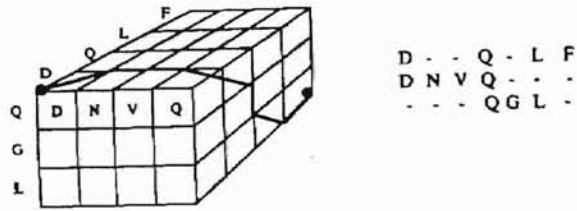


Figure 7. A three-dimensional multiple alignment matrix for 3 protein sequences².
(from Carrillo, H. *et al* 1988)

This approach can generate an optimal alignment but the running time and space are expensive. Suppose that there are N sequences and each sequence is of length L . Then the space required is L^N . From each of the L^N nodes, to find an optimal path, $2^N - 1$ positions need to be considered. For example, if $N = 3$, then 7 positions need be visited from the current position (Figure 8). Then the total running time of the method is $O(2^N \times L^N)$.

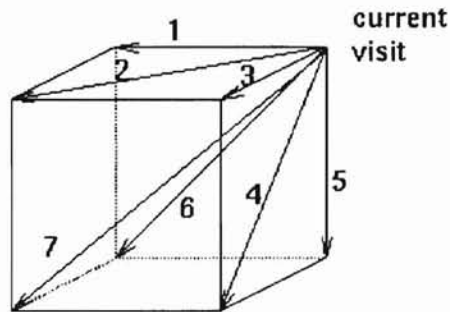


Figure 8. To calculate each node, $2^N - 1$ positions need to be visited

MSA is a method that uses multidimensional dynamic programming approach to align sequences. In MSA, Carillo and Lipman algorithm²⁴ is implemented to reduce the volume of the multidimensional dynamic programming array.

4.1.2 Tree or Hierarchical Method

Although employing Carillo-Lipman algorithm will reduce the time and space greatly, optimal alignment may not be obtained for more than eight sequences with medium size and similarity due to the long time and large memory space required¹⁶. Therefore heuristic alignment algorithms are used to deal with large problems.

Progressive multiple alignment algorithm^{9,22,38} is one of the heuristic alignment algorithms. It builds an alignment progressively by a series of pairwise alignments. First, each pair of sequences is aligned and the distance between them is calculated. Then, based on the distances a guide tree is calculated. At the end, all of the sequences are progressively aligned based on the guide tree. The overview of this process is showed in Figure 9.

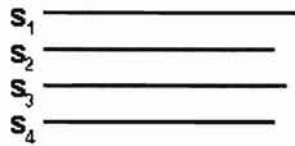
Although progressive alignment algorithms give heuristic results, the results are reasonable in many cases. And this kind of algorithm is simple, fast and efficient. That's why progressive approach is the most used one when handling large problems⁷.

Feng-Doolittle algorithm was one of the first progressive multiple alignment algorithms⁹. CLUSTL W³⁸ is one of the most powerful multiple sequence alignment packages known which performs multiple sequence alignments based on Feng and Doolittle's algorithm.

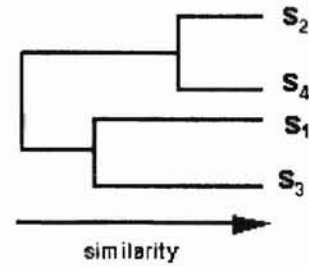
Steps in Multiple Alignment

(A) Pairwise Alignment

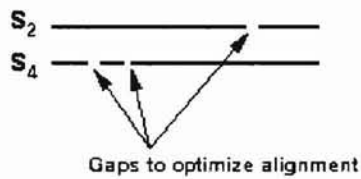
Example - 4 sequences $S_1 S_2 S_3 S_4$



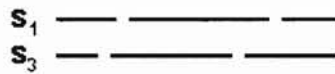
6 pairwise comparisons
then cluster analysis



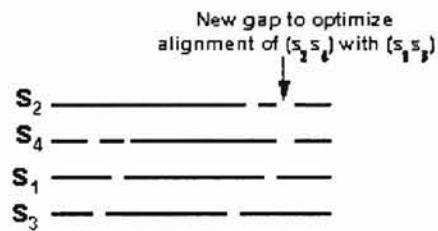
(B) Multiple alignment following the tree from A



align most similar pair



align next most similar pair



align alignments - preserve gaps

Figure 9. The stages in generating a multiple sequence alignment using progressive approach⁴²
(from Sternberg, M.J.E. 1996)

4.1.3 Statistical Approach

If proteins share a common ancestor, they inherit many similarities from their ancestor. It is possible to create a statistical model of a protein family with these similarities⁷. This model is called a Markov model and can be viewed as a finite Markov chain with a starting state (BEGIN) and a stopping state (END).

“A Markov chain can be defined as a sequence of random variables. Each variable can be generated from a finite number of variables preceding it, possibly with some random variable added”⁷. Markov chain can be viewed graphically as a collection of “states”, with arrows between the states. Each arrow has an associated non-negative integer weight (transition probability), which is the probability for the system to go from the current state to a particular new state.

DNA and protein sequences can be viewed as Markov chains of k states, each state corresponds to a particular residue. $k = 4$ for DNA (Figure 10) or $k = 20$ for protein.

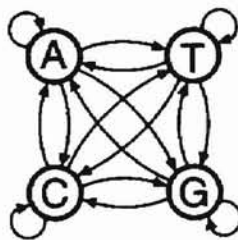


Figure 10. Markov chain of DNA⁷
(from Durbin, R., *et al* 1998)

“A hidden Markov model is a statistical model which describes a series of observations by a “hidden” stochastic process, a Markov process”³⁰. A linear hidden Markov model (HMM) for sequence alignment is a HMM which only models the

primary structure of a protein family, that is, the amino acid sequence of the protein. Hidden Markov models have been used widely in automatic speech recognition and now in molecular biology, especially in database searching and multiple sequence alignment⁷.

HMM is used successfully in multiple sequence alignment because it can capture the statistical details of the multiple alignment, such as position-dependent character distributions and position/length-dependent gap penalties and so on. It therefore can capture the structure intuition of a protein family. Thus HMM has the advantage of characterizing an entire family of sequences. After establishing a HMM, a sequence can align to this model. If this sequence belongs to this model, it will be given a high probability by the model. SAM^{21,23} is one of the well-known software tools that perform multiple sequence alignment based on hidden Markov models.

4.2 Clustal W Analysis

4.2.1 Introduction

Clustal W³⁸ is derived from the series of program Clusters¹⁸, “W” here stands for “weighting”—give different weights to sequences and parameters at different positions in alignment.

Clustal W is based on the Feng-Doolittle method⁹ (Figure 11), however makes many improvements. Feng-Doolittle method is a heuristic multiple sequence alignment method using progressive alignment approach which has two major problems¹⁹, Clustal W method resolves the two problems well.

One problem in Feng-Doolittle method is that any mistakes (misaligned regions) made early in the alignment process cannot be corrected later. This is because of the “greedy” nature of the alignment strategy. Since the alignment process is guided by a guide tree, improving the correctness of guide tree may result in a good alignment. In Feng-Doolittle’s algorithm, UPGMA method³⁴ is used to create a guide tree. In Clustal W, neighbor-joining alignment algorithm is used to build a tree to guide the progressive alignment. It gives better results.

Another problem is the choice of alignment parameters. In Feng-Doolittle method, one weight matrix and two gap penalties (one for opening a new gap and one for extending an existing gap) are used. This choice will give good results only if sequences are all closely related. If sequences are very divergent, different weight matrices should be given at different alignment stages according to the divergence of the sequences to be aligned⁷. Furthermore, in protein alignments, gaps do not occur randomly, position-

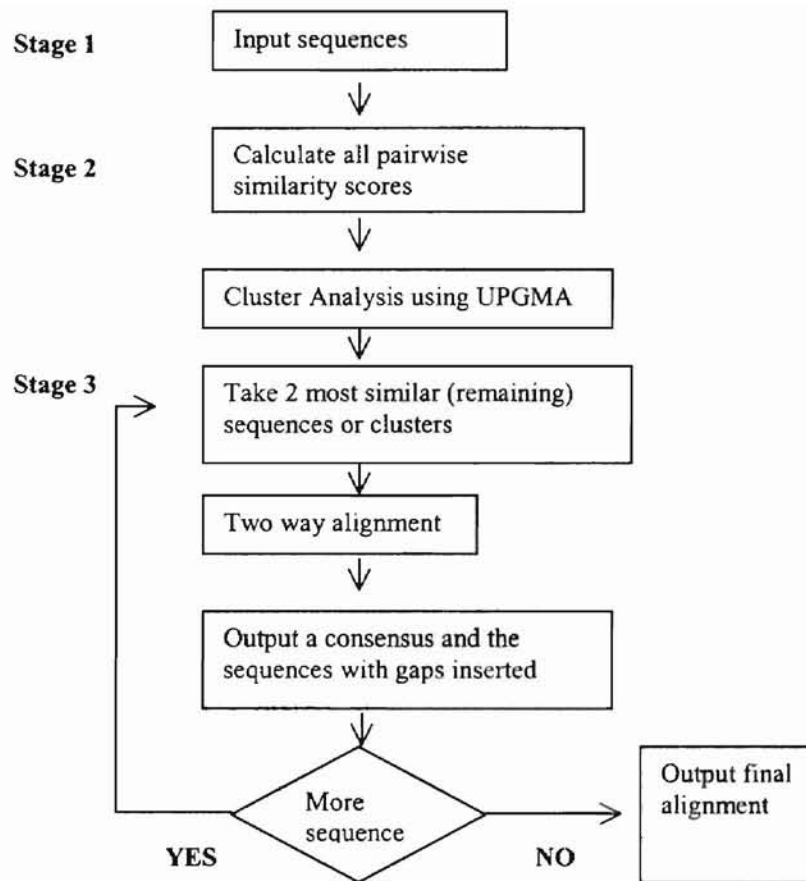


Figure 11. Flow chart of Feng-Doolittle Algorithm¹⁸
 (modified from Higgins D.G., *et al* 1988)

specific gap opening and extension penalties should be provided. Clustal W improves this problem by providing different matrices and position-dependent gap penalties.

Clustal W also gives different weights to different sequences. This is because Clustal W uses the sum-of-pair (SP) scoring scheme to score alignment and there is a problem with sum-of-pair scores⁷. Using SP scores, the relative difference in score between the correct alignment and the incorrect alignment decreases with the number of sequences in the alignment. But in real world the relative difference ought to increase with the sequence number. This shows that evolutionary events are over-counted. This problem increases as the number of sequence increases. To resolve this problem, sequences are weighted. Groups of closely related sequences receive lowered weights because they contain much duplicated information. Highly divergent sequences receive high weights. Sequence weights are calculated directly from the guide tree.

The basic multiple alignment process consists of three major stages:

1. To construct a distance matrix of all pairs by pairwise dynamic programming alignment, giving the evolutionary distance between each pair.
2. To construct a guide tree from the distance matrix by a neighbor-joining clustering algorithm by Saitou and Nei³².
3. To align sequences progressively according to the branch order in the guide tree, using sequence-sequence, sequence-profile, and profile-profile alignment.

An example is shown in Figure 12.

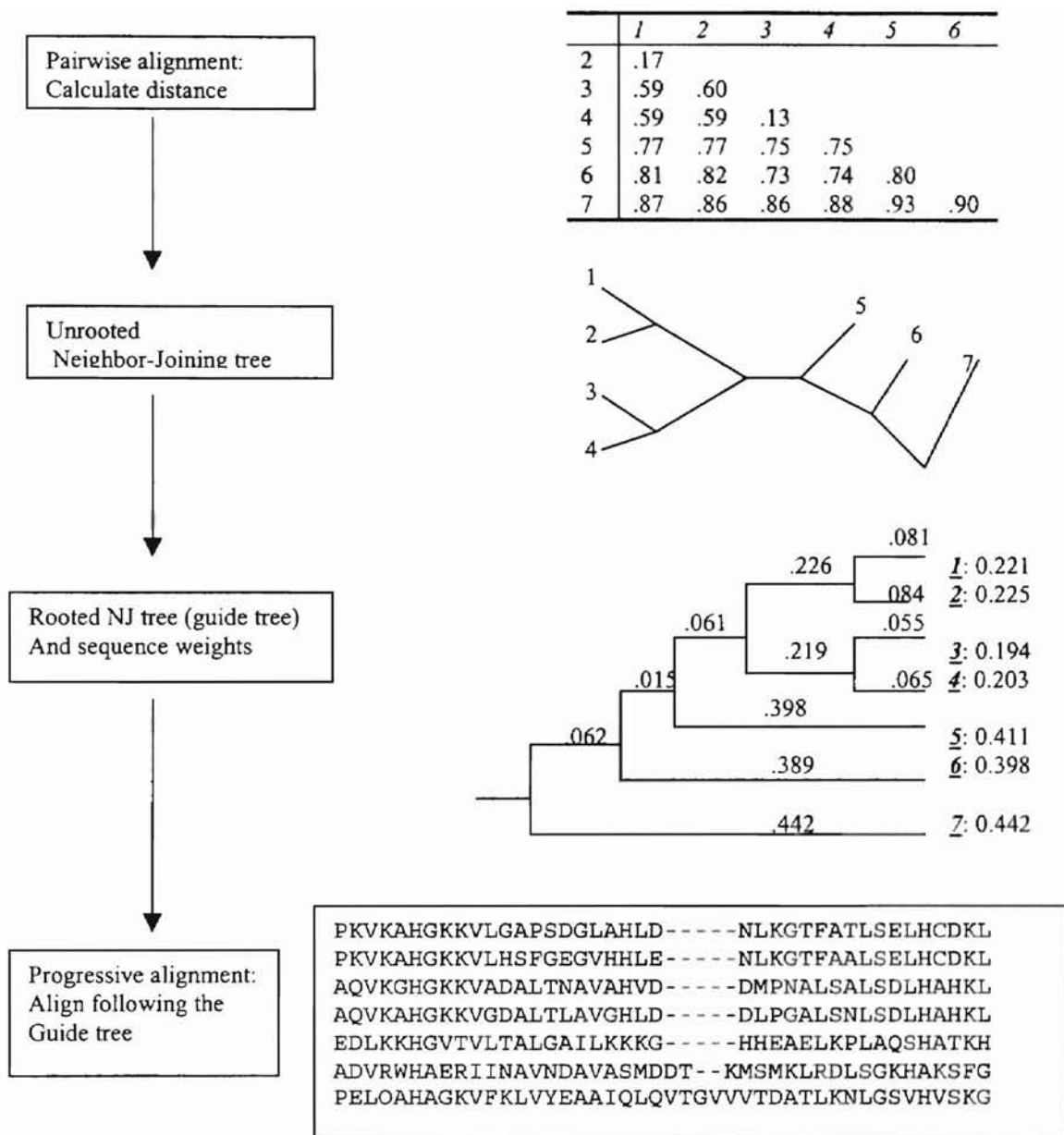


Figure 12. The basic progressive alignment procedure, illustrated using 7 sequences³⁸.
(modified from Thompson, J.D., *et al* 1994)

4.2.2 Algorithm Analysis

1. Pairwise Alignments

The purpose of making pairwise alignments is to build guide tree, which is used to guide the multiple alignment. Because this stage is time-consuming, a heuristic linear time pairwise alignment algorithm was used in previous Clustal W package. Now the full dynamic programming approach is used for the pairwise alignments in Clustal W, because the guide tree is very important to the correctness of alignment.

If there are N sequences, there will be $N(N-1)/2$ pairs of alignments. Suppose each sequence is approximately of the same length L , then each pairwise alignment takes time $O(L^2)$. The total time used by stage 1 is:

$$O(N(N-1)/2 \times L^2) = O(N^2L^2).$$

The distance score between each pair of sequences is calculated by the total number of identical residues divided by the total number of residues compared (gap positions are excluded) in the best alignment. Because only the score of the alignment is required, linear space $O(L)$ is used to make pairwise alignment.

2. The Guide Tree

The guide tree is used to guide the final multiple alignment process. It is calculated from the distance matrix at stage 1. In Clustal W, Saitou and Nei's neighbor-joining method³² is used to construct such tree. The guide tree is also used to derive a weight for each sequence.

Algorithm: Neighbour-Joining

(modified from Saitou, N., 1987³² and Durbin, R., *et al* 1998⁷)

Initialization:

Define T to be the set of leaf nodes, one for each given sequence, and the set $S = T$;

Iteration:

1. find closest i, j in S , that is: $D(i, j) = \min \{ D(k, l): k, l \in S \}$
2. Define a new node c and new set S
cluster $\{i, j\} = c$
 $S \leftarrow S - \{i, j\}$
 $S \leftarrow S \cup c$
3. Add the new node c to T , joining c to i and j with edges of lengths

$$d(c, i) = \left(d(i, j) + \frac{\sum_{k \in S} d(i, k)}{|S| - 2} - \frac{\sum_{k \in S} d(j, k)}{|S| - 2} \right) \times \frac{1}{2}$$

$$\text{and } d(c, j) = d(i, j) - d(c, i)$$

4. Calculate $d(c, k), k \in S$

$$d(c, k) = \frac{1}{2}(d(i, k) + d(j, k) - d(i, j))$$

Termination:

If $|S| = 2$, stop

In this algorithm, D is a matrix of size $O(|S|^2)$ and each element $D(i, j)$ is defined

as:

$$D(i, j) = \frac{\sum_{m < n} d(m, n)}{|S| - 2} - \frac{\sum_{k \in S} d(i, k) + \sum_{k \in S} d(j, k)}{2 \times (|S| - 2)} + \frac{d(i, j)}{2}$$

$\sum_{m < n} d(m, n)$ and $\sum_{i, k \in S} d(i, k)$ can be calculated in the same time. The total calculation

is $O(|S|^2)$. There are $O(|S|^2)$ values in the matrix D , so the total time to calculate the whole matrix D is $O(|S|^2 + |S|^2) = O(|S|^2) = O(N^2)$ since $|S| = N$.

The iteration stage repeats $O(N)$ times where N is the number of sequences.

Therefore the time to create the guide tree is $O(N^3)$.

This neighbor-joining method produces an unrooted tree with branch lengths proportional to the estimated distance along each branch. The tree may be changed to a rooted tree by placing the root at the position where the branch lengths on either side of the root are equal. The weight of each sequence can be calculated as the distance from the root to the leaf. If several sequences share a common branch, they share the weight derived from the shared branch.

3. Progressive Alignment

After finished calculating the guide tree, sequences are aligned progressively, following the branching order of the guide tree. These alignments are a series of pairwise alignments between sequence-sequence, sequence-profile or profile-profile.

Profiles are blocks of pre-aligned sequences. Profile alignment is used to align two existing alignments or to add a series of new sequences to an existing alignment. The profile alignments are an extension of the standard sequence-sequence alignments. Each of the two input alignments is treated as single sequence and the scoring method is changed correspondingly.

C_i and C_j are defined as two profiles and the number of sequences in them are $|i|$ and $|j|$. Define h_1 be a column in C_i and h_2 be a column in C_j . Then the score at position h_1 and h_2 is:

$$W(c_i(h_1), c_j(h_2)) = \frac{\sum_{m=1}^{|i|} \sum_{n=1}^{|j|} w(a_m(h_1), b_n(h_2))}{|i| |j|}$$

After an alignment is completed, gap symbols in this alignment are replaced with a neutral X character. This let the gaps in early stages remain through to later stages. This

rule is called “once a gap, always a gap”. This idea is that the early compared sequences are closely related sequence. Gaps from the result of the early comparison should not be moved because of later alignment with more distantly related sequences.

At each step in the final progressive alignment stage, Myers and Miller²⁷ algorithm is used. This is a memory efficient dynamic programming algorithm. It uses linear space to align two sequences so it can make very large alignments in very little memory.

If there are N sequences ready to be compared, there will have $O(N)$ pairwise alignments, including sequence-sequence, sequence-profile or profile-profile alignments. Suppose two profiles of length L are aligned and the number of sequences in them are $|i|$ and $|j|$ separately, then the time to compute the alignment of the two profiles is $O(L^2|i||j|)$. According to the scoring scheme introduced in page 35 for profile alignment, each individual sequence or a sequence in a profile will be compared with another different sequence once and only once. Since there are $N(N-1)/2$ pairwise comparisons and each comparison takes time proportional to $O(L^2)$, the total time of the progressive alignment stage is $O(\frac{N(N-1)}{2} \times L^2) = O(N^2 L^2)$.

4.3 SAM (Sequence Alignment and Modeling System)

4.3.1 Introduction

SAM is a collection of software tools^{21, 23} for creating, refining and using a type of statistic model called a linear hidden Markov model (HMM) to analyze biological sequences, including sequence alignment and sensitive database searching. This type of HMM is called linear HMM because it only models primary structure (sequence) information of biological sequences.

Linear HMMs are highly effective methods in modeling a family of sequences or a common subsequence within a set of sequences. Given a set of related sequences, the system can automatically train a linear HMM representing the family. The trained HMM can then be used for multiple alignment or database searching.

The primary algorithms and methods used by SAM and other HMM systems were first described by Krogh *et al*^{21, 23}. A basic flow chart for using SAM is shown in Figure 13. The process of making multiple sequence alignment by HMM trained from unaligned sequence contains three stages:

- 1) Initialization: Choose an initial model and initialize parameters.
- 2) Training: Align each test sequence to the model and estimate the model using the Baum-Welch algorithm.
- 3) Multiple alignment: After the final HMM is established, all sequences are aligned to the final model using the Viterbi algorithm and build a multiple alignment by tracing back to the model.

The linear HMM architecture and algorithms used in each stage are analyzed in the following paragraph.

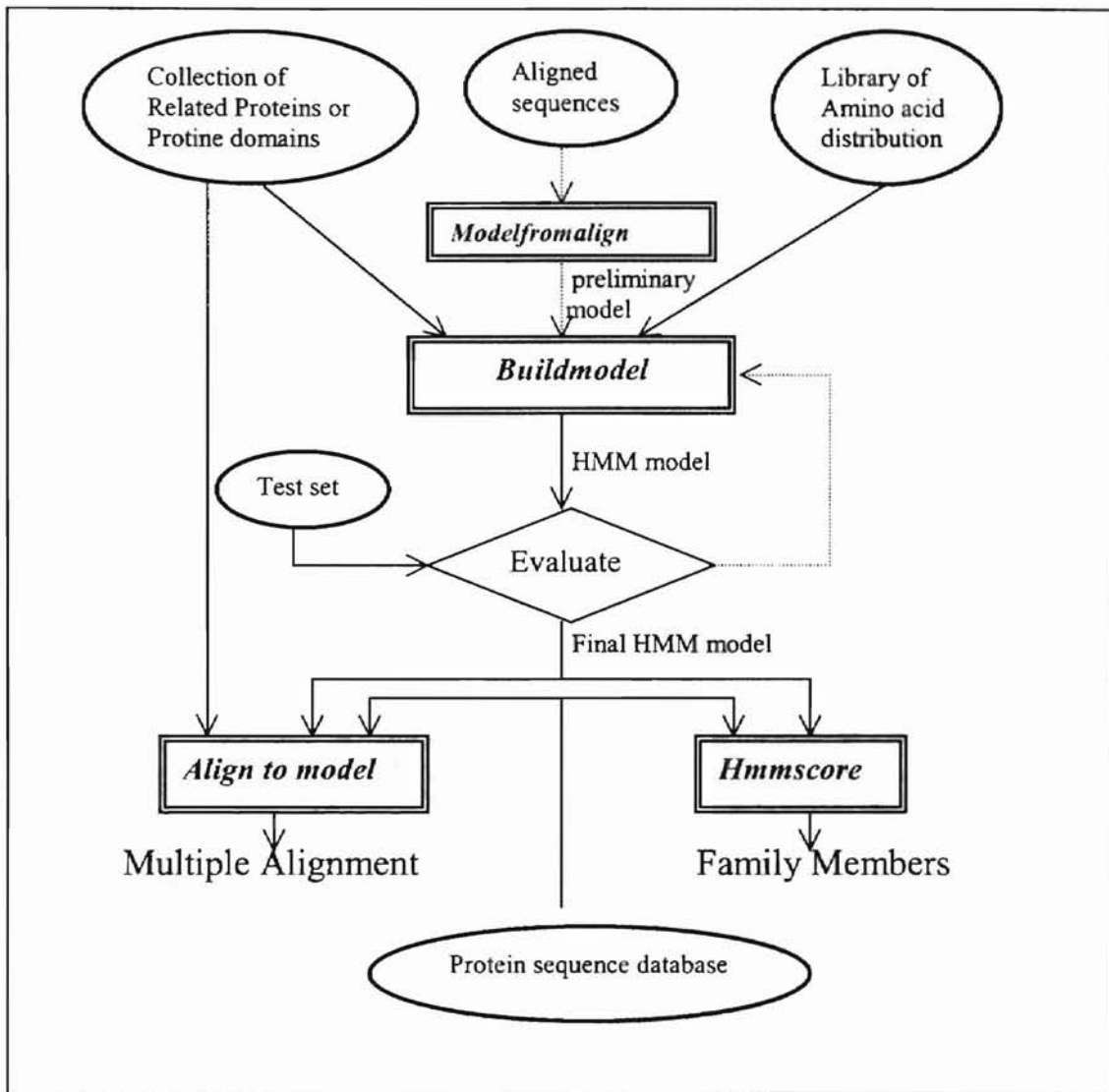


Figure 13. SAM program outline⁴³
 (modified from UCSC computational biology group)

4.3.2 Linear HMM Architecture

A linear HMM for a family of protein sequences can be viewed as a model that generates amino acid sequences by a random process²¹. One type of such model is illustrated in Figure 14.

The linear HMM in Figure 14 contains a set of positions and each position typically has three states: match state (square), insert state (diamond) and delete state (circle). The positions roughly correspond to positions in a protein or columns in a multiple sequence alignment.

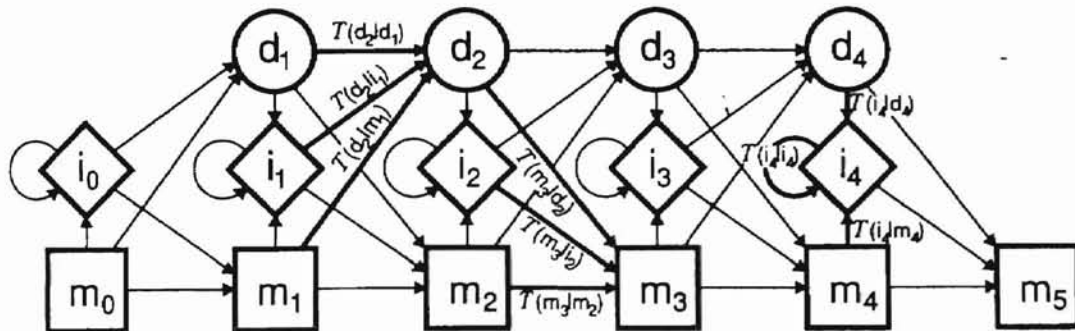


Figure 14. Linear HMM²³
(from Krogh, A., *et al* 1994)

Match state corresponds to matching a single character to a column of the multiple alignment. Insert state corresponds to insert character in alignment. Each of the match states and insert states can emit a character (observation) x from the 20 amino acid alphabet according to an emission probability. At a match state m_k , the emission probability is denoted as: $P(x | m_k)$; At an insert state i_k , this is denoted as: $P(x | i_k)$.

Delete state d_k corresponds to a gap in an alignment. It implies that the character at position k is deleted. Therefore it does not produce any character but skip state k to another state.

In linear HMM, a "BEGIN" and a "END" state are added, denoted by m_0 and m_{M+1} where M is the number of positions. In the model in Figure 14, $M = 4$.

From each state, there are three possible transitions to other state:

$$m_k \rightarrow d_{k+1}, i_k, m_{k+1},$$

$$d_k \rightarrow d_{k+1}, i_k, m_{k+1},$$

$$i_k \rightarrow d_{k+1}, i_k, m_{k+1},$$

The transition is represented by transition probability $T(r | q)$, which denotes the probability of going from state q to state r . Transitions into match or delete states always move forward in the model, whereas transitions into insert states do not move forward. There is a loop on insert state which implies multiple insertions.

A sequence is supposed to be generated by a "random walk" through the model. This random walk starts at state "BEGIN", follows a transition to state m_1 , d_1 or i_0 randomly according to the transition probability $T(m_1 | m_0)$, $T(d_1 | m_0)$ or $T(i_0 | m_0)$. Suppose m_1 is chosen as the current state, m_1 will emit a character x according to the emission probability $P(x | m_1)$. This transition/emission process will continue until reaching the "END" state. At the end of the process, two sequences are generated. One sequence is the observed amino acid sequence $x_1 \dots x_L$, and another sequence is the state sequence $q_0 q_1 \dots q_{M+1}$ where q_0 is the BEGIN state and q_{M+1} is the END state. The state sequence defines a path that generates the amino acid sequence. It is a Markov chain but can not be observed directly.

Given a model, the probability that the path $q_0 q_1 \dots q_{L+1}$ is taken and the sequence $x_1 \dots x_L$ is produced by the path is:

$$\text{Prob}(x_1 \dots x_L, q_1 \dots q_L | \text{model}) = \left[\prod_{i=1}^L T(q_i | q_{i-1}) P(x_i | q_i) \right] \times T(q_{L+1} | q_L) \quad (1)$$

By taking the negative logarithm of equation (1) to avoid underflow, equation (1) can be turned into the equation:

$$-\log \text{Prob}(x_1 \dots x_L, q_1 \dots q_L | \text{model}) = \left[\sum_{i=1}^L -\log T(q_i | q_{i-1}) - \log P(x_i | q_i) \right] - \log T(q_{L+1} | q_L) \quad (2)$$

Many paths can generate the sequence $x_1 \dots x_L$. When aligning a sequence to a model, the path with the highest probability is chosen. This path can be found by a dynamic programming algorithm, called Viterbi algorithm.

Given a model, the full probability that the sequence $x_1 \dots x_L$ is generated is calculated as the sum over all possible paths that could produce this sequence:

$$\text{Prob}(x_1 \dots x_L | \text{model}) = \sum_{q_1 \dots q_L} \text{Prob}(x_1 \dots x_L, q_1 \dots q_L | \text{model}) \quad (3)$$

In practice, equation (3) is used in the negative logarithm form to avoid underflow. The negative logarithm score of equation (3) is called the Negative Log-Likelihood score of a sequence.

$$NLL(x_1 \dots x_L | \text{model}) = -\log \text{Prob}(x_1 \dots x_L | \text{model}) \quad (4)$$

The forward algorithm, which is similar to the Viterbi algorithm, is used to calculate the full probability.

Given a set of training sequences $s(1), \dots, s(j), \dots, s(n)$, where $s(j) = x_1 \dots x_L$, the probability these sequences are generated by the model is evaluated as:

$$\text{Prob}(S(1), S(2) \dots S(n) | \text{model}) = \prod_{j=1, \dots, n} \text{Prob}(S(j) | \text{model}) \quad (5)$$

Therefore, SAM can be divided into three stages. Firstly, a model is initialized. Then the model is trained or estimated in order to maximize the score from equation (5). A maximum likelihood method called Baum-Welch method is used to estimate a model. Finally, sequences are aligned to the model to output the multiple alignment sequence alignment results. The algorithms in each stage are analyzed in the following paragraphs.

4.3.3 Model Initialization

A hidden Markov model can be viewed as a model of five tuples: a finite set of possible states; a finite set of possible observations; a set of transition probabilities; a set of observation probabilities and a set of initial state distributions.

For a given HMM, the possible states and observations are fixed. The transition probabilities, observation probabilities and initial state distribution are the parameters that should be provided by users. After the parameters are given, they will be estimated to build the final HMM.

There are two ways to set the transition and emission probabilities to build a model:

- Train or estimate a model automatically from unaligned sequences. The model learns the parameters automatically from these unaligned sequences by a training algorithm. SAM uses the Baum-Welch algorithm to estimate parameters.
- Build a model from pre-aligned sequences. Sequences are pre-aligned in prior according to the information from the three-dimensional structure of proteins. The

transition or emission probability at each position is calculated. A model built in this way will model protein family better.

In contrast to transition and emission probabilities, a default linear HMM architecture (the number of states, how they are connected by state transitions) must be designed in priori by hand. Given the default architecture, the only way the architecture of the model can be varied is in the length, which is usually represented by the number of match states in HMM. A commonly used rule to set this number is to set it to be the average length of the training sequences.

4.3.4 Model Estimation: the Baum-Welch Algorithm

The most common used estimation method to estimate a model is the Baum-Welch or forward-backward algorithm³⁰. It is a maximum likelihood method and the outline is the following (from Waterman, M.S., 1995)³⁹:

1. Choose an initial model. If no prior information is available, make all transitions equally likely.
2. Align each sequence to the model. Use dynamic programming to find the maximum likelihood path for each sequence.
- 2' Collect the count statistics.
 $n(y) = \# \text{ paths through state } y$
 $n(y'|y) = \# \text{ paths that have } y \rightarrow y'$
 $m(a|y) = \# \text{ times letter } a \text{ was produced at state } y$
3. Reestimate the parameters of the model
 $P(y'|y) = n(y'|y) / n(y)$
 $P(a|y) = m(a|y) / n(y)$
4. Repeat steps 2 to 3 until parameter estimates converge

The probability that the transition $y \rightarrow y'$ is used at position m can be calculated by the forward and backward algorithms. The total number of transitions $y \rightarrow y'$ is

obtained by summing over all positions and over all sequences. The value $m(a|y)$ is calculated similarly.

1. Forward Algorithm

Given a sequence $x_1 \dots x_L$, the probability of the subsequence $x_1 \dots x_i$ generated and the character x_i emitted by state k is:

$$f_k(i) = \text{Prob}(x_1 \dots x_i, \pi_i = k | \text{model})$$

This value can be calculated recursively by the forward algorithm. The recursion function is:

$$f_k(i) = P(x_i | k) \sum_l f_l(i-1) T(k | l)$$

The probability of the sequence $x_1 \dots x_L$ is generated using the following formula:

$$\text{Prob}(x_1 \dots x_L | \text{model}) = \sum_k f_k(L) P(k | 0)$$

As mentioned previously, there are three possible transitions from one state to other state: $m_k \rightarrow d_{k+1}, i_k, m_{k+1}, d_k \rightarrow d_{k+1}, i_k, m_{k+1}, i_k \rightarrow d_{k+1}, i_k, m_{k+1}$ in a linear HMM.

Therefore the forward algorithm for a HMM can be implemented as:

Forward algorithm (from Rabiner, L.R. 1989)³⁰

Initialization: $fM_0(0) = 1$

Recursion:

$$fM_k(i) = P(i | M_k) \times \begin{bmatrix} fM_{k-1}(i-1)T(M_k | M_{k-1}) \\ + fI_{k-1}(i-1)T(M_k | I_{k-1}) \\ + fD_{k-1}(i-1)T(M_k | D_{k-1}) \end{bmatrix}$$

$$fI_k(i) = P(i | I_k) \times \begin{bmatrix} fM_{k-1}(i-1)T(I_k | M_{k-1}) \\ + fI_{k-1}(i-1)T(I_k | I_{k-1}) \\ + fD_{k-1}(i-1)T(I_k | D_{k-1}) \end{bmatrix}$$

$$fD_k(i) = \begin{bmatrix} fM_{k-1}(i)T(D_k | M_{k-1}) \\ + fI_{k-1}(i-1)T(D_k | I_{k-1}) \\ + fD_{k-1}(i-1)T(D_k | D_{k-1}) \end{bmatrix}$$

Termination:

$$fM_{m+1}(L+1) = \begin{bmatrix} fM_m(L)T(M_{m+1} | M_m) \\ + fI_m(L)T(M_{m+1} | I_m) \\ + fD_m(L)T(M_{m+1} | D_m) \end{bmatrix}$$

2. Backward Algorithm

Given a sequence $x_1 \dots x_L$, with the character x_i is emitted by state k , the probability of the subsequence $x_{i+1} \dots x_L$ is generated is:

$$b_k(i) = \text{Prob}(x_{i+1} \dots x_L | \text{model}, \pi_i = k).$$

$b_k(i)$ is calculated by the backward algorithm. The recursion function is:

$$b_k(i) = \sum_l P(x_{i+1} | l)T(k | l)b_l(i+1)$$

Backward algorithm (from Rabiner, L.R. 1989)³⁰

Initialization ($i = L$):

$$\begin{aligned} bM_{m+1}(L+1) &= 1 \\ bM_m(L) &= T(M_{m+1} | M_m) \\ bI_m(L) &= T(M_{m+1} | I_m) \\ bD_m(L) &= T(M_{m+1} | D_m) \end{aligned}$$

Recursion($i = L-1, \dots, 1$):

$$bM_k(i) = \begin{bmatrix} bM_{k+1}(i+1)T(M_{k+1} | M_k)P(x_{i+1} | M_{k+1}) \\ + bI_k(i+1)T(I_k | M_k)P(x_{i+1} | I_k) \\ + bD_{k+1}(i)T(D_{k+1} | M_k) \end{bmatrix}$$

$$bI_k(i) = \begin{bmatrix} bM_{k+1}(i+1)P(M_{k+1} | I_k)T(x_{i+1} | M_{k+1}) \\ + bI_k(i+1)P(I_k | I_k)T(x_{i+1} | I_k) \\ + bD_{k+1}(i)P(D_{k+1} | I_k) \end{bmatrix}$$

$$bD_k(i) = \begin{bmatrix} bM_{k+1}(i+1)T(M_{k+1} | D_k)P(x_{i+1} | M_{k+1}) \\ + bI_k(i+1)T(I_k | D_k)P(x_{i+1} | I_k) \\ + bD_{k+1}(i)T(D_{k+1} | D_k) \end{bmatrix}$$

3. Baum-Welch Algorithm:

Baum-Welch algorithm is used to estimate parameter values of a HMM, including the emission probabilities and transition probabilities.

Given a sequence $x = x_1 \dots x_L$, the probability that the transition $k \rightarrow l$ is used at position i in sequence x is:

$$\begin{aligned} \text{Prob}(\pi_i = k, \pi_{i+1} = l | x, \text{model}) &= \frac{\text{Prob}(x, \pi_i = k, \pi_{i+1} = l | \text{model})}{\text{Prob}(x | \text{model})} \\ &= \frac{f_k(i) \times T(l | k) \times P(x_{i+1} | l) \times b_l(i+1)}{\text{Prob}(x | \text{model})} \end{aligned}$$

The expected number of times that the transition $k \rightarrow l$ is used is calculated over all position in sequence $x = x_1 \dots x_L$:

$$n(k \rightarrow l) = \frac{\sum_i f_k(i) \times T(l | k) \times P(x_{i+1} | l) \times b_l(i+1)}{\text{Prob}(x | \text{model})} \quad (6)$$

Given a model, the probability that the sequence $x_1 \dots x_L$ is generated and the character x_i is emitted by state k is:

$$\begin{aligned} &\text{Prob}(x_1 \dots x_L, \pi_i = k | \text{model}) \\ &= \text{Prob}(x_1 \dots x_i, \pi_i = k | \text{model}) \times \text{Prob}(x_{i+1} \dots x_L | \pi_i = k, \text{model}) \\ &= f_k(i) \times b_k(i) \end{aligned}$$

Then the number of emissions of a character a from state k is:

$$m(a | k) = \frac{\sum_{x_i=a} \text{Prob}(x_1 \dots x_i, \pi_i = k | \text{model})}{\text{Prob}(x | \text{model})} = \frac{\sum_{x_i=a} f_k(i) \times b_k(i)}{\text{Prob}(x | \text{model})} \quad (7)$$

With the expected number of transitions and emissions, the new transition probability and emission probability is given by:

$$P(l | k) = \frac{n(k \rightarrow l)}{\sum_{l'} n(k \rightarrow l')} \quad \text{and} \quad P(a | k) = \frac{m(a | k)}{\sum_{a'} m(a' | k)} \quad (8)$$

According to the architecture of HMM, the expected emission and transition count from sequence x are calculated using equations (6) and (7) as:

$$\begin{aligned} m(a | M_k) &= \frac{1}{P(x)} \sum_{i|x_i=a} f_{M_k}(i) b_{M_k}(i) \\ m(a | I_k) &= \frac{1}{P(x)} \sum_{i|x_i=a} f_{I_k}(i) b_{I_k}(i) \\ n(X_k \rightarrow M_{k+1}) &= \frac{1}{P(x)} \sum_i f_{X_k}(i) T(M_{k+1} | X_k) P(x_{k+1} | M_{k+1}) b_{M_{k+1}}(i+1) \\ n(X_k \rightarrow I_k) &= \frac{1}{P(x)} \sum_i f_{X_k}(i) T(I_k | X_k) P(x_{k+1} | I_k) b_{I_k}(i+1) \\ n(X_k \rightarrow D_{k+1}) &= \frac{1}{P(x)} \sum_i f_{X_k}(i) T(D_{k+1} | X_k) b_{D_{k+1}}(i) \end{aligned}$$

Then the Baum-Welch algorithm used to estimate paths of HMM is:

Baum-Welch algorithm (from Rabiner, L.R. 1989)³⁰:

Initialization: Pick arbitrary model parameters

Recursive Step:

For each sequence $j = 1 \dots n$:

Calculate $f_k(i)$ for sequence j using the forward algorithm

Calculate $b_k(i)$ for sequence j using the backward algorithm

Accumulate the expected emission and transition counts

Calculate the new model parameter using equation (8)

Calculate the new log likelihood of the model

Termination:

Stop if the change in log likelihood is less than some predefined threshold or the maximum number of iterations is exceeded.

4.3.5 Multiple Alignment: the Viterbi Algorithm

After a model is established, the Viterbi algorithm is used to find the most probable path through a model for a sequence. A dynamic programming technique is used to find the best alignment and its probability without going through all the possible alignments. When making multiple sequence alignment, each sequence is aligned to the model. The Viterbi algorithm calculates the alignment for each of the sequences.

Characters aligned to the same match state are aligned in column. For sequences that do not have a match to a certain match state, a gap character is added.

Viterbi algorithm is similar to the forward algorithm, but the recursion function is changed to: $v_k(i) = P(x_i | k) \max_l f_l(i-1)T(k | l)$.

4.3.6 Time and Space Complexity of SAM

The forward, backward and the Viterbi algorithms all use dynamic programming techniques. Suppose the length of the model is M , then the number of states in the model is about $3M$. Let L be the length of a sequence, then it requires $O(ML)$ steps to calculate the whole dynamic programming matrix. At each step, constant number of transitions or emissions is considered. Therefore, the running time is $O(ML)$, and the space required by these algorithms are also $O(ML)$.

When aligning N sequences, the total time is proportional to the sum of all the states over all re-estimation cycles, multiplied by the total length of all the training sequences. In the Baum-Welch algorithm in page 47, the computational time is $O(NML)$ per iteration. This recursive step will repeats many times until a threshold or the maximum number of iterations is exceeded.

4.4 MSA (Multiple Sequence Alignment)

4.4.1 Introduction

MSA is a global multiple sequence alignment program originally written and distributed in 1989²⁴ and later modified by Gupta, Kececioglu and Schaffer¹⁶. It is one of the programs attempt to find an optimal global alignments of multiple DNA or protein sequences by implementing a variant of a multi-dimensional dynamic programming technique – Carrillo-Lipman method²⁴. It reduces the amount of space required by giving an upper bound of the space. Then, a variant of Dijkstra's shortest paths algorithm⁶ is implemented to find an optimal alignment by searching the dynamic programming graph.

Even though MSA reduces memory space required for multiple alignments, it is still uses much more memory than the progressive pairwise technique. Generally speaking, MSA will produce better alignments than most multiple sequence alignments, but in practice this is not always the case²⁴. The drawback with using MSA is that it requires an enormous amount of computer time and memory to align more than a few distantly related sequences. The size of the problems solved by MSA is directly related to the sequence lengths, the number of sequences, and the amount of sequence diversity.

In the following paragraphs, the algorithm of Carrillo-Lipman method used in stage one and the variant of Dijkstra's shortest paths algorithm in stage two are analyzed.

4.4.2 Setting the Bound and Regions – stage I in MSA

1. Basic Idea

Two sequences alignment can be viewed as a path within a two-dimensional array (Figure 15).

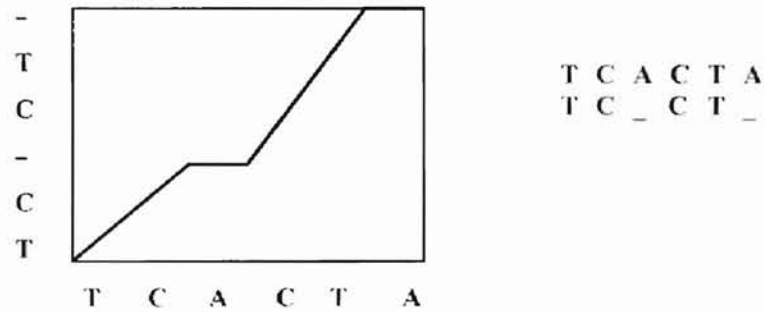


Figure 15. Graphic view of pairwise sequence alignment

For the case of three sequences, the optimal alignment can be viewed as a path pass through a three-dimensional array (Figure 16).

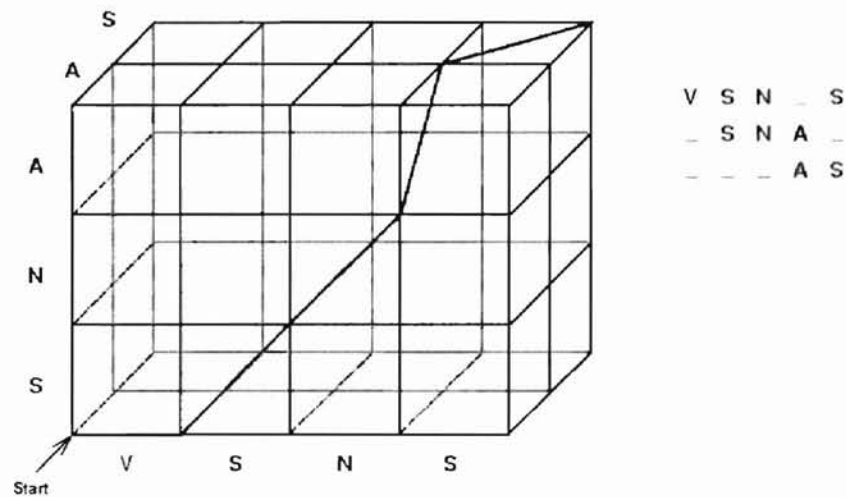


Figure 16. Graphic view of an alignment of three sequences¹¹
(from Fuellen, George 1997)

If there is a good global alignment between two related sequences, the memory cells that contain this alignment would be near the center diagonal of the two dimensional array (Figure 15). Likewise, if there is a good global alignment among three related sequences, it is expected to make use of the memory cells near the center cube of the three-dimensional array (Figure 16). This implies that an optimal alignment path is supposed to be contained in a “polyhedron” close to the main diagonal²⁴. When making multiple sequence alignment, only these memory cells in this “polyhedron” need to be considered. Therefore, in order to get a minimum edge from the current point from one of the 2^N-1 edges, where N is the number of sequences, only these edges coming from “polyhedron” need be considered. If an edge is coming outside, this edge is ignored. This strategy will reduce the computations significantly.

The polyhedron is obtained by the observation that each multiple alignment implies $N(N-1)/2$ pairwise alignments of the N sequences. If N -sequence alignment is treated as a path in the N -dimensional space, the imposed pairwise alignment $A_{i,j}$ of sequences S_i, S_j can be viewed as a projected path on the plane defined by the two sequences S_i and S_j (Figure 17).

Given an optimal alignment A , the projection $A_{i,j}$ can be obtained by copying rows i and j of A except the columns with the gap character ‘-’ in both rows. For example, the alignment projected by L1 (Figure 17) is: $\begin{matrix} SNA- \\ --AS \end{matrix}$, aligned gaps at the position 1 are ignored. The cost of $A_{i,j}$ will always larger or equal to the cost of an optimal alignment of S_i and S_j .

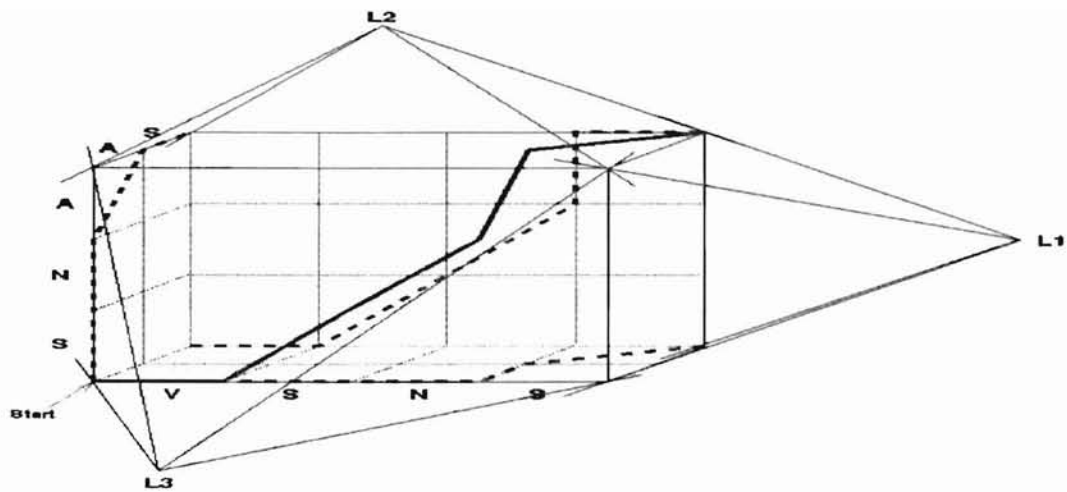


Figure 17. All 3 pairwise projections of the alignment
 - $VSN - S$
 - $SNA -$
 - AS
 (from Fuellen, George 1997)

However, if $N(N-1)/2$ pairwise alignments of N sequences are given arbitrarily, they may not determine a multiple alignment. The $N(N-1)/2$ pairwise alignment must satisfy a Carrillo-Lipman bound. This bound (cost) limits the points through which the optimal pairwise alignment path may pass on each of the two-dimensional plane. It is possible to get the bound for each of the projections of an optimal multiple alignment. Each projection defines a subset of the original N -dimensional array. The intersection of these subsets contains all the paths that may lead to optimal alignments and must be considered by a dynamic programming algorithm to find the final optimal alignment.

2. Calculation of the Carrillo-Lipman Bound

Given a set of sequences s_1, s_2, \dots, s_k , and a cost function c , define

$A =$ any alignment

$c(A)$ = cost of A

A^o = the optimal alignment

$c(A^o) = d(s_1, s_2, \dots, s_k)$ = cost of A^o

A^h = a heuristic alignment

$c(A^h)$ = cost of heuristic alignment

$A_{|i,j}$ = the projection of A on the plane determined by the sequences s_i and s_j .

$c(A^o_{|i,j})$ = the cost of the (i, j) -projection of the optimal alignment

$c(A^h_{|i,j})$ = the cost of the (i, j) - projection of the heuristic alignment

$d(s_i, s_j)$ = cost of the pairwise optimal alignment

MSA assumes that an upper bound value, denoted by U , is known. MSA then searches only these alignments whose costs are less than or equal to U . This upper bound cost must greater than or equal to the cost of an optimal alignment, which is the sum of all pairwise alignments:

$$\begin{aligned} U &\geq c(A^o) = \sum_{i < j} c(A^o_{|i,j}) \\ &= c(A^o_{|p,q}) + \sum_{\substack{i < j \\ (i,j) \neq (p,q)}} c(A^o_{|i,j}) \\ &\geq c(A^o_{|p,q}) + \sum_{\substack{i < j \\ (i,j) \neq (p,q)}} d(s_i, s_j) \\ &= c(A^o_{|p,q}) + \sum_{i < j} d(s_i, s_j) - d(s_p, s_q) \end{aligned}$$

The Carrillo-Lipman bound on the pairwise alignment of sequences s_p and s_q can be got by rearranging this equation:

$$U - \sum_{i < j} d(s_i, s_j) + d(s_p, s_q) \geq c(A^o_{|p,q}) \quad (9)$$

To get an optimal alignment, the Carrillo-Lipman bound for each projection is calculated. Only these paths obey the bound in all their projections are considered. $X_{i,j}$ is define to be the set of all paths whose projections on the plane defined by S_i and S_j have smaller costs than or equal to this bound. Define $R_{i,j}$ be the set of points included in $X_{i,j}$. The intersection of these points $R = \bigcap_{i < j} R_{i,j}$ is the region containing the optimal multiple alignment paths. Points outside this region are ignored.

3. Calculation of the Upper Bound (U)

$$\text{Define } L = \sum_{i < j} d(s_i, s_j)$$

$$\delta = c(A^0 |_{p,q}) - c(s_p, s_q).$$

The upper bound value U in equation (9) can be calculate by $U = L + \delta$.

L is a lower bound value and can be produced by computing all the pairwise alignments of a set of sequences. Because the optimal multiple alignment A^0 is unknown, the δ value can not be calculated directly. There are three ways to obtain this value:

- 1). δ is specified users.
- 2). δ is specified the computation. A heuristic multiple sequence alignment from pairwise alignments at first. Find the projections of this optimal path and calculate the cost of each of the projection. This value is greater than or equal to the optimal pairwise alignment and the difference between them is denoted by $\epsilon_{i,j}$.

$$\epsilon_{i,j} = c(A^h |_{i,j}) - d(s_i, s_j).$$

$$\delta = \sum_{i < j} \epsilon_{i,j}$$

The $\epsilon_{i,j}$ value implies the distance from the optimal pairwise alignment path and paths need be considered. The larger the value is, the more computations are required. Therefore in MSA, a variable *maxepsilon* is defined and set by default to 50. The δ is then computed as

$$\delta = \sum_{i < j} \min(\text{maxepsilon}, \epsilon_{i,j}).$$

Since larger ϵ value may be cut, the full size polyhedron may not have been explored and the optimal alignment may not be found.

3). δ is specified users and computation. Users may supply a file with values for $\epsilon_{i,j}$.

Since all the three options may produce an upper bound U that's too small, MSA cannot guarantee to produce an optimal alignment.

4. Calculation of the Region $R_{i,j}$.

The upper bound value $U = L + \delta$.

L is the lower bound and is computed as the sum over all pairwise alignments.

That is, $L = \sum_{i < j} d(s_i, s_j)$. Suppose there are N sequences and the value of δ is given a

priori, then the time required computing U is $O(\sum_{i < j} |S_i \parallel S_j|)$.

The set $R_{i,j}$ contains a set of points on the plane defined by the sequences S_i and S_j . When point in this set is used by some paths, the best score of the path should be less than or equal to the Carrillo-Lipman bound $U - L + d(S_i, S_j)$.

Suppose the length of S_i is m and the length of S_j is n . If $R_{i,j}(a, b)$ is a point in the set $R_{i,j}$, then the cost of it, represents by $C_{i,j}(a, b)$ is:

$$\begin{aligned} C_{i,j}(a, b) &= d(S_{i,1} \dots S_{i,a}, S_{j,1} \dots S_{j,b}) + d(S_{i,m} \dots S_{i,a}, S_{j,n} \dots S_{j,b}) \\ &\leq d(S_{i,1} \dots S_{i,m}, S_{j,1} \dots S_{j,n}) + U - L \end{aligned}$$

In MSA, the value $U-L$ is set equal to $\epsilon_{i,j}$. This cannot guarantee an optimal alignment any more. That's why in practice, MSA rarely finds a guaranteed optimal alignment¹⁶.

All of these points can be obtained by summing up the forward and reverse scores of a path through these points. The computation required to calculate all the $R_{i,j}$ is

$$O(\sum_{i < j}^N |S_i| |S_j|)$$

The outline of stage I of MSA is as the following:

Stage 1 of MSA (sequence S_1, \dots, S_K ; integer δ) (from Gupta, *et al* 1995)¹⁶:

```

for I ← 1 to K-1 do
  for J ← I+1 to K do
    let  $S_i = a_1 a_2 \dots a_n$  &  $S_j = b_1 b_2 \dots b_m$ 
    for I ← 1 to n do
      for j ← 1 to m do
         $C_{f,I,J}[I, j] = d(a_1 \dots a_I, b_1 \dots b_j)$  and
         $C_{r,I,J}[I, j] = d(a_{I+1} \dots a_n, b_{j+1} \dots b_m)$ 
      od
    od
    for I ← 1 to n do
       $F_{I,J}[I] \leftarrow \{j \mid C_{f,I,J}[I, j] + C_{r,I,J}[I, j] \leq C_{f,I,J}[n, m] + \epsilon_{I,j}\}$ 
    od
     $L \leftarrow L + C_{f,I,J}[n, m]$ 
  od
od
U = L +  $\delta$ 

```

4.4.3 Searching the Optimal Alignment - Stage II in MSA:

In stage 2 of MSA, the optimal alignment is found in the region $R = \bigcap_{i < j} R_{i,j}$.

MSA is a global distance alignment method, it attempts to minimize the cost of the alignment. In stage 2, MSA implements a complex variation of Dijkstra's ⁶ single-source, single-destination, shortest-path algorithm to find a shortest path or an optimal alignment.

To a given set of N sequences S_1, \dots, S_N of length k_1, \dots, k_N , the set of all possible multiple alignments is called a path graph¹⁶. This graph consists of a set of vertices and a set of edges. The vertex set and the edge set are:

$$V = \{ \langle i_1, i_2, \dots, i_N \rangle \mid 0 \leq i_l \leq k_l \}$$

$$E = \{ p \rightarrow q \mid q - p \in \{0, 1\}^N - \{0\}^N \}$$

A multiple sequence alignment can be viewed as a path through the N -dimensional space from the original vertex $\langle 0, \dots, 0 \rangle$ to the end vertex $\langle k_1, \dots, k_N \rangle$

that consists of consecutive edges. For example, the alignment $\left. \begin{array}{l} \text{D--Q-LF} \\ \text{DNWQ---} \\ \text{---QGL-} \end{array} \right\}$ can be

visualized as a path:

$$\begin{aligned} &\langle 0, 0, 0 \rangle \rightarrow \langle 1, 1, 0 \rangle \rightarrow \langle 1, 2, 0 \rangle \rightarrow \langle 1, 3, 0 \rangle \\ &\rightarrow \langle 2, 4, 1 \rangle \rightarrow \langle 2, 4, 2 \rangle \rightarrow \langle 3, 4, 3 \rangle \rightarrow \langle 4, 4, 3 \rangle \end{aligned}$$

With this idea, MSA implements a variant of Dijkstra's shortest paths algorithm to search the basic dynamic programming graph. The fundamental data structures it uses are: Priority queue, which is a queue of edges ranked by distance; Trie, which stores vertices that already exist. The outline of stage 2 is as the following:

Stage 2 of MSA (sequence S_1, \dots, S_K ; integer δ)(from Gupta, *et al* 1995) ¹⁶:

trie T ; priority queue Q ; vertex s, t, v, w ;
point p, q ; edge list E ; edge e, f

$s \leftarrow \text{VERTEX} (s.P = (0, 0, \dots, 0))$
 $t \leftarrow \text{VERTEX} (t.P = (N_1, N_2, \dots, N_k))$
 $T \leftarrow \text{TRIE} ()$; $\text{INSTALL} (s, s.P, T)$; $\text{INSTALL} (t, t.P, T)$
 $e \leftarrow \text{EDGE} (\text{NULL}, s, 0)$
 $E \leftarrow \text{EDGELIST} ()$; $\text{INCLUDE} (e, E)$;
 $Q \leftarrow \text{PRIORITY} ()$; $\text{INSERT} (e, e.D, Q)$;

While NOT EMPTY (Q) do

$e \leftarrow \text{EXTRACT} (Q)$; $v \leftarrow e.\text{head}$;

$q \leftarrow v.P$; $p \leftarrow e.\text{tail}.P$

if $v = t$ then

Output TRACEBACK (t); return

fi

if $e.D + \sum_{i < j} (\text{scale}(S_i, S_j) \times C_{r,i,j}[q,i, q,j]) \leq U$ then

for all $r \leftarrow \text{POINT} (r_1, r_2, \dots, r_k)$ s.t $r - q \in \{0, 1\}^k - \{0\}^k$ and

for all $i < j, r_j \in F_{i,j}[r_i]$) do

$w \leftarrow \text{LOOKUP} (r, T)$

if $w = \text{NULL}$ then

$w \leftarrow \text{VERTEX} (r)$; $\text{INSTALL} (w, w.P, T)$

fi

$f \leftarrow \text{FIND} (q, r, E)$

if $f = \text{NULL}$ then

$f \leftarrow \text{EDGE} (v, w, \infty)$; $\text{INCLUDE} (f, E)$; $\text{INSERT} (f, f.D, Q)$

fi

if $e.D + \text{COST} (f, e) < f.D$ then

$f.D \leftarrow e.D + \text{COST} (f, e)$; $\text{DECREASE} (f, f.D, Q)$;

fi

od

fi

od

output "There does not exist a multiple sequence alignment with cost at most U "

end

CHAPTER V

SUMMARY AND CONCLUSIONS

Sequence alignment is a useful tool in computational biology. Due to the huge number of sequences and limitation of computer hardware resource, it is necessary for the sequence alignment to achieve high throughput with small memory space by efficient algorithms. In order to improve sequence alignment and better use of sequence alignment programs, it is useful to analyze the algorithms in the programs available.

We first introduce the concepts of sequence alignment in computational biology. Then the pairwise sequence alignment algorithms are analyzed. Because of the importance of pairwise alignment, we implement a dynamic pairwise alignment program. We test the time and space complexity of several algorithms in our program. Using standard dynamic programming, the time complexity is $O(N^2)$, however, the space complexity is $O(N^2)$. Using Myers-Miller algorithm, the space required is linear to the sequence length, and the time is $O(N^2)$. Here N is the sequence length.

Pairwise alignment is the base for the multiple sequence alignment. The multiple sequence alignment is much more complicated and useful than pairwise alignment. Currently, three multiple sequence alignment software packages are used mostly. They are Clustal W, SAM and MSA. We analyze the algorithms in each program package, calculate the time and space complexity of each algorithm.

Clustal W uses progressive multiple alignment. The alignment process is divided into three stages: construction of a distance matrix of all pairs, construction of a guide tree and progressive sequence alignment. The time and space complexity at each stage

are $O(N^2L^2)$, $O(N^3)$ and $O(N^2L^2)$ respectively, where L is the length of the sequence (suppose all sequences are of the same lengths) and N is the number of sequences.

SAM uses statistical approach. The alignment process is divided into two stages: establishment of a model and alignment of the sequences to the model. The space complexity is $O(ML)$. Model building is iteration until convergence and the time complexity of each iteration is $O(NML)$, where M is the length of a model, L is the length of sequence and N is the number of sequences. The time complexity of sequence alignment stage is $O(NML)$.

MSA uses multidimensional dynamic programming. Normally, the space and time complexity are $O(L^N)$ and $O(L^N \times 2^N)$ respectively. Carrillo-Lipman algorithm and Dijkstra's shortest paths algorithm are used to reduce the space and computation time. Using these two algorithms, MSA makes multiple sequence alignment in two stages, Setting the cost bound and region and searching the optimal alignment. The model reduces space and time required dramatically. Further study is needed to get time and space complexity of this model.

GLOSSARY

Amino acid: Any of a class of 20 molecules that are combined to form proteins in living things.

Bioinformatics: The discipline of obtaining information about genomic or protein sequence data. This may involve similarity searches of databases, comparing your unidentified sequence to the sequences in a database, or making predictions about the sequence based on current knowledge of similar sequences. Databases are frequently made publically available through the Internet, or locally at your institution.

Consensus Sequence: A derived nucleotide sequence that represents a family of similar sequences. Each base in the consensus sequence corresponds to the base most frequently occurring at that position, in the real sequences.

Dayhoff PAM (percent accepted mutation) family of matrices: scores amino acid pairs on the basis of the expected frequency of substitution of one amino acid for the other during protein evolution. Low PAM is used for closely related sequences, high PAM for distant sequences.

DNA sequence: The relative order of base pairs.

Genetic code: The sequence of nucleotides, coded in triplets (codons) along the mRNA, that determines the sequence of amino acids in protein synthesis.

Genome: Is all the DNA in an organism.

HGP: Is a 13-year project sponsored by the Department of Energy (DOE) and National Institutes of health (NIH). Its goals are to identify the 80,000-100,000 genes (about 3

billion nucleotide base) in human DNA and to develop tools for data analysis by the year 2003.

Homologous protein: proteins which share a common evolutionary history and have similar overall structure and function.

Indel: Insertion and Deletion.

Multiple Sequence comparison: Refers to the search for similarity in three or more sequences.

Mutation: Any heritable change in DNA sequence.

Nucleic acid: A large molecule composed of nucleotide subunits.

Nucleotide: A subunit of DNA or RNA consisting of a nitrogenous base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or cytosine in RNA), a phosphate molecule, and a sugar molecule.

Protein: A large molecule composed of one or more chains of amino acids in a specific order; the order is determined by the base sequence of nucleotides in the gene coding for the protein. Proteins are required for the structure, function, and regulation of the body's cells, tissues, and organs, and each protein has unique functions.

Residue: amino acids in proteins.

REFERENCES

1. Altschul, S.F. (1989). *Gap costs for multiple sequence alignment*. Journal of Theoretical Biology, 138, 297-309.
2. Carrillo, H. and Lipman, D. (1988). *The multiple sequence alignment problem in biology*. SIAM J. Appl. Math., 48, 1073-1082.
3. Chan, S.C., Wong, A.K.C. and Chiu, D.K.Y. (1992). *A survey of multiple sequence comparison methods*. Bull. Math. Biol., 54, 563-598.
4. Corpet F. (1988). *Multiple sequence alignment with hierarchical clustering*. Nucleic Acids Res., 16, 10881-10890.
5. David G., Winona C.B. and Lotis T.H. (1988). *Mutation data matrix and its uses*. Methods in Enzymology, 183, 333-351.
6. Dijkstra, E.W. (1959). *A note on two problems in connection with graphs*. Numerische Mathematik, 1, 269-271.
7. Durbin, R., Eddg, S., Krogh, A. and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.
8. Eddy, S.R. (1995). *Multiple alignment using hidden Markov models*. Ismb, 3, 114-120.
9. Feng, D.F. and Doolittle, R.F. (1987). *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*. J. Molec. Evol., 25, 351-360.
10. Feng, D.F. and Doolittle, R.F. (1996). *Progressive Alignment of Amino Acid Sequences and Construction of Phylogenetic Trees from Them*. Methods in Enzymology, 266, 368-382.
11. Fuellen, George. (1997). *A Gentle Guide to Multiple Alignment*.
<http://www.techfak.uni-bielefeld.de/bcd/Curric/MulAli/node1>.
12. <http://www.ornl.gov/hgmis/>
13. Gotoh, O. (1982). *An improved algorithm for matching biological sequences*. J. Molec. Biol., 162, 705-708.

14. Gotoh, O. (1996). *Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments*. J Molec. Biol., 264(4), 823-38.
15. Gribskov, M., Luthy, R. and Eisenberg, K. (1990). *profile analysis*. Methods in Enzymology, 183, 146-159.
16. Gupta, S.K.J., Kececioglu, A.A. and Schäffer. (1995). *Improving the Practical Space and Time Efficiency of the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignment*. J. Computational Biology, 2(3), 459-472.
17. Henikoff, S. and Henikoff, J.G. (1992). Proc. Natl. Acad. Sci. U.S.A. 89, 10915.
18. Higgins D.G. and Sharp P.M. (1988). *CLUSTAL: a package for performing multiple sequence alignment on a microcomputer*. Gene, 73, 237-244.
19. Higgins, D.G., Thompson, J.D. and Gibson, T.J. (1996). *Using CLUSTAL for Multiple Sequence Alignments*. Methods in Enzymology, 266, 383-401.
20. Hirschberg, D.S. (1975). *A linear space algorithm for computing maximal common subsequences*. Commun. ACM, 18, 341-343.
21. Hughey, R. and Krogh, A. (1996). *Hidden Markov models for sequence analysis: extension and analysis of the basic method*. Comput. Appl. Biosci., 12(2), 95-107.
22. Johnson, M.S. and Doolittle, R.F. (1986). *A method for the simultaneous alignment of three or more amino acid sequences*. J. Molec. Evol., 23, 267-278.
23. Krogh, A., Brown, M., Mian, I.S., Siolander, K., and Haussler, D. (1994). *Hidden Markov Models in computational biology*. J. Molec. Biol., 235, 1501-1531.
24. Lipman, D.J., Altschul, S.F. and Kececioglu, J.D. (1989). *A tool for multiple sequence alignment*. Proc. Natn. Acad. Sci. U.S.A., 86, 4412-4415.
25. McClure M.A., Vasi, T.K., and Fitch W.M. (1994). *Comparative analysis of multiple protein-sequence alignment methods*. Molec. Biol. Evol., 11, 571-592.
26. Miclet, L. (1986). *Structural Methods in Pattern Recognition*. Oxford, U.K., North Oxford Academic.
27. Miller, W. and Myers, E.W. (1988). *Sequence comparison with concave weighting functions*. Bull. Math. Biol., 50, 97-120.
28. Murata, M., Richardson, J.S. and Sussman, J.L. (1985). *Simultaneous comparison of three protein sequences*. Proc. Natl. Acad. Sci. U. S. A., 82, 3073-3077.

29. Pascarella, L. and Argos, P.J. (1992). *Molec. Biol.*, 224, 461.
30. Rabiner, L.R. (1989). *A tutorial on hidden Markov models and selected applications in speech recognition*. *Proc. IEEE*, 77 (2), 256-286.
31. Sankoff, D. (1975). *SIAM J. Appl. Math.*, 78, 35.
32. Saitou, N., Nei, M. (1987). *The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees*. *Molec. Biol. Evol.*, 4, 406.
33. Smith, T.F. and Waterman, M.S. (1981). *Identification of common molecular subsequences*. *J. Molec. Biol.*, 147, 195-197.
34. Sneath, P.H.A, Sokal, R.R. (1973). *Numerical Taxonomy*, Freeman, W.H., San Francisco.
35. Taylor, W.R. (1986). *Identification of protein sequence homology by consensus template alignment*. *J. Molec. Biol.*, 188, 233-258.
36. Taylor, W.R. (1987). *Multiple sequence alignment by a pairwise algorithm*. *Comput. Appl. Biosci.* 3, 81-87.
37. Taylor, W.R. (1996). *Multiple Protein Sequence Alignment: Algorithms and Gap Insertion*. *Methods in Enzymology*, 266, 343-367.
38. Thompson, J.D., Higgins, D.G. and Gibson, T.J. (1994) *CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice*. *Nucleic Acids Research* 22, 4673-4680.
39. Waterman, M.S. (1995). *Introduction to Computational Biology*. Chapman and Hall, London.
40. Waterman, M.S., Smith, T.F. and Beyer, W.A. (1976). *Advan. Math.*, 20, 367-387.
41. Zuker, M. (1999). *What is multiple sequence alignment*.
<http://www.ibr.wustl.edu/~zucker/Bio-5495/>
42. Sternberg, M.J.E.(1996). *Protein Structure Prediction*. Oxford University Press Inc., New York.
43. Modified from UCSC Computational Biology Group
http://www.cse.ucsc.edu/research/compbio/papers/sam_doc

VITA

Xiao Hong Wang

Candidate for the Degree of

Master of Science

Thesis: PAIRWISE AND MULTIPLE SEQUENCE ALIGNMENT
ALGORITHMS ANALYSIS

Major Field: Computer Science

Biographical:

Personal Data: Born in Stone Island, Shandong, China, October 7, 1972, the daughter of RenZheng Wang and XiuHua Jiang.

Education: Received Bachelor of Science from Shandong Normal University, Jinan, China in July, 1994. Received Master of Science from East-China Normal University, Shanghai, China in December, 1996. Completed the requirements for the Master of Science with a major in Computer Science at Oklahoma State University in December, 1999.