USING TRAVELING SALESMAN PROBLEM ALGORITHMS TO DETERMINE MULTIPLE SEQUENCE ALIGNMENT ORDERS

by

WEIWEI ZHONG

(Under the Direction of Robert W. Robinson)

ABSTRACT

Multiple Sequence Alignment (MSA) is one of the most important tools in modern biology. The MSA problem is NP-hard, therefore, heuristic approaches are needed to align a large set of data within a reasonable time. Among existing heuristic approaches, CLUSTALW has been found to be the progressive alignment program that provides the best quality alignments, while the program POA provides very fast alignments.

In this thesis, a new MSA algorithm is implemented and tested extensively. We use a Traveling Salesman Problem (TSP) algorithm to determine a circular tour in which the sequences are aligned. Sequences are aligned progressively by repeatedly merging the closest nodes along the TSP tour. Quality assessment of our algorithm, TspMsa, with CLUSTALW and POA was conducted using the BAliBASE benchmarks. It is found that TspMsa provides alignments which are similar to those from CLUSTALW in most test cases. Both programs give alignments which are significantly better than those from POA. For alignments of large sets of sequences, TspMsa and POA run in considerably shorter execution times than CLUSTALW.

INDEX WORDS: algorithm, multiple sequence alignment, MSA, traveling salesman problem, TSP, CLUSTALW, POA, BAliBASE.

USING TRAVELING SALESMAN PROBLEM ALGORITHMS TO DETERMINE MULTIPLE SEQUENCE ALIGNMENT ORDERS

by

WEIWEI ZHONG

B.S., University of Science and Technology of China, P.R. China, 1997

Ph.D., The University of Georgia, 2002

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2003

USING TRAVELING SALESMAN PROBLEM ALGORITHMS TO DETERMINE

MULTIPLE SEQUENCE ALIGNMENT ORDERS

by

WEIWEI ZHONG

Approved:

Major Professor: Robert W. Robinson

Committee: Eileen T. Kraemer
Russell L. Malmberg

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2003

DEDICATION

This thesis is dedicated to my loving parents.

ACKNOWLEDGEMENTS

My most sincere thanks must go to my major professor Dr. Robert W. Robinson for his invaluable guidance. I have benefited tremendously from his expertise in research and his enormous support. My appreciation for his mentorship goes beyond my words.

I would like to thank my committee members Drs. Eileen T. Kraemer and Russell L. Malmberg for helpful advice and resourceful suggestions. I am especially grateful that Dr. Malmberg also provided me access to his computer for experimentation.

Thanks to Carolyn Lawrence in the Malmberg lab for friendly discussions and alignment evaluations.

A special appreciation goes to all my friends for love and support.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

CHAPTER

1 INTRODUCTION 1

Significance 1

Statement of Problem 2

Basic Dynamic Programming Algorithm 3

Scoring Matrices 7

Gap Penalties 10

Evaluation of an Alignment 11

2 EXISTING MSA PROGRAMS 14

Multidimensional Dynamic Programming 14

Progressive Alignment Methods 17

Iterative Alignment Methods 25

3 ALGORITHM AND IMPLEMENTATION 27

Overall Structure 27

Input/Output Files 30

Pairwise Distances 32

TSP 32

Progressive Alignment 34

Implementation....36
4 RESULTS....38
Test Data and Quality Evaluation Method....38
Starting Point and Direction of TSP....40
Quality Analysis....43
Execution Time Analysis....47
5 DISCUSSION....51
REFERENCES....53
APPENDEX
TSPMSA USER MANUAL....57

CHAPTER 1

INTRODUCTION

Significance

Proteins and DNA are the two most fundamental molecules for all life forms. Proteins are the building blocks for all cells while DNA stores all genetic information. The primary structure of a protein is a linear chain of amino acids. There are twenty amino acids, denoted by *A*, *R*, *N*, *D*, *C*, *Q*, *E*, *G*, *H*, *I*, *L*, *K*, *M*, *F*, *P*, *S*, *T*, *W*, *Y*, and *V*. DNA molecules are chains of nucleotides. There are four different types of nucleotides, denoted by *A*, *T*, *G*, *C*. Therefore, both proteins and DNA molecules can be represented as strings of letters from relatively small alphabets.

A multiple alignment of protein or DNA sequences refers to the procedure of comparing two or more sequences to look for maximum matching of characters (Mount, 2001). It is one of the most important tools in modern biology. Multiple sequence alignment (MSA) provides key information for evolutionary biology and serves as an essential prelude to much molecular biology analysis. Multiple alignments are most commonly used in the following types of analysis: finding consensus regions among several sequences to determine patterns that characterize protein/gene families; detecting homology between new sequences and known protein/gene family sequences; predicting secondary and tertiary structures of new protein sequences; predicting function of new sequences; molecular evolutionary analysis.

MSA has been an active area of study in computer science and bioinformatics for the last thirty years. The problem of finding an optimal solution has been shown to be NP-complete (Wang and Jiang, 1994). A number of heuristic MSA algorithms have been developed for practical usage. These algorithms can only partially solve the computational challenge due to their heuristic nature. Recent genome sequencing projects have led to a rapid explosion of sequence data. Large scale data sets pose new demands on the speed requirement for computing alignments. Therefore MSA continues to be an area of active research.

Statement of Problem

Given the importance of sequence alignments, it is necessary to clearly define the sequence alignment problem. We can describe the problem as follows. Given two sequences of letters, and a scoring scheme for evaluating two matching letters, two mismatching letters, and gap penalties, the goal of the sequence alignment problem is to produce a pairing of letters from one sequence to the other such that the total score is optimal. We can insert gaps at any positions in the two sequences, but the order of characters in each sequence must be preserved. If two sequences are compared, it is called a pairwise alignment. If there are more than two sequences, then it is a multiple sequence alignment problem.

There are two different types of alignment, *global* and *local*. In global alignment, attempts are made to detect the best alignment of the entire sequences (Mount, 2001). In local alignment, the best alignment is constructed for segments of sequences with the highest density of matches, while the rest of the sequences are ignored (Mount, 2001). Examples of global and local alignments are shown in Figure 1.1. In global alignment,

consideration is stretched over the entire lengths of the sequences to match as many characters as possible. In local alignment, islands of highest matching are given the highest priority, and the alignment stops at the ends of such regions. It is very likely that these regions are not aligned in a global alignment in order to favor matching more characters along the entire sequence lengths. An example of this is the region of *FGKG* aligned in the local alignment in Figure 1.1. In the global alignment of the same sequences, these letters are not all aligned. In this thesis, only global alignments are investigated. Hereafter, alignments refer to global alignments.

Global alignment

```
F G K - G K G
| | |   | | |
F G K F G K G
```

Local alignment

```
- - - F G K G K G
      | | | |
F G K F G K G - -
```

Figure 1.1 Global alignment and local alignment of two hypothetical protein sequences.

Basic Dynamic Programming Algorithm

Pairwise alignments can be solved in $O(L^2)$ time by following a dynamic programming approach, where L is the length of the sequences. The key idea is that the best alignment that ends at a given pair of positions in the two sequences is the best alignment previous to the two positions plus the score for aligning the two positions. The algorithm was first proposed by Needleman and Wunsch (Needleman and Wunsch, 1970).

To illustrate the algorithm we use an example adapted from

`http://www.sbc.su.se/~per/molbioinfo2001/dynprog/adv_dynamic.html`.

The two sample sequences to be aligned are

- *G A A T T C A G T T A* (sequence #1) and
- *G G A T C G A* (sequence #2).

Suppose that the scoring scheme is:

- match score = 2, if two letters are identical;
- mismatch score = –1, if two letters are different;
- $g = -2$ for the gap penalty, if a gap is inserted.

The alignment process can be accomplished in the three steps, as follows.

1) Initialization

Let m and n denote the lengths of sequence #1 and sequence #2, respectively. In our example, $m = 11$ and $n = 7$. We create a sequence versus sequence scoring table with $m + 1$ columns and $n + 1$ rows and fill in the first row and first column of the table with 0s (see Figure 1.2).

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
G	0											
A	0											
T	0											
C	0											
G	0											
A	0											

Figure 1.2 The score table after initialization.

2) Table fill

Let $M_{i,j}$ denote the score to be filled in the table cell at row i, column j. We want $M_{i,j}$ to be the best possible score for alignments of the initial segments ending with the character at position i of sequence #1 and the character at position j of sequence #2. The value of $M_{i,j}$ can be calculated recursively using the following formula, where $S_{i,j}$ denotes the value from the scoring scheme if we pair the letter at position i of sequence #1 with the letter at position j of sequence #2:

$$M_{i,j} = max \{ M_{i-1,j-1} + S_{i,j}, M_{i,j-1} + g, M_{i-1,j} + g \}.$$

Here $M_{i-1,j-1} + S_{i,j}$ corresponds to a match or mismatch in the diagonal, $M_{i,j-1} + g$ corresponds to a gap in sequence #1, and $M_{i-1,j} + g$ corresponds to a gap in sequence #2.

In Figure 1.3, an arrow is placed to point back to the table element that leads to the maximum score. If there are two different ways to obtain the maximum score, pointers are placed back to all of the cells that can produce the maximum score; an example is the cell with the value –1 at column 3 and row 2. Using this method, we fill in all the cells of the scoring table.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	2	0	-1	-1	-1	-1	-1	2	0	-1	-1
G	0	2	1	-1	-2	-2	-2	-2	1	1	-1	-2
A	0	0	4	3	1	-1	-3	0	-1	0	0	1
T	0	-1	2	3	5	3	1	-1	-1	1	2	0
C	0	-1	0	1	3	4	5	3	1	-1	0	1
G	0	2	0	-1	1	2	3	4	5	3	1	-1
A	0	0	4	2	0	0	1	5	3	4	2	3

Figure 1.3 The completed scoring table after the table fill step of dynamic programming.

3) Traceback

In this step, we trace back through the scoring table to determine the alignment(s) that result in the maximum score. After the table fill step, the maximum global alignment score for the two sequences is at the (m, n) position in the table. In our example, the score is 3 (see Figure 1.3). We follow the pointer to the predecessor that gives this score. If the pointer is diagonal, we report the two letters as aligned, otherwise, we report a gap in one of the sequences. We reconstruct a path tracing back to the null positions of both sequences.

There can be multiple paths leading to the maximum alignment score. Since all of them produce the same score, we can simply choose one as our optimal solution. Therefore, if there are multiple pointers originated from the same cell in the table, one of the pointers is arbitrarily chosen. In our example there are two possible paths, resulting in the two alignments shown in Figure 1.4. Both of these alignments achieve the maximal alignment score of 3.

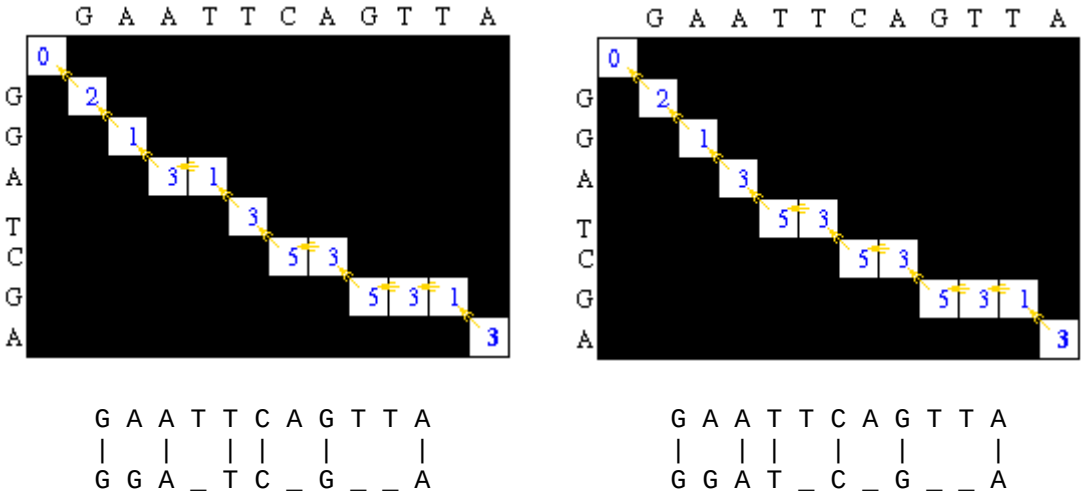


Figure 1.4 Two possible traceback paths and their corresponding alignments.

Scoring Matrices

The scoring used in the previous example is a simplified scheme in which all matches have the same score and all mismatches have the same penalty. In practice, all letters represent chemical compounds such as amino acids. Due to their evolution and chemical properties, some amino acids have higher matching scores than others and some have higher mismatch scores as well. The degree of match between two letters can be represented in a matrix, commonly called a scoring matrix. An example is shown in Figure 1.5. A scoring matrix is a summary of how to map an alphabet to itself. If the scoring matrix is changed, the output alignments will in general be changed.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	–2	6																		
N	0	0	2																	
D	0	–1	2	4																
C	–2	–4	–4	–5	4															
Q	0	1	1	2	–5	4														
E	0	–1	1	3	–5	2	4													
G	1	–3	0	1	–3	–1	0	5												
H	–1	2	2	1	–3	3	1	–2	6											
I	–1	–2	–2	–2	–2	–2	–2	–3	–2	5										
L	–2	–3	–3	–4	–6	–2	–3	–4	–2	2	6									
K	–1	3	1	0	–5	1	0	–2	0	–2	–3	5								
M	–1	0	–2	–3	–5	–1	–2	–3	–2	2	4	0	6							
F	–4	–4	–4	–6	–4	–5	–5	–5	–2	1	2	–5	0	9						
P	1	0	–1	–1	–3	0	–1	–1	0	–2	–3	–1	–2	–5	6					
S	1	0	1	0	0	–1	0	1	–1	–1	–3	0	–2	–3	1	3				
T	1	–1	0	0	–2	–1	0	0	–1	0	–2	0	–1	–2	0	1	3			
W	–6	2	–4	–7	–8	–5	–7	–7	–3	–5	–2	–3	–4	0	–6	–2	–5	17		
Y	–3	–4	–2	–4	0	–4	–4	–5	0	–1	–1	–4	–2	7	–5	–3	–3	0	10	
V	0	–2	–2	–2	–2	–2	–2	–1	–2	4	2	–2	2	–1	–1	–1	0	–6	–2	4

Figure 1.5 PAM 250 scoring matrix (Dayhoff *et al.*, 1978).

Many scoring matrices have been developed and it remains an area of active research. There are four different types of scoring matrices: identity matrices, genetic code matrices, chemical similarity matrices and substitution matrices. Identity matrices

are the simplest form of scoring schemes and are generally considered less effective. In an identity scoring matrix, identical amino acid pairs are given a positive score, whereas non-identical pairs are scored 0. In a genetic code matrix, amino acids are scored based on similarities in the genetic code (three DNA/RNA bases) (Fitch, 1966). Today this scheme is rarely the first choice for scoring alignments of protein sequences. In a chemical similarity matrix, amino acids with similar physical-chemical properties, such as hydrophobicity, polarity, size, shape, and charge, receive high scores (McLachlan, 1972). Substitution matrices are compiled based on statistical observations of the substitution frequencies seen in alignments of sequences. Early substitution matrices were derived by analyzing manually aligned sequences (Dayhoff *et al.*, 1978). More recent substitution matrices have had the benefit of analysis of alignments built on earlier matrices (Henikoff and Henikoff 1992). Extensive experience with substitution matrices suggests that they are superior to simple identity, genetic code, or intuitive physical-chemical property matrices. Therefore, substitution matrices have become the most commonly used scoring schemes.

Two series of substitution matrices are most popular in practice.

1) PAM (***P***ercent ***A***ccepted ***M***utation)

In a PAM matrix, the score given an amino acid pair is a measure of the probability of the change of one amino acid to the other in a family of related proteins (Dayhoff *et al.*, 1978). First, a concept of PAM units was introduced to quantify the amount of evolutionary change in a protein sequence. Two sequences *S* and *T* are defined to be one PAM unit diverged if a series of accepted point mutations (with no

insertion or deletion) can convert *S* to *T* with an average of one amino acid substitution per 100 amino acids (Dayhoff *et al.*, 1978).

In order to derive the PAM1 matrix, closely related sequences within a protein family were aligned. For each pair of amino acids, the frequency of substitutions (aligning) between these two in the alignment was determined. These probabilities were placed into a matrix representing all possible amino acid changes. The matrix was then normalized into values that represented the probability that 1 amino acid in 100 would undergo change (PAM units). For a pair of amino acids (*a*, *b*), let *M* (*a*, *b*) be the observed substitution frequency, and let *P* be the expected frequency; then

$$score\ (a, b) = 10 \log_{10} (M (a, b) / P).$$

The score is rounded up to the next integer, as shown in Figure 1.5. A positive score indicates that substitutions between amino acids *a* and *b* are more likely than random. A zero score indicates that substitutions between *a* and *b* occur at the random base rate. A negative score indicates that substitutions between *a* and *b* are less likely than random.

A series of PAM matrices, such as PAM160 or PAM250, have been constructed based on the PAM1 matrix. A PAM*n* matrix is a look-up table in which scores have been calculated based on sequences that are diverged *n* PAM units apart. The PAM250 matrix has been the most commonly used PAM matrix.

2) BLOSUM (***Blo***cks ***Su***bstitution ***M***atrix)

In BLOSUM matrices, scores for each pair of amino acids are derived from observations of the frequencies of substitutions in blocks of local alignments of related proteins in the BLOCKS database (Henikoff and Henikoff 1992). In BLOCKS there are 3,000 blocks of highly conserved sequences representing hundreds of protein groups.

Similarly to the construction of PAM matrices, a log odds score of substitution frequency for each pair of amino acids is calculated to build a BLOSUM matrix.

There is a series of BLOSUM matrices, each denoted BLOSUM*n* for some *n*, where the number n indicates the similarity of the sequences from which the BLOSUM matrix was derived (Henikoff and Henikoff, 1992). In the BLOSUM80 matrix, for example, the alignment from which scores were derived was created using sequences sharing no more than 80% identity. The BLOSUM62 matrix is believed to be a good all-around matrix while BLOSUM45 is recommended for more divergent sequences and BLOSUM100 is suggested for strongly related sequences (Henikoff and Henikoff, 1992). BLOSUM62 has been found to be most similar to the PAM250 matrix, but is considered more reliable than the PAM250 matrix for finding members of most protein families (Henikoff and Henikoff, 1992).

Gap Penalties

In the example, we used a constant gap penalty for each gap insertion. To provide a better biological modeling scheme, the gap penalty should be a function of the gap length. Based on statistical studies, opening a gap should be more expensive than extending an existing gap (Thompson *et al.*, 1994). In practice, gap penalties are often represented as linear functions. In that case, the score for a gap of length x can be represented as

$$w(x) = g + g'x,$$

where g is the cost of opening a gap and g' is the cost of extending a gap. Accordingly, in the dynamic programming algorithm, the recursive formula for calculating the score at cell (i, j) in the table fill step should be changed to

$$M_{i,j} = max\ \{\ M_{i-1,j-1} + S_{i,j},\ max\{M_{i-x,j-1} + w(x-1)\},\ max\{M_{i-1,j-y} + w(y-1)\}\ \},$$

where $M_{i-1,j-1} + S_{i,j}$ corresponds to a match or mismatch in the diagonal, $M_{i-x,j-1} + w(x-1)$ corresponds to $(x-1)$ gaps in sequence #1, $M_{i-1,j-y} + w(y-1)$ corresponds to $(y-1)$ gaps in sequence #2, and the ranges are $2 \le x \le i$ and $2 \le y \le j$ for the respective max operators.

Evaluation of an Alignment

How do we assess the quality of an alignment? The scoring scheme defines the objective function of the multiple sequence alignment. The goal for a pairwise alignment is simply to achieve the maximum total pairing score. However, when consideration is extended to multiple sequence alignments, several scoring methods have been proposed to evaluate the quality of an alignment. Here we discuss the two most commonly used scoring methods.

1) Sum-of-pairs (SP)

The sum-of-pairs method directly extends the scoring method used in pairwise alignments (Nicholas *et al.*, 2002). It is the most popular scoring method. In this thesis, it is the default scoring method.

The SP score for an aligned column in the MSA is computed by evaluating the matching score between each pair in the column. Let $c_{x,i}$ denote the character located at row x and column i in the MSA, and let $S(c_1, c_2)$ denote the matching score for the characters c_1 and c_2 in the amino acid scoring matrix. Then the SP score for column i, $SP(i)$, is

$$SP(i) = \sum_{x<y} S(c_{x,i}, c_{y,i}).$$

The total SP score for an MSA can be calculated by adding all the column SP scores. Thus

$$SP\ score \ = \sum_{1 \le i \le n} SP(i),$$

where n is the string length of the MSA including gaps. When the SP score is used as the objective function, an optimal MSA solution is one which achieves the maximum SP score.

2) Entropy scoring

The entropy scoring method is preferred in mathematically and statistically oriented studies (Nicholas *et al.*, 2002). The overall entropy for an MSA is the sum of entropies over the columns. The entropy for the i^{th} column in an MSA, denoted by $entropy(i)$, can be computed as:

$$entropy(i) \ = -\sum_{a} \ c_{ia} \log p_{ia}$$

where c_{ia} is the count for character a in column i, and p_{ia} is the probability of character a in column i. That is,

$$p_{ia} \ = \ c_{ia} \ / \ \sum_{a'} \ c_{ia'}.$$

A column receives zero entropy score if all the characters aligned in the column are the same. The more variable the column is, the higher the entropy. The column entropy reaches the highest if there are equal numbers of all possible characters in the column. When entropy scoring is used as the objective function, the goal is to achieve minimum entropy.

In addition to these two scoring methods, several other models such as consistency scoring have been suggested and used in some software packages (Nicholas

et al., 2002). All of the scoring methods try to give a quantitative evaluation of the biological and evolutionary significance of an alignment. However, due to the complex nature of biological data, all scoring methods have their limitations. There is no universal standard to measure the quality of a multiple sequence alignment.

CHAPTER 2

EXISTING MSA PROGRAMS

Multidimensional Dynamic Programming

The Needleman-Wunsh dynamic programming algorithm can be generalized from pairwise alignments to multiple alignments of n sequences by using an n-dimensional scoring table instead of the 2-dimensional scoring table used for normal dynamic programming. As shown in Figure 2.1, in a 2-dimensional scoring table the value in a cell is derived form one of its three neighbors, while a cell in a 3-dimensional scoring table depends for its value on seven neighbors. For n sequences of length L, the time complexity for multidimensional dynamic programming is $O(2^n L^n)$ and the space complexity is $O(L^n)$. This approach is so computationally expensive that it is considered impractical for $n > 4$.

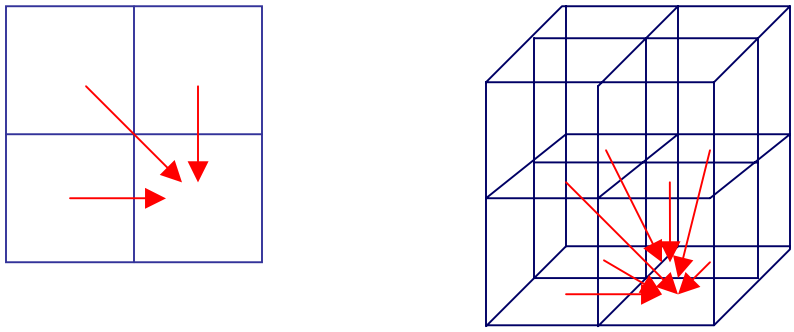


Figure 2.1 2- and 3-dimensional scoring tables.

Multidimensional dynamic programming has been optimized to make it feasible for aligning a moderate number of sequences of reasonable lengths (Lipman *et al.*, 1989). A program called MSA was implemented based on the Lipman optimization. Later,

several improvements were reported which reduced the memory and time requirements of MSA (Gupta *et al.*, 1995). The improved version is freely available for research use and can be downloaded via `ftp://fastlink.nih.gov/pub/msa`.

The key idea of the Lipman algorithm is that the sum-of-pairs score for any pair of sequences in an optimal multiple sequence alignment should be lower than the SP score for the optimal pairwise alignment of the pair of the sequences, and that the total SP score should be higher than the SP score in a heuristic solution. By setting the lower bound and upper bound, only a restricted space needs to be explored in the *n*-dimensional scoring table, as shown in Figure 2.2. This can reduce the computation time greatly. The main steps of an optimized MSA algorithm are described below.

1) Compute the optimal pairwise alignment for each pair of sequences.
2) Apply a fast heuristic multiple sequence alignment program to align all the sequences. Extract the pairwise alignments from the heuristic solution.
3) For each pair of sequences, define the 2-dimensional restricted space using the optimal and extracted pairwise alignments. This space covers the area in the scoring table where alignments have SP scores lower than the optimal but higher than the extracted alignment (see Figure 2.2 A). Normally the 2-dimensional restricted area is extended to somewhat beyond this limit.
4) Lift the 2-dimensional restricted spaces up to the *n*-dimensional scoring table to determine the multidimensional restricted space, as shown in Figure 2.2 B.
5) Use dynamic programming to fill in the cells within the restricted space in the scoring table.
6) Traceback to reconstruct the alignment.

A)

B)

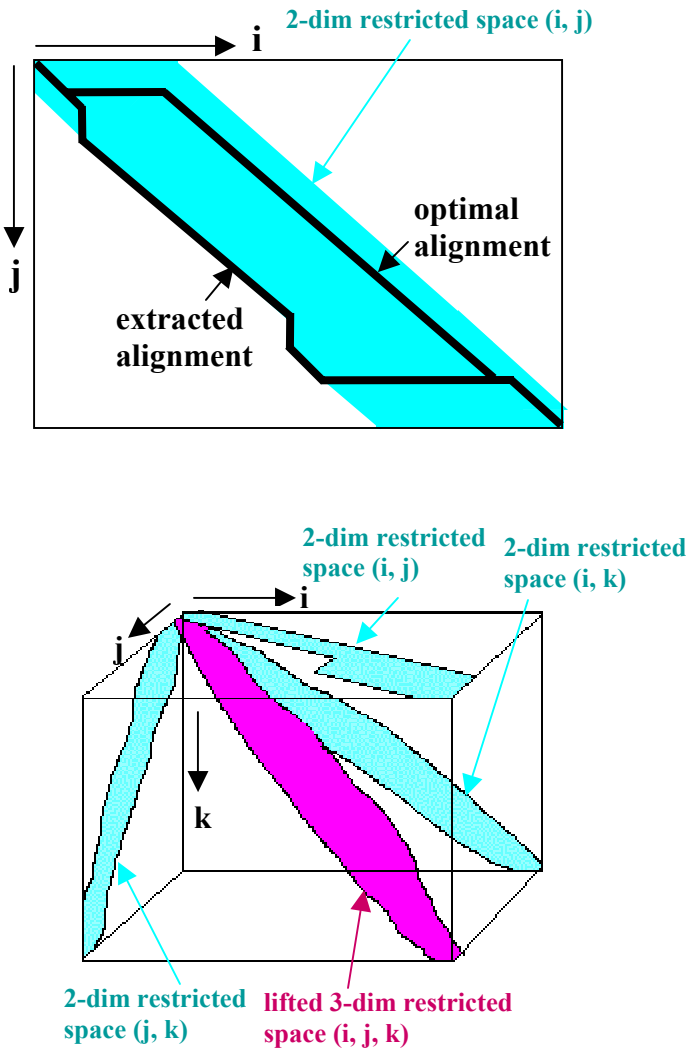


Figure 2.2 An example of the usage of restricted space in the program MSA (modified from Nicholas *et al.*, 2002).

MSA guarantees an optimal solution. However, the computational requirements for CPU time and memory are very high and can easily reach a prohibitive level depending on the number, the lengths and the diversity of the sequences. Using a mini-supercomputer with 4 GB of physical memory, MSA can align 20 phospholipase A2 sequences each of which has approximately 130 characters (Nicholas *et al.*, 2002). At the Pittsburgh Supercomputing Center where an online MSA service is provided

(`http://www.psc.edu/general/software/packages/msa/`), the input data size is limited to fewer than 50 sequences with each sequence having fewer than 150 amino acids, fewer than 25 sequences with each sequence having fewer than 500 amino acids, or fewer than 10 sequences with each sequence having fewer than 1000 amino acids.

Progressive Alignment Methods

Since finding an optimal multiple sequence alignment is extremely expensive computationally, many heuristic approaches have been developed to find near optimal solutions within reasonable lengths of time. Depending on the fundamental strategies used, these heuristic alignment algorithms can be classified into two categories: *progressive* and *iterative* methods (see Figure 2.3). The progressive approach has been the more popular.

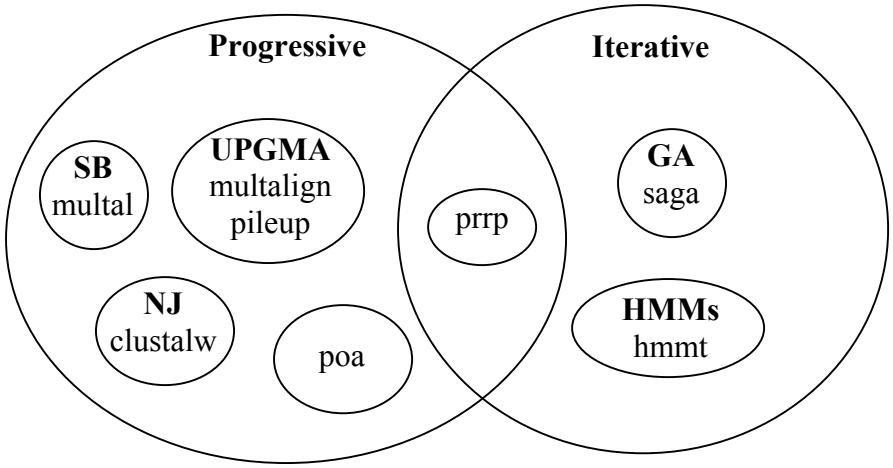


Figure 2.3 Classification of different alignment programs and algorithms (modified from Thompson *et al.*, 1999).

The basic idea of progressive alignment methods is to repeatedly apply the pairwise alignment algorithm instead of aligning all sequences simultaneously. Given n sequences of length L, the running time for progressive alignments is $O(nL^2)$. The major steps of a progressive algorithm are described below.

1) Choose two sequences and align them.

2) Choose another sequence and align it to the current group.

3) Repeat step 2 until all the sequences are aligned.

In some progressive algorithms, sequences are first aligned into subgroups and then subgroups are merged to one group. Although all progressive algorithms adopt the same basic strategy, they apply different modifications to the pairwise dynamic programming algorithm to align two subgroups of sequences. Aligning a sequence to a group of sequences is a special case for aligning two groups in which one group contains only one sequence. Two algorithms for the modified dynamic programming alignment process are introduced below. In the Feng-Doolittle progressive alignment algorithm, groups of sequences are reduced into a sequence of one-dimensional profiles before the pairwise dynamic programming algorithm is applied. In alignments using partial-order graphs, groups of sequences are represented as a partial-order graph and then a dynamic programming algorithm is applied.

1) Feng-Doolittle progressive alignment

Traditionally, the Feng-Doolittle progressive alignment algorithm has been the most popular method. It is based on their landmark work (Feng and Doolittle, 1987). The program ALIGN (Feng and Doolittle, 1996) implements the algorithm and is freely

available at

`http://www-biology.ucsd.edu/~msaier/transport/software.html`.

In the Feng-Doolittle progressive alignment, in order to compute the matching score for two positions from the two groups, matching scores for every pair of characters from the two positions are added up and averaged. For example, in the process of aligning the two groups of sequences in Figure 2.4, the matching score of the position in group 1 with the amino acids *T* and *A* versus the position in group 2 with *V*, *M* and *V* is

$$(S(T, V) + S(T, M) + S(T, V) + S(A, V) + S(A, M) + S(A, V)) / 6,$$

where $S(c_1, c_2)$ denotes the matching score for characters c_1 and c_2 in the scoring matrix. With this scoring scheme, we can compute the dynamic programming scoring table for two groups of sequences using the pairwise alignment algorithm.

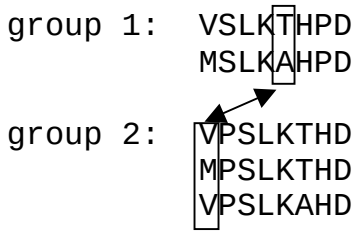


Figure 2.4 Aligning two groups of sequences.

To apply full dynamic programming, one modification at the traceback step must be made in order to align two groups of sequences. At this step, a group of sequences is treated as a whole; if a gap should be added at a certain position, a gap is added at that position for all the sequences in the group. In our example, let "–" denote a gap. Assuming that a gap is needed after the first position in group 1, and a gap should be added before the last position in group 2, the resulting alignment is shown in Figure 2.5.

```
group 1:  V-SLKTHPD
          M-SLKAHPD
group 2:  VPSLKTH-D
          MPSLKTH-D
          VPSLKAH-D
```

Figure 2.5 Inserting gaps in alignment of two groups of sequences.

Notice that any group of sequences to be aligned in a Feng-Doolittle progressive alignment is in fact an alignment of sequences. Gaps resulted from the initial alignments are fixed. In the later alignments, gaps in these sequences are treated as characters. Therefore, the rule "once a gap, always a gap" applies. All earlier gaps are frozen in the alignment. The order in which the sequences are aligned is therefore essential for the quality of the final alignment. The example in Figure 2.6 illustrates this. If we first align *x* with *y*, and *z* with *w*, the resulting alignment is shown in the figure. When we align group *xy* with group *zw*, *y* clearly should be changed to

```
y:   GA-CTT
```

However, early mistakes cannot be corrected in the Feng-Doolittle progressive alignment. Therefore, the algorithm is by nature greedy for a locally optimal solution and thus does not guarantee a global optimal solution (Thompson *et al.*, 1994).

```
x:   GAAGTT
y:   GAC-TT

z:   GAACTG
w:   GTACTG
```

Figure 2.6 An example of a progressive alignment.

2) Programs based on Feng-Doolittle progressive alignment

The Feng-Doolittle progressive alignment is the most common heuristic technique. A variety of programs, such as MULTAL, MULTALIGN, PILEUP and CLUSTALW are

developed based on the Feng-Doolittle alignment process (see Figure 2.3). They differ only in how to determine the order in which sequences or sequence groups are aligned. All of them share the same major steps:

1) Calculate the $n(n\text{-}1)/2$ distances between all pairs of n sequences by standard pairwise alignment.

2) Construct a guide tree using a clustering algorithm

3) Align sequences. Align the closest sequences first, then, add the more distant ones in the order specified by the guide tree.

Several algorithms have been used in practice to construct guide trees. MULTAL uses a Sequential Branching (SB) method to determine the alignment order (Taylor, 1988). The closest pair is aligned first. The sequence that has the highest pairwise SP score to either of the aligned sequences will be added next. The core grows until all the sequences are incorporated. If all the unaligned sequences have low SP scores with those in the core, then a new chain is started by aligning the closest pair among the unaligned sequences. A second pass will be needed to fuse all the chains together.

MULTALIGN (Barton and Sternberg, 1987) and PILEUP (Wisconsin package v.8 from Genetics Computer Group; `http://www.gcg.com`) use the UPGMA method (Sneath and Sokal, 1973) to construct the guide tree. UPGMA stands for unweighted pair-group method using arithmetic averages. Using this method, we first identify from among all the sequences the two sequences that are most similar to each other and align them. These two sequences are then treated as a new sequence. I will call it a composite sequence. The pairwise distances between this new composite sequence and other sequences are recalculated. The distance between a simple sequence and a composite

sequence is the average of the distances between the simple sequence and each simple sequence in the composite sequence. For example, let *AB* denote a composite sequence composed of sequences *A* and *B*, let *C* be another sequence, and let *dist*(*A*, *C*) denote the pairwise distance between sequences *A* and *C*. Then the distance between *AB* and *C* is

$$dist(AB, C) = (dist(A, C) + dist(B, C)) / 2 .$$

In this way, all pairwise distances for the new group of sequences can be calculated. We subsequently select the pair with the highest similarity to align. Then new pairwise distances are recalculated, and the whole cycle is repeated until we have only one composite sequence as our final alignment.

CLUSTALW (Thompson *et al.*, 1994) and its other version called CLUSTALX, which provides a graphical user interface, use a complicated scheme called the Neighbor-Joining (NJ) method (Saitou and Nei, 1987) to determine the guide tree. First, unrooted trees where each leaf represent a sequence are derived based on pairwise distances. These unrooted NJ trees have branch lengths proportional to the sequence divergence. Then, a midpoint root position is calculated and a rooted NJ tree is constructed where each branch is associated with a weight. These weights are used later to derive a weight for each sequence. The rooted NJ tree will serve as the guide tree for the progressive alignment.

CLUSTALW (Thompson *et al.*, 1994) has made important improvements to the progressive alignment process. Two gap penalties, a gap opening penalty and a gap extension penalty, are used. CLUSTALW dynamically calculates the gap penalties to customize the values for each set of sequences. It also automatically changes the scoring matrix (within the same matrix series, such as the BLOSUM series) during the course of

building the alignment based on the divergence of the sequences. CLUSTALW assigns a weight to each sequence based on the guide tree. In the dynamic programming table fill step, weighted scores are used in the objective function. Using our previous example of two groups of sequences, with each sequence assigned a weight (w_1 through w_5, as indicated in Figure 2.7), then the weighted matching score of the two marked positions is $(w_1w_3S(T, V)+w_1w_4S(T, M)+w_1w_5S(T, V)+w_2w_3S(A, V)+w_2w_4S(A, M)+w_2w_5S(A, V)) / 6$.

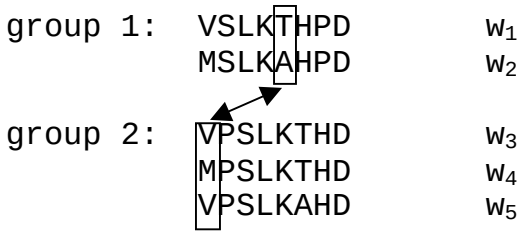


Figure 2.7 Aligning two groups of weighted sequences.

These efforts have made CLUSTALW one of the most successful alignment packages. It has been shown that CLUSTALW provides the best quality alignments compared to other progressive alignment programs (Thompson *et al.*, 1999). CLUSTALW has been one of the most commonly used alignment programs since 1994 (Thompson *et al.*, 1999). The European Bioinformatics Institute provides online CLUSTALW service at `http://www.ebi.ac.uk/clustalw/`. CLUSTALW packages for different operation systems are freely available at `ftp://ftp.ebi.ac.uk/pub/software/`.

3) Partial order alignment (POA)

In the Feng-Doolittle alignment, a group of aligned sequences is averaged into a 1-dimensional profile when developing the dynamic programming table. For each column in the alignment, the 1-dimensional profile keeps character and gap frequencies.

This data structure does not keep information as to which sequence a character comes from. The POA method (Lee *et al.*, 2002) tries to correct this by using a partial order multiple sequence alignment (PO-MSA) data structure to represent a group of aligned sequences as a partial order graph. In the PO-MSA format, each character is represented as a node and directed edges are drawn between consecutive characters in each sequence (see Figure 2.8). Characters that are aligned and identical are fused into one node, while characters that are aligned but not identical are represented as separated nodes with a record marking that they are aligned to each other (represented by dashed circles in the figure). This data structure makes it possible to trace the path of each individual sequence.

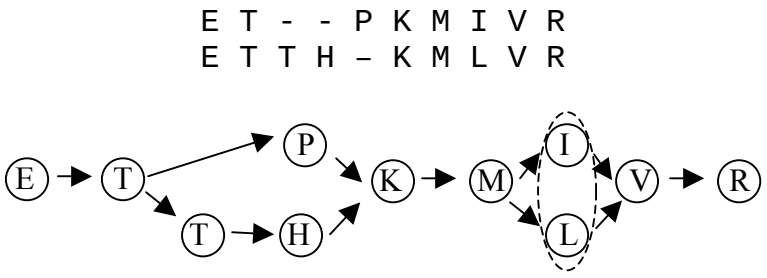


Figure 2.8 PO-MSA representation of a group of aligned sequences (modified from Lee *et al.*, 2002).

A modified version of the Needleman-Wunsch pairwise alignment has been developed to extend dynamic programming to the PO-MSA data format (Lee *et al.*, 2002). In the dynamic programming scoring table, multiple surfaces arise at the positions corresponding to the branches in the PO-MSA, as shown in Figure 2.9. On a given surface, the traditional dynamic programming procedure can be applied, and the value in

a cell is derived from one of its three neighbors. At junctions where multiple surfaces fuse, more move directions are needed to allow entry from any of the surfaces. As illustrated in the figure, at the surface junction where two surfaces fuse a cell's value depends on the values in five possible neighbors on the two surfaces.

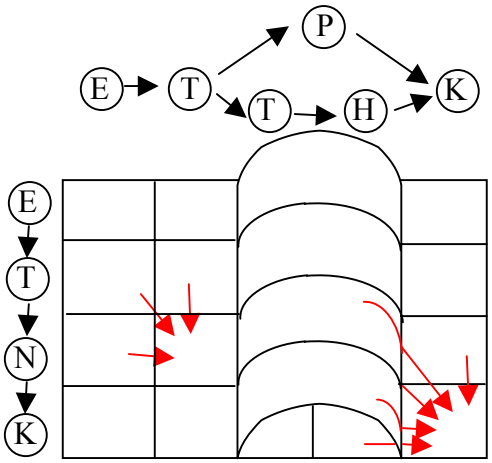


Figure 2.9 Dynamic programming table for the PO-MSA data format (modified from Lee *et al.*, 2002).

The program POA implements this algorithm (Lee *et al.*, 2002) and is available at `http://www.bioinformatics.ucla.edu/poa`. POA is very efficient, and is capable of handling large data sets. It can align 5000 sequences in 4 hours on a Pentium II (Lee *et al.*, 2002).

Iterative Alignment Methods

Several iterative methods (indicated in Figure 2.3) applying different techniques have been developed recently, providing fresh approaches to solve the alignment problem. Iterative methods construct a low quality alignment first, and then refine the alignment in a series of iterative steps.

The program SAGA (Notredame and Higgins, 1996) uses a Genetic Algorithm (GA) to create an evolving population of alignments and then select for possible solutions. The alignments optimize the objective function COFFEE, which measures the consistency between a multiple alignment and a library of pairwise alignments (Notredame and Higgins, 1996).

With a solid statistical base in probability theory, Hidden Markov Models (HMMs) have also been introduced into the multiple sequence alignment field (Eddy, 1995). HMMs are used to represent the consensus of a protein family. The program HMMT maximizes the possibility that an HMM represents the sequences to be aligned (Eddy, 1995).

The PRRP program (Gotoh, 1996) optimizes a progressive alignment by iteratively dividing the sequences into two groups. Subsequently, the groups are realigned using a global group-to-group alignment algorithm until an optimal alignment is obtained.

Compared to progressive methods, iterative methods trade off efficiency for quality. Iterative methods in general provide more alignment accuracy, although the iteration process may be unstable sometimes (Thompson *et al.*, 1999). However, iterative methods have a big disadvantage in computational time (Thompson *et al.*, 1999; Nicholas *et al.*, 2002). Iterative approaches can sometimes even exceed the multidimensional dynamic programming in running time requirements (Nicholas *et al.*, 2002).

CHAPTER 3

ALGORITHM AND IMPLEMENTATION

Overall Structure

Based on an approach proposed by Korostensky and Gonnet (Korostensky and Gonnet, 2000), we have developed and implemented a heuristic progressive alignment algorithm TspMsa. The basic idea is to use a traveling salesman tour to determine the order in which the sequences are aligned. Preliminary work (Wang, 2002) indicated that this algorithm produces alignments with better SP scores than those using CLUSTALW or PILEUP and that the SP score is related to the starting point and the direction of the TSP circle. However, there are several deficiencies in the preliminary work. First, the evaluation method is not appropriate. The biological importance of a higher SP score is unclear. Further, as CLUSTALW dynamically changes the scoring matrix and the gap penalty for each sequence, using a fixed scoring matrix to evaluate the final alignment will unfairly penalize CLUSTALW. Second, the pairwise distance calculation is not accurate. Third, the TSP solver used is slow. This limits the practical usage of the program. This thesis presents a more thorough study in which we have corrected the above deficiencies and also provided several improvements to the algorithm.

The basic algorithm consists of three main stages.

1. For each pair of the sequences, a pairwise alignment is calculated in order to provide an SP score. All pairwise SP scores are converted to distances and a distance matrix is generated.

2. A Traveling Salesman Problem (TSP) solver is applied to the pairwise distance matrix to determine a circular tour that has minimum total distance.
3. The Feng-Doolittle progressive alignment algorithm is applied to the sequences in an order which follows the TSP tour.

A flowchart of the procedure and the output at each step are illustrated in Figure 3.1 using a set of five cold shock protein sequences. Each step will be described in more detail below.

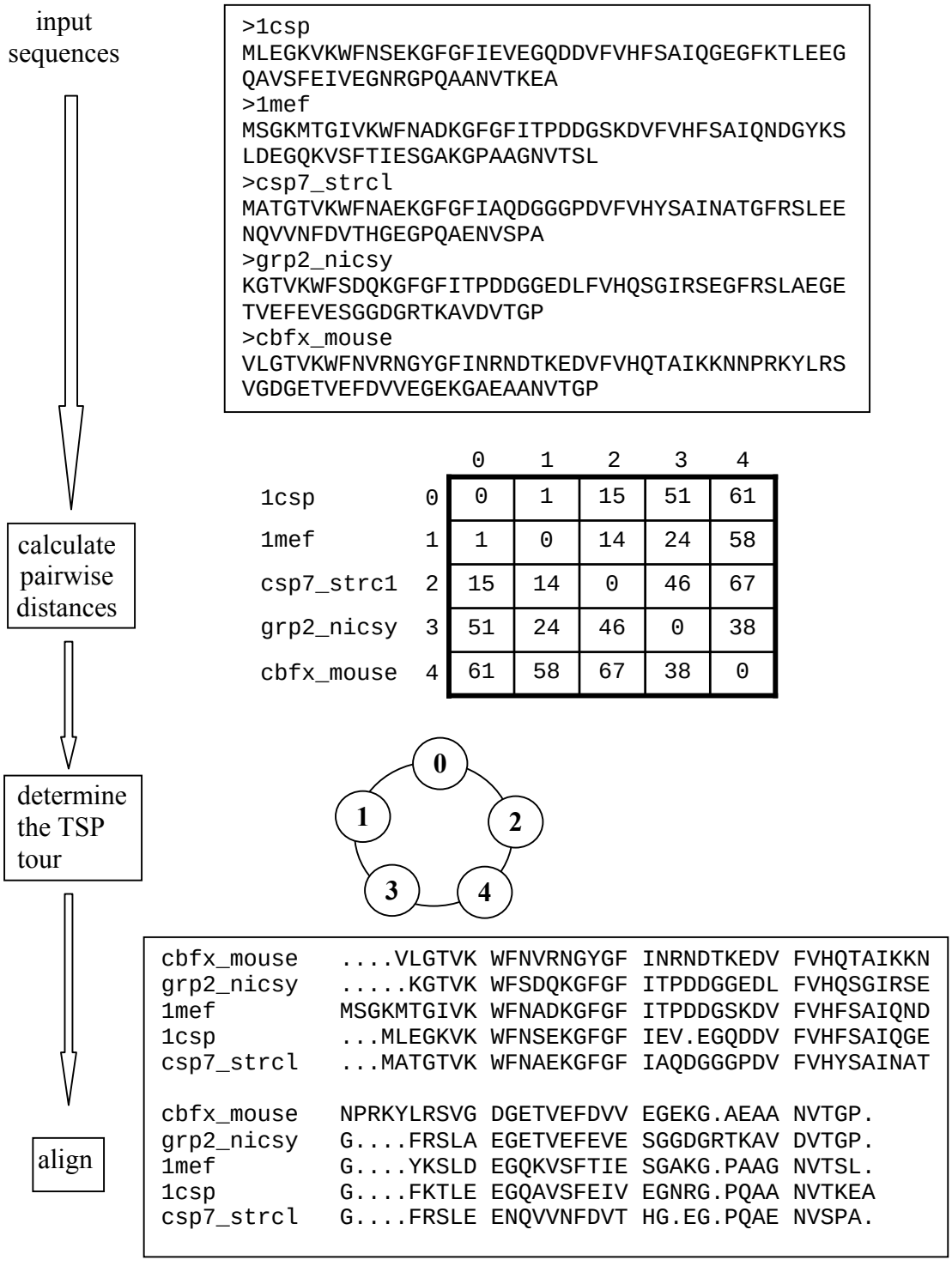


Figure 3.1 The basic alignment procedure of TspMsa.

Input/Output Files

The input sequence file format is the FASTA format used by the National Center for Biotechnology Information (NCBI). This format is accepted for many multiple sequence alignment programs. A description for the FASTA format can be found on `http://www.ncbi.nlm.nih.gov/BLAST/fasta.html`.

Sequences in FASTA formatted files are listed one sequence right after another. Each sequence is preceded by a single-line description (see Figure 3.2). The description line is distinguished from the sequence data by starting with a ">" symbol. The first word on this line is the name of the sequence. The rest of the line is an optional description of the sequence. Following the description line are lines of sequence data. Blank lines in a FASTA file are ignored, and so are spaces or other illegal or undefined symbols in a sequence.

```
> seq1 This is the description of my first sequence.
AGTACGTAGTAGCTGCTGCTACGTGCGCTAGCTAGTACGTCA
CGACGTAGATGCTAGCTGACTCGATGC
> seq2 This is the description of my second sequence.
AGTACGTAGTAGCTGCTGCTACGTGCGCTAGCTAGTACGTCA
CGACGTAGATGCTAGCTGACTCGATGC
> seq3 This is the description of my third sequence.
AGTACGTAGTAGCTGCTGCTACGTGCGCTAGCTAGTACGTCA
CGACGTAGATGCTAGCTGACTCGATGC
```

Figure 3.2 A hypothetical FASTA file illustrating the file format.

The output multiple sequence alignment file format is the MSF format defined by the Genetics Computer Group (GCG). The MSF format was originally used by programs of the GCG suite such as PILEUP. Presently, it is widely supported by other software such as CLUSTALW. MSF ignores blank lines. Some of the hallmarks of an MSF

formatted file are listed below. The format is illustrated in Figure 3.3. In my program, all optional descriptions are skipped in the output.

```
PileUp

   MSF:   61  Type: P    Check:  6540   ..

 Name: 1idy oo  Len:   61  Check:  9109  Weight:  10.0
 Name: 1hstA oo  Len:   61  Check:  6853  Weight:  10.0
 Name: 1tc3C oo  Len:   61  Check:  9591  Weight:  10.0

//

1idy       MEVKKTSWTE EEDRILYQAH KRLG.NRWAE IAKLLP.... ..GRTDNAIK
1hstA      ...SHPTYSE MIAAAIRAEK SRGG.SSRQS IQKYIKSHYK VGHNADLQIK
1tc3C      ....RGSALS DTERAQLDVM KLLN.VSLHE MSRKIS.... ...RSRHCIR

1idy       NHWNSTMRRK V
1hstA      LSIRRLLAAG V
1tc3C      VYLKDPVSYG T
```

Figure 3.3 An example of an MSF file illustrating the file format.

- First is an optional line describing the program that produced the alignment.
- Next is an optional description line having the following keywords: “MSF”, followed by a number specifying the alignment length; “Type”, followed by either the letter P for protein sequences or N for nucleotide sequences; “Check”, followed by a number which is the checksum made up of the ASCII values of the sequence characters. This line ends with two periods. After the periods the data starts, and anything preceding the data may be omitted.
- Next, and preceding the alignment, there is an alignment description. A line in this part starts with the sequence name following the keyword “Name”. The names have to be unique. No blank is accepted within a name. Following the name are the optional fields “Len”, “Check”, and “Weight”.

- A line with "//" is added as the terminator for the header list. After this, the alignment begins.
- In the alignment part, sequences are aligned with each other. Each line starts with a sequence name listed in the header. Each period denotes a gap. Blank spaces are ignored.

Pairwise Distances

Pairwise alignments are performed using the Needleman-Wunsch dynamic programming alignment. Three scoring matrices, PAM250, BLOSUM62, and GONNET, are offered for user selection. Once a scoring matrix is selected, it is used throughout the program for all the sequences. Similarly, a fixed gap penalty associated with the scoring matrix is used throughout the alignment. Once the table fill step of the dynamic programming is completed, the final SP score is extracted.

Since SP scores are matching scores, we have to convert them to distances in order to use the TSP solver. After SP scores have been calculated for each pair of sequences, the maximum score among all pairs is recorded as SP_{max}. All SP scores are then converted to distances by using the following function:

$$distance_{i,j} = SP_{max} - SP_{i,j} + 1,$$

where $distance_{i,j}$ denotes the distance for sequence i and sequence j, and $SP_{i,j}$ denotes the SP score for sequences i and j.

TSP

The Traveling Salesman Problem is the following: given a set of n nodes and distances for each pair of nodes, find a round trip of minimal total length visiting each

node exactly once. In the symmetric traveling salesman problem, the distance from node *i* to node *j* is the same as from node *j* to node *i*.

Although TSP is NP-hard, it has been studied extensively. Using a branch and bound algorithm (reviewed by Kreher and Stinson, 1999), optimal solutions can be solved within a few hours for up to 100 nodes and in a few seconds for up to 30 nodes. Many heuristic approaches have been developed. Among them is the Lin-Kernighan algorithm (reviewed by Rego and Glover, 2002), which can give a near optimal solution within 200 seconds for up to 10,000 nodes (Johnson and McGeoch, 2002).

In my program, each sequence is considered a node. If there are fewer than 10 nodes, TSP3, an exact TSP solver using the branch and bound algorithm, is applied. TSP3 is freely available from `http://www.math.mtu.edu/~kreher/cages.html`. If there are more than 10 nodes, the Lin-Kernighan heuristic is applied. The implementation used is LINKERN, which is part of the CONCORDE suite. The code can be downloaded from `http://www.math.princeton.edu/tsp/concorde.html`.

The input format for TSP3 is the number of nodes followed by a distance matrix, as shown in Figure 3.4A. The input for LINKERN must be in TSPLIB format, which is illustrated in Figure 3.4B. TSPLIB is a library of sample instances for the TSP and related problems of various types. Examples can be found from `http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html`.

The TSPLIB formatted matrix file begins with several lines of descriptions followed by the distance matrix and ends with EOF. In the description lines, the fields NAME, COMMENT and TYPE, which specify the name of the data set, any comments and the

type of the TSP problem (TSP or TSP related problems), respectively, are optional lines for LINKERN. In the DIMENSION description line the number of nodes is specified. The distance data are listed after the keyword EDGE_WEIGHT_SECTION. After all the pairwise sequence distances are calculated, the distance matrix is written into a file in proper format according to the number of sequences. This file then serves as the input for the TSP solver.

A)

```
5
 0 1 15 51 61
 1 0 14 24 58
 15 14 0 46 67
 51 24 46 0 38
 61 58 67 38 0
```

B)

```
NAME: csp_ref1
COMMENT: code shock proteins
TYPE: TSP
DIMENSION: 5
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
 0 1 15 51 61
 1 0 14 24 58
 15 14 0 46 67
 51 24 46 0 38
 61 58 67 38 0
EOF
```

Figure 3.4 A distance matrix TSP3 format and in TSPLIB format.

Progressive Alignment

After the alignment order is extracted from the TSP tour, sequences are aligned using the basic Feng-Doolittle progressive alignment (Feng and Doolittle, 1996). Unlike CLUSTALW, my program uses only a simple scoring scheme. Sequences are not weighted. The gap penalty and the scoring matrix are constant. The same gap penalty and scoring matrix values are used for the progressive alignment and for the prior pairwise alignments.

Aligned sequences are represented as an array of strings. However, this is not computationally efficient for large data sets. Consider the process of aligning two groups of sequences, Seq_1 and Seq_2, with n_1 sequences in Seq_1 and n_2 sequences in Seq_2. The matching score between column i in Seq_1 and column j in Seq_2 is given by

$$\left(\sum_{m,\, n} S(C_{i,\, m},\, C'_{j,\, n}) \right) / (n_1 n_2)$$

where $C_{i,\, m}$ denotes the i^{th} character of sequence m in the sequence group Seq_1, $C'_{j,\, n}$ denotes the j^{th} character of sequence n in Seq_2, and $S(c_1, c_2)$ is the matching score for characters c_1 and c_2. Since we need to add up the matching scores for every pair of letters from the two columns, this calculation will take $O(n_1 n_2)$ time as there are $n_1 n_2$ pairs of characters. This is not efficient when n_1 or n_2 is large.

For large data sets, a vector associated with each column was used to ensure that progressive alignment would run in linear time. The improvement takes advantage of the fact that the alphabet for all the sequences has no more than 20 characters. When a group of sequences has more than 20 individual sequences, a profile is constructed for each column of the group. The profile records the frequency for each of character in the column. For example, for column i of Seq_1, we have profile $P(i) = [f_A, f_B, \ldots, f_Y]$, where f_A represents the frequency of the letter A in column i, and so on for the other characters. Similarly, the profile for column j in Seq_2 has the form $P'(j) = [f_A', f_B', \ldots, f_Y']$. Then the matching score between the two columns is

$$\left(\sum_{a} \sum_{b} f_a f_b' S(a, b) \right) / (n_1 n_2),$$

where a and b range independently over the alphabet. Since there are at most 20 characters in the alphabet, there are at most $20 \times 20 = 400$ possible (a, b) pairs.

Therefore, the calculation runs in constant time. After the two groups are aligned, if column *i* is aligned with column *j*, then the profile for the new column is $[f_A+f_A', f_B+f_B', \ldots, f_Y+f_Y']$. This profile approach makes the running time for the progressive alignment step $O(mL)$ where m is the number of sequences and L is the maximum sequence length.

Implementation

The package TspMsa is written in Java. In this section, the major classes implemented are briefly introduced.

The class `TspMsa` serves as the control module. `TspMsa` takes the input from the user to determine the input file name. The user can also specify parameters such as output file name, scoring matrix, and running mode (quick/slow). `TspMsa` uses other modules to perform individual tasks.

The class `Sequences` serves as an input/output module. It extracts sequences from a text file and stores them as an array of strings. The class `Sequences` is also maintained as a data structure. The actual sequence data, including profiles for the group of sequences, are associated with the class.

The `DistCalculator` class performs dynamic programming alignment on two groups of sequences to compute the final SP score. A single sequence is considered a group that has only one sequence. `DistCalculator` has a field that specifies which scoring matrix will be used.

The `TspCircle` class implements a circular linked list. It is the data structure used to represent the sequences as a TSP tour. Each node contains sequence data. Nodes

are assembled in the order of the TSP tour. In the progressive alignment step, the sequences in one node can only be aligned with the sequences in one of its two neighbors.

The `Align` class aligns two groups of sequences using the Feng-Doolittle progressive alignment.

A summary of the major classes and their functions is presented in Table 3.1.

Class	Method	Function
`TspMsa`	main	Control the entire alignment process.
`Sequences`	constructor	Read a FASTA file, convert sequence data to an array of strings.
	buildProf	Construct the profiles for each column when there are more than 20 sequences in the group.
	write	Write the aligned sequences into a MSF formatted file.
`DistCalculator`	calculate	Calculate the maximum SP score for two groups of sequences using full dynamic programming alignment.
	pairDist	Calculate the SP score for a pair of sequences using a fast approximate method (see Chapter 4).
`TspCircle`	insert	Insert a node into the TspCircle.
	merge	Align the sequence in a node with the sequence in one of its neighbors. Merge the two nodes into one that carries the newly aligned sequences.
`Align`	align	Construct the alignment for two groups of sequences.

Table 3.1 Major classes and their principle methods in the TspMsa package.

CHAPTER 4

RESULTS

Test Data and Quality Evaluation Method

For evaluation and comparison of alignment algorithms, a standard set of reference alignments is needed. Currently, BAliBASE (Thompson *et al.*, 1999) is the only available benchmark for sequence alignments. BAliBASE stands for benchmark alignment database. It is a database of manually-refined multiple sequence alignments specifically designed for comprehensive studies of multiple sequence alignment programs. The alignments have been verified and corrected by superimposition of all known three-dimensional protein structures. BAliBASE is available at `http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE2/`.

BAliBASE 2.0 consists of 220 alignments, containing more than 1000 sequences. All sequences in BAliBASE are real protein sequences. The alignments are categorized into eight reference sets by sequence length, similarity, and presence of insertions and N/C- terminal extensions. The reference sets represent some of the most common multiple sequence alignment problems encountered in practice. Reference 1 contains equidistant sequences with various levels of similarity among sequences. The percent identity between two sequences is within a specified range. All the sequences are of similar length, with no large insertions or extensions. Reference 2 aligns families of at least 15 closely related sequences with up to three highly divergent "orphan" sequences (less than 25% identical). Reference 3 consists of up to 4 sub-groups, with less than 25%

residue identity between sequences from different groups. Reference 4 and reference 5 contain alignments of up to 20 sequences including N/C-terminal extensions (up to 400 residues), and insertions (up to 100 residues). Reference 6 contains alignments of sequences with repeated fragments. It is recommended that an alignment be performed with the repeat domains only. Reference 7 consists of transmembrane protein sequences. Reference 8 consists of proteins with circular permutations. For each family in reference 8, an independent alignment of each permuted domain is recommended. Proper alignments of the reference sets 6 and 8 require a preprocessing step for all the sequences, so the test data in these two reference sets are not used in this study.

In order to accurately measure the performance of an alignment program, a quantitative evaluation method is needed to measure the consistency between the test alignment and the reference alignment. A standard evaluation has been developed (Thompson et. al, 1999). Let n be the number of sequences in the alignment, and suppose the alignment consists of m columns. Let $A_{i1}, A_{i2}, A_{i3},\ldots,A_{in}$ denote the characters in the i^{th} column in the alignment. For each pair of characters A_{ij} and A_{ik}, we define P_{ijk} such that $P_{ijk} = 1$ if letters A_{ij} and A_{ik} are aligned with each other in the reference alignment, and $P_{ijk} = 0$ if not. Then the column score for the i^{th} column, SC_i, is

$$SC_i = \sum_{j=1}^{n} \sum_{k=1,\, k\neq j}^{n} P_{ijk} \ .$$

The total score for the alignment, S, is computed as

$$S = \left(\sum_{i=1}^{m} SC_i \right) \Big/ \left(\sum_{i=1}^{m'} SC_i' \right)$$

where m' is the column number in the reference alignment and S_i' is the column score for the i^{th} column in the reference alignment. The score lies between 0 and 1, with 1 being the best and 0 the worst.

To evaluate the program TspMsa, CLUSTALW and POA were used in this study for performance comparisons. Comparable parameters were set for all programs in the alignments using the BAliBASE data as test cases. The scoring matrix BLOSUM 62 was used for TspMsa and POA, while the BLOSUM series was set for CLUSTALW. CLUSTAL was set to run in the slow and accurate mode unless indicated otherwise below.

Starting Point and Direction of TSP

The TSP tour provides a circular order for the alignment. In a circle, there is no canonical starting node or direction. However, the progressive alignment is heuristic. It is sensitive to alignment order. To test the effect of assembly order, alignment scores starting from different nodes in the TSP circle and following different directions of the circle were tested. Given a starting node and a direction, the sequence in the starting node is first aligned with the sequence in the next node on the TSP circle in the given direction. Then the sequence in the third node is aligned with the group. This step is repeated, adding one sequence at a time, until they have all been aligned. It was found that different starting points and different directions gave different qualities of alignment. This is illustrated in Figure 4.1 using the representative example *kinase_ref3*, a reference set of 23 sequences in BAliBASE. The two numbers linked to each node by two arrows are the scores for alignments starting from this node, while the arrows indicate the directions of the tour taken. In Figure 4.1, the number inside a node is the sequence ID of

the sequence associated with the node (see Table 4.1). The number outside the circle on an edge between two nodes is the distance between the two nodes. The number inside on the edge is the rank of the distance. The shortest distance between two nodes is ranked 0, while the longest is ranked 22. The lengths of the edges are not drawn to scale.

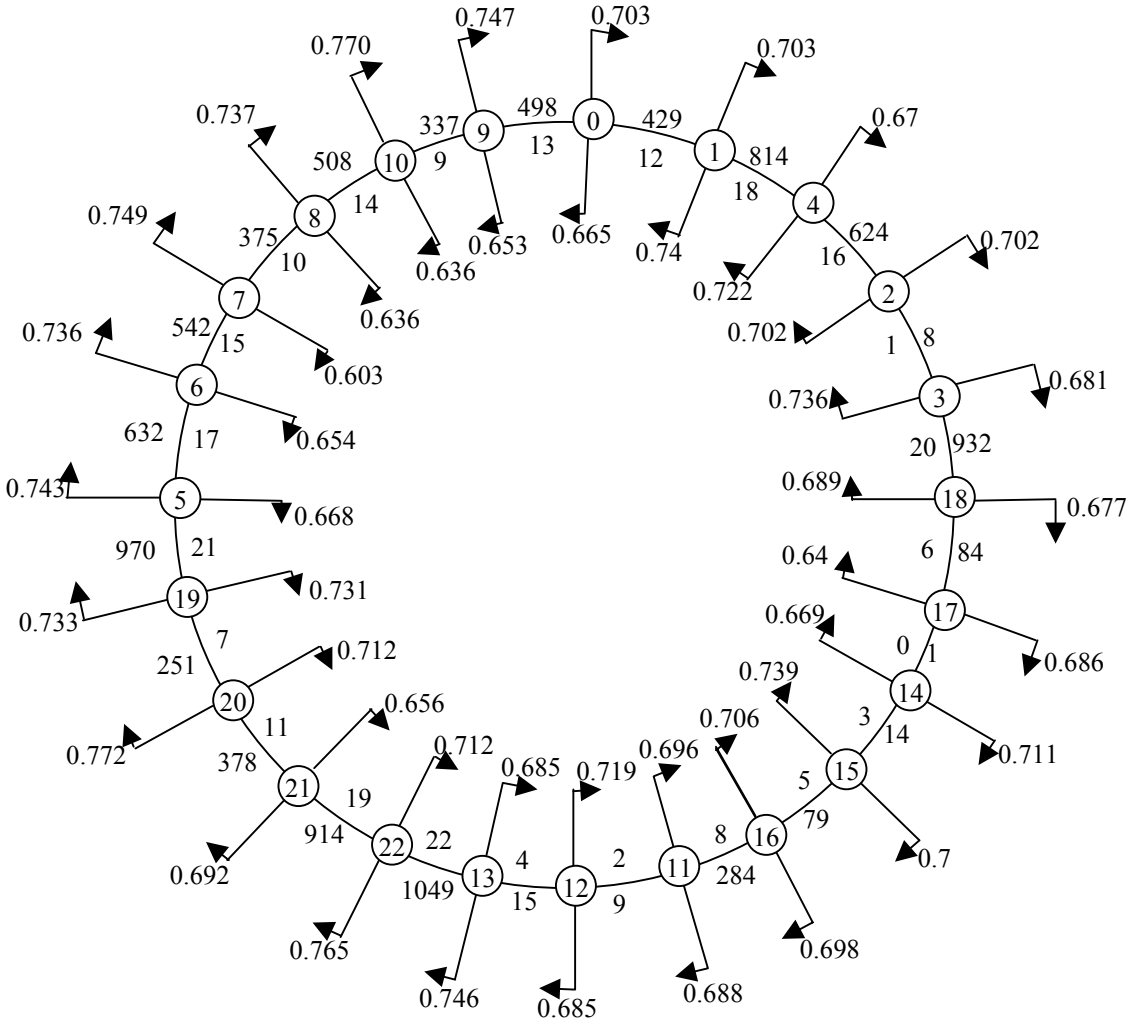


Figure 4.1 Different alignment scores for *kinase_ref3*.

0	1	2	3	4	5
1cdkA	*dc1*	*kcc1_yeast*	*kcc2_yeast*	*psk-h1*	*Bark*

6	7	8	9	10	11
ks62_human	*pkc-beta*	*kpc1_yeast*	*ypk2_yeast*	*krac_dicdi*	*1csn*

12	13	14	15	16	17
rag8_klula	*ck13_yeast*	*kc1e_human*	*hr25_yeast*	*hhp2_schpo*	*kc1a_drome*

18	19	20	21	22
kc1b_bovin	*dckii*	*cka1*	*cka2*	*erk1*

Table 4.1 The sequence IDs (used in the TSP tour) and corresponding names for the sequences in *kinase_ref3*.

As shown in Figure 4.1, starting from different nodes and following different directions can vary the alignment score for *kinase_ref3* from 0.603 to 0.772. The average of all the alignment scores is 0.701. As a control, CLUSTALW gives an alignment score of 0.747 and POA has the score 0.71.

The TspMsa program was modified to avoid the irregularities resulting from the choice of starting point and direction, and to possibly improve the alignment quality. First the two nodes with the shortest distance are aligned. Once aligned, the two nodes are merged into one node in the TSP circle and the edge between them is deleted. In our example of *kinase_ref3*, sequences 14 and 17 are aligned first. The distance data for edges will not be changed. In the example, the distance between the new node and node 18 remains 84, while the distance between the new node and node 15 is 14. The step of aligning the shortest edge is then repeated. In our example, it will be nodes 2 and 3. This process continues until there is only one node left. In the implementation, the edge lengths are sorted in ascending order once the TSP tour has been determined. The sorted list guides the order in which sequences along the TSP tour are aligned. The score for the

resulting alignment is increased to 0.836. Since this modified TspMsa program gives better performance, it is the default version of TspMsa for the remainder of the thesis.

Since two nodes are merged at each step, the alignment process can be mapped to a binary tree in which a leaf node represents an original sequence and an internal node represents an alignment resulting from merging two nodes (see Figure 4.2). From this point of view, the modified TspMsa program follows the traditional progressive alignment pattern of generating a guide tree to determine the order of alignment. However unlike other progressive alignment programs, the guide tree for TspMsa is restricted by a TSP tour.

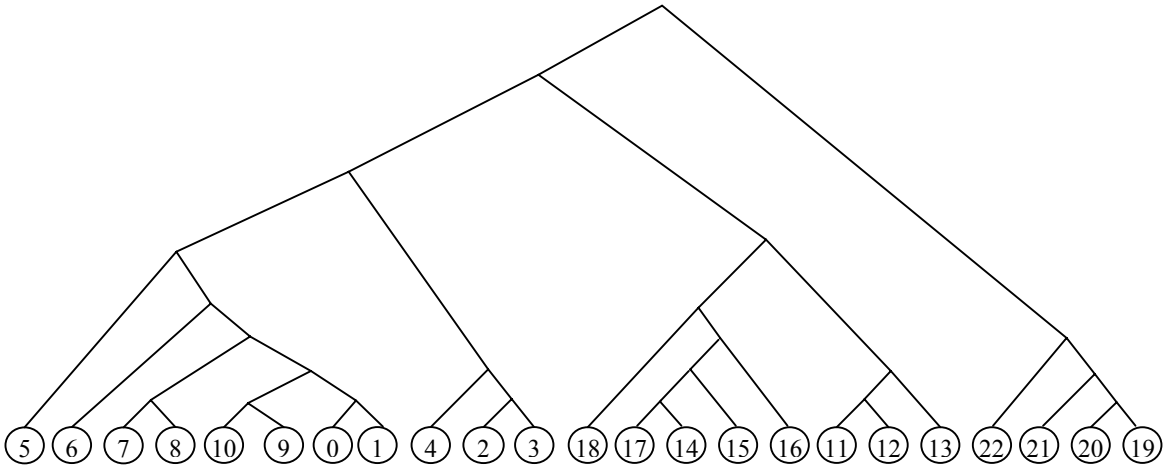


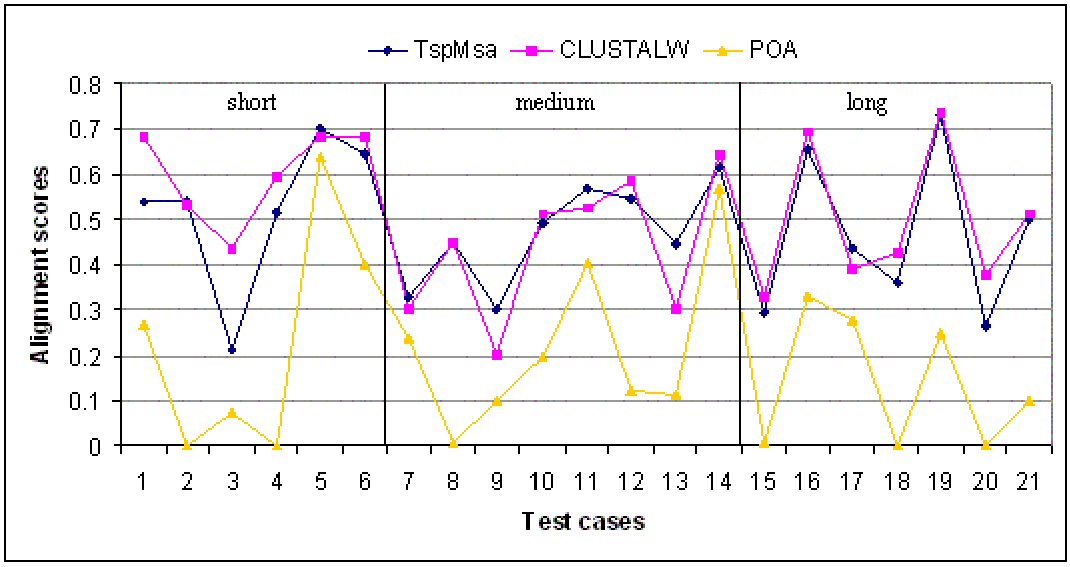
Figure 4.2 Aligning order of TspMsa, illustrated for *kinase_ref3*.

Quality Analysis

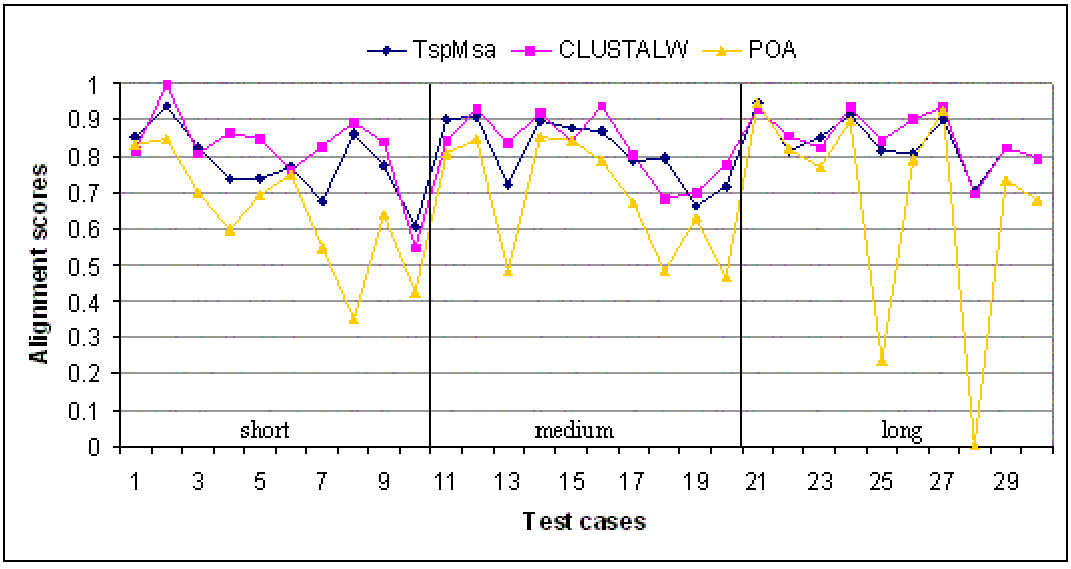
A comprehensive evaluation of the quality of TspMsa alignments was performed using BAliBASE benchmarks. CLUSTALW and POA were used for comparison. The results are shown in Figure 4.3 and listed in Table 4.2. In Figure 4.3, parts A – C display alignment scores for reference 1, with test sequences in part A having less than 25%

identity, sequences in part B having 20-40% identity, and sequences in part C having more than 35% identity. Figure 4.3D and E display alignment scores for reference 2 and 3 respectively. Figure 4.3F displays alignment scores for reference sets 4, 5 and 7. In Table 4.2, for TspMsa and POA, the number in the second line indicates the percentage of cases that the program gave a better alignment than CLUSTALW.

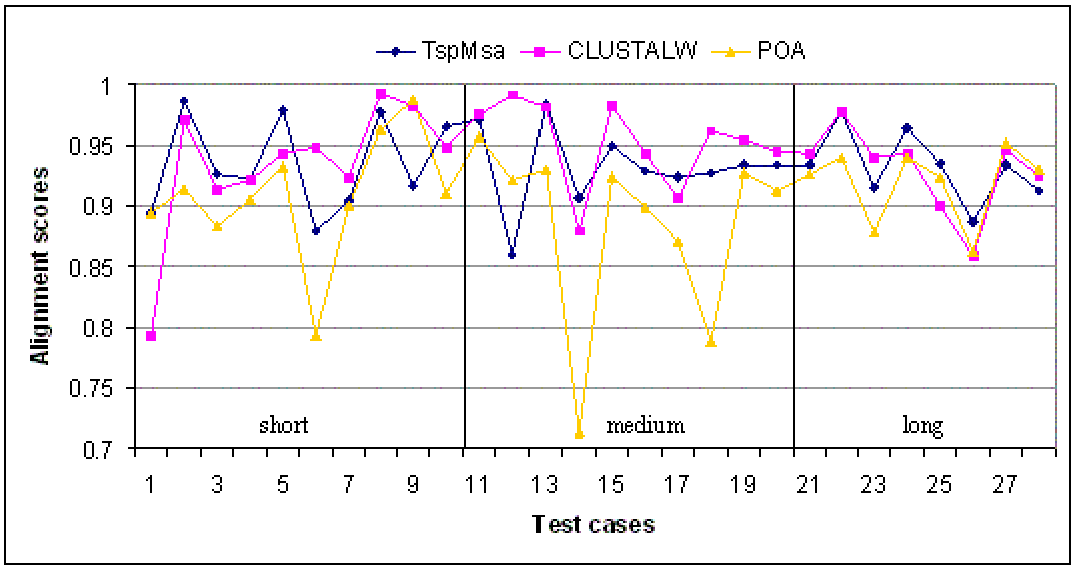
A)



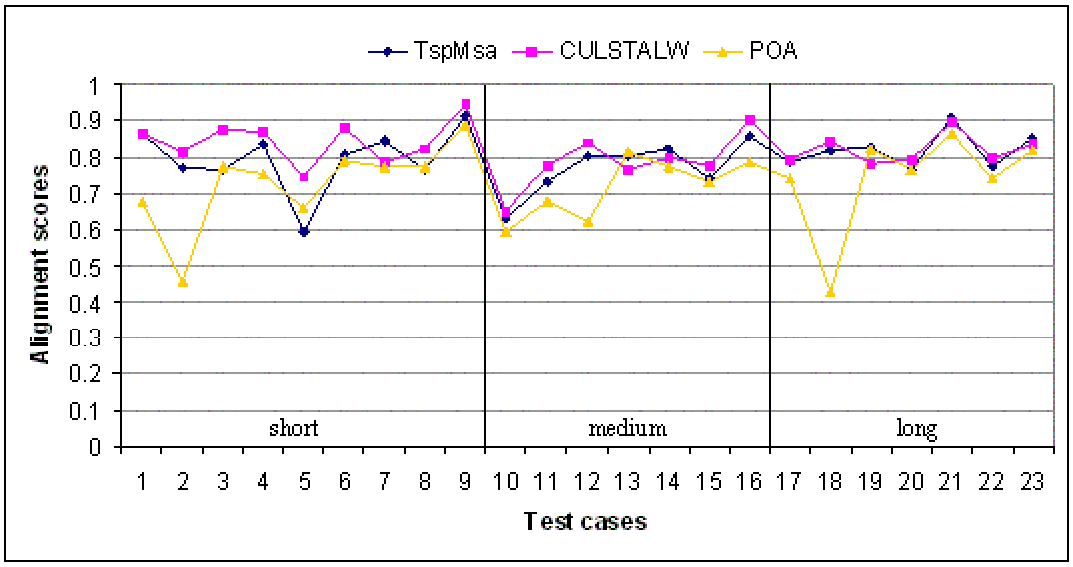
B)



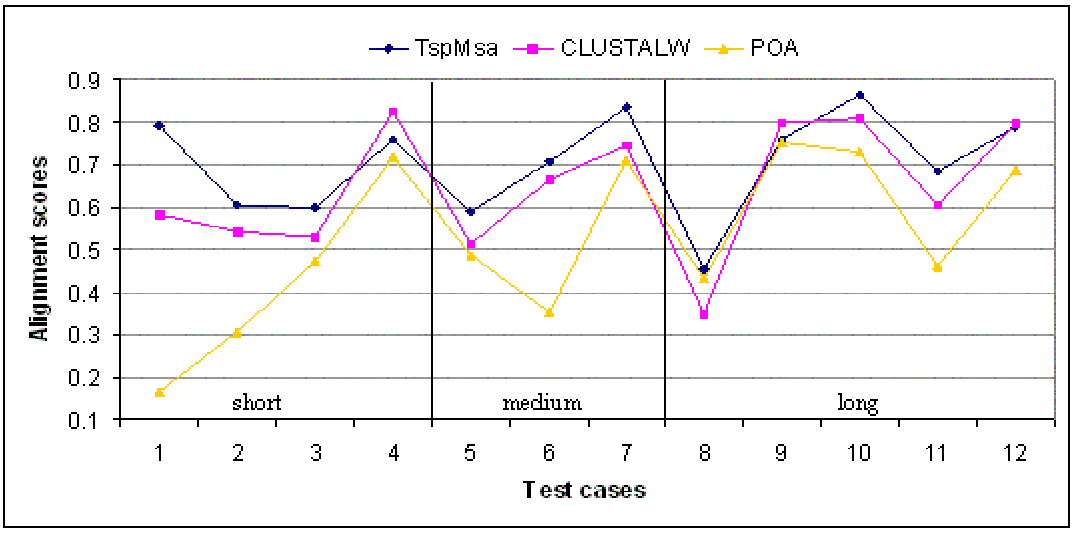
C)


TspMsa
CLUSTALW
POA
Alignment scores
1
0.95
0.9
0.85
0.8
0.75
0.7
short
medium
long
1 3 5 7 9 11 13 15 17 19 21 23 25 27
Test cases

D)


TspMsa
CULSTALW
POA
Alignment scores
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
short
medium
long
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Test cases

E)


TspMsa
CLUSTALW
POA
Alignment scores
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
short
medium
long
1 2 3 4 5 6 7 8 9 10 11 12
Test cases

F)

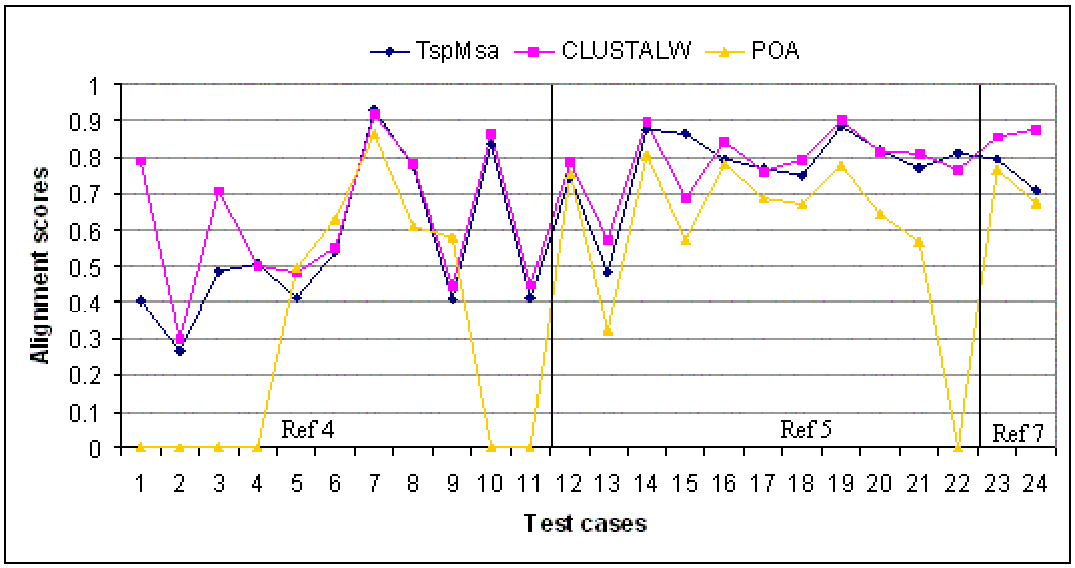


Figure 4.3 Alignment scores for TspMsa, CLUSTALW and POA for all test cases in BAliBASE.

	Reference 1								
	< 25% identity			20-40% identity			>35% identity		
	short	medium	long	short	medium	long	short	medium	long
clustalw	0.603	0.441	0.492	0.820	0.832	0.846	0.934	0.952	0.929
TspMsa	0.525 33%	0.469 63%	0.465 14%	0.777 40%	0.824 33%	0.826 42%	0.935 60%	0.932 20%	0.933 50%
POA	0.230 0	0.218 0	0.137 0	0.637 0	0.711 11%	0.660 8%	0.908 20%	0.884 0	0.919 50%

	Reference 2			Reference 3			Ref. 4	Ref. 5	Ref. 7
	short	medium	long	short	medium	long			
clustalw	0.844	0.787	0.820	0.620	0.641	0.671	0.636	0.758	0.868
TspMsa	0.795 22%	0.769 29%	0.819 43%	0.689 75%	0.711 100%	0.710 60%	0.557 20%	0.749 33%	0.752 0
POA	0.727 0	0.714 14%	0.740 14%	0.415 0	0.516 0	0.612 20%	0.319 30%	0.550 0	0.721 0

Table 4.2 Average alignment scores from TspMsa, CLUSTALW and POA for different data groups.

From the data, it is seen at once that CLUSTALW and TspMsa give better quality alignments than POA in all reference sets. A detailed analysis is needed to compare CLUSTALW with TspMsa. For reference 1 sequences with less than 25% identity, TspMsa gives alignments with quality comparable to that of CLUSTALW in the groups of medium and long sequences, while CLUSTSALW is slightly better for short sequences (see Figure 4.3A). The average scores for CLUSTALW and TspMsa have less than 0.03 in difference in both long and medium length groups, as shown in Table 4.2. The same pattern can be found in Figure 4.3B for sequences with 20-40% identity. For sequences with more than 35% identity, the two programs show similar average scores in all groups, with less than 0.02 in difference (see Figure 4.3C and Table 4.2). Similarly, for reference 2 CLUSTALW and TspMsa are comparable for medium and long sequences, with less than 0.02 in difference in average scores, while CLUSTALW gives higher quality alignments for short sequences (see Figure 4.3D and Table 4.2).

While comparable average scores can be found for reference 5, with less than 0.01 in difference, TspMsa is significantly better than CLUTSALW in all three groups in reference 3 as seen in Figure 4.3 parts E and F, and in Table 4.2. In contrast, CLUSTALW is superior to TspMsa for reference 4 and reference 7 (see Figure 4.3F and Table 4.2).

Execution Time Analysis

Most data sets in BAliBASE have fewer than 50 sequences. All three programs can complete the alignments within seconds. When computing an alignment for a large number of sequences, the most time-consuming step for TspMsa is the calculation of pairwise distances. When computing more than 500 sequences, this step can take hours,

while the TSP tour can be computed in seconds and the progressive alignment can be completed in minutes. The similar problem is solved in CLUSTALW by using a fast approximate matching method instead of full progressive programming for pairwise distance computations. CLUSTALW allows the user to determine whether to run in this fast approximate mode or the slow accurate mode.

A similar approach is taken by TspMsa. In the fast mode of TspMsa, an approximation method is used to evaluate the similarity of two sequences. To approximate the matching score of two sequences Seq_1 and Seq_2, assuming that n is the length of the shorter sequence, the score is

$$\sum_{i=1}^{n} S(C_i, C'_i) * 100 / n .$$

Here C_i denotes the i^{th} character of Seq_1, C'_i denotes the i^{th} character of Seq_2, and $S(c_1, c_2)$ is the matching score for characters c_1 and c_2 in the scoring matrix.

To assess the impact of this fast method on alignment quality, TspMsa and CLUSTALW were both tested in fast mode using the same BAliBASE benchmark (see Table 4.3). Note that the percentage in Table 4.3 indicates the proportion of cases for which TspMsa (fast mode) gave a better alignment than CLUSTALW (fast mode). As seen in Tables 4.3 and 4.2, the alignments using fast mode TspMsa and fast mode CLUSTALW have lower quality than those using slow modes, but they are still better than the alignments provided by POA. In fast mode TspMsa provides better alignments than fast mode CLUTSALW for data sets in reference 1 that have sequences with less than 25% identity. The two programs have comparable performance for the remaining test data sets in reference 1, the data sets in reference 2, the data sets composed of

medium length sequences in reference 3, and the data sets in reference 5. In fast mode CLUSTALW is better for short sequences in reference 3, while fast mode TspMsa is better for long sequences in reference 3. Finally, CLUSTALW performs better for sets in references 4 and 7.

	Reference 1								
	< 25% identity			20-40% identity			>35% identity		
	short	medium	long	short	medium	long	short	medium	long
clustalw	0.445	0.378	0.408	0.784	0.821	0.821	0.923	0.934	0.910
TspMsa	0.507 67%	0.446 75%	0.461 57%	0.791 60%	0.800 33%	0.829 45%	0.927 44%	0.923 40%	0.925 75%

	Reference 2			Reference 3			Ref. 4	Ref. 5	Ref. 7
	short	medium	long	short	medium	long			
clustalw	0.782	0.748	0.785	0.542	0.641	0.641	0.620	0.726	0.853
TspMsa	0.790 67%	0.750 57%	0.795 86%	0.517 50%	0.647 33%	0.686 100%	0.521 20%	0.706 33%	0.780 0

Table 4.3 Average alignment scores of TspMsa and CLUSTALW running in fast mode.

To test the scalability of TspMsa, 1000 tubulin sequences downloaded from `http://pfam.wustl.edu/cgi-bin/getdesc?name=tubulin` were used as large scale test data for execution time analysis. All programs are tested on a Linux i686 computer. Both CLUSTALW and TspMsa were run in fast mode for these execution time comparisons. TspMsa was executed with Java version 1.4.0 with JIT (Just-In-Time) compiler enabled. The results are shown in Figure 4.4. For up to 200 sequences, the three programs have similar execution time. For more than 500 sequences, CLUSTALW takes significantly longer than either TspMsa or POA. Even in fast mode it took CLUSTALW a good 186 minutes to align 1000 sequences, while POA only needed 32 minutes and TspMsa (in fast mode) completed the alignment within 25 minutes.

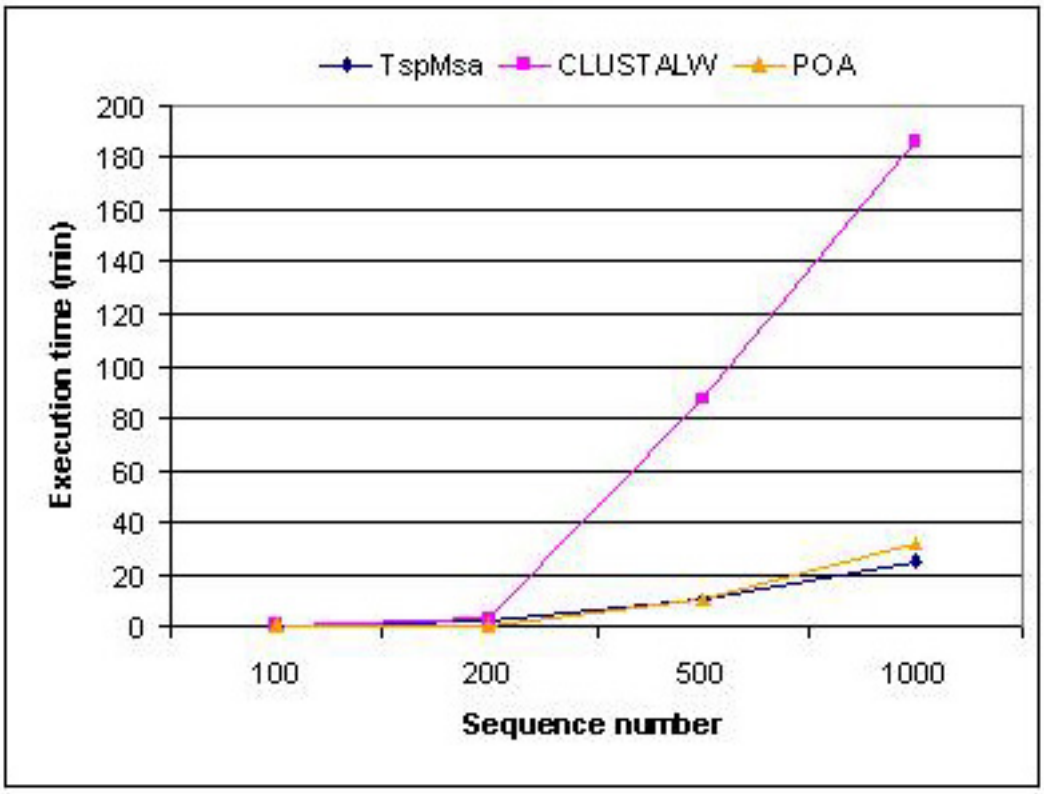


Figure 4.4 Execution time comparison.

CHAPTER 5

DISCUSSION

Over the relatively long period of time that the multiple sequence alignment problem has been studied, CLUSTALW has been found to be the progressive alignment program that provides the best quality alignments (Thompson *et al.*, 1999), while POA has been used in practice to provide fast alignments on large scale data sets (Lee *et al.*, 2002). In this thesis, an alternative heuristic method using the traveling salesman problem to restrict the guide tree for progressive alignment has been comprehensively tested and evaluated. The program, TspMsa, offers alignments with quality comparable to that of CLUSTALW at a running speed comparable to that of POA.

TspMsa produces alignments which are similar or better than those from CLUSTALW in most BAliBASE benchmark test cases except for sequences with N/C-terminal extensions in reference 4 and the two transmembrane data sets in reference 7 (see Tables 4.2 and 4.3). Both programs give alignments which are significantly better than those from POA, as seen in Table 4.2 and Figure 4.3. For alignments of large scale data sets, TspMsa and POA require considerably shorter execution times than CLUSTALW (see Figure 4.4).

CLUSTALW dynamically changes scoring matrices and gap penalties based on pairwise sequence similarities during the alignment process. In TspMsa, a much simpler progressive alignment method with a constant matrix and a constant gap penalty is used throughout the alignment process. It would be interesting to explore whether changing

the scoring matrix and gap penalty for each sequence when building the alignment would improve the quality of TspMsa alignments. Within the existing framework of TspMsa, this could be achieved by re-implementing the Align class to adopt the CLUSTALW approach.

A preprocess for extracting protein domains would allow TspMsa to be better suited to certain types of data, such as BAliBASE reference 6 sequences with repeated domains. It would also help to increase the accuracy of the fast method used to calculate pairwise distances, because a quick match of a conserved protein domain provides a better estimate of the divergence and similarity of the two sequences than does matching two unrelated domains.

In summary, the research in this thesis has provided a promising method that constructs high quality multiple sequence alignments quickly. This was implemented as the TspMsa package and tested extensively. The research has also laid the ground work for future improvements to TspMsa.

REFERENCES

Barton, G. J. and Sternberg, M. J. E. (1987) A strategy for the rapid multiple alignment of protein sequences – confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, 198: 327-337.

Dayhoff, M. O., Schwartz, R. and Orcutt, B.C. (1978) A model of evolutionary change in proteins. In Dayhoff, M. O. (ed.) *Atlas of Protein Sequence and Structure*. Vol. 5, Suppl. 3, pp. 345-352. National Biomedical Research Foundation, Washington, DC.

Eddy, S. R. (1995) Multiple alignment using hidden Markov models. *Proc. Int. Conf. Intell. Syst. Mol. Biol.,* 3:114-120.

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

Feng, D. and Doolittle, R. (1996) Progressive alignment of amino acid sequences and construction of phylogenetic trees from them. *Methods Enzymol.,* 266:368-382.

Fitch, W. M. (1966) An improved method of testing for evolutionary homology *J. Mol. Biol.,* 16: 9-16.

Gotoh, O. (1996) Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.,* 264: 823-838.

Gupta, S. K., Kececioglu, J. and Schäffer, A. A. (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comput. Biol.*, 2: 459-472.

Henikoff, S. and Henikoff, J. G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Nat. Acad. Sci. USA*, 89: 10915-10919.

Johnson, D. S. and McGeoch, L. A. (2002) Experimental analysis of heuristics for the STSP. In Gutin, G. and Punnen, A. P. (eds.) *The Traveling Salesman Problem and its Variations*, pp. 415-424. Kluwer Academic Publishers, Dordrecht, the Netherlands.

Korostensky, C. and Gonnet, G. H. (2000) Using traveling salesman problem algorithms for evolutionary tree construction. *Bioinformatics*, 16: 619-627.

Kreher, D. L. and Stinson, D. R. (1999) *Combinatorial Algorithms: Generation, Enumeration, and Search.* CRC Press, Boca Raton, FL.

Lee, C., Grasso, C. and Sharlow, M. F. (2002) Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18: 452-464.

Lipman, D. J., Altschul, S. F. and Kececioglu, J. D. (1989) A tool for multiple sequence alignment. *Proc. Nat. Acad. Sci. USA* 86: 4412-4415.

McLachlan, A. D. (1972) Repeating sequences and gene duplication in proteins. *J. Mol. Biol.*, 64: 417-437.

Mount, D. W. (2001) *Bioinformatics: Sequence and Genome Analysis.* Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY.

Needleman, S. B. and Wunsch, C. D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48: 443-453.

Nicholas, H. B. Jr, Ropelewski, A. J. and Deerfield, D. W. (2002) Strategies for multiple sequence alignment. *Biotechniques*, 32: 572-578.

Notredame, C. and Higgins, D. G. (1996) SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Res.*, 24: 1515-1524.

Rego, C. and Glover, F. (2002) Local search and metaheuristics. In Gutin, G. and Punnen, A. P. (eds.) *The Traveling Salesman Problem and its Variations*, pp. 321-327. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4: 406-425.

Sneath, P. and Sokal, R. (1973) *Numerical Taxonomy: the Principles and Practice of Numerical Classification.* W. H. Freeman and Company, San Francisco, CA.

Taylor, W. R. (1988) A flexible method to align large numbers of biological sequences. *J. Mol. Evol.*, 28: 161-169.

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

Thompson, J. D., Plewniak, F. and Poch, O. (1999) A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res.*, 27: 2682-2690.

Wang, L. and Jiang, T. (1994) On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1: 337-348.

Wang, X. (2002) *Multiple Sequence Alignment Using Traveling Salesman Problem Algorithms.* M.S. Thesis, University of Georgia, Athens, GA.

APPENDEX

TSPMSA USER MANUAL

To use the TspMsa program, simply type a command with several arguments as in the following example:

```
java TspMsa tubulin.fasta tubulin.msf –B –fast.
```

The general format of the command line is

```
java TspMsa [input] [output] –[matrix] –[mode].
```

Here, [input] specifies the input file name. If the file is in the same folder as TspMsa, then simply type the file name, for example, tubulin.fasta. If the file is under another folder, then a path is needed, for example /home/sarrac/weiwei/data/tubulin.fasta. Among all of the command line options, input is the only necessary argument and the others are optional.

Users can specify the desired output file name with the [output] argument. As in the case of the [input] argument, a path is needed if the desired output destination is not in the TspMsa folder. If the [output] argument is missing, a default output file name will be used. The default output file name is the input file name (without its file extension name) plus a postfix ".msf". For example, the command

```
java TspMsa tubulin.fasta
```

will produce an output file named tubulin.msf in folder TspMsa.

The [matrix] argument specifies the scoring matrix with “B” for BLOSUM62, “P” for PAM250, and “G” for GONNET. If the user does not specify this option, the default value is BLOSUM62.

The [mode] option specifies the running node of the program with “fast” for the fast heuristic mode and “slow” for the slow accurate mode. If the argument is missing in the user command, the slow mode will be used.