XUTING WANG
Multiple Sequence Alignment Using Traveling Salesman Problem Algorithms
(Under the direction of Dr. ROBERT W. ROBINSON)

The construction of multiple sequence alignments (MSAs) is a fundamental problem in computational biology, but computing an optimal MSA is NP-hard. It is therefore necessary to develop heuristics to come as close to the optimum as possible but operate within reasonable time and space bounds. In this thesis, an approach pioneered by Korostensky and Gonnet is developed and tested extensively. The idea is to apply a Traveling Salesman Problem (TSP) algorithm to find an optimal circular order to build an MSA progressively. The sum-of-pairs (SP) score is used for computing pairwise alignments and evaluating the quality of an MSA. Using the reference alignments from the benchmark alignment database BALISBASE, the performance of my program, **tspMSA**, was evaluated extensively with more than 60 reference data sets. Except for one test case, the SP scores obtained from the TSP algorithm are significant better than the scores obtained from two popular progressive alignment programs, **PILEUP** and **CLUSTALW**. The SP scores of MSAs built from different starting points and different directions of an optimal circular order are studied.

INDEX WORDS: multiple sequence alignment, MSA, traveling salesman problem, TSP, algorithms, branch-and-bound, global alignments, phylogenic tree, progressive alignment.

MULTIPLE SEQUENCE ALIGNMENT USING TRAVELING SALESMAN PROBLEM ALGORITHMS

by

XUTING WANG

B.S., Shandong University, China, 1988

M.S., Jilin University, China, 1991

Ph.D., The University of Georgia, 2001

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

2002

MULTIPLE SEQUENCE ALIGNMENT USING TRAVELING SALESMAN

PROBLEM ALGORITHMS

by

XUTING WANG

Approved:

Major Professor: Robert W. Robinson

Committee: Thiab R. Taha
E. Rodney Canfield

Electronic Version Approved:

Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
May 2002

DEDICATION

This dissertation is dedicated to my beloved wife, children, parents, and parents-in-law, whose love, support, and encouragement made this possible.

ACKNOWLEDGMENTS

First, I want to express my most sincere thanks to my major professor Dr. Robert W. Robinson for his guidance, encouragement, and help during my M.S. research at the Department of Computer Science.

Many thanks to my other committee members, Drs. E. Rodney Canfield and Thiab R. Taha, who offered valuable guidance and suggestion.

Special thanks to Dr. Liming Cai for his very valuable help, suggestion and attentions on my research.

Finally, thanks are extended to Dr. Harry W. Dickerson, who supported me during my completion of this degree.

TABLE OF CONTENTS

CHAPTER 1

BACKGROUND

Two fundamental building blocks of living creatures are DNA (deoxyribonucleic acid) and proteins, which are long linear chains of chemical components. The DNA molecule is made up of four different nucleotides, each denoted by one of the letters A, C, G or T. Proteins are made up of 20 different amino acids (or residues) which are denoted by 20 different letters of the alphabet. So, the sequence of either a DNA or a protein molecule can be simply viewed as a string of letters.

Alignment of DNA and protein sequences is one of the most important primitive operations in computational biology, serving as a basis for many other more complex manipulations. The basic concept of this operation is conceptually simple. However, the problem of computing an optimal solution is NP complete (13). It is therefore necessary to develop heuristics that come as close to the optimum as possible but operate within reasonable time and space bounds.

1. Definition of sequence alignment

Sequence alignment is the procedure of comparing two or more sequences by looking for a series of individual characters or character patterns that are in the same order in the sequences. If we align two sequences, the operation is called a pair-wise alignment; if the number of sequences to be aligned is more than two, the operation is called a multiple sequence alignment.

To align sequences by hand, they are written across a page in rows. Identical or similar characters are placed in the same column, and non-identical characters can either be placed in the same column as a mis-match or opposite a gap in one of the other sequences. In an optimal alignment, non-identical characters and gaps are so placed to bring as many identical or similar characters as possible into vertical agreement. Two types of sequence alignment have been recognized, global and local, as illustrated below.

2. Global and local alignment of protein sequences

Global alignment compares sequences over their entire lengths, and is appropriate for application to sequences that are expected to share similarity over the whole length. In global alignment, an attempt is made to maximize regions of similarity and to minimize gaps.

Local alignment finds subsequences of the input sequences which align at the highest score. In other words we are ready to ignore any deletions which happen at the beginning or end of any of the aligned fragments. A fragment is a consecutive subsequence (or substring) of a sequence. In local alignment, stretches of sequence with the highest density of matches are given the highest priority, thus generating one or more islands of matches in the aligned sequences.

Figure 1.1 illustrates a global alignment and a local alignment of two hypothetical protein sequences (24). Vertical bars between the sequences indicate the presence of identical amino acids. Dashes indicate fragments that are not included in the alignment.

The global alignment is stretched over the entire sequence lengths to include as many matching amino acids as possible. Although there is an obvious region of identity

in this example (the sequence FGKG), a global alignment may not align such regions in order to favor matching more amino acids along the entire sequence lengths.

The local alignment of the same sequences tends to stop at the ends of regions of identity or strong similarity. A much higher priority is given to finding these local regions than to extending the alignment to include more neighboring amino acid pairs. This type of alignment favors finding conserved amino acid motifs in related protein sequences.

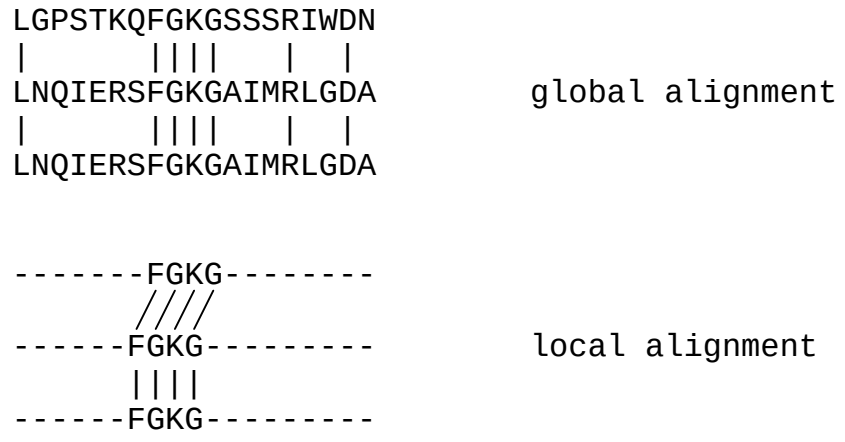


Figure 1.1. Alignment of two protein sequences.

3. Why perform sequence alignments?

Multiple sequence alignments are usually undertaken in order to perform a function such as one of the following:

- Determination of consensus regions of several sequences;
- Characterization of protein families by identifying shared regions;
- Molecular evolution analysis of a gene/protein family;
- Prediction of the secondary and tertiary structures of new sequences;
- Prediction of the function of new sequences.

There are many other functions that might be the goal of a multiple sequence alignment. For example, by making a multiple sequence alignment, we can find that two or more sequences are *similar*. If many characters in one sequence are in the same order as they are in the other sequence, then we say they are similar. We know 1) or 2) may be true for similar sequences: 1) the sequences may share a common origin - a common ancestor sequence. If the similarity is sufficiently convincing or if we have additional evidence for an evolutionary relationship, then we say that the sequences are homologous. 2) the sequences may have the same or related structure and function. The stronger the alignment between sequences, the more likely they are to be related. Very similar sequences that are almost identical along their lengths almost certainly have the same function. Sequences that are only weakly similar may or may not be related, and no firm conclusions can be drawn about their relationship.

4. Algorithms for multiple sequence alignment

This thesis is devoted to global alignment of multiple protein sequences, so hereafter the terms alignment and MSA mean global alignment of multiple protein sequences.

Definition: Global Multiple Sequence Alignment (MSA)

Given a set of sequences $S = \{s_1, s_2, \ldots, s_k\}$ with $s_i \in \Sigma^*$ where Σ^* is a finite alphabet, a multiple sequence alignment is a set of sequences $A = < a_1, a_2, \ldots, a_k >$ with $a_i \in \Sigma'^*$ where $\Sigma' = \Sigma \cup \{\text{“-”}\}$and $\{\text{“-”}\} \notin \Sigma$. All sequences in A have some common length *n* and therefore can be arranged in a matrix of *k* rows and *n* columns. The sequence obtained from $a_i \in A$ by removing all “-” gap characters is equal to s_i.

The problem of finding an MSA can be solved by either a dynamic programming (DP) algorithm or a heuristic algorithm.

4.1. Dynamic programming algorithm

Basically, dynamic programming is a method for successively optimizing the alignments of all pairs of prefixes of the two sequences, adding just one new character at a time to a prefix.

Needleman and Wunsch (26) first used dynamic programming in the comparison of two protein sequences. This algorithm sets up a 2-dimensional matrix where each sequence is placed along the sides of the matrix. Each element in the matrix represents the two residues of the sequences being aligned at that position. To calculate the score in position (i, j), one looks at the alignment that has already been made up to that point, and finds the best way to continue. Having gone through the entire matrix in this way, one can go back and trace which way through the matrix gives the best alignment.

To evaluate the quality of an alignment, we need some scoring scheme. A popular scoring scheme is the so-called sum-of-pairs (SP) function, which adds up the scores of each pair of aligned residues. For the SP score function, the higher the score, the better the quality of an alignment. So, the optimal alignment is an alignment with the maximum SP score.

The following is an example of pairwise sequence alignment using Needleman/Wunsch techniques. Two sequences to be globally aligned are:
sequence 1 = G A A T T C A G T T A, and sequence 2 = G G A T C G A

A simple scoring scheme is assumed where

- $S_{i,j}$ = 1 if the residue at position i of sequence 1 is the same as the residue at position j of sequence 2 (match score); otherwise
- $S_{i,j}$ = -1 (mismatch score)
- g = -2 (gap penalty)

In the dynamic programming algorithm, alignment is performed in three steps:

1) Initialization

Create a matrix with m + 1 columns and n + 1 rows where m and n correspond to the lengths of the sequences to be aligned (Figure 1.2). The first row and first column of the matrix are filled with 0s.

		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. Initialized matrix

2) Matrix fill (scoring)

For each position, $M_{i,j}$ is defined to be the maximum score at position i,j;

i.e., $M_{i,j} = \max\{ M_{i-1, j-1} + S_{i,j}$ (match/mismatch in the diagonal),

$M_{i,j-1} + g$ (gap in sequence 1),

$M_{i-1,j} + g$ (gap in sequence 2) }

Optionally, an arrow is placed to point back to the cell that led to the maximum score. The filled matrix is shown below (Figure 1.3).

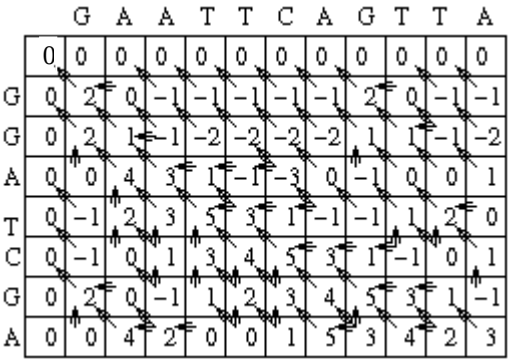


Figure 1.3. Score Matrix with pointers for tracing back

3) Traceback (alignment)

The traceback step determines the actual alignment that resulted in the maximum score. This step begins in the (n, m) position. It takes the current cell and looks to the neighbor cells that could be direct predecessor. This means it looks to the neighbor to the left (gap in sequence 2), the diagonal neighbor (match/mismatch), and the neighbor above it (gap in sequence1). Since we have kept pointers back to all optimal predecessors, the traceback step is simple. At each cell, we just need to find where we move next according to the pointers. In the example here, we can get the two possible paths shown in Figure 1.4.

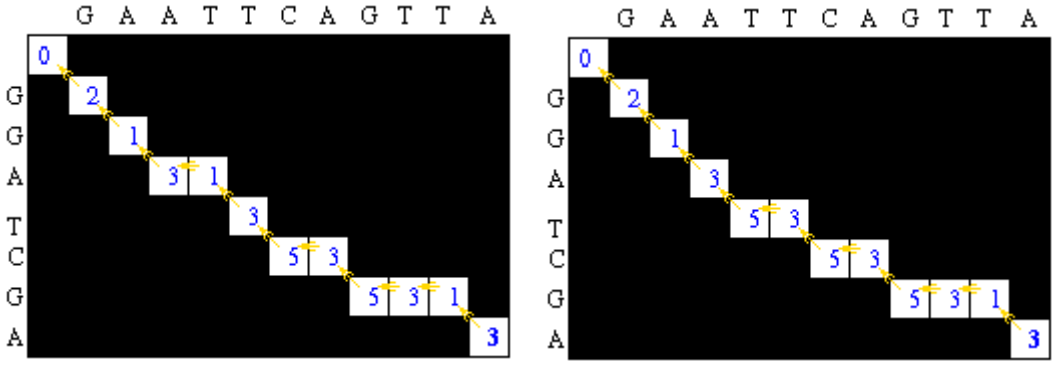


Figure 1.4. Traceback step

The alignments corresponding to these two paths are:

```
G A A T T C A G T T A        G A A T T C A G T T A
|   |   | |   |      |       |   | |   |   |      |
G G A _ T C _ G _ _ A        G G A T _ C _ G _ _ A
```

This method also has been extended directly to the comparison of three sequences using a reduced three-dimensional matrix by Murata *et al* (25) with $O(n^3)$ computational complexity, where n is the longest length of sequences to be aligned.

The dynamic programming algorithm guarantees an optimal MSA on a set of given sequences. However, the space complexity is $O(n^k)$ and the time complexity is $O(k^2 2^k n^k)$ if we use the sum-of-pairs score, where k is the number of sequences and n is the longest length of the sequences to be aligned.

In 1989, Lipman, Altschul, and Kececioglu implemented a more refined version of this algorithm in their program "MSA". This program in practice is restricted to aligning 5-7 protein sequences of 200-300 residues each (21).

4.2. Heuristic algorithms

Since searching for the optimal MSA using a DP algorithm is not realistic for more than 10 sequences, a number of heuristic algorithms have been developed to carry out a multiple global alignment in a reasonable amount of time with a reasonable chance of finding a near-optimal alignment.

These algorithms are heuristic in that they are not guaranteed to find an optimal solution. But they run fast, use reasonable memory in practice, and get MSAs with good quality at most cases.

4.2.1 Progressive Alignment

This approach begins with the alignment of the two most closely related sequences (as determined by pairwise analysis) and subsequently adds the next closest sequence or sequence group to this initial pair (4, 5). This process continues in an

iterative fashion, adjusting the positioning of indels in all sequences. An indel is an insertion or a deletion which appears in any aligned sequence.

The major shortcoming of this approach is that a bias may be introduced in the inference of the ordered series of motifs (homologous parts) because of an over-representation of a subset of sequences.

A concrete implementation of the progressive algorithm was developed by Feng and Doolittle in 1987 (5). The steps can be summarized as follows:

a) Calculate an above the diagonal matrix of k(k-1)/2 pairwise distances between all pairs of sequences s_1, s_2, …,s_n using the dynamic programming pairwise sequence alignment (PSA) algorithm (26),

b) These similarity scores are used to create a clustering order that can be represented as a dendrogram. The clustering strategy represented by the dendrogram is called UPGMA, which stands for **u**nweighted **p**air-**g**roup **m**ethod using **a**rithmetic averages (32).

c) Construct a guide tree from the distance matrix computed in step b);

d) Build the multiple alignment by first aligning the most similar pair of sequences, then the next most similar pair and so on. Once an alignment of two sequences has been made, then it is fixed and can only be modified by the insertion of common gaps;

e) Repeat step d) until all sequences have been aligned.

A program called "**PILEUP**" (a component of the GCG software package) creates a MSA using the progressive alignment method of Feng and Doolittle (4-6).

CLUSTALW is another widely used progressive MSA program. It is very similar to the Feng-Doolittle algorithm and works as follows:

a) Construct a distance matrix of all k(k-1)/2 unordered pairs of sequences by pairwise sequence alignment. Then convert the similarity scores to evolutionary distances using a specific model of evolution proposed by Kimura in 1983 (14);

b) Construct a guide-tree from this matrix using a clustering method called neighbor-joining proposed by Saitou and Nei in 1987 (30);

c) Progressively align nodes of the tree in order of decreasing similarity.

The time complexity of either progressive alignment algorithm is $O(k^2n^2)$.

4.2.2. Divide-and-Conquer Alignment

The Divide-and-Conquer Alignment is a fast heuristic algorithm for multiple sequence alignment which provides near-to-optimal results for sufficiently similar sequences (33, 37). The main idea is first to cut the sequences several times at certain points to reduce the length of the sequences, second to align the fragments, and third to concatenate the multiple alignments. The method needs time $O(k^{n-1})$ and space $O(k^2n^2)$.

5. Scoring matrices for sequence alignments

Early sequence alignment programs used a simple scoring function that scores all matches the same and penalizes all mismatches the same. This approach ignores protein evolution and structure. Construction of biologically significant alignments should take into account the fact that protein evolution is constrained by the chemical properties of amino acids, and by the degeneracy of the genetic code. Chemically conservative replacements tend to occur more frequently than replacements with amino acids that are chemically different. For example, it is far more likely that Leucine will be replaced with Isoleucine (both of which are non-polar), than with Aspartic acid (which is negatively charged).

Many scoring matrices have been constructed to replace the simple scoring functions described above. The commonly used scoring matrices are substitution matrices based on evolutionary distances. Substitution matrices are constructed by observing the frequencies of amino acid replacements in large samples of protein sequences.

The PAM (Point Accepted Mutations) matrices are the most commonly used scoring matrices. They were proposed by Margaret Dayhoff in 1978 (2, 31). Dayhoff carefully aligned all of the proteins in several families of proteins and then constructed phylogenetic trees for each family. Each phylogenetic tree was examined for the substitutions found on each branch. This leads to a table of the relative frequencies with which amino acids replace each other over a short evolutionary period. This table and the relative frequency of occurrence of the amino acids in the proteins studied were combined in computing the PAM family of scoring matrices. Therefore, PAM matrices show probability scores of replacement of amino acids by each other based on natural mutation rates in related protein families. For a given replacement, the PAM value is proportional to the natural log of the frequency with which that replacement was observed to occur. A positive score assigned to two amino acids indicates that these two replace each other more often than expected by chance alone, *i.e.*, they are functionally exchangeable. A negative score indicates that the two amino acids are rarely interchangeable. One PAM unit is defined as the amount of evolutionary change that yields, on average, one substitution in 100 amino acid residues. The traditional PAM matrix, the PAM250 matrix (Figure 1.5), often referred to as the Dayhoff Matrix, assumes the occurrence of 250 point mutations per 100 amino acids or 300 nucleotides in the gene.

Cys	12																		
Gly	-3	5																	
Pro	-3	-1	6																
Ser	0	1	1	1															
Ala	-2	1	1	1	2														
Thr	-2	0	0	1	1	3													
Asp	-5	1	-1	0	0	0	4												
Glu	-5	0	-1	0	0	0	3	4											
Asn	-4	0	-1	1	0	0	2	1	2										
Gln	-5	-1	0	-1	0	-1	2	2	1	4									
His	-3	-2	0	-1	-1	-1	1	1	2	3	6								
Lys	-5	-2	-1	0	-1	0	0	0	1	1	0	5							
Arg	-4	-3	0	0	-2	-1	-1	-1	0	1	2	3	6						
Val	-2	-1	-1	-1	0	0	-2	-2	-2	-2	-2	-2	-2	4					
Met	-5	-3	-2	-2	-1	-1	-3	-2	0	-1	-2	0	0	2	6				
Ile	-2	-3-	2-	-1	-1	0	-2	-2	-2	-2	-2	-2	-2	4	2	5			
Leu	-6	-4	-3	-3	-2	-2	-4	-3	-3	-2	-2	-3	-3	2	4	2	6		
Phe	-4	-5	-5	-3	-4	-3	-6	-5	-4	-5	-2	-5	-4	-1	0	1	2	9	
Tyr	0	-5	-5	-3	-3	-3	-4	-4	-2	-4	0	-4	-5	-2	-2	-1	-1	7	10
Trp	-8	-7	-6	-2	-6	-5	-7	-7	-4	-5	-3	-3	2	-6	-4	-5	-2	0	0	17
	Cys	Gly	Pro	Ser	Ala	Thr	Asp	Glu	Asn	Gln	His	Lys	Arg	Val	Met	Ile	Leu	Phe	Tyr	Trp

Figure 1.5. PAM250 matrix

The BLOSUM (Blocks Substitution Matrix) series of matrices are also commonly used. BLOSUMs are substitution matrices derived from the observed frequencies of amino acid replacements in highly conserved regions of ungapped local alignments. The data for the substitution scores in these matrices come from about 2000 blocks of aligned sequence segments characterizing more than 500 groups of related proteins (11).

Substitution matrices have theoretical advantages over alternative methods of scoring schemes. From a biological point of view, substitution matrices are based on observed mutations. Thus they contain information about the processes that generate mutations as well as the criteria that are important in selection and in fixing a mutation within a population. From a statistical point of view, substitution matrices are the most accurate description of the changes in amino acid composition that are expected after a given number of mutations that can be derived from the data used in creating the matrices. Thus the highest scoring alignment is the statistically most likely to have been generated by evolution rather than by chance.

6. Gap Penalty

A gap penalty is designed to reduce the score when an alignment has been broken by an insertion in one of the sequences. The value should be small enough to allow a previously accumulated alignment to continue with an insertion in one of the sequences but should not be so large that this previous alignment score is removed completely. The gap penalty scheme is often used to have a larger gap opening penalty followed by a much smaller gap extension penalty. Thus, the score becomes larger as a linear function of gap length. One extreme is to allow a constant gap penalty regardless of gap length.

The gap penalty used in this study is the same value used by **PILEUP** and **CLUSTALW**.

CHAPTER 2

RELATED WORK

Multiple sequence alignment analysis has become an essential tool that enables biologists to find characteristic motifs and conserved regions in protein families, determine evolutionary linkage, and predict secondary and tertiary structure. With the explosive increase in the number of known protein sequences, notably from the genome sequencing projects, searching protein databases for homologous sequences, followed by the alignment of a new sequence with a large closely related group is now standard practice. The development of accurate, reliable multiple alignment programs capable of handling these divergent sets of data is therefore of major importance. Although a dynamic programming algorithm (10) guarantees a mathematically optimal alignment, the method is limited to a small number of short sequences because the computing power required for larger data sets becomes prohibitive. To overcome this problem, various heuristic approaches have been developed leading to a variety of programs based on very different algorithms. This chapter summarizes some of the most commonly used programs.

1. Software implementing a dynamic programming approach: MSA

MSA is a global optimal multiple sequence alignment program originally written by John Kececioglu, Stephen Altschul, David Lipman, and Robert Miner and distributed in 1989 (21). Credit for improvements in release 2.0 of the code belong to Sandeep K Gupta and Alejandro A Schaffer with some guidance from John Kececioglu (10).

MSA utilizes a variant of a multi-dimensional dynamic programming to produce an optimal global alignment between several sequences. **MSA** implements a branch-and-bound technique together with a variant of Dijkstra's shortest path algorithm to greatly reduce the amount of space required to solve the the basic dynamic programming problem. Generally speaking, **MSA** will produce better alignments than other multiple sequence alignment programs such as **CLUSTALW** or **PILEUP**. The drawback to using **MSA** is that it requires an enormous amount of both computer time and memory to align more than a few distantly related sequences. The size of the problems solved by **MSA** are directly related to the sequence lengths, the number of sequences, and the amount of sequence diversity. At the Pittsburgh Supercomputing Center (PSC, http://www.psc.edu/general/software/packages/msa), there are three versions of **MSA** compiled for problems of different sizes:

* msa_50_150 - Align fewer than 50 sequences. Each sequence has fewer than 150 residues;

* msa_25_500 - Align fewer than 25 sequences. Each sequence has fewer than 500 residues;

* msa_10_1000 - Align no more than 10 sequences. Each sequence has fewer than 1000 residues.

MSA restricts the amount of memory needed by computing bounds that approximate the center of a multi-dimensional hypercube. The first bound is producing by computing pairwise alignments between the set of sequences. Weights are applied to this value to produce the lower bound used by the program. Next a heuristic alignment is produced for the sequences. Weights are applied to this value to produce the upper bound

used by the program. A *delta* value is then computed as the difference between these two values. The *epsilon* for each sequence pair may be input by the user or estimated. An heuristic alignment is computed and the *epsilons* are taken to be the differences between the projected and pairwise costs. The delta and epsilon values are preliminary measures of the divergence between the set of sequences. Thus, closely related sequences will have low epsilons and deltas while distantly related sequences will have high epsilons and deltas.

Even though **MSA** reduces the space required to produce a multiple alignment dramatically, it is still uses much more memory than the progressive pairwise technique.

2. Software implementing the progressive pairwise approach

The progressive pairwise approach relies on exhaustive pairwise alignments between all of the sequences to produce a measure of sequence relatedness. From this measure, an algorithm (UPGMA in **PILEUP**, Neighbor Joining in **CLUSTALW**) is used to develop a joining order. This joining order corresponds to a tree that is used to produce the multiple sequence alignment. It should be noted that this tree is **not an evolutionary tree.** After the joining order has been determined, sequences close to each other are aligned first.

(a) PILEUP

PILEUP creates a multiple sequence alignment from a group of related sequences using progressive, pairwise alignments. It implements a simplification of the progressive alignment method of Feng and Doolittle (4-6). The method used is similar to the method described by Higgins and Sharp (12). **PILEUP** is usually distributed as part of the

Wisconsin Package from the Genetics Computer Group (GCG) (**http://www.gcg.com**), which is licensed to numerous bioinformatics services on the Internet.

PILEUP begins by doing pairwise alignments that score the similarity between every possible pair of sequences. Each pairwise alignment in **PILEUP** uses the method of Needleman and Wunsch. These similarity scores are used to create a clustering order that can be represented as a dendrogram. The clustering strategy represented by the dendrogram is called UPGMA, which stands for **u**nweighted **p**air-**g**roup **m**ethod using **a**rithmetic averages (32).

PILEUP uses this clustering order and first aligns the two most-related sequences to each other in order to produce the first cluster. It then aligns the next most related sequence to this cluster. Alignments continue in a progressive fashion until all sequences have been included in the final alignment.

As a general rule, **PILEUP** can align up to 500 sequences, with any single sequence in the final alignment restricted to a maximum length of 5,000 characters (including gap characters inserted into the sequence by **PILEUP** to create the alignment).

(b) CLUSTALW

CLUSTALW is one of the most popular multiple nucleotide or protein sequence alignment programs. It uses a progressive alignment approach (35).

CLUSTALW uses a neighbor-joining method to construct a guide tree. This determines the order in which the sequences are incorporated into the alignment. Progressive multiple alignments are created by first aligning the most similar of a set of sequences and then incorporating less similar sequences successively into the alignment. A comparison of multiple sequence alignment programs revealed that **CLUSTALW**

performs well when aligning equidistant sequences of a similar length and when aligning small to large families of similar sequences, in which a few divergent sequences are also included in the alignment (36). In both cases, the performance of **CLUSTALW** was maintained from short sequences of less than 100 residues to those of over 400 residues.

CLUSTALW is available on several Internet servers, such as the European Bioinformatics Institute (EBI) (http://www2.ebi.ac.uk/clustalw/).

3. Other software

In addition to the software packages mentioned above, numerous new alignment algorithms have recently been developed which offer fresh approaches to the multiple alignment problem. A common point of interest has been the application of iterative strategies to refine and improve the initial alignment. The **PRRP** program (8) optimises a progressive, global alignment by iteratively dividing the sequences into two groups, which are subsequently realigned using a global group-to-group alignment algorithm. **SAGA** (27) uses a genetic algorithm to select from an evolving population the alignment which optimizes the **COFFEE** Objective Function (OF) (28). The OF is a measure of the consistency between the multiple alignment and a library of **CLUSTALW** pairwise alignments. Hidden Markov models (HMMs) have also been used as statistical models of the primary structure consensus of a sequence family (1, 18). The program **HMMT** (3) uses a simulated annealing method to maximize the probability that an HMM represents the sequences to be aligned.

CHAPTER 3

DESIGN OF ALGORITHMS

Adapting an approach pioneered by Korostensky and Gonnet (13), I have developed a heuristic method for multiple sequence alignment (MSA) which provides near-optimal results for protein sequences. It is assumed that the sequences are related to each other. To compute an MSA, this method first calculates the pairwise sum-of-pairs scores for each pair of sequences using the dynamic programming algorithm. Then the resulting scores are converted to a cost matrix and sent to a Traveling Salesman Problem (TSP) solver. Finally the circular tour determined via the TSP solver is used to provide a linear order to assemble the MSA. I will describe these steps in detail below.

Step 1. Calculating pairwise sum-of-pairs score

The Needleman-Wunsch dynamic programming algorithm (26) is used to find the optimal alignment of two sequences. This algorithm ensures the optimal global alignment by exploring all possible alignments and choosing the best. In a pairwise alignment, two sequences are padded by gaps to achieve the same length and to display the maximum similarity/conservation on a character-by-character basis.

To find the optimal alignment of two sequences, we first need to **fill a score matrix** by the following algorithm.

fillMatrix (s, t)

// Input: a pair of sequences s and t

// Output: a score matrix a[m, n]

```
m ← length (s)
n ← length (t)
for  i ← 0 to m do
        a[i, 0] ← 0
for  j ← 0 to n do
        a[0, j] ← 0
for  i ← 1 to m
        for  j ← 1 to n do
                a[i, j] ← max (a[i-1, j] + g,  a[i-1, j-1] + w(i, j),  a[i, j-1]+g)
                // w(i, j) is the score from the substitute matrix, g is the gap penalty
return a[m, n]
```

Then, we **traceback** the matrix to print out the optimal alignment by calling traceback(m, n, len), where len = max(m, n).

traceback(i, j, l)

```
// Input: a score matrix with pointers
// Output: an alignment in sequences align-s and align-t with length len
if i = j = 0 then
        return
else if a[i, j] = a[i-1, j] + g then
        traceback(i-1, j, len-1)
        align-s[len] ← s[i]
        align-t[len] ← -
```

```
else if a[i, j] = a[i-1, j-1] + w(i, j) then
        traceback(i-1, j-1, len-1)
        align-s[len] ← s[i]
        align-t[len] ← t[j]
else if a[i, j] = a[i, j-1] + g then
        traceback(i, j-1, len-1)
        align-s[len] ← -
        align-t[len] ← t[j]
```

An optimal alignment is an alignment which has an optimal sum-of-pairs score. The sum-of-pairs (SP) score is defined as the sum of all scores between all pairs of letters in the columns of the pairwise alignment.

For a pairwise alignment $a(s, t)$ made of a pair of sequences s and t and with the same length len, the SP score is calculated by:

$$SP(a) = \sum_{j=1}^{len} w(s[j], t[j])$$

where $s[j]$ or $t[j]$ is a residue in sequence s or t, and $w(s[j], t[j])$ is the score from the substitute matrix if $s[j]$ and $t[j]$ form a match or a mismatch. Otherwise $w(s[j], t[j])$ = gap penalty.

The SP-score of each optimal pairwise alignment is calculated and saved into an kxk distance matrix (where n is the number of sequences to be aligned). Since the matrix is symmetric and has 0's on the diagonal, we just store the k*(k -1)/2 entries which lie above the diagonal.

Step 2. Constructing a cost matrix

Initialize a kxk cost matrix (where k is the number of sequences to be aligned);

Find the largest value SP_{max} in the distance matrix;

Fill the cost matrix with the function: $cost(i,j) = SP_{max} - SP(i, j) + 1$.

Step 3. Finding the order of MSA by TSP algorithm

Using the symmetric Traveling Salesman Problem (TSP) algorithm, an optimal circular order based on the distances between the given sequences can be found. This optimal circular order can be used to construct a MSA (15).

In TSP, we are given a matrix M that contains the k*(k-1)/2 distances of n cities, and we need find the shortest tour where each city is visited once. In our case, the cities correspond to the sequences and the distances are the scores of the pairwise alignments. However, we are interested in the longest, not shortest tour, as we are interested in the maximum SP-score for a MSA. To be able to use any available TSP algorithm, we built a new cost matrix with the function $cost(i,j) = SP_{max} - SP(i, j) + 1$ in Step 2.

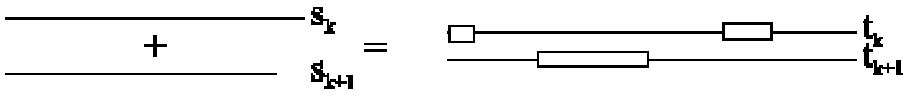
The Traveling Salesman Problem is NP hard (20, pp.1-15). But it is very well studied and optimal solutions can be calculated within a few hours for up to 1000 cities and in a few seconds for up to 100 cities (16). For real applications we have seldom more than 100 sequences to compare simultaneously. In my software, the branch-and-bound technique is used to solve the TSP. The algorithm and source code come from Drs. Donald L. Kreher and Douglas R. Stinson (17, pp.127-143).

Step 4. Assembling the MSA

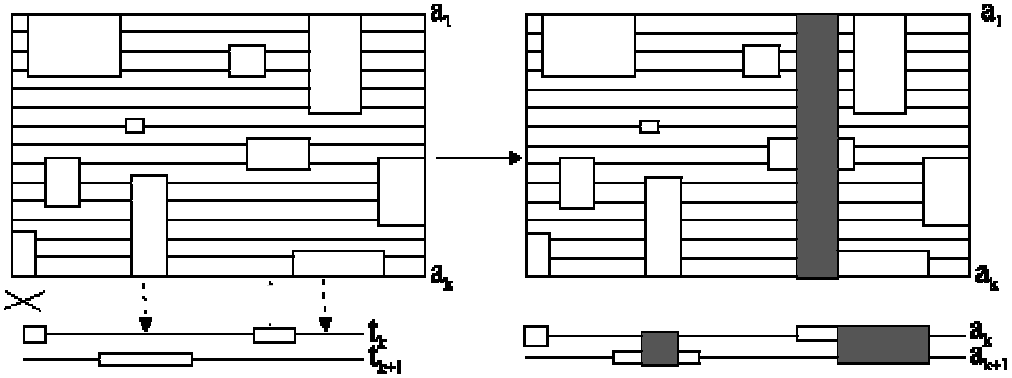
Assuming the optimal circular order is $\{s_1, s_2, .., s_n\}$, an MSA is assembled progressively. First, do an optimal alignment of sequences s_1 and s_2 using the Needleman-

Wunsch algorithm. The sequences in this alignment with the gaps are called t_1 and t_2. Then, align the remaining sequences one-by-one against previously aligned sequences until all n sequences are in the MSA. In detail, the procedure to add a sequence s_{k+1} into a partially aligned MSA $A' = \langle a_1, a_2, \ldots, a_k \rangle$ can be done in the following two steps (15):

1) Take sequence s_k and calculate an optimal pairwise alignment with the next sequence s_{k+1}. The sequences in this alignment with the gaps are called t_k and t_{k+1}.



2) Insert all gaps from t_k that are not already present in a_k into *all* previously aligned sequences $A' = \langle a_1, a_2, \ldots, a_k \rangle$. Insert all the gaps that were present in a_k into both, t_k and t_{k+1}, except for the gaps that are already present in t_k. Add a_{k+1} to the alignment A'.



Step 5. Calculating sum-of-pairs score of MSA

The SP score of the resulting MSA is used to evaluate the quality of the MSA. The SP score is defined as the sum of all pairwise scores between all pairs of letters in the columns of the multiple alignment A, and is calculated by the following algorithm.

SPscore(A)

```
// Input: a MSA A = { a1, a2, .., ak}

// Output: sop (SP score) of A

sop ← 0

for  j ← 1 to len

        for  i ← 1 to k do

                sop ← sop + w(ai[h], aj[h]) Figure
                // w(ai[h], aj[h]) = score from the substitute matrix if match or
                // mismatch. Otherwise w(aih, ajh) = gap penalty

return sop
```

CHAPTER 4

SOFTWARE DESIGN AND IMPLEMENTATION

Architecture

My software package **tspMSA** is an implementation of the progressive alignment of multiple protein sequences using a Traveling Salesman Problem approach. The package has five major modules; Sequence Reader, Distance Calculator, TSP Solver, MSA Builder, and MSA printer. Its architecture is depicted in Figure 4.1.

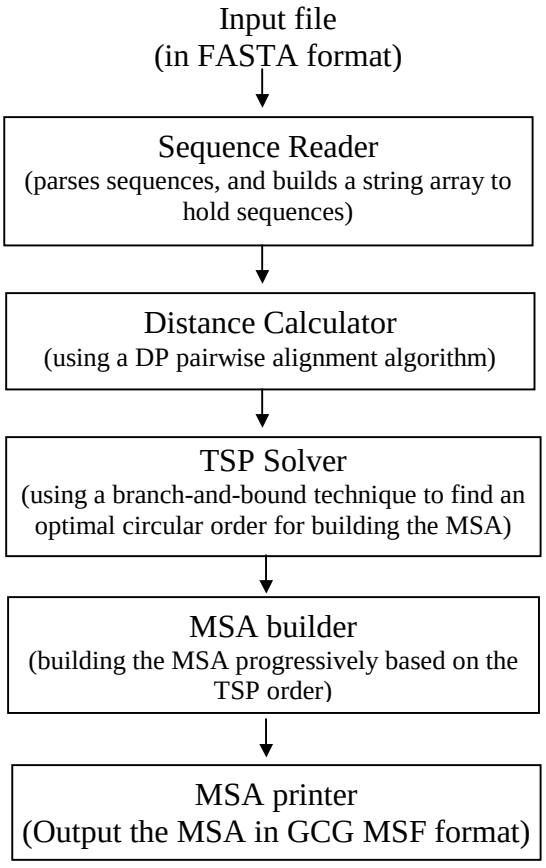


Figure 4.1. The architecture of **tspMSA**

Process Description

- **Input file**

The format for input files is a modified version of FASTA, which is very popular and compatible with most software used in biological research.

A sequence in FASTA format begins with a single line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater than (">") symbol in the first column. An example sequence in FASTA format is:

```
> SequenceName description here
ATGTCGTTACCGTCGTCGGGACCGACCATG
AGAGCGA
```

In order to allow for easy termination of the Sequence Reader, I add two '>' s at the end of each input file. The following file is an example input file that contains three sequences.

```
> Randseq1 first randomly generated seq
GGTGGTTACTAACCGTAAGAGATGATGTCGCCGTGGTCGCGTGGCGCCGCGGACCCAG
TGTACTTCTCTGAGTCGTTCTAGATCGACCAGTCTTCTAGCTTGCCCGTGAGGTATGG
AGCCGCATATTGCCCACAAT
> Randseq2 second randomly generated seq
GCGACGCGTCTCTACACCAGACGCTTCTGTTGAGGAAGAGTGCCTGAGTGCAGGTCCT
AGAACCCACTGGAACTTGAAGGGCGCGTCTCACTGGTCGTGAGAAGGCTCCGTCGATA
AAAGTCCATGCCAAGGACAT
> Randseq3 third randomly generated seq
GGCGAGTCTGAACTCACAAATATTGCACGAGAGTTTAGTGTATGTTCCTCTTAGGCTG
AACAATAGTTTAGTGAGCGGAAATGCAACCGCGAGGCGGTCCCCTGCGCTTGTAATGG
ACCTGTTGCCCGTCGGATAT
>>
```

Figure 4.2. A sample of a modified FASTA format file of 3 sequences

- **Sequence Reader**

The input file is parsed, and two string arrays are built from parsed strings. One string array, seqt[], is made of sequence titles, each of which comes from the first

4 letters of the sequence description. The other string array, seq[], contains protein sequences.

- **Distance Calculator**

First, an upper diagonal matrix of k(k-1)/2 pairwise distances between all pairs of sequences s_1, s_2, …,s_k is calculated using the dynamic programming pairwise sequence alignment algorithm (26). In the distance matrix, each element is the sum-of-pairs score of the corresponding pair of sequences.

In order to use TSP Solver, we need to build a new cost matrix with the function $cost(i,j) = SP_{max} - SP(i, j) + 1$. Since we are interested in an optimal circular order to build an MSA with the maximum SP-score, we need look for the longest with our TSP Solver. However, the TSP algorithm used finds an optimal shortest tour. This program is easily solved by subtracting each SP score in the distance matrix from a large number (here I use the maximum SP score in the distance matrix plus 1).

- **TSP Solver**

The Traveling Salesman Problem is NP hard. But it is very well studied and optimal solutions can be calculated within a few hours for up to 100 cities and in a few seconds for up to 30 cities. For real applications we seldom have more than 100 sequences to compare simultaneously. In **tspMSA** a branch-and-bound technique is used to solve the TSP. The algorithm and source code come from Drs. Donald L. Kreher and Douglas R. Stinson (17, pp.127-143).

- **MSA Builder**

Assuming the optimal circular order is $\{s_1, s_2, .., s_n\}$, a MSA is assembled progressively. First, do an optimal alignment of sequences s_1 and s_2 using the Needleman-

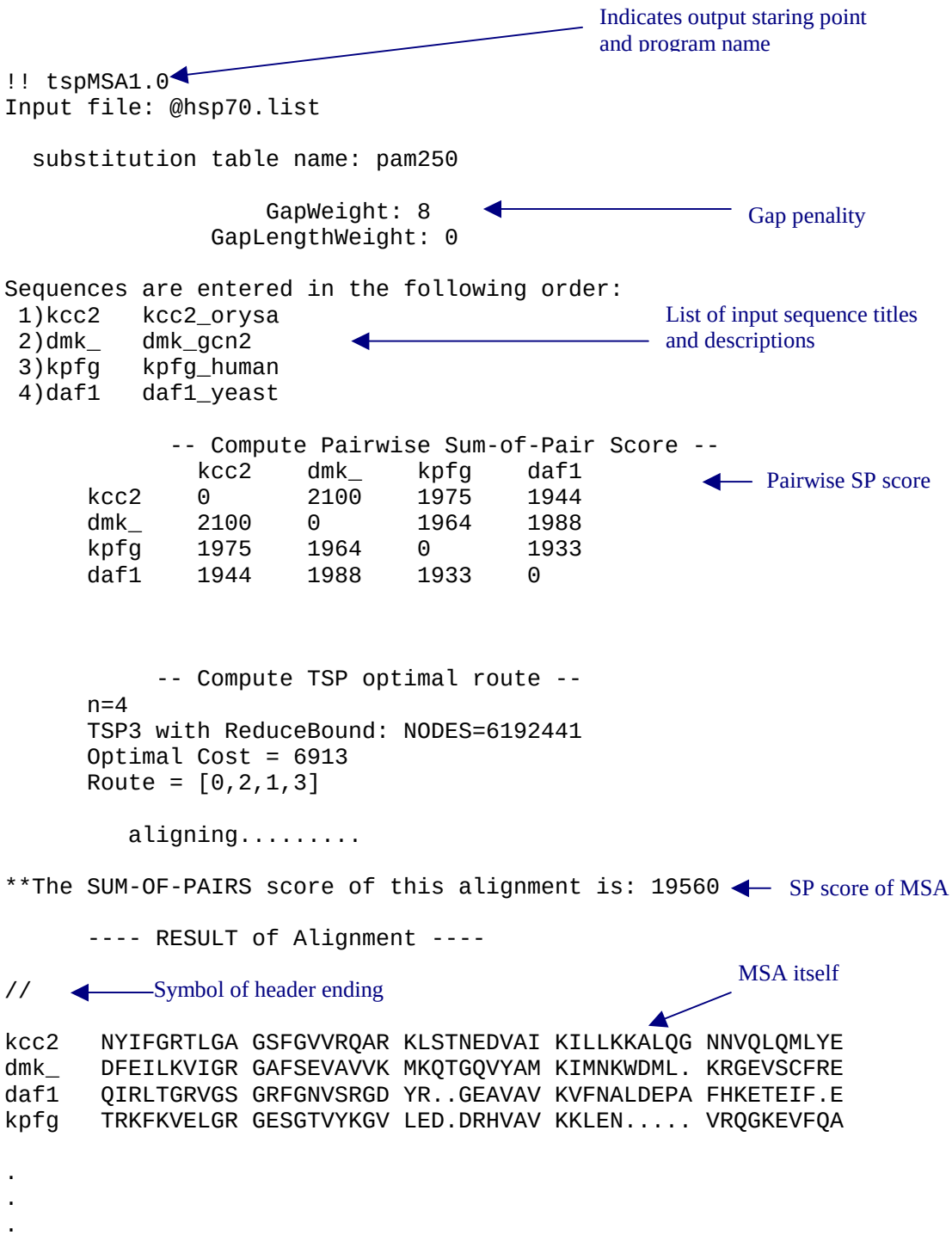
Wunsch algorithm. The sequences in this alignment with the gaps are called t_1 and t_2. Then, align the rest of the sequences one-by-one against the previously aligned sequences until all *n* sequences are in the MSA.

The SP score of the resulting MSA is calculated and used to evaluate the quality of the MSA.

- **MSA printer**

The output file is produced in a format similar to the MSF format. MSF is the multiple sequence alignment format of the GCG sequence analysis package. MSF is accepted by many bioinformatics software packages. Also, I include the SP score of the MSA in the output file generated by my program. These features will be helpful for comparing my output with that of others.

An example of my output file containing the SP score and the multiple sequence alignment is shown below.




```
!! tspMSA1.0
Input file: @hsp70.list

  substitution table name: pam250

                 GapWeight: 8
           GapLengthWeight: 0

Sequences are entered in the following order:
 1)kcc2   kcc2_orysa
 2)dmk_   dmk_gcn2
 3)kpfg   kpfg_human
 4)daf1   daf1_yeast

          -- Compute Pairwise Sum-of-Pair Score --
            kcc2    dmk_    kpfg    daf1
     kcc2    0       2100    1975    1944
     dmk_    2100    0       1964    1988
     kpfg    1975    1964    0       1933
     daf1    1944    1988    1933    0



          -- Compute TSP optimal route --
     n=4
     TSP3 with ReduceBound: NODES=6192441
     Optimal Cost = 6913
     Route = [0,2,1,3]

        aligning.........

**The SUM-OF-PAIRS score of this alignment is: 19560

     ---- RESULT of Alignment ----

//
kcc2   NYIFGRTLGA GSFGVVRQAR KLSTNEDVAI KILLKKALQG NNVQLQMLYE
dmk_   DFEILKVIGR GAFSEVAVVK MKQTGQVYAM KIMNKWDML. KRGEVSCFRE
daf1   QIRLTGRVGS GRFGNVSRGD YR..GEAVAV KVFNALDEPA FHKETEIF.E
kpfg   TRKFKVELGR GESGTVYKGV LED.DRHVAV KKLEN..... VRQGKEVFQA

.
.
.
```

Implementation

My program, **tspMSA**, described above is written in the C language and run under the UNIX operation system.

My implementation includes two free programs, "**ALIGN**" from D. F. Feng and R. F. Doolittle (4-6), and "TSP3" from D. L. Kreher and D. R. Stinson (17, pp.127-143), which are available for free downloading from:

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

http://www.math.mtu.edu/~kreher/cages.html

ALIGN has the following major functions (Table 4.1):

Name	Function	Comment
read_seq( )	Parse input file	Only recognizes files in the rarely used Old Atlas format
align( )	Pairwise sequence alignment	
bord( )	Build a distance matrix and compute the guide tree	Uses a special scoring function
mulalign( )	Output MSA	
main( )	Control data flow and assemble MSA	Adds sequences one-by-one to MSA by iteratively calling align( )

Table 4.1. Functions in **ALIGN**

Based on the ALIGN and TSP3, first I integrated TSP3 into ALIGN in order to achieve my goal, then I added new functions and improved some existing functions. My program, **tspMSA**, has the following major functions (Table 4.2):

Name	Function	Comment
read_seq( )	Parse input file, establish string arrays which hold sequences.	Re-implemented from **ALIGN** to read the most popular format, FASTA.
align( )	Do pairwise alignment using SP Score.	Modified from **ALIGN** .
distCalc( )	Calculate pairwise SP scores, build a distance matrix, build a cost matrix.	Re-implemented from **ALIGN** to compute the distance matrix using SP score function.
tspOrder( )	For a given cost matrix, compute the optimal route (circular order for building MSA).	Modified from **TSP3** . A timer function is added to terminate tspOrder() after a certain time.
RandomOrder()	Generate a randomized order of the given sequences.	Implemented to build the MSA from a random ordering of the given sequences.
sop( )	Compute the SP score of an MSA (also parse the MSF file and convert the aligned sequences to FASTA format).	Implemented to evaluate the quality of the MSA, to parse the MSF file and convert the aligned sequences in FASTA format.
printMSA( )	Output the MSA.	Improved from **ALIGN** to output the MSA as a GCG MSF format file.
main( )	Control data flow and generate the MSA.	Modified from **ALIGN** .

Table 4.2. Funtions in **tspMSA**

CHAPTER 5

EVALUATION OF THE PROGRAM

TEST DATA

In order to comprehensively evaluate a new alignment program, we need a large number of accurate reference alignments which can be used as test cases. It has been shown (23) that the performance of alignment programs depends on the number of sequences, the degree of similarity between the sequences, the lengths of the sequences, andthe existence of large insertions and N/C-terminal extensions.

Thompson *et al* have developed a benchmark alignment database called BAliBASE (Benchmark Alignment dataBASE) specifically for evaluating new alignment programs (36). It is available on the World Wide Web at http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE. The sequences included in this database are selected from alignments in either the FSSP (**F**old classification based on **S**tructure-**S**tructure alignment of **P**roteins) database or HOMSTRAD (**Hom**ologous **Str**ucture **A**lignment **d**atabase), or from manually constructed structural alignments taken from the literature. The alignments of sequences sharing the same three-dimensional fold have been validated to ensure the alignment of functional and other conserved residues. The VAST (**V**ector **A**lignment **S**earch **T**ool) Web server (22) is used to confirm that the sequences in each alignment are structural neighbors and can be structurally superimposed. Functional sites are identified using the PDBsum database (19) and the alignments are manually

verified and adjusted, in order to ensure that conserved residues are aligned as well as the secondary structure elements.

BAliBASE2.0 currently consists of 142 reference alignments, containing over 1000 sequences. Those high-quality alignments are organized into 8 reference sets that represent some of the most common problems currently encountered when aligning real families of proteins.

- **Reference set 1** contains alignments of equi-distant sequences, *i.e.*, the percent identity between two sequences is within a specified range.
- **Reference set 2** aligns up to three "orphan" sequences (less than 25% identical) from reference set 1 with a family of at least 15 closely related sequences.
- **Reference set 3** consists of up to 4 sub-groups, with less than 25% residue identity between sequences from different groups.
- **Reference set4** contains alignments of up to 20 sequences with N/C-terminal extensions (up to 400 residues),
- **Reference set 5** contains alignments of sequences with internal insertions (up to 100 residues).
- **Reference set 6** contains alignments of sequences with repeated fragments.

METHODOLOGY

To assess the the quality of an alignment, the sum-of-pairs score is calculated. The higher the SP score of a MSA, the better the quality of the MSA.

A program, **sop**, was implemented to calculate the SP score of a MSA. Also, the **sop** program parses the input reference alignment file to extract the original protein sequences and saves these sequences into an output file of FASTA format. Later on, the

resulting file can serve as the input for my program, **tspMSA**, or the programs that I want to compare with, such as **PILEUP** and **CLUSTALW**.

RESULTS AND DISCUSSION

1) Performance of TSP Solver

An efficient TSP Solver, which can find the optimal circular order in reasonable time and computer memory, is critically important for my program, **tspMSA**.

The TSP3 program from Drs. Donald L. Kreher and Douglas R. Stinson is based on a branch-and-bound TSP algorithm . To evaluate it, I generated random instances of the TSP problem on n cities. The distances between any pair of cities were randomly chosen integers between 0 and MAX. The time of computation and the number of nodes in the state space trees are shown in Table 5.1.

cities(n)	MAX	nodes	Time (s)
10	200	74	0.01
15	200	1082	0.19u
20	200	31203	2.29u
25	200	105975	63.14
30	200	1167178	1072.68
10	***2000***	***108***	***0.01***
15	***2000***	***3450***	***0.37***
20	***2000***	***50028***	***14.45***
25	***2000***	***1109759***	***544.1***
30	***2000***	***5766607***	***3890.56***
10	20000	280	0.02
15	20000	4771	0.63
20	20000	205071	55.91
25	20000	1361174	699.46
30	20000	6446124	6000

Table 5.1 The number of nodes in the state space trees and the running time to find the optimal route on random instances of the TSP problem.

The results show that TSP Solver can find an optimal solution on an instance of 30 cities with MAX of 2000 in 3890.56 seconds and the number of nodes in the state space trees is 5766607. If we assume a node takes 1 byte, we need 5.8 mega bytes, which is not a problem for most computers. I also tested **tspMSA** on a real problem with 162 protein sequences and MAX of 4000 for 100 minutes. The number of nodes was 8916700 when the program was terminated, not yet having reported an optimal solution.

In fact, I found that TSP Solver, **TSP3**, never reported an optimal solution within 100 minutes on instances of more than 50 cities. Some researchers report that optimal solutions can be calculated within a few hours for up to 1000 cities and in a few seconds for up to 100 cities (29). However their code was not available for experimentation

2) Execution times and quality of solutions

For large scale problems, it is unrealistic to ask for optimal solutions with limited computer memory and time. However, there are heuristics for large scale problems that calculate near optimal solutions that are within 1% to 2% of the optimum (9, 29).

To evaluate the performance of TSP Solver on large scale data sets, I explored the relationship of the execution time and the quality of the solution. The results are summarized in the Table 5.2.

The results showed that, after running for 20 minutes, TSP Solver found a near optimal solution for the instances of up to 300 cities. For instances of MAX =20000, the costs obtained by running either 20 or 100 minutes were very close or exactly same, while the number of nodes increased by 150% up to 660%. For other instances, the cost obtained by running 20 minutes decreased at most 4.5% compared with that obtained by running 100 minutes, while the number of nodes increased by 40% up to 450%.

Based on these results, I set the default execution time of **tspMSA** to 20 minutes.

MAX	n	Exec Time (min.)	Cost	Nodes
100	50	5	217	410375
	50	10	206	777771
	50	***20***	***194***	***1454733***
	50	100	194	2633672
	100	5	284	371897
	100	10	283	683685
	100	***20***	***283***	***1175990***
	100	100	278	5326311
	300	5	*infinity*	15690
	300	10	267	63122
	300	***20***	***262***	***205490***
	300	100	258	1131275
2000	50	5	4806	396718
	50	10	4646	673568
	50	***20***	***4646***	***917608***
	50	100	4428	4350398
	100	5	5356	162302
	100	10	5356	388742
	100	***20***	***5257***	***722723***
	100	100	5129	2737193
	300	5	*infinity*	15690
	300	10	7057	66729
	300	***20***	***7003***	***241179***
	300	100	7003	1080409
20000	50	5	54027	490154
	50	10	54027	948164
	50	***20***	***54027***	***1420971***
	50	100	54006	5710316
	100	5	60288	258405
	100	10	60168	487423
	100	***20***	***60168***	***874483***
	100	100	60164	3498741
	300	5	*infinity*	15690
	300	10	73203	74273
	300	***20***	***73203***	***254773***
	300	100	73203	1690807

Table 5.2 Execution time, cost, and nodes

The cost *infinity* indicates that no TSP tour was found; This only happened when the time was limited to 5 minutes.

3) Comparison with SP scores of MSAs generated by other programs

To evaluate my **tspMSA** on multiple sequence alignments, I selected some reference data sets from the BALIBASE (version 2.0) and fed them to my **tspMSA** and **CLUSTALW**1.8. These reference data sets were multiple sequence alignments generated by **PILEUP**.

The SP scores and execution times are shown in Table 5.3. Using the reference data sets, the SP scores obtained by my **tspMSA** were higher than those obtained by either **PILEUP** or **CLUSTALW**, except for only one case, ref5: kinase53. We should note that some sequences of that exceptional case have long internal insertions, which might cause the SP scores to be decreased. More experiments were carried out on this data set to explore the order of sequences to be aligned and the SP score (see later in this section, and in section 6).

The versions of **PILEUP** availabe at University of Georgia is installed at the Research Computing Resource (RCR). However, I could not test the execution time because all tasks on RCR are submitted to a task queue and no value on the execution time is returned. Therefore in Table 5.3 the execution times of **PILEUP** are not included.

data set	feature (n, s--l)[1]	SP score on MSAs and Execution Time				
		PILEUP	**CLUSTALW**		**tspMSA**	
Ref1: 1ubi	4, 76--94	2376	3158	(0.16s)	3205	(0.09s)
1pfc	5, 116--108	9823	9955	(0.47s)	10002	(0.16s)
1fmb	5, 98--104	6040	6044	(0.26s)	6061	(0.10s)
1pii	4, 246--252	11969	12081	(1.63s)	12274	(0.41s)
1thm	5, 269--279	15573	15594	(1.82s)	15630	(0.44s)
gal4	5, 335--395	20983	23683	(4.42s)	23777	(1.01s)
1bgl	4, 938--991	50835	51024	(22.75s)	51287	(3.35s)
1gpb	5, 796--828	82666	82351	(22.58s)	82860	(4.34s)
Ref2: 1csy	19, 75--99	112407	112507	(1.80s)	115971	(172.81s)
kinase2	18, 257--287	344437	344283	(22.82s)	353901	(703.24s)
1cpt	20, 347--389	330650	333998	(24.79s)	334700	(21.54s)
Ref3: 1wit	19, 85--102	115898	119723	(2.12s)	121146	(1193.71s)
kinase3	23, 243--312	518529	523320	(30.69s)	537566	(303.05s)
1ped	21, 324--388	592142	595400	(34.65s)	607333	(1200s^{+})
Ref4: kinase41	7, 289--481	43234	45446	(10.93s)	48939	(5.43s)
kinase42	18, 257--631	145749	137424	(76.23s)	184349	(1193.71s)
Ref5: kinase51	5, 285--358	21599	22285	(4.75s)	22429	(2.39s)
kinase53	19, 256--384	331007	315367	(30.53s)	320587*	(1159.77s)
Ref6: deah	22, 578--2176	1145541	1179323	(105.93s)	1237392	(1200s^{+})
ion	53, 496--2039	2352858	2453544	(134.70s)	2508615	(1200s^{+})

Table 5.3 The SP scores and execution times obtained from different alignment programs

[1] 'n', 's', and 'l' mean the number of sequences, the shortest sequence, and the longest sequence in the data set;
* The SP score is lower than that obtained by **PILEUP**.
$^{+}$No optimal solution is returned.
Ref 1: equi-distant sequences with various levels of conservation;
Ref 2: families aligned with a highly divergent "orphan" sequence;
Ref 3: subgroups with < 25% residue identity between groups;
Ref 4: sequences with N/C-terminal extensions;
Ref 5: sequences with internal insertions (up to 100 residues);
Ref 6: sequences with repeated fragments.

4) Execution times and SP scores

For some reference data sets, I did not obtain an optimal circular order when **tspMSA** was executed up to 20 minutes. To evaluate the quality of MSAs built from near optimal circular orders, I ran **tspMSA** on the following data sets for certain times. The SP scores of these MSAs were calculated and listed in Table 5.4.

Time(s) / data set	120	300	600	**1200**	6000
Ref3:1ped	603648	607333	607333	**607333**	608008
Ref4:kinase42	187054	189141	189141	**194349**	200114
Ref5:kinase53	332338	329297	320587	**320587**	320587
Ref2:1cpt	335678	334700	334700	**334700**	334700
Ref6:ion	2508615	2508615	2508615	**2508615**	2508615
Ref6:deah	1233887	1237185	1237392	**1237392**	1230908
big.seq	8091586	8094191	8094191	**8094191**	8099394

Table 5.4. The SP cores of MSAs obtained from different execution times

The results show that the relative difference of SP scores obtained from 20 minutes and 100 minutes is within a range of -0.5% to 2.9%. This indicates that the default execution time (20 minutes) is reasonable. The negative values indicate that a better TSP tour does not always promise a better alignment.

5) Starting point and direction of the circular order and SP score of the MSA

Basically, the TSP circular order is a heuristic method of determining a linear order in which to apply progressive alignment to the given sequences. From an optimal circular order, we only get a near optimal MSA.

To investigate the effect of assembling order on the SP score, I tested to see if we could increase the SP score by starting at different points in the circular order and going different directions.

First, I selected the Ref2:kinase2 as a normal test case. The optimal circular oder for this data set is [**0**,1,17,16,2,3,4,6,8,9,11,12,15,5,7,13,14,10]. The results are shown in Table 5.5.

starting point	SP score (Forward)	SP score (Backward)
0	**353901+**	353772+
1	352383+	351338+
17	349806+	345276+
16	352713+	347244+
2	**354526*+**	343670
3	350964+	343786
4	350077+	343913
6	348867+	348273+
8	351661+	347753+
9	348754+	348398+
11	345526+	351753+
12	347820+	353480+
15	345785+	348929+
5	350113+	353745+
7	347738+	350068+
13	345460+	350216+
14	346159+	345670+
10	352734+	346129+

Table 5.5. Starting points, directions of the circular order and SP scores of the MSAs for Ref2:kinase2

* The SP score of this MSA is higher than that of the MSA built from an optimal TSP circular order;

+ The SP score of this MSA is higher than that of the MSA generated by either **PILEUP** or **CLUSTALW**.

The results show that, when the ref2:kinase2 was tested, among the all possible combination of starting points and directions , only one starting point (indicated by the *) with the forward direction led to a higher SP score than the SP score of the MSA starting from the first sequence in the optimal circular order. All MSAs built from TSP orders with the forward direction, no matter where they start, have a higher SP score than that of MSAs generated by either **PILEUP** or **CLUSTALW.** The circular orders in backward direction led to lower SP scores than that from the optimal TSP order, no matter starting from which sequence. In some cases, the SP scores were lower than that of MSA generated by either **PILEUP** or **CLUSTALW**.

I also did same experiment on the test case Ref5:kinase53 (which my **tspMSA** did not beat **PILUP**). The result is shown in Table 5.6. Please notices that: the first 5 sequences in Ref5:kinase53 have long internal insertions at different points. The optimal circular order is [**0**,2,1,4,5,6,3,7,12,16,11,10,17,13,14,18,15,9,8].

The result is interesting. With the forward direction, when two or more sequences with long insertions were assembled first, we certainly got a bad SP score (Table 5.6 column 2, rows 1, 2, 3, and 19). When normal sequences were assembled first, the SP scores became significantly higher (Table 5.6 column 2, rows 5 to 18, except for row 16). When I simply put all sequences with long insertions to the end of the input file, and the SP score was 338152, which is significantly higher than that of MSA generated by either **PILEUP** or **CLUSTALW**.

The circular orders in backward direction led to MSAs with higher SP scores, which were not only higher than that of MSA from optimal TSP order, but also higher than that of MSA generated by either **PILEUP** or **CLUSTALW** except for 1 case (Table 5.8 row 10).

starting point	SP score (Forward)	SP score (Backward)
0	**320587**	**339409*+**
2	328447*	335280*+
1	328512*	336359*+
4	333175*+	332506*+
5	334498*+	338266*+
6	336012*+	334328*+
3	334659*+	335849*+
7	340500*+	335529*+
12	340590*+	333311*+
16	339501*+	*328601**
11	338732*+	335556*+
10	340863*+	335648*+
17	337537*+	339418*+
13	334937*+	339717*+
14	334937*+	336997*+
18	330978*	332875*+
15	336679*+	338785*+
9	332193*+	338889*+
8	314934	339202*+

Table 5.6. Starting points, directions of the circular order and SP scores of the MSAs for Ref3:kinase53

* The SP score of this MSA is higher than that of the MSA built from an optimal TSP circular order;

+ The SP score of this MSA is higher than that of the MSA generated by either **PILEUP** or **CLUSTALW**.

One possible explanation is that: when MSA was built from the optimal TSP order, we, by chance, started from a sequence with long insertions (sequence 0), and subsequently we added more sequences with long insertions (sequences 2, 1, and 4). This led to more gaps in overall MSA, and thus we get a lower SP score. However, when using the reversed optimal TSP order, although we started with sequence 0, we added sequences with long insertions after we added normal sequences, therefore we got a higher SP score (Table 5.6 column 3, row 1). The SP score will tend to lower if we align more sequences with long insertions before we do other normal sequences (Table 5.6 column 3, rows 2, 3 and 4). This result suggests that, when we have sequences with long insertions, in order to get a higher SP score, we should avoid to build MSA from an order that has more than one sequences with long insertions at the beginning.

The progressive alignment follows the rule, "once a gap, always a gap". If we start alignment with sequences containing long insertions, the gaps derived from insertions will propagate in the susequent process. If we start alignment with normal sequences and add sequences with long insertions at the end of alignment, we will get fewer gaps and a higher SP score.

For the future improvement, we can design a step to select the maximum SP score of MSAs built from all possible circular orders, because after finding the optimal circular order (the most time-consuming step), to assemble the MSA only takes about 3 seconds.

6) Random order progressive alignments

To explore whether a randomly generated linear order could lead to a better SP score than the optimal circular order generated by TSP algorithm does, I implemented a

random sequence generating function and tested 500 randomly generated orders aligning the Ref2:kinase2 data set. I found only 1 case out of 500 that gave a higher SP score than the one derived from the optimal circular order generated by TSP.

The running time for building an MSA from an optimal circular order generated by TSP is 703.24 seconds, and the running time for building an MSA from a randomly generated circular order is much shorter (3.80 seconds). However, within the same time used by the TSP strategy, it is unlikely that a better score will be found using the random strategy.

7) Large data set

The last experiment was to test my program with a large data set. I selected 162 protein sequences from BALIBASE and made a test file, big.seq in the modified FASTA format.

The result was very encouraging. Using my program, **tspMSA**, the SP score at 380 seconds was 8094191, at 20 minutes was 8094191, and at 100 minutes was increased to 8099394. When I input the big.seq to **CLUSTALW**, the running time was 379.53 seconds and the SP score was much lower, 930301. This result suggests that my program may perform much better when dealing with a set containing many sequences.

CHAPTER 6

CONCLUSIONS

The traveling salesman problem algorithm shows promise for the ordering of multiple protein sequences using pairwise SP scores. Except for one test case, the SP scores obtained from progressive alignments based on the TSP algorithm were significantly better than the scores obtained from two popular progressive alignment programs, **PILEUP** and **CLUSTALW**.

When aligning more than 20 sequences with lengths over 200, my program, **tspMSA**, may need a longer execution time to find the optimal circular order to build an MSA. Usually, by running the TSP Solver for 20 minutes, the MSA obtained by my program has a higher SP score than that obtained from either **PILEUP** or **CLUSTALW**.

For a test data set which contains sequences with long internal insertions, a linear order starting from two or more sequences containing long insertions can lead to a bad SP score.

Within the same time used by the TSP strategy, it is unlikely that a better score will be found using a randomly generated order for the progressive alignment.

On a large data set containing 162 sequences, my program performs much better than **CLUSTALW**.

REFERENCES

1. **Baldi, P., Y. Chauvin, T. Hunkapiller, and M. A. McClure.** 1994. Hidden Markov models of biological primary sequence information. Proc. Natl. Acad. Sci. USA **91:**1059-63.
2. **Dayhoff, M., R. Schwartz, and B. Orcutt.** 1978. A model of evolutionary change in proteins, p. 345-352. *In* M. Dayhoff (ed.), Atlas of Protein Sequence and Structure, vol. 5. Natl Biomedical Research Foundation, Washington, DC.
3. **Eddy, S. R.** 1995. Multiple alignment using hidden Markov models. Proc. Int. Conf. Intell. Syst. Mol. Biol. **3:**114-120.
4. **Feng, D., and R. Doolittle.** 1990. Progressive alignment and phylogenetic tree construction of protein sequences. Methods Enzymol. **183:**375-387.
5. **Feng, D., and R. Doolittle.** 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J. Mol. Evol. **25:**351-360.
6. **Feng, D. F., and R. F. Doolittle.** 1996. Progressive alignment of amino acid sequences and construction of phylogenetic trees from them. Methods Enzymol **266:**368-82.
7. **Gonnet, G., C. Korostensky, and S. Benner.** 1999. Evaluation measures of multiple sequence alignments. J. Comput. Biol. **7:**261-276.
8. **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.

9. **Grotschel, M., and O. Holland.** 1991. Solution of large-scale symmetric traveling salesman problems. Math. Programming **51:**141 - 202.
10. **Gupta, S., J. Kececioglu, and A. Schaffer.** 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.
11. **Henikoff, S., and J. Henikoff.** 1992. Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA **89:**10915-10919.
12. **Higgins, D., and P. Sharp.** 1989. Fast and sensitive multiple sequence alignments on a microcomputer. Computer Applications in the Biosciences **5:**151-153.
13. **Jiang, T., and L. Wang.** 1994. On the complexity of multiple sequence alignment. J. Comp. Biol. **1:**337-348.
14. **Kimura, M.** 1983. The Neutral Theory of Molecular Evolution, Cambridge University Press, New York.
15. **Korostensky, C., and G. Gonnet.** 1999. Near Optimal Multiple Sequence Alignments Using a Travelling Salesman Problem Approach. String Processing and Information Retrieval Symposium & International Workshop on Groupware**:**105-114.
16. **Korostensky, C., and G. Gonnet.** 2000. Using traveling salesman problem algorithms for evolutionary tree construction. Bioinformatics **16:**619-627.
17. **Kreher, D., and D. Stinson.** 1999. Combinatorial Algorithms : Generation, Enumeration, and Search. CRC Press, Boca Raton.
18. **Krogh, A., I. Mian, and D. Haussler.** 1994. A hidden Markov model that finds genes in E. coli DNA. Nucleic Acids Res. **22:**4768-4778.

19. **Laskowski, R., E. Hutchinson, A. Michie, A. Wallace, M. Jones, and J. Thornton.** 1997. PDBsum: a Web-based database of summaries and analyses of all PDB structures. Trends Biochem. Sci. **22:**488-490.

20. **Lawler, E., J. Lenstra, A. Rinnooy Kan, and D. Shmoys.** 1985. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester.

21. **Lipman, D., S. Altschul, and J. Kececioglu.** 1989. A Tool for Multiple Sequence Alignment. Proc. Natl. Acad. Sci. USA **86:**4412-4415.

22. **Madej, T., J. Gibrat, and S. Bryant.** 1995. Threading a database of protein cores. Proteins **23:**356-369.

23. **McClure, M., T. Vasi, and W. Fitch.** 1994. Comparative analysis of multiple protein-sequence alignment methods. Mol. Biol. Evol. **11:**571-592.

24. **Mount, D.** 2001. Multiple sequence alignment, Bioinformatics: Sequence and Genome Analysis. Cold Spring Harbor Press, Cold Spring Harbor.

25. **Murata, M., J. Richardson, and J. Sussman.** 1985. Simultaneous comparison of three protein sequences. Proc. Natl. Acad. Sci. USA **82:**3073-3077.

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

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

28. **Notredame, C., L. Holm, and D. Higgins.** 1998. COFFEE: an objective function for multiple sequence alignments. Bioinformatics **14:**407-422.

29. **Padberg, M., and G. Rinaldi.** 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM (The Society for Industrial and Applied Mathematics) Review **33:**60-100.

30. **Saitou, N., and M. Nei.** 1987. Neighbour Joining. Mol. Biol. Evol. **4:**406-425.

31. **Schwartz, R., and M. Dayhoff.** 1978. Matrices for detecting distant relationships, pp.353-358. *In* M. Dayhoff (ed.), Atlas of protein sequence and structure, vol. 5. Natl Biomedical Research Foundation, Washington DC.

32. **Sneath, P., and R. Sokal.** 1973. Numerical Taxonomy: The principles and practice of numerical classification. W.H. Freeman and Company, San Francisco.

33. **Stoye, J.** 1998. Multiple sequence alignment with the divide-and-conquer method. Gene **211**(2)**:**45-56.

34. **Thompson, J., T. Gibson, F. Plewniak, F. Jeanmougin, and D. Higgins.** 1997. The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. Nucleic Acids Res. **25:**4876-4882.

35. **Thompson, J., D. Higgins, and T. Gibson.** 1994. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. Nucleic Acids Res. **22:**4673-4680.

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

37. **Tönges, U., S. Perrey, J. Stoye, and A. Dress.** 1996. A General Method for Fast Multiple Sequence Alignment. Gene **172:**33-41.