DYNAMIC PROGRAMMING AND ITS APPLICATION TO PAIRWISE BIOLOGICAL SEQUENCE ALIGNMENT

By

YANWEN GUO

Bachelor of Science

Wuhan University

Wuhan, China

1987

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2000

DYNAMIC PROGRAMMING AND ITS

APPLICATION TO PAIRWISE

BIOLOGICAL SEQUENCE

ALIGNMENT

Thesis Approved:

[illegible]

Thesis Advisor

J Chandler

[illegible]

Wayne B. Powell

Dean of the Graduate Collage

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major adviser, Dr. George E. Hedrick for his guidance, encouragement, patience, advice and friendship throughout my graduate program. I would like to thank Dr. John P. Chandler, Dr. Robert L. Burnap for serving my graduate committee, and for their helpful suggestions throughout the study.

I deeply appreciate Dr. Robert L. Matts, my supervisor in the department of biochemistry and molecular biology, for his kindness and understanding. I took courses in computer science in part time while I was a full-time employee working in Dr. Matts' lab. I could not finish my graduate study in such a short time without his generous permission.

My eternal gratitude goes to my wonderful wife Ms. Hui Zeng, who gave me the most substantial support and encouragement. Her appreciation for this project, and her love and affection for me gave me the strength to endure difficult times, and the privilege to work on an advanced degree and at the meantime to have a perfect family.

My whole-hearted appreciation also extends to my dear parents, brothers and sisters, especially to my great mother, Ms. Zhonghui Long who gave me so much love and support during my entire life.

TABLE OF CONTENTS

Chapter Page

I INTRODUCTION 1

II BIOINFORMATICS AND SEQUENCE ALIGNMENT 3
- 2.1 Definition of Bioinformation 3
- 2.2 Biological Preliminaries 4
- 2.3 Definition of Sequence Alignment 7
- 2.4 The Application of Sequence Alignment 8

III LITERATURE REVIEW OF ALGORITHMS FOR BIOLOGICAL SEQUENCE ALIGNMENT 9
- 3.1 The Importance of Sequence Alignment in Molecular Biology 9
- 3.2 The Scoring Scheme 10
  - 3.2.1 Identity Scoring 10
  - 3.2.2 Genetics Code Scoring 11
  - 3.2.3 Chemical Similarity Scoring 11
  - 3.2.4 Observed Substitutions 12
    - 3.2.4.1 The Dayhoff Mutation Data Matrix 13
    - 3.2.4.2 PET91 – An Updated Dayhoff Matrix 14
    - 3.2.4.3 BLOSUM – Matrix From Ungapped Alignment 15
  - 3.2.5 The Selection of a Suitable Scoring Matrix 15
- 3.3 Gap Penalty 17
  - 3.3.1 Motivation 18
  - 3.3.2 Gap Penalties Types 20
  - 3.3.3 Constant Gap Weight Model 20
  - 3.3.4 Affine Gap Penalty 21
- 3.4 Pairwise Sequence Alignment 23
  - 3.4.1 Alignment Without Gaps – Fixed Length Segments 24
    - 3.4.1.1 Correlation Methods 25
    - 3.4.1.2 Variable Length Segments 26
  - 3.4.2 Alignment with Gaps 26
    - 3.4.2.1 Finding the Best Alignment with Dynamic Programming 27
    - 3.4.2.2 Alternative Weight for Gaps 30
  - 3.4.3 Identification of Local Similarities 30
    - 3.4.3.1 Finding Subsequence - Best Local Alignment 31
- 3.5 Multiple Sequences Alignment 31
  - 3.5.1 Introduction 31

Chapter Page

3.5.2 Multiple Alignment Algorithms 32

IV DYNAMIC PROGRAMMING AND PAIRWISE BIOLOGICAL SEQUENCE ALIGNMENT. 35
4.1 Introduction 35
4.2 Needleman-Wunsch Algorithm and Global Sequence Alignment 36
4.3 Smith-Waterman Algorithm and Local Sequence Alignment 39
4.4 Repeated Matches Algorithm 42
4.5 Overlap Matches Algorithm 44
4.6 Dynamic Programming and Linear Space Algorithm 46

V PROGRAM DESIGN AND IMPLEMENTATION 49
5.1 System Requirements 49
5.2 Algorithms Used for Programming 49
5.3 Input and Output 50
5.3.1 The Format of Input 50
5.3.2 The Sample of Output 50
5.4 Implementation 53
5.4.1 Implementation of Global Alignment Algorithm 55
5.4.1.1 Initialization 55
5.4.1.2 Filling in the Matrix 55
5.4.2 Implementation of Local Alignment Algorithm 56
5.4.2.1 Initialization 56
5.4.2.2 Filling in the Matrix 56
5.4.3 Implementation of Repeated Matches Algorithm 57
5.4.3.1 Initialization 57
5.4.3.2 Filling in the Matrix 57
5.4.4 Implementation of Overlap Matches Algorithm 58
5.4.4.1 Initialization 58
5.4.4.2 Filling in the Matrix 59
5.4.4.2.1 Finding Maximal Score on Left-hand and Upper Borders 59
5.4.4.2.2 Finding Maximal Score on Right-hand and Bottom Borders 59

VI SUMMARY, CONCLUSIONS AND FUTURE WORK 60

APPENDIXES 63
APPENDIX A - GLOSSARY 63
APPENDIX B - TABLE OF ACRONYMS AND ABBREVIATIONS 66

REFERENCES 67

LIST OF TABLES

Table		Page
Table 2.1	DNA nucleotides	5
Table 2.2	The twenty amino acids	6
Table 3.2.5	BLOSUM 50 matrix	17
Table 4.2.1	Initial setup.	37
Table 4.2.2	Half way through the second step	38
Table 4.2.3	Trace the alignment	39
Table 4.3.1	An example of Smith-Waterman algorithm	41
Table 5.1	System requirement	49
Table 5.2	Algorithms used in program	50
Table 5.4.1	Description of main classes	54
Table 6.1	Performance of the algorithms used in the program	61

LIST OF FIGURES

Figure   Page

Figure 2.1 Scope of Bioinformatics 3

Figure 2.2 Translation of a DNA sequence into amino acid sequence 6

Figure 2.3 An example of sequence alignment 8

Figure 3.2.3 Classification of different amino acids 12

Figure 3.3 Alignment with gaps 18

Figure 3.4.2.1 The difference between global and local alignments 28

Figure 4.5.1 Difference types of overlap matches 45

Figure 4.6 Linear space alignment algorithm 48

CHAPTER I

INTRODUCTION

Bioinformatics, also known as computational biology, is a new explosively expanding research area that uses computational techniques to analyze biological data obtained from experimentation.

The most important aspect in bioinformatics at the molecular level is sequence alignment providing an explicit mapping between two or more biological sequences such as DNA and proteins. Analyzing the similarities and differences at the level of individual bases or amino acids by using sequence alignment, this allows us to predict and to find the structural, functional, and evolutionary relationships among the sequences under study.

There are two basic methods for sequence alignment: pairwise sequence alignment and multiple sequence alignment. And there are many algorithms used for these two sequence analysis methods. We may find that different algorithms may generate different results with different efficiency, sensitivity and selectively, so finding a good algorithm that can produce an optimal result is very important for sequence alignment program.

The object of this study is to analyze and to compare several dynamic programming algorithms for pairwise alignment. Such algorithms are the basis of most methods currently used to identify distantly related proteins by analyzing sequence similarity. A newly designed program implements and demonstrates how the several algorithms work to together and to produce the results.

In this thesis, chapter II gives a brief introduction to bioinformatics and the concept of sequence alignment. Chapter III is a literature review of algorithms for biological sequence alignment, focusing on the pairwise and multiple sequence alignment and the selection of scoring scheme. Chapter IV introduces the details of dynamic programming algorithms and their application to pairwise biological alignment. Chapter V is the detailed information of program design and implementation. Finally, the summary, conclusions and future work recommendations are given in Chapter VI. Also, a glossary, a table of acronyms and abbreviations are appended at the end of the thesis.

CHAPTER II

BIOINFORMATICS AND SEQUENCE ALIGNMENT

2.1 Definition of Bioinformatics

Bioinformatics, sometimes used interchangeably with the term computational biology, is the science of developing computer databases and algorithms for the purpose of accelerating and enhancing biological research. The most notable use of bioinformatics is in the Human Genome Project (Cooper 1994, Hudson 1995 and Schuler 1996), the effort to identify the 80,000 genes in human DNA. It brings together systematic biological data (e.g. genomes) with the analytic theory and practical tools of computer science and mathematics. The Figure 2.1 shows the scope of bioinformatics:

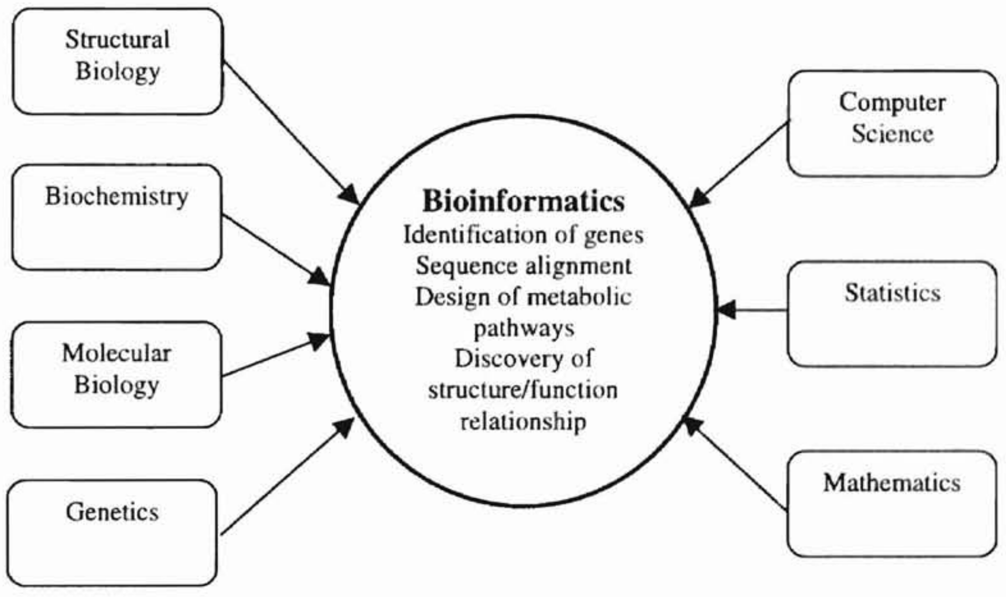

Figure 2.1 Scope of Bioinformatics (from Lee, 1999)

It is possible to use computer techniques to solve a large variety of biologically motivated problems primarily involving strings or sequences. For example:

- Comparing two or more strings for similarities;
- Storing, retrieving and comparing DNA strings;
- Searching databases for related strings and substrings;
- Exploring frequently occurring patterns of nucleotides;
- Finding informative elements in protein and DNA sequences;
- Reconstructing long sequences of DNA from overlapping string fragments;
- Determining physical and genetic maps from probe data under various experiments;

Many of these research problems show the functionality or structure of a protein or DNA without experimentation and without physically constructing the molecular sequences. The basic idea is that similar sequences produce similar proteins. In order to predict the characteristics of a protein using only its sequence data, we use the structure/function information on known proteins with similar sequences available from a database. This thesis explores algorithms for biological sequence alignment, especially dynamic programming algorithms and their application to pairwise biological sequence alignment.

2.2 Biological Preliminaries

An overview of molecular biology can be found in any modern textbook on biology, biochemistry or molecular biology (Stryer 1988, Alberts *et al* 1989). Goad (1986) gave a short review of computational methods in biological sequence analysis and recently several books summarizing problems and methods have been published (Doolittle 1986, Heijne 1987 and Lesk 1988).

DNA (deoxyribonucleic acid) and proteins are biological macromolecules built as long linear chains of chemical components. In the case of DNA, these components are the so-called nucleotides, of which there are four different ones, each denoted by one of the letters A, C, G, T (Table 2.1).

Adenine	Guanine	Cytosine	Thymine
A	G	C	T

Table 2.1 DNA nucleotides

Proteins, however, are made up of 20 different amino acids (or "residues") which are denoted by 20 different letters of the alphabet. The table 2.2 shows the names of the different amino acids and their single-letter code and three-letter code.

Name	**Three-Letter Code**	**One-Letter Code**
Alanine	Ala	A
Cysteine	Cys	C
Aspartic Acid	Asp	D
Glutamine Acid	Glu	E
Phenylalanine	Phe	F
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Lysine	Lys	K
Leucine	Leu	L

Methionine	Met	M
Asparagine	Asn	N
Proline	Pro	P
Glutamine	Gln	Q
Arginine	Arg	R
Serine	Ser	S
Threonine	Thr	T
Valine	Val	V
Tryptophan	Trp	W
Tyrosine	Tyr	Y

Table 2.2 The twenty amino acids

DNA plays a fundamental role in the processes of life in two respects. First it contains the templates for the synthesis of proteins, which are essential molecules for any organism. DNA can be transcribed into RNA, then RNA is translated into amino acid, this means that a linear string of DNA is translated into a corresponding linear string of amino acids; i.e., a protein. Figure 2.2 gives an example.

```
DNA sequence:            ... GAA CTA CAC ACG TGT AAC ...
Amino Acid Sequence      ...  E   L   H   T   C   N  ...
```

Figure 2.2 Translation of a DNA sequence into amino acid sequence

The amino acid sequence of a protein, also called its primary structure, is only one of many levels at which it can be examined. To fulfill its natural role a protein assumes a certain three dimensional structure, its tertiary structure. The term "secondary structure" refers to the local folding of the amino acid chain into small regular elements. The major classes of secondary structure are called β-sheets and α-helices. The three dimensional (tertiary) structure of a protein is usually built up of elements of α- and/or β- structure together with loop regions between them. It is the three dimensional folding of the chain which determines the biological function of a protein.

The second role of DNA is as a medium to transmit information (namely the building plans for proteins) from generation to generation. Watson and Crick in 1953 found the double helical structure of DNA. The linear chain does not really occur on its own but is paired to a complementary strand. The complementary stems from the ability of the nucleotides to establish specific pairs (A-T and G-C). The pair of complementary strands then forms the famous double helix. Each strand therefore carries the entire genetic information and the biochemical machinery and guarantees that the information can be copied over and over again even when the "original" molecule has long since vanished.

During this process of copying, changes (known as mutations) are introduced into the DNA sequence. The kinds of mutations that are important to sequence comparison are base changes, insertions of nucleotides into the chain and deletions from the chain. Those changes do not affect the translated protein when the changes do not alter protein's tertiary structure, but sometimes such alteration is possible. Based on this theory, the sequence alignment or comparison is very important in predicting the structure and function relationship between DNA and protein molecules.

2.3 Definition of Sequence Alignment

Sequence alignment, also called string matching or inexact string matching, refers to 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.

For example, given two strings *S* and *T*, the alignment between *S* and *T* is obtained by first inserting chosen spaces (gaps or dashes), into *S* and *T*, and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string (Gusfield 1997). Figure 2.3 shows an example of sequence alignment,

```
QAC-DBD
QAWX-B-
```

Figure 2.3 An example of sequence alignment

In this alignment, character C is mismatched with W, both the Ds and X are opposite space, and all characters match their counterparts in the opposite string.

2.4 The Application of Sequence Alignment

Sequence alignment is a very useful tool. It commonly it used to search a word, check spelling error in a text, or even do file comparison. One of the widest applications of sequence alignment is in comparing the biological sequences. Sequence alignment attempts to align two or more biological sequences (DNA or proteins) such that regions of structural or functional similarity between the molecules are highlighted.

In modern molecular biological research, when scientists find an unknown new gene from a cDNA library or genomic library, they usually send this gene to a gene bank to search for any similar sequences in the database. By asking whether the unknown sequence is in any way similar to known sequences (and ideally to sequences of known function or structure), scientists can identify an unknown sequence and predict its structure and function.

CHAPTER III

LITERATURE REVIEW OF ALGORITHMS FOR BIOLOGICAL SEQUENCE ALIGNMENT

3.1 The Importance of Sequence Alignment in Molecular Biology

Sequence alignment, or sequence comparison, particularly when combined with the systematic collection of data and search of database containing biomolecular sequences, has become essential in modern molecular biology. Biological sequence alignment make use of the fact that high sequence similarity usually implies significant structural or functional similarity. When we find two or more sequences can be related by aligning them; that is, many characters in one sequence are in the same order as they are in the other sequence, then we say that the two sequences are similar. Later, we calculate a similarity score, which gives the probability that the sequences are related. The following may be true for similar sequences: (A) The sequences may share a common origin – a common ancestor sequence. If the similarity is sufficiently convincing, or if there is additional evidence for an evolutionary relationship, then the sequences are homologous; (B) 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.

As more is known about more sequences in an increasingly larger number of organisms, we find the organisms share many similar sequences, but some sequences also are unique to an organism or group of organisms. The shared sequences all can be compared to each other to try to define the common domains that provide a particular function. In other cases, they can be used to predict which of the organisms are most closely related based upon the degree of similarity.

3.2 The Scoring Scheme

All algorithms to compare biological sequences rely on some scheme to score the equivalence of each of the 210 possible pairs of amino acids (i.e. 190 pairs of different amino acids + 20 pairs of identical amino acids). Most scoring schemes represent the 210 pairs of scores as a 20 x 20 matrix of similarities where identical amino acids and those of similar character (e.g. I, L, see Figure 3.2.3) give higher scores compared to those of different character (e.g. I, D, sec Figure 3.2.3). Since the first protein sequences were obtained, many different types of scoring scheme have been devised. The most commonly used are those based on observed substitution and of these, the Blosum matrix set (Henikoff and Henikoff 1992) and Dayhoff matrix for 250 PAMS (Dayhoff 1978) are good examples of widely used matrices. Different scoring schemes are discussed in the following sections.

3.2.1 Identity Scoring

This is the simplest scoring scheme: amino acid pairs are classified into two types: identical and non-identical. Non-identical pairs are scored 0 and identical pairs give a positive score (usually 1). The scoring scheme is generally considered less effective than schemes that weight non-identical pairs, particularly for the detection of weak similarities (Feng 1985 and Schwartz 1978). The normalized sum of identity scores for an alignment is popularly quoted as "percentage identity", but this value can be useful to indicate the overall similarity between two sequences, there are pitfalls associated with the measure.

3.2.2 Genetic Code Scoring

Whereas the identity scoring scheme considers all amino acid transitions with equal weight, genetic code scoring, introduced by Fitch (1966), considers the minimum number of DNA/RNA base changes (0,1,2 or 3) that would be required to interconvert the codons for the two amino acids. The scheme has been used both in the construction of phylogenetic trees and in the determination of homology between protein sequences having similar three-dimensional structures (Cohen 1981). However, today it is rarely the first choice for scoring alignments of protein sequences.

3.2.3 Chemical Similarity Scoring

Chemical similarity scoring schemes give greater weight to the alignment of amino acids with similar physic-chemical properties that other schemes. This is desirable since major changes in amino acid type could reduce the ability of the protein to perform its biological role and hence the protein would be selected against during the course of evolution. The intuitive scheme developed by McLachlan (1972) classified amino acids on the basis of polar or non-polar character, size, shape and charge and gives a score of 6 to interconversions between identical rare amino acids (e.g. F, F) reducing to 0 for substitutions between amino acids of quite different character (e.g. F, E). Figure 3.2.3 shows the classification of different amino acids. Feng et al. (1985) encoded features similar to McLachlan's (1972) by combining information from the structural features of the amino acids and the redundancy of the genetic code.

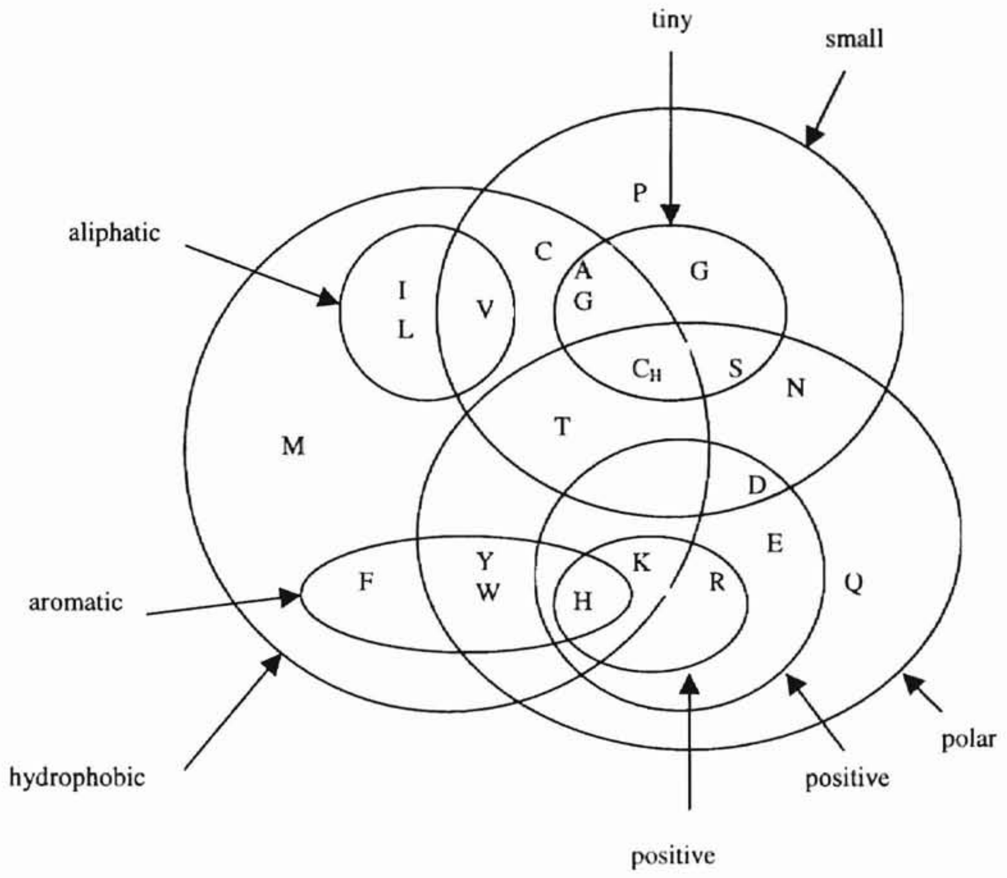

Figure 3.2.3 Classification of different amino acids (from Taylor 1986)

3.2.4. Observed Substitutions

Scoring schemes based on observed substitutions are derived by analyzing the substitution frequencies seen in alignments of sequences. Early schemes based on observed substitutions worked from closely related sequences that could easily be aligned by eye. More recent schemes have had the benefit of the earlier substitution matrices to generate alignments on which to build. Experience with scoring schemes based on observed substitutions suggests that they are superior to simple identity, genetic code, or intuitive physical-chemical property schemes.

3.2.4.1 The Dayhoff Mutation Data Matrix

Possibly the most widely used scheme for scoring amino acid pairs is that developed by Dayhoff and his co-workers (Dayhoff 1978). The system arose out of a general model for the evolution of proteins. Dayhoff and co workers examined alignments of closely similar sequences where the likelihood of a particular mutation (e.g. A-D) being the result of a set of successive mutations (e.g. A-x-y-D) was low. Since relatively few families were considered, the resulting matrix of accepted point mutations included a large number of entries equal to 0 or 1. A complete picture of the mutation process including those amino acids which did not change was determined by calculating the average ratio of the number of changes a particular amino acid type underwent to the total number of amino acids of that type present in the database. This was combined with the point mutation data to give the mutation probability matrix (M) where each element $M_{i,j}$ gives the probability of the amino acid in column j mutating to the amino acid in row i after a particular evolutionary time.

The mutation probability matrix is specific for a particular evolutionary distance, but may be used to generate matrices for greater evolutionary distances by multiplying it repeatedly by itself. The PAM (point accepted mutation) matrix which was invented by Dayhoff and his co-workers can be used for measuring the evolutionary distance. At the level of 2,000 PAM Schwartz (1978) and Dayhoff (1978) suggest that all the information present in the matrix has degenerated except that the matrix element for Cys-Cys is 10% higher than would be expected by chance. At the evolutionary distance of 256 PAMs one amino acid in five remains unchanged but the amino acids vary in their mutability; 48%

of the tryptophans, 41% of the cysteines and 20% of the histidines would be unchanged, but only 7% of serines would remain.

When used for the comparison of protein sequences, the mutation probability matrix is usually normalized by dividing each element $M_{i,j}$ by the relative frequency of exposure to mutation of the amino acid i. This operation results in the symmetrical "relatedness odds matrix" with each element giving the probability of amino acid replacement per occurrence of i per occurrence of j. The logarithm of each element is taken to allow probabilities to be summed over a series of amino acids rather than requiring multiplication. The resulting matrix is the "log-odds matrix" which is frequently referred to as "Dayhoff's matrix" and often used at a distance of close to 256 PAM since this lies near to the limit of detection of distant relationships where approximately 80% of the amino acid positions are observed to have changed (Schwartz 1978).

3.2.4.2 PET91 – An Updated Dayhoff Matrix

The 1978 family of Dayhoff matrices was derived from a comparatively small set of sequences. Many of the 190 possible substitutions were not observed at all so suitable weights were determined indirectly. Recently, Jones et al. (Jones 1992) have derived an updated substitution matrix by examining 2,621 families of sequences in the SWISSPROT database release 15.0. The principal differences between the Jones et al. (1992) matrix (PET91) and the Dayhoff matrix are for substitutions that were poorly represented in the 1978 study. However, the overall character of the matrices is similar. Both reflect substitutions that conserve size and hydrophobicity, which are the principle properties of the amino aids (Taylor 1986).

3.2.4.3 BLOSUM – Matrix From Ungapped Alignment

Dayhoff-like matrices derive their initial substitution frequencies from global alignments of very similar sequences but they do have limitations. An alternative approach has been developed by Henikoff and Henikoff using local multiple alignments of more distantly related sequences (Henikoff and Henikoff 1992). First a database of multiple alignments without gaps for short regions of related sequences was derived. Within each alignment in the database, the sequences were clustered into groups where the sequences are similar at some threshold value of percentage identity. Substitution frequencies for all pairs of amino acids were then calculated between the groups and this used to calculate a log odds BLOSUM (blocks substitution matrix) matrix. Different matrices are obtained by varying the clustering threshold. For example, the BLOSUM 80 matrix was derived using a threshold of 80% identity.

3.2.5 The Selection of a Suitable Scoring Matrix

The general consensus is that matrices derived from observed substitution data (e.g. the Dayhoff or BLOSUM matrices) are superior to identity, genetic code or physical property matrices (e.g. see Feng *et al.* 1985). However, there are Dayhoff matrices of different PAM values and BLOSUM matrices of different percentage identity and which of these should be used?

Schwartz and Dayhoff (Schwartz *et al.* 1978) recommended a mutation data matrix for the distance of 250 PAMs as a result of a study using a dynamic programming procedure (Needleman and Wunsch 1970) to compare a variety of proteins known to be distantly related. The 250 PAM matrix was selected since in Monte Carlo studies matrices reflecting this evolutionary distance gave a consistently higher significance score than other matrices in the range 0-750 PAM. The matrix also gave better scores when compared to McLachlan's substitution matrix (McLachlan 1972), the genetic code matrix and identity scoring. Recently, Altschul (1991) has examined Dayhoff style mutation data matrices from an information theoretical perspective. For alignments that do not include gaps he concluded, in broad agreement with Schwarz and Dayhoff, that a matrix of 200 PAMS was most appropriate when the sequences to be compared were thought to be related. However, when comparing sequences that were not known in advance to be related, for example when database scanning, a 120 PAM matrix was the best compromise. When using a local alignment method Altschul suggests that three matrices should ideally be used: PAM40, PAM120 and PAM250, the lower PAM matrices will tend to find short alignments of highly similar sequences, while higher PAM matrices will find longer, weaker local alignments. Similar conclusions were reached by Collins and Coulson (1988) who advocate using a compromise PAM100 matrix, but also suggest the use of multiple PAM matrices to allow detection of local similarities of all types.

Henikoff and Henikoff (1993) have compared the BLOSUM matrices to PAM, PET, Overington et al (1990), Gonnet (1992) and multiple PAM matrices by evaluating how effectively the matrices can detect known members of a protein family from a database when searching with the ungapped local alignment program BLAST (Altschul *et al.* 1990). They concluded that overall the BLOSUM 62 matrix is the most effective for ungapped matching. Pearson (1996) found that BLOSUM 50 is perhaps better for alignment with gaps. However, all the substitution matrices investigated perform better than BLOSUM 62 for a proportion of the families. This suggests that no single matrix is the complete answer for all sequence comparisons. It is probably best to compliment the BLOSUM 62 matrix with comparisons using PET91 at 250 PAMS, and Overington structurally derived matrices. It seems likely that as more protein three-dimensional structures are determined, substitution tables derived from structure comparison will give the most reliable data. The BLOSUM 50 matrix is employed for gapped sequence alignment. Table 3.2.5 shows the entire BLOSUM 50 matrix used in our program.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**																			
R	-2	**7**																		
N	-1	-1	**7**																	
D	-2	-2	2	**8**																
C	-1	-4	-2	-4	**13**															
Q	-1	1	0	0	-3	**7**														
E	-1	0	0	2	-3	2	**6**													
G	0	-3	0	-1	-3	-2	-3	**8**												
H	-2	0	1	-1	-3	1	0	-2	**10**											
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	**5**										
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	**5**									
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	**6**								
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	**7**							
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	**8**						
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	**10**					
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	**5**				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	**5**			
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	**15**		
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	**8**	
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	**5**

Table 3.2.5 BLOSUM 50 matrix (entries on the main diagonal for identical residue pairs are highlighted in bold)

3.3 Gap Penalty

Until now, the central constructions used to measure the value of an alignment have been matches, mismatches and spaces. Now we introduce another important construct, gaps. Gaps help create alignments that conform to underlying biological models better and that more closely fit patterns that one expects to find in meaningful alignment. They take account of the number of continuous gaps and rather than only the number of spaces when calculating an alignment mark. This section presents a gap penalty model for evaluating the weight of a sequence of consecutive indel operations. The model states that consecutive indel operations have different total weight than simply the sum of their weights.

A gap is any maximal, consecutive run of spaces in a single string of a given alignment. Figure 3.3 is a example:

```
S = a t t c - - g a - t g g a c c
T = a - - c g t g a t t - - - c c
```

Figure 3.3 Alignment with gaps

There are four gaps containing a total of eight spaces in this example. That alignment would be described as having seven matches, no mismatch, four gaps and eight spaces.

The length of the gap will be the number of indel operations in it. The number of gaps in the alignment will be denoted as # gaps.

3.3.1 Motivation

The concept of a gap in an alignment is important in many biological applications, because the insertion or deletion of an entire substring often occurs as single mutational event. Moreover, many of these single mutational events can create gaps of quite varying sizes. At the protein level, two protein sequences might be relatively similar over several intervals but differ in intervals where one contains a protein subunit that the other does not.

One concrete illustration of the use of gaps in the alignment model comes from the problem of cDNA matching (Gusfield 1997). In this problem, one string is much longer than the other, and the alignment best reflecting their relationship should consist of a few regions of very high similarity interspersed with 'long' gaps in the shorter string. Note that the matching regions can have mismatches and spaces, but these should amount only to a small fraction of the region.

An RNA molecule is transcribed from DNA of the gene. That RNA transcript is a complement of the DNA in the gene in that each A in the gene is replaced by U in the RNA, each T is replaced by A, each C by G, and each G by C. Moreover, the RNA transcript covers the entire gene, introns as well as exons. Then in a process that is not understood completely, each introns-exon boundary in the transcript is located, the RNA corresponding to the introns is spliced out, and the RNA regions corresponding to exons are concatenated. Additional processing occurs. The resulting RNA molecule is called the messenger RNA (mRNA): it leaves the cell nucleus and is used to create the protein it encodes.

Usually, each cell contains a copy of all the chromosomes; and hence, of all the genes of the entire individual, yet in each specialized cell (a liver cell for example) only a small fraction of the genes are expressed. That is, only a small fraction of the proteins encoded in the genome are actually produced in that specialized cell. A standard method to determine which proteins are expressed in the specialized cell line, and to hunt for the location of the encoding genes, involves capturing the mRNA in that cell after it leaves the cell nucleus. That mRNA is then used to create a DNA sequence complementary to it. This sequence is called cDNA (complementary DNA). Compared to the original gene, the cDNA sequence consists only of the concatenation of exons in the gene. After cDNA is obtained, the problem is to determine where the gene associates with that cDNA resides, and it becomes one of aligning the cDNA sequence against the longer DNA sequence in a way that reveals the exons.

3.3.2 Gap Penalties Types

There are four types of gap penalties as follows:

- Constant: solved in $O(mn)$ time.
- Affine: solved in $O(mn)$ time
- Convex: solved in $O(mnlogm)$ time
- Arbitrary: solved in $O(mn^2 + nm^2)$ time

3.3.3 Constant Gap Weight Model

The simplest choice is the constant gap weight, where each individual space is free, and each gap is given a weight of Ws independent of the number of spaces in the gap. Letting σ denote the weights of match and mismatch only $(\sigma(x, -) = (-, x) = 0$ for every character x). Thus we have to find an alignment that maximizes:

$$\Sigma\sigma(S_i', T_i') + W_g \text{ x (No. of gaps)}$$

where S' and T' represent S and T after inserting space(s). A generalization of the constant gap weight model is to add a weight W_s for each space in the gap. In this case, W_g represents the cost of starting a gap, and W_s represents the cost of extending the gap by one space. This leads us to the *affine gap weight model*. This is called affine gap weight model because the weight contributed by a single gap of length q is given by the affine function $W_g + qW_s$. The constant gap weight model is simply the affine model with $W_s = 0$. Thus we have to find an alignment that maximizes:

$$\Sigma\sigma(S_i', T_i') + W_g \text{ x (No. of gaps) + Ws x (No. of spaces)}$$

while S' and T' represent S and T after inserting space and $\sigma(x, -) = (-, x) = 0$ for every character x.

It has been suggested that some biological phenomena are better modeled by a gap weight function where each additional space in a gap contributes less to the gap weight than the preceding space. In other words, a gap weight that is a convex, but not affine function of its length. An example is the function $W_g + log q$, where q is the length of the gap. Finally, the most general gap weight that might be considered is the *arbitrary gap weight*, where the weight of a gap is an arbitrary function $\omega(q)$ of its length q. The constant, affine and convex weight models are of course restricted cases of the arbitrary weight model.

3.3.4 Affine Gap Penalty

To align strings S, T, consider as usual the prefixes $S_{1...i}$ of S and $T_{1...j}$ of T. Any alignment of these two prefixes is one of the following three types:

1.
```
S------i
T------j
```

Alignment of $S_{1...i}$ and $T_{1...j}$ where characters $S(i)$ and $T(j)$ are aligned opposite each other. This includes both the case that $S_i = T_j$ and that $S_i \neq T_j$.

2.
```
S------i_ _ _ _ _ _ _
T------------j
```

Alignment of $S_{1...i}$ and $T_{1...j}$ where character S_i is aligned to a character strictly to the left of character T_j. Therefore, the alignment ends with a gap in S.

3.
```
S --------------i
T ------j_ _ _ _ _ _ _ _
```

Alignment of $S_{1...i}$ and $T_{1...j}$ where character S_i is aligned to a character strictly to the right of character T_j. Therefore, the alignment ends with a gap in T.

We will use $G(i, j)$ to denote the maximum value of any alignment of type 1, $E(i, j)$ as the maximum value of any alignment of type2 and $F(i, j)$ as the maximum value of any alignment of type3. We finally define $V(i, j)$ as the maximum value of the three terms $E(i, j)$, $F(i, j)$, $G(i, j)$. Hence the base conditions are:

$$V(i, 0) = E(i, 0) = W_g + iW_s$$

$$V(0, j) = F(0, j) = W_g + jW_s$$

and the recursive computation of $V(i, j)$ will be:

$$V(i, j) = \max \{ E(i, j), F(i, j), G(i, j)\}$$

while

$$G(i, j) = V(i - 1, j - 1) + \sigma(S_i, T_j)$$
$$E(i, j) = \max \{ E(i, j - 1) + W_s, V(i, j - 1) + W_g + W_s\}$$
$$F(i, j) = \max \{ F(i - 1, j) + W_s, V(i - 1, j) + W_g + W_s\}$$

The optimal value alignment is the maximum value in the nth row or mth column. The complexities of affine gap penalty are:

- Time complexity - As before $O(nm)$, as we compute four small matrices instead of large one.

- Space complexity - There's a need to save four matrices (for E, F, G, and V respectively) during the computation. Hence, $O(nm)$ space is needed for the trivial implementation.

3.4 Pairwise Sequence Alignment

There are two major methods for sequence alignment: Pairwise alignment and multiple sequence alignment. Pairwise alignment, as the term suggests, aligns two sequences at one time.

Pairwise sequence alignment is a cornerstone of sequence alignment. Molecular biologists frequently compare biosequences to determine whether any similarities can be found hoping that what is true of one sequence either physically or functionally is true of its analogue. Generally, such comparison involves aligning sections of the two sequences in a way that exposes the similarities between them.

Given a scoring scheme, the next problem is how to compare the sequences to decide how similar they are, then to generate an alignment. This problem may be subdivided into alignment methods for two sequences, multiple alignment methods, and methods that incorporate additional non-sequence information; for example, from the tertiary structure of the protein.

The simplest two-sequence comparison methods do not consider insertions and deletions (gaps) explicitly. More sophisticated methods make use of dynamic programming to determine the best alignment including gaps (see Section 3.2.2.1 and Chapter IV).

3.4.1 Alignment Without Gaps – Fixed Length Segments

Given two sequences *S* and *T* of length *m* and *n*, respectively, all possible overlapping segments having a particular length (sometimes called a 'window length') from *S* are compared to all segments of *T*. This requires of the order of $m \times n$ comparisons to be made. For each pair of segments the amino acid pair scores are accumulated over the length of the segment. For example, consider the comparison of two 7-residue segments: ALGAWDE and ALATWDE using identity scoring. The total score for this pair would be $1 + 1 + 0 + 0 + 1 + 1 + 1 = 5$.

In early studies of protein sequences, statistical analysis of segment comparison scores was used to infer homology between sequences. For example, Fitch (1966) applied the genetic code scoring scheme to the comparison of α- and β-hemoglobin and showed the score distribution to be non-random. Today, segment comparison methods are most commonly used in association with a "dot plot" or "diagram" (Gibbs and McIntyre 1970) and can be a more effective method of finding repeated sequences than using dynamic programming, but they have less effective in global and local alignment.

The scores obtained by comparing all pairs of segments from *S* and *T* may be represented as a comparison matrix *R* where each element $R_{i,j}$ represents the score for matching an odd length segment centered on residue S_i with one centered on residue T_j. This matrix can provide a graphic representation of the segment comparison data particularly if the scores are contoured at a series of probability levels to illustrate the most significantly similar regions. Collins and Coulson (1987) have summarized the features of the "dot-plot". The runs of similarity can be enhanced visually by placing a dot at all the contributing match points in a window rather than just at the center.

McLachlan (1972) introduced two further refinements into segment comparison methods. The first was the inclusion of weights in the comparison of two segments in order to improve the definition of the ends of regions of similarity. For example, the scores obtained at each position in a 5-residue segment comparison might be multiplied by 1,2,3,2,1 respectively before being summed. The second refinement was the development of probability distributions which agreed well with experimental comparisons on random and unrelated sequences and which could be used to estimate the significance of an observed comparison.

3.4.1.1 Correlation Methods

Several experimental, and semi-empirical properties have been derived associated with amino acid types, for example hydrophobicity (e.g. Jones 1975), and propensity to form an α-helix (eg. Chou and Fasman 1978). Correlation methods for the comparison of protein sequences exploit the large number of amino acid properties as an alternative to comparing the sequences on the basis of pair scoring schemes.

Kubota et al. (1982) gathered 32 property scales from the literature and through application of factor analysis selected 6 properties which for carp parvalbumin gave good correlation for the comparison of the structurally similar CE- and EF-hand region Ca^{2+} binding sites and poor correlation in other regions. They expressed their sequence comparisons in the form of a comparison matrix similar to that of McLachlan (1972) and demonstrated that their method could identify an alignment of α-lytic protease and *Streptomyces Griseus* protease A which agrees with that determined from comparison of the available crystal structures.

Argos (1987) determined the most discriminating properties from a set of 55 by calculating correlation coefficients for all pairs of sequences within 30 families of proteins that had been aligned on the basis of their three-dimensional structures. The correlation coefficients for each property were then averaged over all the families to leave 5 representative properties. Unlike Kubota *et al.* (1982), Argos applied the correlation coefficients from the five properties in addition to a more conventional segment comparison method using the Dayhoff matrix scoring scheme. He also combined the result of using more than one segment length on a single diagram such that the most significant scores for a particular length always prevail.

3.4.1.2 Variable Length Segments

The best local ungapped alignments of variable length may be found either by dynamic programming with a high gap-penalty, or using heuristic methods. Since the heuristic methods are primarily used for database searching.

3.4.2 Alignment with Gaps

The segment based techniques described in section 3.2.1 do not consider insertions and deletions explicitly. Deletions are often referred to as "gaps", while insertions and deletions are collectively referred to as "indels". Insertions and deletions are usually needed to align accurately even quite closely related sequences such as the α- and β-globins. The naive approach to finding the best alignment of two sequences including gaps is to generate all possible alignments, add up the scores for equivalencing each amino acid pair in each alignment then select the highest scoring alignment. However, for two sequences of 100 residues there are more than 10^{75} alternative alignments so such an approach would be time consuming and infeasible for longer sequences. Fortunately, there is a group of algorithms that can calculate the best score and alignment in the order of *mn* steps. These dynamic programming algorithms were first utilized for protein sequence comparison by Needleman and Wunsch (1970), though similar methods were independently devised during the late 1960's and early 1970's for use in the fields of speech processing and computer science (Kruskal 1983).

3.4.2.1 Finding the Best Alignment with Dynamic Programming

Dynamic programming algorithms are discussed in detail in Chapter IV. Dynamic programming algorithms may be divided into those that find a global alignment of the sequences and those that find local alignments. The difference between global and local alignment is illustrated in Figure 3.4.2.1. Global alignment is appropriate for sequences that are known to share similarity over their whole length. Local alignment is appropriate when the sequences may show isolated regions of similarity, for example multiple domains or repeats. Local alignment is best applied when scanning a database to find similarities or when there is no *a priori* knowledge that the protein sequences are similar.

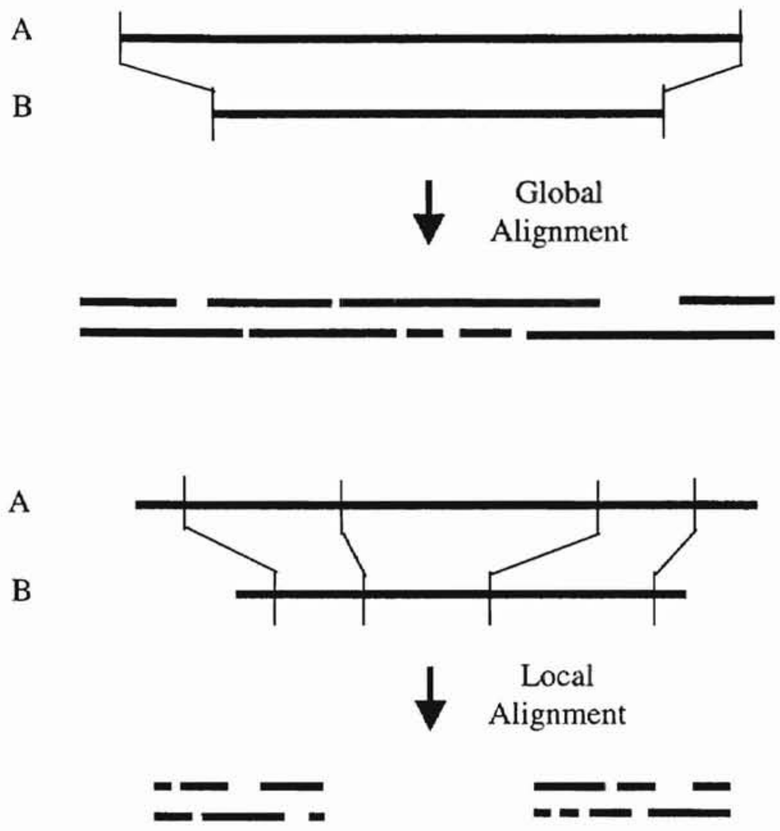

Figure 3.4.2.1 The difference between global and local alignments

There are many variations on the theme of dynamic programming applied to protein comparisons. Here is a brief account of a basic method for finding the *globally* best score for aligning two sequences. Kruskal and Sankoff (1983) give a clear and detailed explanation of dynamic programming.

Let the two sequences of length m and n be $S = (S_1, S_2, \ldots S_m)$, $T = (T_1, T_2, \ldots T_n)$ and the symbol for a single gap be Δ. At each aligned position there are three possible events:

1). $\omega(S_i, T_j)$ substitution of S_i by T_j.

2). $\omega(S_i, \Delta)$ deletion of S_i.

3). $\omega(\Delta, T_j)$ deletion of T_j.

The substitution weight $\omega(S_i, T_j)$ is derived from the chosen scoring scheme, perhaps Dayhoff's matrix. Gaps, Δ, normally are given a negative weight often referred to as the "gap penalty" since insertions and deletions are usually less common than substitutions.

The maximum score M for the alignment of S with T may be represented as $\delta(S_{1\ldots m}, T_{1\ldots n})$. This may be found by working forward along each sequence successively finding the best score for aligning $S_{1\ldots i}$ with $T_{1\ldots j}$ for all i, j where $1 \leq i \leq m$ and $1 \leq j \leq .n$. The values of are stored in a matrix H where each element of H is calculated as follows:

$$H_{i,j} = \max \left\{ \begin{array}{l} H_{i-1,j-1} + \omega_{Si,\,Tj} \\ H_{i,j-1} + \omega_{Si,\,\Delta} \\ H_{i-1,j} + \omega_{\Delta,\,Tj} \end{array} \right\}$$

The element $H_{m,n}$ contains the best score for the alignment of the complete sequences.

If the alignment is required as well as the best score, then the alignment path may be determined by tracing back through the H matrix. Alternatively, a matrix of pointers can be recorded to indicate which of the three possibilities was the maximum at each value $H_{i,j}$.

3.4.2.2 Alternative Weight for Gaps

The above scheme showed a simple length-dependent weighting for gaps. Thus two isolated gaps give the same score as two consecutive gaps. It is possible to generalize the algorithm to allow gaps of length greater than 1 to carry weights other than the simple sum of single gap weights (Waterman et al. 1976). Such gap weighting can give a more biologically meaningful model of transitions from one sequence to another since insertions and deletions of more than one residue are not uncommon events between homologous protein sequences. Most computer programs that implement dynamic programming allow gaps to be weighted with the form $v + uk$ where k is the gap length and v and u are constants ≥ 0, since this can be computed efficiently (Gotoh 1982).

3.4.3 Identification of Local Similarities

Although segment based comparison methods (see section 3.2.1) rely on local comparisons, when insertions and deletions have occurred, then the match may be disrupted for a region of the order of the length of the segment. In order to circumvent these difficulties algorithms which are modifications of the basic global alignment methods have been developed to locate common subsequences including a consideration of gaps (e.g.Bosewell 1984, Smith and Waterman 1981a and Sellers 1984). For protein sequences, the most commonly used local alignment algorithm that allows gaps is that described by Smith and Waterman (1981b). This is essentially the same as the global alignment algorithm described in section 3.2.2.1, except that a zero is added to the recurrence equation.

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + \omega_{S_i,T_j} \\ H_{i,j-1} + \omega_{S_i,\Delta} \\ H_{i-1,j} + \omega_{\Delta,T_j} \\ 0 \end{array} \right\}$$

Thus, all $H_{i,j}$ must have a value ≥ 0. The score for the best local alignment is simply the largest value in a cell of H and the corresponding alignment is obtained by tracing back from this cell.

3.4.3.1 Finding Subsequence - Best Local Alignment

The Smith-Waterman algorithm returns the single best local alignment, but two proteins may share more than one common region. Waterman and Jones (1990) have shown how all local alignments may be obtained for a pair of sequences with minimal recalculation. Recently, Barton (1993) has described how for a simple length dependent gap-penalty, all locally optimal alignments may be determined in the order of steps without recalculation.

3.5 Multiple Sequences Alignment

This thesis mainly focuses on pairwise sequence alignment, so we only briefly introduce multiple sequence alignment here.

3.5.1 Introduction

Multiple sequence alignment is the process of aligning three or more sequences with each other to bring as many similar sequences characters (nucleotides or amino acids) into register as possible. The resulting alignments can be used for two purposes:

1) To find regions of similar sequence in all of the sequences that define a conserved consensus pattern or domain.

2) If the alignment is particularly strong, to use the aligned positions to try and derive the possible evolutionary relationships among the sequences.

When dealing with a sequence of unknown function, the presence of similar domains in several similar sequences implies a similar biochemical function or structural folding that may become the basis of further experimental investigation. A group of similar sequences may define a protein family that may share a common biochemical function or evolutionary origin.

3.5.2 Multiple Alignment Algorithms

Two types of methods are used for aligning sequences: global alignment and local alignment. Global methods attempt to find an 'optimal' alignment throughout the length of sequences (Barton and Sternberg 1987a, 1987b; Feng and Doolittle 1987; Taylor 1987; Lipman *et al.* 1989; Subbian and Harrison 1989; Higgins and Fuchs 1992). A sub-class of global methods attempts first to identify an ordered series of motifs and then proceeds to align the remaining regions (Martinez 1988; Vingron and Argon 1991). Local methods, such as the Multiple Alignment Construction Workbench (MACAW; Schuler *et al.* 1991) and PIMA (Smith and Smith 1990; Waterman and Jones 1990), which use the Smith-Waterman algorithm, only attempt to identify motifs and do not attempt to align regions between motifs. Global methods are best for both evolutionary and structural studies of unequivocally related sequences, whereas local methods are best for seeking distant relationships and for motif identification.

Global methods rarely attempt to compare all sequences simultaneously because of the computational expense of doing so. Usually the two most closely related sequences (determined by pairwise comparisons) are aligned and the others, in order of similarity, are added progressively. This method of progressive multiple alignment was introduced in 1987 (Feng and Doolittle 1987; Taylor 1987). The alignment continues in an iterative fashion, adding gaps where required to achieve alignment, but only to all members of each growing cluster. The most widely used variant of this approach is CLUSTAL V (Higgins and Fuchs 1992). The shortcoming of this approach is the inherent bias when one subset of sequences is over-represented (Altschul *et al* 1989). Other methods directly produce a progressive multiple alignment or an optimal alignment defined in some way (Subbiah and Harrison 1989; Lipman *et al* 1989; Vingron and Argos 1991). Barton and Sternberg's method (1987b) allows the user to analyze sequences following either strategy.

A comparative study of global and local methods was made by McClure *et al.* (1994). Four data sets, each containing sequences from the globin, kinase, retroid aspartic acid protease and ribonuclease H sequence families, were aligned and used to test the ability of twelve global and local methods to accurately identify their characteristic motifs. It was found that CLUSTAL V and Dfalign (also called PILE-UP in the GCG package) perform better than the others; the global methods were better than the local methods, PIMA was slightly better than MACAW. CLUSTAL V is flexible and user-friendly, its instructions and interactive interface are clear, it can align both nucleic acids and proteins and there are DOS, VMS and UNIX versions.

Other new approaches to multiple sequence alignment are being tested. One such is the use of Hidden Markov Models (HMM) with adaptive algorithms for parameter training, which has also been used for speech recognition (Baldi *et al.* 1994; Krogh *et al.* 1994). The method creates a stochastic model using the sequences being studied as a 'training set'. At each position of a sequence there is the probability that a residue, a deletion or an insertion will occur. The model starts with a uniform probability of each type; one chance out of three for a nucleotide, deletion or insertion and one out of 20 for a particular amino acid for the residue or insertion states. As the model "learns" from each sequence these probabilities are adjusted. Thus the differences in a family of sequences are incorporated in to the model, so there is no need for a separate dissimilarity matrix. The final model is used as a template for aligning individual sequences to produce a multiple alignment, and for identifying related sequences.

None of the currently available computer methods is foolproof, and manual refinement of the alignment may be required. A skilled scientist can identify and correct small regions of similarities not detected by multiple alignment programs.

CHAPTER IV

DYNAMIC PROGRAMMING AND PAIRWISE BIOLOGICAL SEQUENCE ALIGNMENT

4.1 Introduction

Dynamic programming is a term from operations research where it was first used to describe a class of algorithms for the optimization of dynamic systems (Bellman 1957). In dynamic programming the principle of divide-and-conquer is used extensively: subdivide a problem that is too large to be computed into several smaller problems that may be computed efficiently. Then the answers are assembled to give a solution for the large problem. Here, this principle is carried to the extreme: when we do not know which smaller problem to solve, we simply solve all smaller problems, store the answers and assemble them later to a solution for the larger problem. Dynamic programming methods assume the principle of optimality, stating that each part of a globally optimal solution is itself an optimal solution to its corresponding partial problem. This inference of global properties from local properties is, unfortunately, not valid for all biological problems, but the principle can be used to find globally optimal sequence alignments very efficiently.

Globally optimal alignment is a difficult problem. The major difficulty comes from the fact that one cannot simply slide one sequence along another and sum over the similarity scores looked up in the appropriate mutation data matrix. This does not work, because biological sequences may have gaps or insertions of sequences relative to each other. Thus one cannot know exactly which residue is paired with which other residue, and in principle one should try all possible combinations to find the optimal one, maximizing the number and quality of matches while at the same time minimizing the number and length of necessary gaps in the alignment. Obviously, this is a problem that quickly explodes in its requirements for computational resources with the size of the sequences to be searched.

4.2 Needleman-Wunch Algorithm and Global Sequence Alignment

Saul Needleman and Christian Wunsch were the first to use dynamic programming approach for aligning biological sequences (Needleman and Wunsch 1970). The algorithm is a way to find an alignment that maximizes a particular score (The score can be calculated in a variety of methods - as is indicated below). The overall method is reminiscent of the dot plot (though this was not popular in the 1970's, so there is probably no connection). The first step is to place the two sequences along the margins of a matrix as shown in Table 4.2.1

In this first step, simply place a 1 anywhere the two sequences match and a 0 elsewhere. If done on a larger scale than is shown in Table 4.2.1, this exactly recreates the dot plot shown in Figure 1. In this case however, we search for a path through this matrix to define a more conventional alignment. For example, proceeding along the diagonal with no deviations would imply an alignment without any gaps. The introduction of a gap (either by an insertion or a deletion - an indel) in either sequence would correspond to moving either above or below the main diagonal.

To find the best route, Needleman and Wunsch suggested that one modify the matrix to represent this idea of tracing different pathways through the matrix. However, one wants to include all possible pathways and from among these choose only that one that is best (in the sense of maximizing some score). Their method consists of two passes through the matrix. The first traces a score for all possible routes and moves right to left, bottom to top. Once the score for all possible routes are found, the maximum can be chosen (somewhere on the topmost row or leftmost column) and a second pass can be carried out, this time running left to right, top to bottom to find the alignment that gives the maximum score.

$$
\begin{array}{c|ccccccccccccc|}
 & A & B & C & N & J & R & Q & C & L & C & R & P & M \\
\hline
A & 1 & & & & & & & & & & & & \\
J & & & & & 1 & & & & & & & & \\
C & & & 1 & & & & & 1 & & 1 & & & \\
J & & & & & 1 & & & & & & & & \\
N & & & & 1 & & & & & & & & & \\
R & & & & & & 1 & & & & & 1 & & \\
C & & & 1 & & & & & 1 & & 1 & & & \\
K & & & & & & & & & & & & & \\
C & & & 1 & & & & & 1 & & 1 & & & \\
R & & & & & & 1 & & & & & 1 & & \\
B & & 1 & & & & & & & & & & & \\
P & & & & & & & & & & & & 1 & \\
\hline
\end{array}
$$

Table 4.2.1 Initial Step

The way to trace a score for all possible paths is shown in Table 4.2.2. For each element $M_{i,j}$ in the matrix is computed as:

$$M_{i,j} = M_{i,j} + max\ (M_{i-k,j}, M_{i,j-l})$$

where k is any integer larger than i and l is any integer larger than j. The process alters the matrix by adding to each element the largest element from the row just below and to the right of that element and from the column just to the right and below the element of interest. This row and column for one element are shown in Table 4.2.2 by boxes. The number contained in each cell of the matrix after this operation is completed is the largest number of identical pairs that can be found if that element is the origin for a pathway which proceeds to the upper left.

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	1												
J					1								
C			1					1		1			
J					1								
N				1									
R						$\boxed{1}$	4	3	3	2	2		
C	3	3	4	3	3	3	3	4	3	3	1		
K	3	3	3	3	3	3	3	3	3	2	1		
C	2	2	3	2	2	2	2	3	2	3	1		
R	2	1	1	1	1	2	1	1	1	1	2		
B	1	2	1	1	1	1	1	1	1	1	1		
P												1	

Table 4.2.2 Half way through the second step

To have an alignment which covers the entire sequence, we find on the upper row, or on the left column, the element of the matrix with maximum value. An alignment must begin at this point then proceed to the lower right. This is the second pass through the matrix. At each step of this pass, starting from the maximum, one moves one row and column to the lower right and finds the maximum in this row or column. The alignment must proceed through this point. The alignment finishes with a hit of either the bottom row or the rightmost column. This tracing pattern is shown in Table 4.2.3. The optimal alignment is not necessarily unique. Two alignments both give the optimal score of 8 matches.

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
J	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
J	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 4.2.3 Trace the alignment

These two alignments can be written in more familiar form as either

```
ABCNJ-RQCLCR-PM
*  *  *  *  *  ** *
AJC-JNR-CKCRBP-
```

or as

```
ABC-NJRQCLCR-PM
*  *  *  *  *  ** *
AJCJN-R-CKCRBP-
```

both with 8 asterisks to denote the 8 matches. Note that in this particular case, gaps are given the same penalty as a mismatch. They simply do not add to the score.

4.3 Smith-Waterman Algorithm and Local Sequence Alignment

The Needleman-Wunsch algorithm creates a global alignment. That is, it tries to take all of one sequence and align it with all of a second sequence. Short and highly similar subsequences may be missed in the alignment because they are outweighed by the rest of the sequence. Hence, one would like to create a locally optimal alignment. Smith and Waterman (1981a) find an alignment that determines the longest/best subsequence pair that gives the maximum degree of similarity between the two original sequences. This means that not all of the sequences might be aligned together.

Only minimal changes to the Needleman-Wunsch algorithm are required. These are:

1. A negative score/weight must be given to mismatches.
2. Zero must be the minimum score recorded in the matrix.
3. The beginning and end of an optimal path may be found anywhere in the matrix - not just the last row or column.

The first point is required to cause the score to drop as more mismatches are added. Hence, the score rises in a region of high similarity, then falls outside of this region. If there are two segments of high similarity then these must be close enough to a path between them to be linked by a gap or they will be left as independent segments of local similarity.

The second point is computed so that each pathway begins fresh. Thus each short segment of similarity should begin with a score of zero. The third point indicates that the entire matrix must be searched for regions with high local similarity.

Each element in the matrix comes from the computation:

$$M_{i,j} = M_{i,j} + max\ (M_{i-k,j}, M_{i,j-l})$$

In this case it is easier to go left to right, top to bottom in the matrix -- here k is any integer smaller than i and l is any integer smaller than j. Also, for a local alignment $M_{i,j}$ must be some negative penalty if residue i is not the same as residue j.

As an example the previous alignment can be reproduced with a penalty of -0.5 for each mismatch. The matrix will then be as given in Table 4.3.1.

	A	B	C	N	J	R	Q	C	L	C	R	P	M
A	1	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0.5	0.5	0.5	2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
C	0	0.5	1.5	0	0	1.5	1.5	3	1.5	3	1.5	1.5	1.5
J	0	0.5	0	1	2.5	1.5	1	1	2.5	2.5	2.5	2.5	2.5
N	0	0.5	0	2.5	0.5	2	2	2	2.5	2	2.5	2	2
R	0	0.5	0	1	2	3.5	2	2	2.5	2	4	2	2
C	0	0.5	1.5	1	2	2	3	4.5	3	4.5	3	3.5	3.5
K	0	0.5	0	1	2	2	3	2.5	4	4	4	4	4
C	0	0.5	1.5	1	2	2	3	4	4	5	4	3.5	3.5
R	0	0.5	0	1	2	3.5	3	2.5	4	3.5	6	4.5	4.5
B	0	2	0	1	2	2	3	3	4	3.5	4.5	5.5	5.5
P	0	0.5	1.5	1.5	2	2	3	2.5	4	3.5	4.5	7	5

Table 4.3.1 An example of Smith-Waterman algorithm (with penalty of – 0.5)

In this case the same alignment is found. However, the Smith-Waterman algorithm does not include the final M/P mismatch in its path as it is not part of the locally optimal solution. More generally, large chucks of each sequence may be missing from the local alignment (as in the alignment presented by BLAST).

It is not always the case that these two methods give the same answer; for example, a global and a local alignment of TTGACACCCTCCCAATTGTA with ACCCCAGGCTTTACACAT.

If the sequences are not known to be homologous throughout their entire length, a local alignment should be the method of choice. Often the two methods will give similar answers (as above) but if the homology is distant, a local alignment will be more likely to find the remaining patches of homology.

4.4 Repeated Matches Algorithm

Local alignment gives the best single local match between two sequences. If one or both of the sequences is long, it is possible that there are many different local alignments with a significant score, and in most cases all are of interest. For example, we may find many copies of repeated domains or motifs in a protein sequence. To find those repeated matches effectively, we need the method called the repeated matches algorithm.

If there are two different sequences: x = HEAGAWGHEE and y = PAWHEAE, repeated matches of y in x is that all of x must be aligned with some (possibly repeated) subsequences from y:

```
HEAGAWGHEE
HEA.AW-HE.
```

A dash (-) indicates that the corresponding x_i is matches by a gap in a y subsequence, a dot $(\cdot)$ means that the corresponding x_i is matched by no subsequence of y. Every two matched subsequences of x are separated by one or more unmatched subsequences

This method is asymmetric: it finds one or more non-overlapping copies of sections of one sequence (e.g. the domain or motif) in the other. An example of the repeated matches algorithm is given in Figure 4.4. We again use the matrix M, but the recurrence is now different, as is the meaning of $M(i, j)$. In the final alignment, x is partitioned into regions that match parts of y in gapped completed match region as being its standard gapped alignment score minus the threshold T. All theses matches scores will be positive. $M(i, j)$ for $j \geq 1$ is now the best sum of match scores to $x_{1...i}$, assuming that x_i is in a matched region, and the corresponding match ends in x_i and y_j (they may not actually be aligned, if this is a gapped section of the match). $M(i, 0)$ is the best sum of completed match score to the subsequence $x_{i...i}$, i.e. assuming that x_i is in an unmatched region.

To achieve the desired goal, we start by initializing $M(0, 0) = 0$ as usual, and then fill the matrix using the following recurrence relations:

$$M(i, 0) = \max \begin{cases} M(i-1, 0), \\ \\ M(i-1, j) - T, \qquad j = 1, \ldots, m; \end{cases} \tag{4.4.1}$$

$$M(i, j) = \max \begin{cases} M(i, 0), \\ M(i-1, j-1) + S(x_i, y_j), \\ \\ M(i-1, j) - \mathrm{d}, \\ M(i, j-1) - \mathrm{d}. \end{cases} \tag{4.4.2}$$

Equation (4.4.1) handles unmatched regions and ends of matches, only allowing matches to end when they have score at least T. Equation (4.4.2) deals with starts of matches and extensions. The total score of all the matches is obtained by adding an extra cell to the matrix, $M(n + 1, 0)$, using (4.4.1). This score will have T subtracted for each match; if there were no matches of score greater than T it will be 0, obtained by repeated application of first option in (4.4.1).

The individual match alignments can be obtained by tracing back from cell

$(n + 1, 0)$ to $(0, 0)$, at each point going back to the cell that was the source of the score in the current cell in the max operation. This traceback procedure is a global procedure, showing what each residue in x will be aligned to. The resulting global alignment will contain sections of more conventional gapped local alignments of subsequences of x to subsequences of y.

The algorithm obtains all local matches in one pass. It finds the maximal scoring set of matches, in the sense of maximizing the combined total of the excess of each match score above the threshold T. Changing the value of T changes what the algorithm finds. Increasing T may exclude matches. Decreasing it may split them, as well as finding new weaker ones. A locally optimal match in the sense of the preceding section will be split into pieces if it contains internal subalignemnts scoring less than $-T$. However, this may be what is wanted: given two similar high scoring sections significant in their own right, separated by a non-matching section with a strongly negative score, it is not clear whether it is preferable to report one match or two.

4.5 Overlap Matches Algorithm

If there are two different sequences: x = HEAGAWGHEE and y = PAWHEAE, an overlap matches means that a prefix or suffix of x must be aligned with a prefix or suffix of y:

```
GAWGHEE
PAW-HEA
```

This is similar to local alignment, but the alignment must begin on the left-hand or top border and must end on the right-hand or bottom border, that means an alignment cannot begin or end inside the matrix.

Overlap Matches Algorithm uses an algorithm similar to the algorithm of Wilbur and Lipman (1983) to compare one sequence (the query) to any group of sequences. We may think of the comparisons as a set of dot-plots with the query as the vertical sequence and the group of sequences to which the query is being compared as the different horizontal sequences (the search set). This often occurs when comparing fragments of genomic DNA sequence to each other, or to larger chromosomal sequences. Several different types of configuration can occur, as shown Figure 4.5.1 (a) and (b):

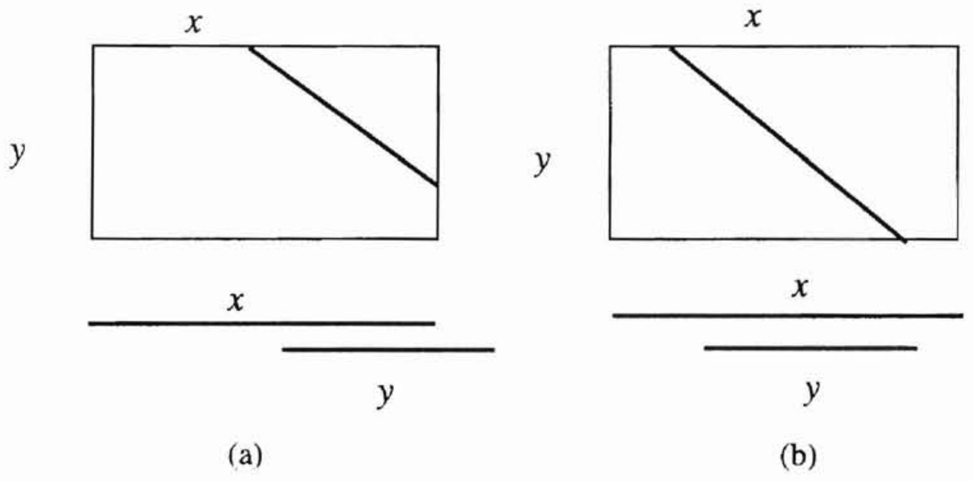

Figure 4.5.1 Different types of overlap matches

To find a global alignment that does not penalize overhanging ends a match starts on the top or left border of the matrix, and finishes on the right or bottom border. The initialization equations are therefore that $M(i, 0) = 0$ for $i = 1,\ldots,n$ and $M(0, j) = 0$ for $j = 1,\ldots,m$, and the recurrence relations within the matrix are simply those for a global alignment (See Equation 4.5.1). We set $M_{\max}$ to be the maximum value on the right border (i, m), $i = 1,\ldots,n$, and the bottom border (n, j), $j = 1,\ldots,m$. The traceback starts from the maximum point and continues until reaching the top or left edge.

There is a repeated match version of this overlap match algorithm, in which the analogues of (4.5.1) and (4.5.2) are:

$$M(i, 0) = \max \begin{cases} M(i-1, 0), \\ \\ M(i-1, m) - T; \end{cases} \qquad (4.5.1)$$

$$M(i, j) = \max \begin{cases} M(i-1, j-1) + S(x_i, y_j), \\ M(i-1, j) - \mathrm{d}, \\ M(i, j-1) - \mathrm{d}. \end{cases} \qquad (4.5.2)$$

The line (4.5.1) in the recurrence for $M(i, 0)$ is now just looking at complete matches to $y_{1\ldots m}$, rather than all possible subsequences of y as in (4.4.1) in the previous section. However, (4.4.1) is still used in its original form for obtaining $M(n + 1, 0)$, so that matches of initial subsequences of y to the end of x can be obtained.

4.6 Dynamic Programming and Linear Space Algorithm

The standard dynamic programming algorithm requires storage of at least on m x n matrix (m and n are the length of two sequences, respectively) in order to calculate the alignment. On modern computers, this is not a problem for alignment of sequences with short length, but for computing large DNA sequences or complete genomes, space requirements can be prohibitive.

Myers and Miller (1988) introduced a linear space algorithm for dynamic programming to overcome the space requirement for long sequence alignment. This algorithm gives the optimal alignment in limited memory of order $n + m$ rather than nm. This algorithm does not propagate the traceback pointer $c(i, j)$, but instead finds the alignment midpoint (u, v) by combining the results of forward and backward dynamic programming passes at row u. Figure 4.6 provides detailed pseudo-code for the linear space alignment algorithm.

```
**shared strings** $a_1a_2 \cdots a_M$, $b_1b_2 \cdots b_N$
**shared temporary integer arrays** $S^-[0\ldots N]$, $S^+[0\ldots N]$
**procedure** *Align(M, N)*
        **if** $M = 0$ **then**
                **for** $j \leftarrow 1$ to $N$ **do**
                        **write** $[\begin{smallmatrix}-\\ b_j\end{smallmatrix}]$
        **else**
                *path*(0, 0, *M*, *N*)

**recursive procedure** $path(i_1, j_1, i_2, j_2)$
        **if** $i_1 + 1 = i_2$ or $j_1 = j_2$ **then**
                **write** aligned pairs for maximum-score path from $(i_1, j_1)$ to $(i_2, j_2)$
        **else**
                $mid \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$

                /* find maximum path scores from (i1, j1) */
                $S^-[j_1] \leftarrow 0$
                **for** $j \leftarrow j_1 + 1$ to $j2$ **do**
                        $S^-[j] \leftarrow S^-[j-1] + \sigma([\begin{smallmatrix}-\\ b_j\end{smallmatrix}])$

                **for** $i \leftarrow i_1 + 1$ to *mid* **do**
                        $s \leftarrow S^-[j_1]$
                        $S^-[j_1] \leftarrow c \leftarrow S^-[j_1] + \sigma([\begin{smallmatrix}a_i\\ -\end{smallmatrix}])$
                        **for** $j \leftarrow j_1 + 1$ to $j_2$ **do**
                                $c \leftarrow \max\{ S^-[j_1] + \sigma([\begin{smallmatrix}a_i\\ -\end{smallmatrix}]), s + \sigma([\begin{smallmatrix}a_i\\ b_j\end{smallmatrix}]),$
                                                $c + \sigma([\begin{smallmatrix}-\\ b_j\end{smallmatrix}])\}$
                                $s \leftarrow S^-[j]$
                                $S^-[j] \leftarrow c$

                /* find maximum path scores to $(i_2, j_2)$ */
                $S^+[j_2] \leftarrow 0$
                **for** $j \leftarrow j_2 - 1$ **down to** $j_1$ **do**
        $S^+[j] \leftarrow S^+[j+1] + \sigma(\left[\begin{smallmatrix} - \\ b_{j+1} \end{smallmatrix}\right])$
for $i \leftarrow i_2 - 1$ **down to** *mid* **do**
        $s \leftarrow S^+[j_2]$
        $S^+[j_2] \leftarrow c \leftarrow S^+[j_2] + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ - \end{smallmatrix}\right])$
for $j \leftarrow j_2 - 1$ **down to** $j_1$ **do**
        $c \leftarrow \max\{S^+[j] + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ - \end{smallmatrix}\right]), s + \sigma(\left[\begin{smallmatrix} a_{i+1} \\ b_{j+1} \end{smallmatrix}\right]), c + \sigma(\left[\begin{smallmatrix} - \\ b_{j+1} \end{smallmatrix}\right])\}$
        $s \leftarrow S^+[j]$
        $S^+[j] \leftarrow c$

/* find where maximum-score path crosses row *mid* */
$j \leftarrow$ value $x$ $[j_1, j_2]$ that maximizes $S^-[x] + S^+[x]$
$path(i_1, j_1, mid, j)$
$path(mid, j, i_2, j_2)$
```

Figure 4.6 Linear space alignment algorithm (from Chao et al 1994)

In this project, we want to combine linear space alignment algorithm and dynamic programming methods to reduce the space complexity from $O(mn)$ to $O(m + n)$.

CHAPTER V

PROGRAM DESIGN AND IMPLEMENTATION

This project focuses on the dynamic programming and its application to pairwise biological sequence alignment. The program was written in Java language. The program design and implementation are described in this chapter. Details see as follows:

5.1 System Requirements

Table 5.1 summarizes system requirements for the user side and server side.

Server Side	• Unix System
User Side	• Any platform with the Java Virtual Machine

Table 5.1 System Requirement

5.2 Algorithms Used For Programming

Nine algorithms are covered in this thesis to demonstrate dynamic programming and its applications to pairwise biological sequence alignment. Table 5.2 shows the brief description of each algorithm used in the program.

Algorithms	**Description**
NWSimple	Global alignment with simple linear gap costs, using the Needlemam-Wunsch algorithm
SWSimple	Local alignment with simple linear gap costs, using the Smith-Waterman algorithm
RMSimple	Repeated Matches with simple linear gap costs
OMSimple	Overlap matches with simple linear gap costs
NWAffine	Global alignment with affine gap costs, using the Needlemam-Wunsch algorithm
NWSmart	Global alignment with simple linear gap costs. Where all the above implementations require space proportional to the product of the sequence lengths, this one requires

	only space proportional to the sum of the sequence lengths. It reconstructs the full alignment using a recursive divide-and-conquer algorithm, calling NWSimple to solve the base cases(when one of the sequences have length 1)
SWSmart	Local alignment with simple linear gap costs, using the smart linear space algorithm (as in NWSmart)
AlignSmartAffine	Global alignment with affine gap costs, using the smart linear space algorithm (as in NWAffine and NWSmart)
SWSmartAffine	Local alignment with affine gap costs, using the smart linear space alignment (as in AlignSmartAffine and SWSmart)

Table 5.2 Algorithms Used in Program

5.3 Input and Output

The program enables users to input two sequences to be compared and then the results of all algorithms running will be displayed. The format of input is as follows:

5.3.1 The Format of Input

Under the UNIX environment, at the prompt sign, just type java, and then followed by the program named PairAlign, then followed by the stings of two sequences which are going to be aligned:

```
$ java PairAlign EFHGHYYTRRICKQK AEGHYRICK
```

Press "Enter" key, the program will run immediately.

5.3.2 The Sample of Output

After the program run, the result will be output like the followings:

```
GLOBAL ALIGNMENT:
Score = 8
The F matrix:
   0   -8  -16  -24  -32  -40  -48  -56  -64  -72  -80  -88  -96 -104 -112 -120
  -8   -1   -9  -17  -24  -32  -40  -48  -56  -64  -72  -80  -88  -96 -104 -112
 -16   -2   -4   -9  -17  -24  -32  -40  -48  -56  -64  -72  -80  -87  -94 -102
 -24  -10   -6   -6   -1   -9  -17  -25  -33  -41  -49  -57  -65  -73  -81  -89
 -32  -18  -11    4   -4    9    1   -7  -15  -23  -31  -39  -47  -55  -63  -71
 -40  -26  -14   -4    1    1   17    9    1   -7  -15  -23  -31  -39  -47  -55
 -48  -34  -22  -12   -7    1    9   16    8    8    0   -8  -16  -24  -32  -40
```

```
   -56   -42   -30   -20   -15    -7     1     8    15     7     4     5    -3   -11   -19   -27
   -64   -50   -38   -28   -23   -15    -7     0     7    11     3     2    18    10     2    -6
   -72   -58   -46   -36   -30   -23   -15    -8    -1    10    14     6    10    24    16     8
An optimal alignment:
EFHGHYYTRRICKQK
A-EGH-Y--RIC--K

LOCAL ALIGNMENT:
Score = 33
The F matrix:
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     6     0     0     0     0     0     0     0     0     0     0     0     1     2     1
     0     0     2     0     8     0     0     0     0     0     0     0     0     0     0     0
     0     0     0    12     4    18    10     2     0     0     0     0     0     0     1     0
     0     0     4     4     9    10    26    18    10     2     0     0     0     0     0     0
     0     0     0     4     1     9    18    25    17    17     9     1     0     3     1     3
     0     0     0     0     0     1    10    17    24    16    13    14     6     0     0     0
     0     0     0     0     0     0     2     9    16    20    12    11    27    19    11     3
     0     1     0     0     0     0     0     1     8    19    23    15    19    33    25    17
An optimal alignment:
GHYYTRRICK
GH-Y--RICK

REPEATED MATCHES:
Score = 17
The F matrix:
     0     0     0     0     0     0     0     6     6     6     6     6     6    11    17    17
     0     0     0     0     0     0     0     6     6     6     6     6     6    11    17    17
     0     6     0     0     0     0     0     6     6     6     6     6     6    11    17    18
     0     0     2     0     8     0     0     6     6     6     6     6     6    11    17    17
     0     0     0    12     4    18    10     6     6     6     6     6     6    11    17    17
     0     0     4     4     9    10    26    18    10     6     6     6     6    11    17    17
     0     0     0     4     1     9    18    25    17    17    13     6     6    11    17    20
     0     0     0     0     0     1    10    17    24    16    13    18    10    11    17    17
     0     0     0     0     0     0     2     9    16    20    12    11    31    23    17    17
     0     1     0     0     0     0     0     6     8    19    23    15    23    37    29    23
An optimal alignment:
EFHGHYYTRRICKQK
...GHY...RICK..

OVERLAP MATCH:
Score = 30
The F matrix:
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0    -1    -3    -2     0    -2    -2    -2     0    -2    -2    -1    -1    -1    -1    -1
     0     6    -2    -3    -5     0    -4    -4    -3     0    -2    -6    -4     0     1     0
     0    -2     2    -4     5    -3    -3    -7    -6    -6    -3    -6    -9    -6    -2    -1
     0     0    -3    12     4    15     7    -1    -9    -6    -6    -7    -9    -9    -5    -2
     0    -2     4     4     9     7    23    15     7    -1    -7    -7   -10   -11   -10    -7
     0     0    -4     4     1     9    15    22    14    14     6    -2   -10    -7   -10    -7
     0    -4     0    -4     0     1     8    14    21    13    10    11     3    -5   -10   -13
     0    -3    -6    -3    -7    -3     0     6    13    17     9     8    24    16     8     0
     0     1    -7    -6    -5    -7    -5    -2     5    16    20    12    16    30    22    14
An optimal alignment:
FHGHYYTRRICK
AEGH-Y--RICK

AFFINE GLOBAL:
Score = 20
The F matrix:
F[0]:
     0  -Inf  -Inf
  -Inf    -1   -11
  -Inf    -2    -4
  -Inf   -15    -6
  -Inf   -16   -11
  -Inf   -22   -10
  -Inf   -24   -21
  -Inf   -32   -22
  -Inf   -35   -28
```

```
 -Inf  -35  -34
F[1]:
    0   -8  -12
 -Inf -Inf   -9
 -Inf -Inf  -10
 -Inf -Inf  -23
 -Inf -Inf  -24
 -Inf -Inf  -30
 -Inf -Inf  -32
 -Inf -Inf  -40
 -Inf -Inf  -43
 -Inf -Inf  -43
F[2]:
    0 -Inf -Inf
   -8 -Inf -Inf
  -12   -9  -19
  -16  -10  -12
  -20  -14  -14
  -24  -18  -18
  -28  -22  -18
  -32  -26  -22
  -36  -30  -26
  -40  -34  -30
An optimal alignment:
EFHGHYYTRRICKQK
A-EGHY---RIC--K

SMART GLOBAL:
Score = 8
The F matrix:
 -112 -120
 -104 -112
  -94 -102
  -81  -89
  -63  -71
  -47  -55
  -32  -40
  -19  -27
    2   -6
   16    8
An optimal alignment:
EFHGHYYTRRICKQK
A-EGH-Y--RIC--K

SMART LOCAL:
Score = 33
The F matrix:
    0    0
    0    0
    2    1
    0    0
    1    0
    0    0
    1    3
    0    0
   11    3
   25   17
An optimal alignment:
GHYYTRRICK
GH-Y--RICK

SMART AFFINE LOCAL:
Score = 41
The F matrix:
F[0]:
    0    0
    0    0
    2    1
    0    0
    1    0
    0    0
```

```
     1    3
     0    0
     7    3
    29   29
F[1]:
     0    0
    -8   -8
    -7   -6
    -8   -8
    -8   -7
    -8   -8
    -3   -7
     6    2
    23   19
    33   29
F[2]:
     0    0
    -4   -4
    -8   -8
    -6   -7
    -8   -8
    -7   -8
    -8   -8
    -7   -5
    -8   -8
    -1   -5
An optimal alignment:
GHYYTRRICK
GHY---RICK
```

From the output, user may easily find out the result of different algorithms' running.

5.4 Implementation

This project implements dynamic programming and its application to pairwise biological sequence alignment. The program contains 21 classes for different purposes, the main classes and their descriptions are shown in Table 5.4.1.

Main Classes	Description
Substitution	The class of substitution (scoring) matrices
Blosum50	The BLOSUM50 substitution matrix for amino acids
Align	Implements pairwise sequence alignment
AlignSimple	Implement alignment with simple gap costs

Traceback	Implements traceback objects
Traceback2	Implements traceback objects for simple gap costs
Traceback3	Implements traceback3 objects for Affine gap costs
NW	Implements global alignment with the Needleman-Wunsch algorithm (simple gap costs)
SW	Implements local alignment with the Smith-Waterman algorithm (simple gap costs)
RM	Implements repeated matches (simple gap costs)
OM	Implements overlap matching (simple gap costs)
AlignAffine	Implements alignment with affine gap costs
NWAffine	Implements global alignment using the Needleman-Wunsch algorithm (affine gap costs)
AlignSmart	Implements alignment with simple gap costs; smart linear-space algorithm
NWSmart	Implements global alignment (simple gap costs, smart linear-space algorithm)
SWSmart	Implements local alignment with the Smith-Waterman algorithm (simple gap costs, smart linear space algorithm)
AlignSmartAffine	Implements alignment with affine gap costs; smart linear-space algorithm
SWSmartAffine	Implements local alignment with the Smith-Waterman algorithm (affine gap costs, smart linear space algorithm)
PairAlign	Test all eight alignment algorithms and call other classes
Output	Auxiliary classes for output
SystemOut	Called by Output class

Table 5.4.1 Description of main classes

Some of the implementations of algorithms are as follows:

5.4.1 Implementation of Global Alignment Algorithm

5.4.1.1 Initialization

Upper border position $(i, 0)$ represents the alignment of $x_{1...i}$ to the empty prefix of y. That is the prefix $x_{1...i}$ has been matched with gaps in y. With the simple linear gap costs, the score is $-d * i$. The traceback pointer at $(i, 0)$ points to $(i - 1, 0)$, and the same thing happens on the left-hand border. So the border can be initialized as follows:

```
for (int i=1; i<=n; i++) {
      F[i][0] = -d * i;
      B[i][0] = new Traceback2(i-1, 0);
    }
    for (int j=1; j<=m; j++) {
      F[0][j] = -d * j;
      B[0][j] = new Traceback2(0, j-1);
    }
```

5.4.1.2 Filling in the Matrix

Position (i, j) may be reached:

- From $(i-1, j-1)$ with a match, adding *score* $[x_i][y_j]$ to the score;
- From $(i-1, j)$ with a gap in y, substracting d from the score; or
- From $(i, j-1)$ with a gap in x, substracting d from the score.

The traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$. Thus:

```
for (int i=1; i<=n; i++)
      for (int j=1; j<=m; j++) {
            int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
            int val = max(F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d);
            F[i][j] = val;
            if (val == F[i-1][j-1]+s)
                  B[i][j] = new Traceback2(i-1, j-1);
            else if (val == F[i-1][j]-d)
                  B[i][j] = new Traceback2(i-1, j);
            else if (val == F[i][j-1]-d)
                  B[i][j] = new Traceback2(i, j-1);
        }
    B0 = new Traceback2(n, m);
```

The start B0 of the traceback is cell (n, m).

5.4.2 Implementation of Local Alignment Algorithm

5.4.2.1 Initialization

The position of upper border $(i, 0)$ represents the alignment of a suffix $x_{1 \ldots i}$ to an empty sequence. An empty match, with score 0, is the best we can do (provided gaps have negative scores). Then $(i, 0)$ is the start of a new local alignment, and the traceback pointer at $(i, 0)$ points nowhere. The left-hand border is similar. So the border cells can be initialized to 0 and the traceback to null (this require no action in Java).

5.4.2.2 Filling in the Matrix

Positions (i, j) may be reached:

- From nowhere, with score 0, because we can always start a local alignment;
- From $(i-1, j-1)$ with a match, adding *score* $[x_i][y_j]$ to the score;
- From $(i-1, j)$ with a gap in y, substracting d from the score; or
- From $(i, j-1)$ with a gap in x, substracting d from the score.

The traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$, if any. Thus:

```
for (int i=1; i<=n; i++)
      for (int j=1; j<=m; j++) {
            int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
            int val = max(0, F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d);
            F[i][j] = val;
            if (val == 0)
                  B[i][j] = null;
            else if (val == F[i-1][j-1]+s)
                  B[i][j] = new Traceback2(i-1, j-1);
            else if (val == F[i-1][j]-d)
                  B[i][j] = new Traceback2(i-1, j);
            else if (val == F[i][j-1]-d)
                  B[i][j] = new Traceback2(i, j-1);
      }
}
```

The start B0 of the traceback must be set some cell (i, j) in F with maximal score.

5.4.3 Implementation of Repeated Matches Algorithm

5.4.3.1 Initialization

The position $(0, j)$ on left-hand border represents the best alignment of an empty subsequence of x to a subsequence of y. This must have score 0. The traceback pointer at $(0, j)$ points nowhere.

5.4.3.2 Filling in the Matrix

Position $(i, 0)$ may be reached:

- From $(i - 1, 0)$ by letting x_i be unmatched be nay part of y, keeping the old score; or
- From $(i - 1, j)$ by completing a match score is at least T, substracting T from that score.

Position (i, j) for $j > 0$ may be reached:

- From $(i, 0)$, because we start a new local alignment, keeping the old score;
- From $(i - 1, j - 1)$ with a match, adding score $[x_i][y_j]$ to the score;
- From $(i - 1, j)$ with a gap in y, substracting d from the score; or
- From $(i, j - 1)$ with a gap in x, substracting d from the score.

As always, the traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$.

Let `maxj(i - 1)` be $j > 0$ if $F(i-1, j) - T$ is greater than $F(i-1, 0)$ and maximal; otherwise 0. This gives:

```
for (int i=1; i<=n; i++) {
      int maxj = maxj(i-1);
      F[i][0] = maxjval(i-1, maxj);
      B[i][0] = new Traceback2(i-1, maxj);
      for (int j=1; j<=m; j++) {
            int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
            int val = max(F[i][0], F[i-1][j-1]+s, F[i-1][j]-d,
                  F[i][j-1]-d);
            F[i][j] = val;
            if (val == F[i][0])
                  B[i][j] = new Traceback2(i, 0);
            else if (val == F[i-1][j-1]+s)
                  B[i][j] = new Traceback2(i-1, j-1);
            else if (val == F[i-1][j]-d)
                  B[i][j] = new Traceback2(i-1, j);
            else if (val == F[i][j-1]-d)
                  B[i][j] = new Traceback2(i, j-1);
      }
}
```

The start B0 of the traceback is (n, `maxj(n)`). That is, (n, j) if there is a last match with score $> T$; otherwise $(n, 0)$, if some suffix of x is unmatched.

5.4.4. Implementation of Overlap Matches Algorithm

5.4.4.1 Initialization

The position $(0, j)$ on left-hand border represents the best alignment of an empty subsequence of x to a subsequence of y. Similarly, the position $(i, 0)$ on right-hand border represents the best alignment of a subsequence of x to an empty subsequence of y. The traceback pointers at either $(0, j)$ or $(i, 0)$ point nowhere.

5.4.4.2. Filling in the Matrix

5.4.4.2.1 Finding Maximal Score on Left-hand and upper Borders

```
for (int i=1; i<=n; i++)
      for (int j=1; j<=m; j++) {
            int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
            int val = max(F[i-1][j-1]+s, F[i-1][j]-d,
                        F[i][j-1]-d);
            F[i][j] = val;
            if (val == F[i-1][j-1]+s)
                  B[i][j] = new Traceback2(i-1, j-1);
            else if (val == F[i-1][j]-d)
                  B[i][j] = new Traceback2(i-1, j);
            else if (val == F[i][j-1]-d)
            B[i][j] = new Traceback2(i, j-1);
  }
```

5.4.4.2.2 Finding Maximal Score on Right-hand and Bottom Borders

```
      int maxi = -1, maxj = -1;
      int maxval = NegInf;
      for (int i=0; i<=n; i++)
            if (maxval < F[i][m]) {
            maxi = i;
            maxval = F[i][m];
            }
                  for (int j=0; j<=m; j++)
                  if (maxval < F[n][j]) {
                  maxj = j;
                  maxval = F[n][j];
            }
            if (maxj != -1)
            B0 = new Traceback2(n, maxj);
            else
            B0 = new Traceback2(maxi, m);
  }
}
```

CHAPTER VI

SUMMARY, CONCLUSIONS AND FUTURE WORK

This project implements nine dynamic programming algorithms and their application in biological sequence alignment. Sequence alignment is a very important tool in analysis of the structures and functions of DNA and protein molecules, so the selection of better algorithms for aligning those sequences is very critical to get an optimal result.

This study focuses on dynamic programming algorithms because dynamic programming algorithms are guaranteed to find the optimal scoring alignment or set of alignments. In this project, global alignment algorithm, local alignment algorithm, repeated alignment algorithm and overlap alignment algorithm are discussed and analyzed extensively.

A program is designed and implemented to demonstrate the output of each algorithm with different gap costs. In this program, the time complexity and space complexity of each algorithm with simple linear gap costs are $O(mn)$. If affine gap costs is used, the time and space complexity remain same as $O(mn)$, but it is desirable when gaps of a few residues are expected almost as gaps of a single residue because affine gap costs allows long insertions and deletions to be penalized less than what they would be by linear gap cost. The time complexity of each algorithm with linear space algorithm is $O(mn)$, but the space complexity is reduced to $O(m + n)$. Table 6.1 shows the different running and space complexity among the algorithms used:

Algorithm Name	Gap Penalty Used	Space Algorithm	Time Required	Space Required
NWSimple	Constant Gap	Regular	$O(N^2)$	$O(N^2)$
SWSimple	Constant Gap	Regular	$O(N^2)$	$O(N^2)$
RMSimple	Constant Gap	Regular	$O(N^2)$	$O(N^2)$
OMSimple	Constant Gap	Regular	$O(N^2)$	$O(N^2)$
NWAffine	Affine Gap	Regular	$O(N^2)$	$O(N^2)$
NWSmart	Constant Gap	Linear Space	$O(N^2)$	$O(N)$
SWSmart	Constant Gap	Linear Space	$O(N^2)$	$O(N)$
SWSmartAffine	Affine Gap	Linear Space	$O(N^2)$	$O(N)$

Table 6.1 Performance of the algorithms used in the program

The advantages of the designed program are:

- Gives all output of each dynamic programming algorithm with simple gap costs, affine gap costs and linear space algorithm, respectively.
- Can easily find the repeated matches and overlap matches between two biological sequences
- Because it is written in Java, the implementation is modular so that new scoring matrices can be used and new alignment algorithms can be developed very easily.

The program is designed to run under UNIX. In the future, a Java Applet should be added in the program so that any user can run the program through web browser such as Netscape and Microsoft IE, and different scoring matrices might be changed for different alignment purpose.

Appendix A

Glossary

α-helix A secondary structure in proteins; the right-handed helical folding of a polypeptide such that amide nitrogens share their hydrogen atoms with the carbonyl oxygens of the fourth amide bonds towards the C-terminal end of the polymer.

amino acid Any of a class of 20 molecules that are combined to form proteins in living things. The sequence of amino acids in a protein and hence protein function are determined by the genetic code.

β-sheet A form of secondary structure of a protein in which the amide hydrogens of a peptide bond of one extended polypeptide sequence are shared with the carbonyl oxygens of a peptide bond on a second polypeptide sequence. A sheet that often consists of three or more polypeptide sequences is said to be parallel (i.e. both adjacent strands run in the same direction; N- to C-terminal) or anti-parallel.

base sequence analysis A method, sometimes automated, for determining the base sequence.

bioinformatics The study of the application of computer and statistical techniques to the management of biological information. In genome projects, bioinformatics includes the development of methods to search databases quickly, to analyze DNA sequence information, and to predict protein sequence and structure from DNA sequence data.

BLAST BLAST (Basic Local Alignment Search Tool) is a popular program for searching biosequences against databases. BLAST was developed and is maintained by a group at the National Center for Biotechnology Information (NCBI).

cDNA Complementary DNA; DNA that is synthesized, by reverse transcriptase, from an mRNA template, and therefore has no introns.

cDNA library A collection of cells, usually E. *coli*, transformed by DNA vectors each of which contains a different cDNA insert synthesized from a collection of mRNA species.

complementary sequences: Nucleic acid base sequences that can form a double-stranded structure by matching base pairs; the complementary sequence to G-T- A- C is C- A- T- G.

conserved sequence A base sequence in a DNA molecule (or an amino acid sequence in a protein) that has remained essentially unchanged throughout evolution.

DNA Deoxyribonucleic acid; a macromolecule formed of repeating deoxyribonucleotide units linked by phosphodiester bonds between the 5'-phosphate group of one nucleotide and the 3'-hydroxy group of the next. DNA appears in Nature in both double-stranded (the Watson-Crick model) and single-stranded forms, and functions as a repository of genetic information that is encoded in its base sequence.

DNA sequence The relative order of base pairs, whether in a fragment of DNA, a gene, a chromosome, or an entire genome. See base sequence analysis.

domain A discrete portion of a protein with its own function. The combination of domains in a single protein determines its overall function.

exons The protein- coding DNA sequences of a gene. Compare introns.

FASTA FASTA is a similarity search program which can be used to search a nucleotide sequence database with a nucleotide query sequence, or a protein sequence database with a protein query sequence. Its companion program TFastA (or FastA-Trans) is used to search a 6 frames translation of a nucleotide sequence database with a protein query sequence. FASTA accelerates database searching by using several passes over the database and only retaining a `best matching' subset for further analysis at each pass, therefore `pruning down' the database progressively.

folding Also protein folding, the process of newly synthesized protein forming a certain 3D structure so that it can function correctly.

gene The fundamental physical and functional unit of heredity. A gene is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e., a protein or RNA molecule). See gene expression.

gene expression The process by which a gene coded information is converted into the structures present and operating in the cell. Expressed genes include those that are transcribed into mRNA and then translated into protein and those that are transcribed into RNA but not translated into protein (e.g., transfer and ribosomal RNAs).

genetic code The sequence of nucleotides, coded in triplets (codons) along the mRNA, that determines the sequence of amino acids in protein synthesis. The DNA sequence of a gene can be used to predict the mRNA sequence, and the genetic code can in turn be used to predict the amino acid sequence.

genomic DNA DNA that has been isolated from a cell and therefore contains introns, as opposed to cDNA.

genomic library A collection of transformed cells, each of which contains DNA fragments; the entire population represents the total genome of an organism, e.g., a rat library containing DNA fragments which together comprise the entire rat genome. Appropriate screening methods can select a single transformed cell that contains a specific gene.

homology Sequence similarity which is attributed to evolutionary descent from a common ancestor. In molecular biology, homology is often inferred from a high degree of sequence similarity. Sequence similarity does not necessarily infer homology. In general, if two sequences are longer than 100 residues and are more than 25% identical (after suitable gapping), they are very likely homologous. If two sequences are less than 15% identical, they are probably not homologous.

introns The DNA base sequences interrupting the protein- coding sequences of a gene; these sequences are transcribed into RNA but are cut out of the message before it is translated into protein. Compare exons.

messenger RNA (mRNA) RNA that serves as a template for protein synthesis. See genetic code.

PAM An acronym for Percent Accepted Mutations. 1 PAM means there has been 1 mutation per 100 residues. 250 PAM means there has been 250 mutations per 100 residues or 2.5 mutations per residue. The PAM concept was developed by M.O. Dayhoff in the 1960s to measure the evolutionary pressure that had been placed on a protein sequence.

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

RNA Ribonucleic acid; a macromolecule formed of repeating ribonucleotide units linked by phosphodiester bonds between the 5'-phosphate group of one nucleotide and the 3'-hydroxy group of the next. RNA has several biological functions, most of which depend upon its ability to form sequence-specific interactions with DNA. RNA comprises the genome of some viruses.

sequence The order of nucleotide bases in a DNA molecule or the order of amino acid in protein molecule.

Appendix B

Table of acronyms and abbreviations

Acronym or Abbreviation	Meaning
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
ALIGN	a program used for sensitive sequence comparison
BLAST	basic local alignment search tool
BLOCKS	a database of protein motifs
BLOSUM	an amino acid substitution matrix derived from BLOCKS
cDNA	complementary or copy DNA
CLUSTAL	a program used for profile-based progressive multiple alignment
DNA	deoxyribonucleic acid
dNTP	a mixture of dATP (A), dGTP (G), dCTP (C), and dTTP (T)
FASTA	a program for sequence alignment, short for "fast-all"
GCG	genetic computer group
MOTIFS	a recurring pattern of protein supersecondary structure
MSA	a program used for multiple sequences alignment
PAM	point accepted mutation, used as a unit of evolutionary distance
PROSITE	a regular expression database for significant patterns in protein
RNA	ribonucleic acid

REFERENCES

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K. and Watson J.D. (1989) *Molecular Biology of the Cell.* Garland Publishing, New York and London.

Altschul S.F., Carroll R.J. and Lipman D.J. (1989) Weights for data related by a tree. *Journal of Molecular Biology* 207:647-753.

Altschul S.F. (1991) Amino acid substitution matrices from an information theoretic perspective. *Journal of Molecular Biology* 219:555-565.

Altschul S.F., Gish W., Miller W., Myers E.W. and Lipman D.J. (1990) Basic local alignment search tool. *Journal of Molecular Biology* 215:403-410.

Argos P. (1987) A sensitive procedure to compare amino acid sequences. *Journal of Molecular Biology* 193:385-396.

Baldi P., Chauvin Y., Hunkapiller T. and McClure M.A. (1994) Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Science of the USA* 91:1059-1063.

Barton G.J. and Sternberg M.J.E. (1987a) A strategy for the rapid multiple alignment of protein sequences. *Journal of Molecular Biology* 198:327-337.

Barton G.J. and Sternberg M.J.E. (1987b) Evaluation and improvements in the automatic alignment of protein sequences. *Protein Engineering* 1:89-94.

Barton G.J. (1993) An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps. *Computer Applications in the Biosciences* 9:729-734.

Bellman R. E. (1957) *Dynamic programming*. Princeton University Press.

Boswell D.R. and Mclachlan A.D. (1984) Sequence comparison by exponentially damped alignment. *Nucleic Acid Research* 12:457-464.

Chao K. M., Hardison R. C. and Miller W. (1994) Recent developments in linear-space alignment methods: A Survey. *Journal of Computational Biology.* 1:271-291.

Chou P.Y. and Fasman G.D. (1978) Prediction of the secondary structure of proteins from their amino acid sequences. *Advances in Enzymology* 47:45-148.

Cohen F.E., Novotny J., Sternberg M.J.E., Campbell D.G. and Williams A.F. (1981)

Analysis of structural similarities between brain Thy-1 antigen and immunoglobulin domains: evidence for an evolutionary relationship and a hypothesis for its functional significance. *Biochemical Journal* 195:31-40.

Collins J.F. and Coulson A.F W. (1987) In Bioshop M.J. and Rawlings C.J. (ed.), *Nucleic acid and protein sequence analysis – A practival approach*, pp 323-358 IRL Press.

Collins J.F., Coulson A.F.W. and Lyall A. (1988) The significance of protein sequence similarities. *Computer Applications in the Biosciences* 4:67-71.

Cooper N. (1994) *The Human Genome Project*. Univ. Science Books, Mill Valley, CA.

Dayhoff M.O., Schwartz R.M. and Orcutt B.C. (1978) A model of evolutionary change in proteins. matrices for detecting distant relationships. In Dayhoff M.O. (ed.), *Atlas of protein sequence and structure*, 5:345-358 National biomedical research foundation Washington DC.

Doolittle, R.F. (1986) Of Urfs and Orfs: *A primer on how to analyze derived amino acid sequences*. University Science Books, Mill Valley, CA.

Feng D.F., Johnson M.S. and Doolittle R.F. (1985) Aligning amino acid sequences: Comparison of commonly used methods. *Journal of Molecular Evolution* 21:112-125.

Feng D.F. and Doolittle R.F. (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* 25:351-360.

Fitch W.M. (1966) An improved method of testing for evolution homology. *Journal of Molecular Biology* 16:9-16.

Gibbs A.J. and McIntyre G.A. (1970) The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences. *European Journal of Biochemistry* 16:1-11.

Goad, W. (1986) Computational analysis of genetic sequences. *Annual Review in Biophysics and Chemistry* 15:79-95.

Gonnet G.H., Cohen M.A. and Benner S.A. (1992) Exhaustive matching of the entire protein sequence database. *Science* 256:1443-1445.

Gotoh O. (1982) An improved algorithm for matching biological sequence. *Journal of Molecular Biology* 162:705-708.

Gusfield D. (1997) *Algorithms on strings, trees, and sequences: Computer science and* *computational biology*. Cambridge University Press.

Heijne, G. (1987) *Sequence analysis in molecular biology: Treasure trove or trivial pursuit?* Academic Press, London,

Henikoff S. and Henikoff J.G. (1992) Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the USA* 89:10915-10919.

Henikoff S. and Henikoff J.G. (1993) Performance evaluation of amino acid substitution matrices. *Proteins* 17:49-61.

Higgins D.G. and Fuchs R. (1992) CLUSTAL V: improved software for multiple sequence alignment. *Computer Applications in the Biosciences* 8:189-191.

Hudson T.J., Lander E.S. (1995) An STS-based map of human genome. *Science*, 270:1945-1954.

Jones D.D. (1975) Amino acid properties and side-chain orientation in protein: A cross correlation approach. *Journal of Theoretical Biology* 50:167-183.

Jones D.T., Taylor W.R. and Thornton J.M. (1992) The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences* 8:275-282.

Krogh A., Brown M., Mian I.S., Olander K. and Haussler D. (1994) Hidden Markov models in computational biology: applications to protein modeling. *Journal of Molecular Biology* 235:1501-1531.

Kruskal J.B. (1983) An overview of sequence comparison, In Sankoff D. and Kruskal J.B., (ed.) *Time warps, string edits and macromolucules: The theory and practice of sequence comparison*, pp1-44, Addison Wesley.

Kubota Y., Nishikawa K., Takahashi S. and Ooi T. (1982) Correspondence of homologies in amino acid sequence and tertiary structure of protein molecules. *Biochemical Et Biophysical Acta* 701:242-252.

Lee, C. (1999) Bioinformatics Interdisciplinary Program Proposal. *http://www.doe-mbi.ucla.edu/people/leec/UCLA-bioinf/bioinf-RFC.html*

Lesk, A. (1988) ed.: *Computational molecular biology: sources and methods for* *sequence analysis*. Oxford University Press, Oxford,

Lipman D.J., Altschul S.F. and Kececioglu J.D. (1989) A tool for multiple sequence alignment. *Proceedings of National Academy of Sciences of the USA* 86:4412-4415.

Martinez H.M. (1988) A flexible multiple sequence alignment program. *Nucleic Acids Research* 16:1683-1691.

McClure M.A., Vasi T.K. and Fitch W.M. (1994) Comparative analysis of multiple protein sequence alignment methods. *Journal of Molecular Evolution* 11:571-592.

McLachlan A.D. (1972) Repeating sequences and gene duplication in proteins. *Journal Molecular Biology* 64:417-437.

Myers E. W. and Miller W. (1988) Optimal alignments in linear-space. *Computer Applications in the Biosciences* 4:11-17.

Needleman S.B. and Wunsch C.D. (1970) A general method application to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48:443-453.

Overington J., Johnson M.S., Sali A. and Blundell T.L. (1990) Tertiary structural constraints on protein evolutionary diversity: templates, key residues and structure prediction. *Proc. R. Soc. Lond. B Biol. Sci.*, 241:132-145.

Pearson, W. R. (1996) Effective protein sequence comparison. *Methods in Enzymology* 266:227-258.

Risler J.L., Delorme M.O., Delacroix H. and Henaut A. (1988) Amino acid substitutions in structurally related proteins: a pattern recognition approach, determination of a new and efficient scoring matrix. *Journal of Molecular Biology* 204:1019-1029.

Schuler G.D., Altschul S.F. and Lipman D.J. (1991) A workbench for multiple alignment construction and analysis. *Proteins* 9:180-190

Schuler G.D., Lander E.S. and Hudson T.J. (1996) A gene map for the human genome. *Science*, 274:540-546.

Schwartz R.M. and Dayhoff M.O. (1978) In Dayhoff M.O. (ed.), *Atlas of protein sequence and structure*, 5:353-362 National biomedical research foundation Washington DC.

Sellers P.H. (1984) Pattern recognition in genetic sequences by mismatch density. *Bulletin in Mathematical Biology* 46:705-708.

Smith R.F. and Smith T.F. (1990) Automatic generation of primary sequence patterns from sets of related protein sequences. *Proceedings of the National Academy of Sciences of the USA* 87:118-122.

Smith T.F. and Waterman M.S. (1981) Identification of common molecular subsequences. *Journal of Molecular Biology* 147:195-197.

Smith T.F. and Waterman M.S. (1981) Comparison of bio-sequence. *Advances in Applied Mathematics* 2:482-489.

Stryer, L. (1988) *Biochemistry*. W.H. Freeman, New York.

Subbiah S. and Harrison S.C. (1989) A method for multiple sequence alignment with gaps. *Journal of Molecular Biology* 209:539-548.

Taylor W.R. (1986) The classification of amino acid conservation. *Journal of Theoretical Biology* 119:205-218.

Taylor W.R. (1987) Multiple sequence alignment by a pairwise algorithm. *Computer Applications in the Biosciences* 3:81-87.

Vingron M. and Argos P. (1991) Motif recognition and alignment for many sequences by comparison of dot-matrices. *Journal of Molecular Biology* 218:33-43.

Waterman M.S., Smith T.F. and Beyer W.A. (1976) Some biological sequence metrics. *Advances in Mathematics*, 20:367-387.

Waterman M.S. and Jones R. (1990) Consensus methods for DNA and protein sequences alignment, in *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, (eds. Doolittle R.F.) 183:221-237, Academic Press, Inc., San Diego.

Wilbur W.J. and Lipman D.J. (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences of the USA* 80: 726-730

VITA

Yanwen Guo

Candidate for the Degree of

Master of Science

Thesis: DYNAMIC PROGRAMMING AND ITS APPLICATION TO PAIRWISE BIOLOGICAL SEQUENCE ALIGNMENT

Major Field: Computer Sciences

Biographical:

Personal Information: Born in Jinling, Hubei Province, the People's Republic of China, on October 15, 1966, the son of Fajun Yuan and Zhonghui Long.

Education: Graduated from the First High School, Gongan County, Hubei, China, in July 1983; received Bachelor of Science degree in Cell Biology from Wuhan University, Wuhan, Hubei, China, in July 1987; was Ph.D candidate in Biochemistry at The Chinese University of HongKong from January 1994 to November 1995. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 2000.

Professional Experience: Research Assistant and Associate, Shanghai Institute of Cell Biology, Academia Sinica, from July 1987 to July 1992. Visiting Scholar, The Chinese University of HongKong, from September 1992 to March 1993. Ph.D candidate at The Chinese University of HongKong, from January 1994 to November 1995. Laboratory Technologist at Oklahoma State University, from July 1996 – present.

Professional Memberships: Association for Computing Machinery