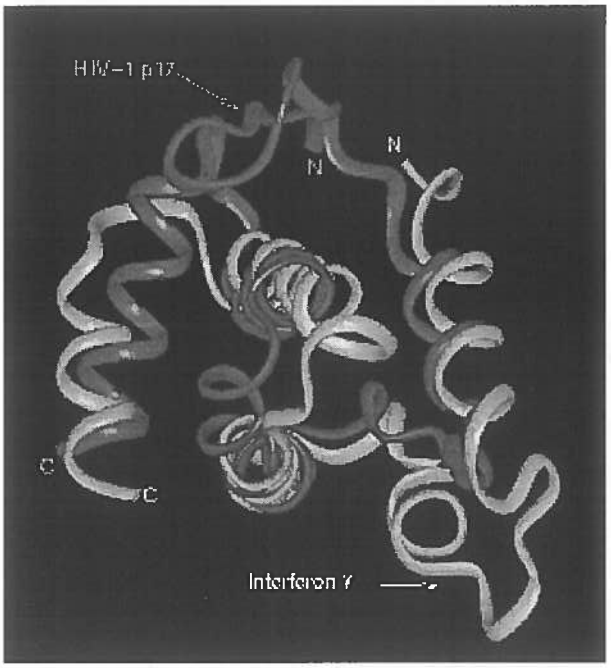
Optimisation of Amino Acid Scoring Matrices for Improved Protein Homologue Detection

Ketan Patel



MSc in Artificial Intelligence
Department of Artificial Intelligence
University of Edinburgh
1998

Abstract

The problem of protein structure prediction remains one of the major unsolved problems in structural biochemistry. The most successful method to date for predicting protein tertiary structure from primary sequence data, is homology modelling based on alignment with similar sequences of known structure. The premier component utilised in this process is a scoring matrix which determines how similar one protein is to another.

The aim of this work is to improve upon the scoring matrices currently used, in an effort to detect proteins which are more distantly related to each other. To this end an optimisation approach was taken, using a type of evolutionary algorithm. Experiments were conducted to evolve both a general scoring matrix, and a matrix which only detects a specific class of protein.

The results showed that although the optimisation approach works, the matrices derived were not superior to those already in use, however they were competitive. Good results were achieved when optimising matrices to detect a particular class of protein, and this suggests that a combination of such matrices could be used as a set when performing homology searches.

Acknowledgements

I would first of all like to thank my supervisors, Bob Fisher, Andrew Tuson and Andrew Coulson for their excellent guidance, advice and support throughout the project. Thanks also to Shane Sturrock from the Biocomputing Research Unit for giving me access to a fast computer and the use of the `sss_align` package. I would also like to thank Dr. Stephen Altschul who helped me out on tricky theoretical questions. Finally I wish to thank EPSRC for funding me.

Contents

1 Introduction **1**
- 1.1 The Problem . . . 1
- 1.2 Solving the problem by optimisation . . . 3
  - 1.2.1 Aim of optimising scoring systems . . . 3
- 1.3 The evolutionary approach to optimisation . . . 3
- 1.4 Expectations and results . . . 4

2 Background to protein biochemistry **5**
- 2.1 Proteins, what are they? . . . 5
  - 2.1.1 Chemistry . . . 6
  - 2.1.2 Structure . . . 8
- 2.2 Importance of identifying protein structure . . . 9
  - 2.2.1 Medicine . . . 9
  - 2.2.2 Agriculture . . . 10
  - 2.2.3 Industry . . . 11
- 2.3 Protein homology modelling . . . 11
- 2.4 Searching protein databases . . . 12
- 2.5 Summary . . . 13

3 Review of literature **15**
- 3.1 Overview of scoring systems . . . 15
- 3.2 Dynamic programming algorithms . . . 17
  - 3.2.1 Global alignment . . . 17
  - 3.2.2 Local alignment . . . 20
- 3.3 Methods for deriving scoring matrices . . . 21
  - 3.3.1 Genetic code scoring . . . 21
  - 3.3.2 Chemical similarity scoring . . . 22
  - 3.3.3 Observed substitution scoring schemes . . . 22
- 3.4 Comparison of matrices . . . 26
  - 3.4.1 What makes a good matrix? . . . 28
  - 3.4.2 Different matrices for different tasks? . . . 29

3.5 Summary 29

4 Design of the optimiser **31**
4.1 Problem formulation 31
4.2 Choosing a suitable algorithm 32
4.3 Differential evolution 34
4.3.1 The DE algorithm 34
4.4 Design of representation and operators 36
4.4.1 Encoding of matrix features 37
4.4.2 Move operators 37
4.5 Evaluation of weight matrices 38
4.6 Parameters 38
4.7 Implementation 39
4.7.1 Hardware 39
4.7.2 Software 39
4.8 Summary 40

5 Design of the evaluation function **41**
5.1 Possible approaches to evaluating matrices 41
5.1.1 Using statistical models 42
5.1.2 Using results of database searches 43
5.2 Final choice of evaluation function 44
5.2.1 How it works 45
5.2.2 The sss_align program 48
5.2.3 Evaluation time 48
5.3 Subset selection methods 48
5.3.1 Subset size 50
5.4 Training data 51
5.4.1 Which proteins should we include? 51
5.4.2 The SCOP database 51
5.5 Preparation of training data 53
5.6 Problems with training data 54
5.7 Summary 54

6 Results **55**
6.1 Experimental approach 55
6.2 Choice of parameters 56
6.2.1 Preliminary experiments 56
6.3 Results 58
6.3.1 Results for first set of experiments 59
6.3.2 Discussion 59

6.3.3 Validation tests for first set of experiments 60
6.3.4 Discussion of test results 60
6.3.5 Results for second set of experiments 61
6.3.6 Discussion 62
6.3.7 Validation tests for second set of experiments 62
6.3.8 Discussion of test results 63
6.4 Analysis of results 65
6.4.1 Significance of results 67
6.5 Summary 68

7 Conclusion **69**
7.1 Aims and achievements 69
7.1.1 Objectives 69
7.1.2 Achievements 70
7.2 Significant results 70
7.3 Further work 72
7.4 Concluding remarks 74

Appendices **81**

A The best evolved matrices **83**

B Training and test data **87**
B.1 Training data for general matrix 87
B.2 Test data for general matrix 89
B.3 Training data for specific matrix: SCOP class 1 91
B.4 Testing data for specific matrix: SCOP class 1 92
B.5 Training data for specific matrix: SCOP class 3 93
B.6 Testing data for specific matrix: SCOP class 3 95

C Glossary **97**

List of Figures

2.1 Illustration of the basic information flow within a cell 6
2.2 A small polypeptide chain 7
2.3 The three dimensional structure of Hemoglobin. 10

3.1 The difference between Global and Local Alignment. 18
3.2 Array used to perform dynamic programming algorithm 20
3.3 A Phylogenetic tree. 24
3.4 Plots of results from a database search 30

5.1 The evaluation process 49

6.1 Comparison of convergence with and without seed matrix . . . 57
6.2 Graph showing convergence for run2 (SCOP class 1) 62

List of Tables

2.1 Single letter codes for each amino acid. 9
2.2 The PAM250 scoring matrix 14

6.1 Parameters for the first set of experiments 58
6.2 Parameters for the second set of experiments 58
6.3 Results for the first set of experiments using the training data 59
6.4 Test results for the general set of matrices. 61
6.5 Results for SCOP Class 1 experiments on training data 63
6.6 Results for SCOP Class 3 experiments on training data 64
6.7 Results for SCOP Class 1 experiments on the test data 65
6.8 Results for SCOP Class 3 experiments on the test data 66

A.1 The best general scoring matrix (evolved in run5) 84
A.2 The best scoring matrix for SCOP class 1 85

Chapter 1

Introduction

This chapter gives an introduction to the problem of *protein structure prediction* (Section 1.1), and in particular the technique of homology modelling. Section 1.2 talks about one way in which we could improve this technique, and the reasons for doing so. The methods used in this work are overviewed in Section 1.3. Finally what we expect to find, and the results of the work are discussed in Section 1.4.

1.1 The Problem

The problem of protein structure prediction remains one of the major unsolved problems in structural biochemistry. The problem is to predict the three-dimensional structure of a protein from its one dimensional amino acid sequence. The sequence consists of a string of letters; each letter represents one amino acid (there are 20 different types). This sequence contains information on how the protein will fold, and thus is the key to identifying the structure of an unknown protein. Chapter 2 gives an introduction to protein biochemistry.

The principal real-world application of protein structure determination and model building is in designing drug agents to interact with new diseases.

Once the 3-D structure of a malevolent protein is known, then chemists can go about designing a suitable molecule to stop that protein from working, thus fighting the disease. Also it is important to find out how different proteins are related, as this can give some ideas as to the evolution of different species, and the relationships between them.

There are several methods which have been developed to solve the problem of protein structure prediction, however no one technique can take the credit for solving the problem in its entirety. There are algorithms for predicting the secondary structure (see Section 2.1.2) of a protein from its amino acid sequence, the current accuracy is about 70% [Rost & Sander 93]. However to build an accurate model of the active site for the purposes of designing drugs, the biologist must have the tertiary (3D) structure (see Section 2.1.2) of the protein. There are different ways that this can be done, these include:

- Ab Initio methods: simulation of the folding process.
- Fold Threading: Hidden Markov Model based methods [Krogh *et al.* 94].
- Fold Recognition methods: Homology modelling.

Currently the most productive technique [CAS96] is to compare a new sequence against a database of sequences which have a known structure. If this comparison yields a protein from the same 'family' as the unknown protein, straightforward methods can be used to construct a useful 3-D model. The process is known as *homology modelling*. Similarity between two proteins is defined by aligning the sequences and summing scores taken from weight matrices to 'score' the relatedness of each pair of amino acids. We discuss this process in detail in Chapter 3.

By improving these weight matrices, we can improve the detection of proteins which are distantly related, but are still structurally similar. The research described here aims to improve the detection of distantly related proteins by optimising the weight matrices used in protein database searches.

1.2 Solving the problem by optimisation

The improvement of weight matrices has been done before using a variety of methods (see Chapter 3). However a direct optimisation of weight matrices has not been attempted. The research described here aims to show whether or not it is possible to improve upon the current 'state of the art' weight matrix, by using such an approach.

1.2.1 Aim of optimising scoring systems

So what are we trying to achieve? At present there are hundreds of new proteins being discovered each year, and some of these have no detectable homologues. This is either because they are from an entirely new family of protein, in which case there will be no homologues in the database, or they are so distantly related that the search does not find the correct homologue.

It is this latter case that we hope to improve upon. If we can improve the scoring matrix so that we can detect more distantly related proteins, then this clearly would be advantageous. Once these cases are detected, the structure of these unknown proteins could be predicted. Also it would help biologists better understand how proteins are related to each other, and to what extent the sequence determines the final structure.

1.3 The evolutionary approach to optimisation

Evolutionary computation (EC) approaches use search methods inspired by natural evolution. The solution to a problem is formulated into some suitable representation, and then a population of solutions is evolved over many generations, each generation becoming more 'fitter' than the previous one. One component common to all EC algorithms is the generation of random

perturbations, or mutations, and the presence of a *fitness function* that is used to assess the quality of a given point and filter out mutations that are not useful.

Evolutionary computation techniques have many applications, and they have been used extensively in optimisation. For example *Genetic Algorithms* (GA's) [Holland 75] have been used for a number of applications from optimising aircraft wings, to improving job shop scheduling. In the canonical GA the solution is represented as a coded binary string. Furthermore, in addition to mutations, new points are generated by a number of other operations mimicking genetic operators and sexual reproduction, such as crossover.

Other evolutionary algorithms use different operators and representations [Reeves 93]. The research presented here aims to use an EC approach to optimise the weight matrices used in protein database searches. The choice of which algorithm to use and the design of the representation and operators is described in Chapter 4. The design of the fitness function is discussed in Chapter 5.

1.4 Expectations and results

We expect to find that not much improvement can be made on a general matrix for finding all proteins, because many matrices have already been created for this purpose, and have been studied extensively (the reason for this will be discussed in Chapter 3). However we can perhaps evolve matrices which perform better for certain classes of protein. These matrices, could be used as a set to detect distantly related proteins of the same class. The results confirmed these hypotheses and can be found in Chapter 6. The significance of these results and some ideas for further work are given in Chapter 7.

Chapter 2

Background to protein biochemistry

In this chapter we give a brief introduction to protein biology, for further information please refer to [Voet & Voet 95]. An introduction to what proteins actually are is given in Section 2.1, this is followed by an introduction to protein chemistry in Section 2.1.1. A description of protein structure is given in Section 2.1.2, and we discuss the importance of identifying structures in Section 2.2. The technique of homology modelling is described in Section 2.3, as one of the techniques which can enable prediction of protein structure. The main tool used in such homology modelling is searching of protein databases, and this is discussed in Section 2.4

2.1 Proteins, what are they?

Proteins are fundamental to all life and no sustained biological activity is possible without them. They are the most abundant organic components of cells and typically constitute 50% of the dry weight of the cell. Proteins come in a vast array of different types, and it is estimated that there are between 10^{10} and 10^{12} different proteins in all living organisms. In the human body

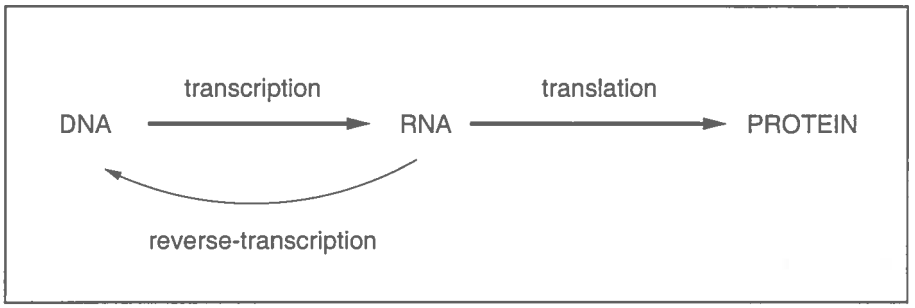


Figure 2.1: Illustration of the basic information flow within a cell, reverse transcription is also shown.

alone there are around 100,000 seperate types of protein.

Proteins are made by complex cellular machinery in each cell. Each protein is unique and is derived from the genetic sequence found in the DNA of all cells. The DNA is first transcripted into RNA, this is then translated into a code of amino-acids, each protein is defined by this amino-acid sequence (see Figure 2.1). This is the *central dogma of molecular biology* as defined by Crick, one of the co-discoverers of DNA structure. The figure also shows reverse transcription which only occurs in certain viruses known as retroviruses, e.g. HIV.

2.1.1 Chemistry

Proteins are linear polymers of amino acids, and are made up of carbon, hydrogen, oxygen and nitrogen atoms, together with phosphorus and sulphur. Amino acids possess an *amino* group, $_NH_2$, at one end of the molecule, and a *carboxl* group, COOH, at the other (Figure 2.2 shows how two amino acids can form peptide bonds to become a polypeptide chain). A single type of protein always has the same number and composition of monomers, but different proteins have a range of monomer units, from a few tens to approximately a thousand. There are 20 types of amino acids which themselves

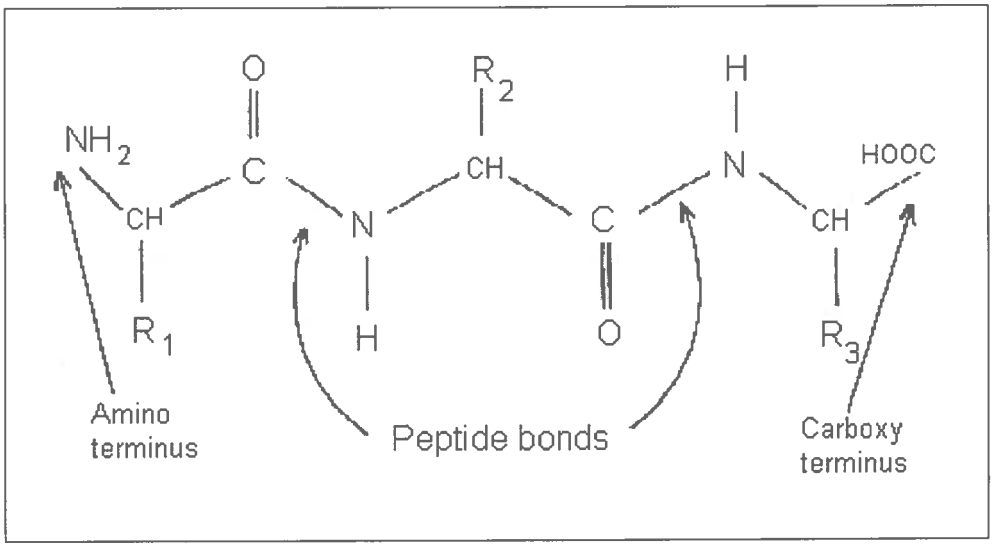


Figure 2.2: A small polypeptide chain (a tripeptide), the peptide bonds show where two amino acids have bonded together.

have a range of chemical properties. There is therefore a great diversity of possible protein sequences and structures. The linear chains fold into specific three-dimensional conformations (spatial arrangements), which are determined by the sequence of amino acids; proteins are generally self-folding. The three-dimensional structures of proteins are therefore also extremely diverse, ranging from completely *fibrous*, to *globular*.

Fibrous proteins are the basic structural elements in the connective tissue of higher animals, they are usually physically tough and are insoluble in aqueous solution. For example, Collagen is a fibrous protein found in all multicellular animals, occurring in almost every tissue. It is the most abundant vertebrate protein; approximately a quarter of mammalian protein is collagen.

Globular proteins are tightly folded into compact globular or spherical shapes which gives them more interesting chemical properties than fibrous proteins. Globular proteins come in two types: those that naturally occur in aqueous solution within the cell, and those that occur embedded in cell

membranes (for more on this see [Voet & Voet 95]).

Proteins unfolded *in vitro*[1] fold back to their original (or 'native') state when solution conditions are returned to those in which the folded protein exists. All the information for the native fold appears therefore to be contained within the primary structure since proteins are self-folding (although *in vivo*[2], polypeptide folding is often assisted by additional molecules known as molecular chaperones). This has the implication that if we know the amino acid sequence of a protein then we must somehow be able to elucidate it's final three-dimensional structure from this information. As we shall see, this is not as easy as it sounds.

2.1.2 Structure

The initial polypeptide chain is known as the *primary structure* of the protein. This is usually given as a single sequence of one-letter amino-acid codes (see Table 2.1). The protein can the be further analysed and can be divided into regions of structural similarity. The *secondary structure* of a protein, breaks it down into structural regions which are either alpha-helices, beta-sheets or loops. The secondary structure can be fairly reliably predicted, since for example only certain sequences of residues can form alpha-helices.

However the main interest to biologists is to predict the *tertiary structure* (or 3D structure) of a protein (see Figure 2.3), since this gives us the most pertinent clues as to the function of the protein. So far there has been no reliable way to predict tertiary structure, and so protein structures need to be determined. Protein structures can be determined to an atomic level by X-ray diffraction and neutron-diffraction studies of crystallised proteins, and more recently by nuclear magnetic resonance (NMR) spectroscopy of proteins in solution. However there are many proteins whose structures cannot yet

[1] In vitro means 'In the test tube.'

[2] In vivo means 'in a living system'.

Alanine	A	Leucine	L
Arginine	R	Lysine	K
Asparagine	N	Methionine	M
Aspartate	D	Phenylalanine	F
Cysteine	C	Proline	P
Glutamate	E	Serine	S
Glutamine	Q	Threonine	T
Glycine	G	Tryptophan	W
Histidine	H	Tyrosine	Y
Isoleucine	I	Valine	V

Table 2.1: Single letter codes for each amino acid.

be determined. For a more detailed introduction to protein structure, please refer to [Branden & Tooze 91].

2.2 Importance of identifying protein structure

So why is this problem so important? The benefits of protein structure identification cannot be overstated, to illustrate this here are a few ways in which the technology can or has already been used.

2.2.1 Medicine

The understanding of enzyme function allows the design of drugs which inhibit specific enzyme targets for therapeutic purposes. Proteins can themselves be designed to an extent; for example, a current area of research focuses on the engineering of insulin so that it dissociates into its active form more

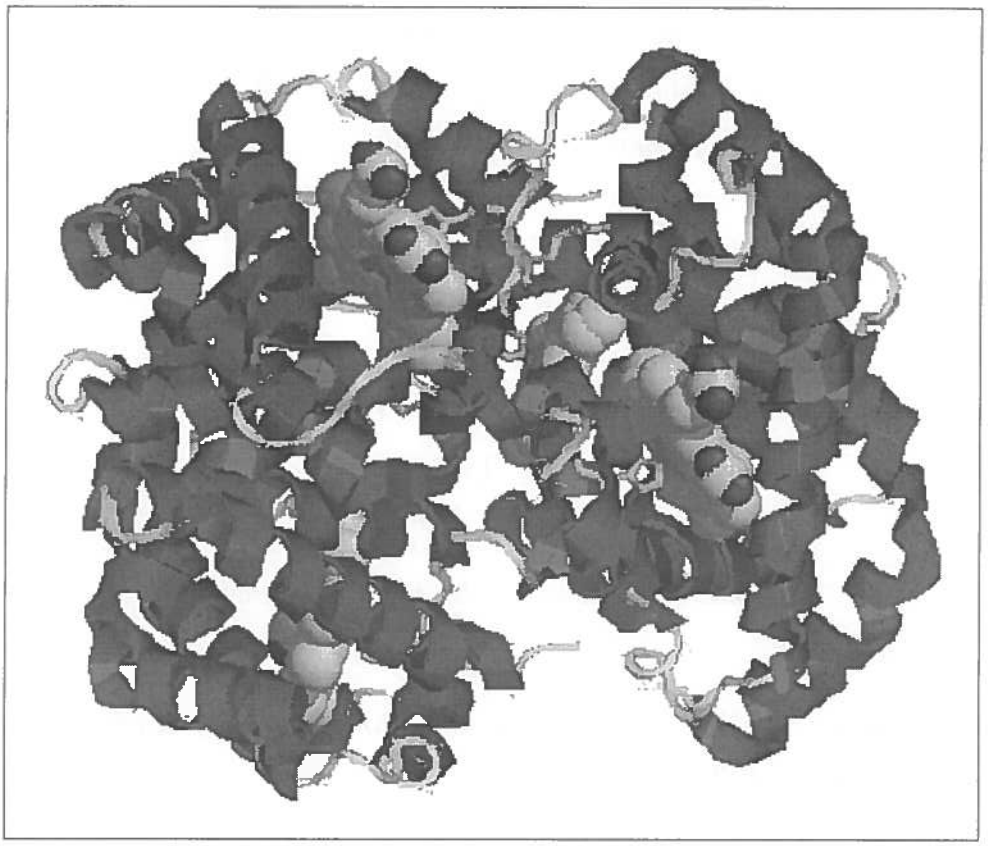


Figure 2.3: The three dimensional structure of Hemoglobin.

readily, and therefore would quicken its response if injected into a diabetic patient. Gene technology has also allowed the mass production of human insulin in microorganisms, for use in the treatment of diabetes.

2.2.2 Agriculture

Just as therapeutic proteins and drugs can be produced for medical and vetinary purposes, so can knowledge of protein structure and function be used to treat diseases of plants, and to modify growth and development of crops; for example the production of *stay-ripe* fruits.

2.2.3 Industry

Protein engineering has potential for the synthesis of enzymes to carry out various industrial processes on a mass scale. For instance, there is currently a good deal of research into the use of lipases for the industrial breakdown of fats. A domestic example of the application of protein science is the introduction of biological detergents, containing enzymes.

2.3 Protein homology modelling

There is a practically infinite number of different possible primary structures; this is the basis for the great diversity of three-dimensional structures, and functions, of proteins. Consider how many different primary structures are possible for a polypeptide of 200 residues (which is in fact a relatively small polypeptide in terms of those found in nature; polypeptides of over 1,000 residues exist). From the size of the number above, it is apparent that only a fraction of possible primary structures actually exist (or have ever existed).

As a corollory it is very unlikely that two proteins with similar amino acid sequences have independently evolved. Such similarities therefore indicate that the two proteins must be related and share a common ancestor. Related proteins are termed *homologous*. Over evolutionary time-spans, proteins mutate: i.e. their primary structure becomes altered, generally by one amino acid at a time (although more drastic single modifications can also occur). Such alterations are caused by mutations in the genes (linear sequences of nucleotides) which encode them. Not only point mutations (the substitution of one amino acid for another) occur; a protein sequence may lose some of its amino acids (deletion mutation) or have amino acids inserted (insertion mutation). However, if two primary sequences are more than approximately 20% identical (making reasonable allowance for insertions and deletions) then they can be assumed to be homologous.

In order to build a model based on homology, one first needs to determine a homologue which has a close relationship to the protein of unknown structure. Once a homologue has been found, this can be used to build a model of the unknown protein. As the homologue should have the same basic structure as the unknown protein, the biologist can then use the structure of the known protein, as a guide when doing experiments. These will then determine the essential differences between the two proteins. Once these differences are known, then an accurate model can be built and used as a tool for subsequent study.

2.4 Searching protein databases

The main way to establish homology between two proteins is to directly compare their amino acid sequences. This can be done with any two proteins, using a scoring system which determines whether one protein is indeed similar to the other. This search is often performed using a database of proteins with known three-dimensional structure. The unknown sequence is compared against every member of the database, and the entry which achieves the highest score is the most similar. This is similar to case based reasoning [Kolodner 93], where a database of known cases is used to identify an unknown case, by matching a set of features. In this case the set of features is determined to some extent by the scoring system used.

Of course the scores achieved are highly dependent on the type of scoring system used, and it is this scoring system which we seek to improve. In this work, all algorithms to compare protein sequences rely on some scheme to score the comparisons 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 by 20 symmetric matrix of similarities where identical amino acids and those of similar char-

acter give higher scores compared to those of different character. (see Table 2.2 for an example scoring matrix).

The most simple scoring scheme is to score a '1' for each match, i.e. each pair-wise comparison which has the same amino-acid residue, and a '0' for each mismatch, such a system does not need a matrix of similarity scores. This *identity* scoring system gives us the percentage identity of two sequences, e.g. if two sequences were said to be 34% identical, then 34% of their amino-acid residues match. However some proteins are less than 20% identical, but have still been proved to be homologous (using more traditional methods). To identify similarities between these proteins is difficult, and calls for more complex scoring schemes. These scoring schemes do make use of similarity scoring weight matrices. Each value in the matrix indicates how similar a particular residue pair is. The way in which these matrices are derived is introduced and discussed in Chapter 3.

2.5 Summary

To summarise, proteins are linear polypeptide chains of amino acids. There are 20 different types of amino acid, and these for peptide bonds to become polypeptide chains. These chains then fold into complex three-dimensional structures and become proteins. Protein structure determination has many useful applications in medicine, agriculture and industry. One method to solve the 3D structure of an unknown protein is to look for similar proteins of known structure. This similar protein can be used to build a model of the protein whose structure is not known. The method for finding homologues of proteins, is to search for them in a large database of proteins whose 3D structure is known. A scoring system is used to detect whether a certain protein is homologous or not. This scoring system usually takes the form of a set of scores for each amino acid substitution found in the comparison

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	2																			
C	-2	12																		
D	0	-5	4																	
E	0	-5	3	4																
F	-3	-4	-5	-5	9															
G	1	-4	1	0	-4	5														
H	-1	-3	1	1	-2	-2	7													
I	-1	-2	-2	-2	1	-3	-3	4												
K	-1	-5	0	0	-5	-2	0	-2	5											
L	-2	-6	-4	-3	2	-4	-2	2	-3	6										
M	-1	-5	-3	-2	0	-3	-2	2	1	4	6									
N	0	-4	2	1	-3	0	2	-2	1	-3	-2	2								
P	1	-2	-1	0	-4	0	0	-2	-1	-2	-2	0	6							
Q	0	-5	2	2	-4	-1	3	-2	1	-2	-1	1	0	4						
R	-2	-3	-1	-1	-4	-3	2	-2	3	-3	0	0	0	1	6					
S	1	0	0	0	-3	1	-1	-1	0	-3	-2	1	1	-1	0	2				
T	1	-2	0	0	-3	0	-1	0	0	-2	-1	0	0	-1	-1	1	3			
V	0	-2	-2	-2	-1	-1	-3	4	-2	2	2	-2	-1	-2	-2	-1	0	4		
W	-6	-7	-7	-7	1	-7	-4	-6	-4	-4	-5	-4	-6	-5	2	-2	-5	-7	17	
Y	-3	0	-4	-4	7	-5	0	-1	-5	-1	-3	-2	-5	-4	-5	-3	-3	-2	1	10

Table 2.2: The PAM250 scoring matrix

between the two sequences.

Chapter 3

Review of literature

In this chapter we will review the literature on scoring matrices and also sequence comparison. An overview of scoring systems is then given in Section 3.1, followed by a discussion on algorithms which use these scoring systems in Section 3.2. The methods used to derive the scoring matrices are described in Section 3.3. Finally the comparison of such matrices is discussed in Section 3.4, this is then followed by a brief summary.

3.1 Overview of scoring systems

A scoring scheme in the context of protein database searches is usually a 20 by 20 symmetric matrix of scores, which tells us how similar an amino acid is to another. There are well formed theories on how these matrices should be calculated. These are discussed in detail in Section 3.3 below. Essentially the scores can be represented as implicit target frequencies, which specify the probability of the juxtaposition of two amino acids [Altschul 91]. Thus the matrix gives us an indication of how similar one protein sequence is to another.

The complexity of the matrix determines how sensitive it is. For example if we used a simple identity matrix, then the score would tell us how similar

two matrices were, simply based on their identity (i.e. A's matching with A's, B's matching with B's etc.). This method was used early on, to define homology of two proteins. However more recently it has been found that sequences diverging greatly in identity (i.e. less than 20% identical), are still structurally homologous. It is clear that the information must lie in the sequence somewhere, so more sophisticated scoring schemes were developed. These have different scores for all combinations of amino acids, and represent the likelihood that one amino can be changed into another one, without loss or change of function. This could happen as a result of evolution, as the genetic sequence is mutated over time. Thus two sequences which were related closely (by identity) can, as a result of evolution, diverge but still retain similar structural characteristics.

Also there is the problem of gaps. Sometimes two sequences are similar in places, but have gaps where amino-acids have been deleted, or extra segments where amino-acids have been inserted. These are known as insertions and deletions and are a very common phenomenon. The scoring system must take these into account, since two sequences which have similar parts of their sequence (perhaps with a gap in the middle), may have a high structural similarity. These *indels* (as they are known) are represented as gap penalties in the scoring system, and are usually fixed. So every time there is a gap a penalty is deducted from the score, the penalty is usually proportional to the length of the gap. There are two types of gap penalties, normal gap costs have the same penalty for starting a gap as they do for extending the gap. *Affine* gap costs, however, have different penalties for starting and extending gaps. Affine gap costs are more biologically meaningful, since two consecutive gaps score higher than two isolated gaps. The processing of gaps is very important to the scoring scheme, as you very rarely get homologous proteins which have the same length.

3.2 Dynamic programming algorithms

The main method for assessing the similarity of two sequences is by performing a sequence alignment, using the scoring system of your choice. The original sequence alignment algorithm assessed the distance between two proteins by performing a pairwise comparison of the amino acid residues. As discussed above, the problem with comparing protein sequences is that you are very unlikely to get two sequences of the same length. Thus the alignment algorithm needs to take this into account (using the gap penalties given). The naive approach to finding the best alignment of two sequences including gaps is to generate all possible alignments, add up the score for equivalencing each amino acid pair in each alignment, then select the highest scoring alignment. However, for two sequences of 100 residues there are $> 10^{75}$ 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, where m and n are the sequence lengths. These *Dynamic Programming* algorithms were first developed for protein sequence comparison by Needleman and Wunsch [Needleman & Wunsch 70], though similar methods were independently devised during the late 1960's and early 1960's for use in the fields of speech processing and computer science (see [Sankoff & Kruskal 83]).

3.2.1 Global alignment

Dynamic programming algorithms may be divided into those that find a global alignment of the sequences and those that find local alignments. A global alignment is across the whole of the two sequences, and a local alignment only covers parts of the two sequences (as shown in Figure 3.1). So what does an alignment look like? Below is an alignment generated by the `sss_align` program [Sturrock 97].

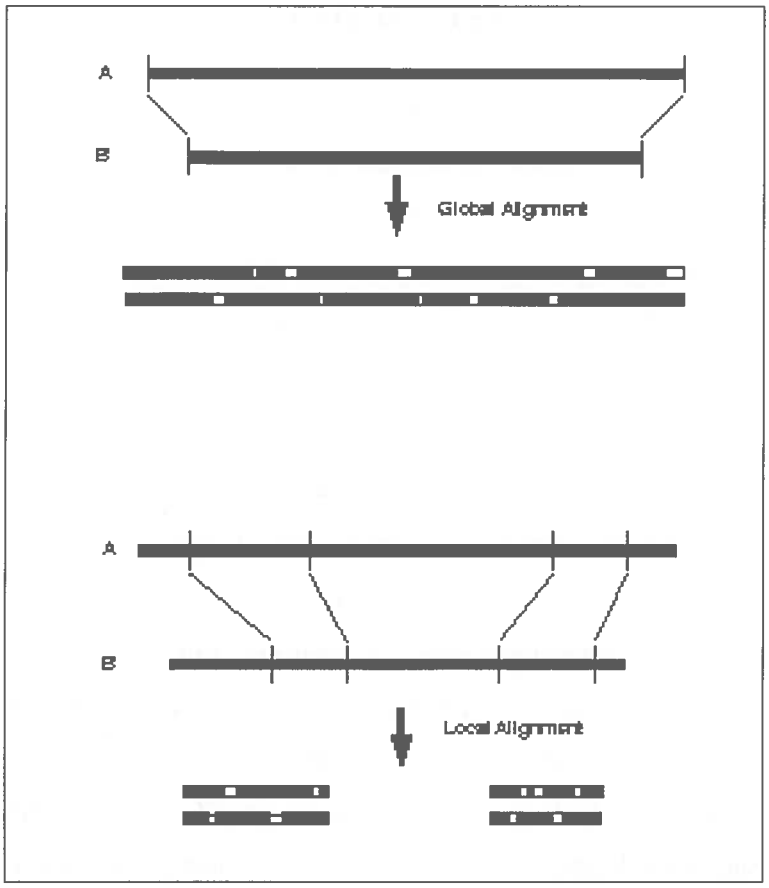


Figure 3.1: The difference between Global and Local Alignment.

```
SEQ IDENTITY 56.00%;  SEQ CONSERVATION 60.00%;

Matches 14; Conservative 1; Mismatches 10; Indels 2; Gaps 2;

              tt    ttt
Db       1 CKLKGQSCRKTSYDCCSGSCGRSGK-C 26
           || .| || .|||:||  ||.. .| |
Qy       1 CKSXGSSCSXTSYNCCR-SCNXYTKRC 26
```

The matches are indicated by the vertical lines, and the mismatches are indicated by the gaps, single dots and double dots. This notation indicates the degree of similarity between the two mismatching residues. Gaps are

indicated by the dashed lines, and it is clear that the alignment is between two sequences of differing length.

The dynamic programming algorithm finds the optimal alignment as follows. In order to optimise the distance, a value is give to each point in the alignment (i.e. each juxtaposition of amino acids). The value depends on whether there was a match, indel or substitution. The maximum of these values is taken at each point. We want the maximum because we want to find the alignment with the best score (i.e. the optimal alignment). Let the two sequences of length m and n be A $= (A_1, A_2, ...A_m)$, B $= (B_1, B_2, ...B_n)$ and the symbol for a single gap be Δ. At each aligned position there are three possible events.

1. $w(A_i, B_j)$ substitution of A_i by B_j.
2. $w(A_i, \Delta)$ deletion of A_i.
3. $w(\Delta, B_j)$ deletion of B_j.

The substitution weight $w(A_i, B_j)$ is derived from the chosen scoring scheme. Gaps Δ are normally given a negative weight (the gap penalty), since insertions and deletions are usually less common than substitutions.

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

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i-1,j} + w_{A_i,\Delta} \\ H_{i,j-1} + w_{\Delta,B_j} \end{cases}$$

The element $H_{m,n}$ contains the best score for the alignment of the complete sequences. The matrix H is illustrated in Figure 3.2. If the alignment

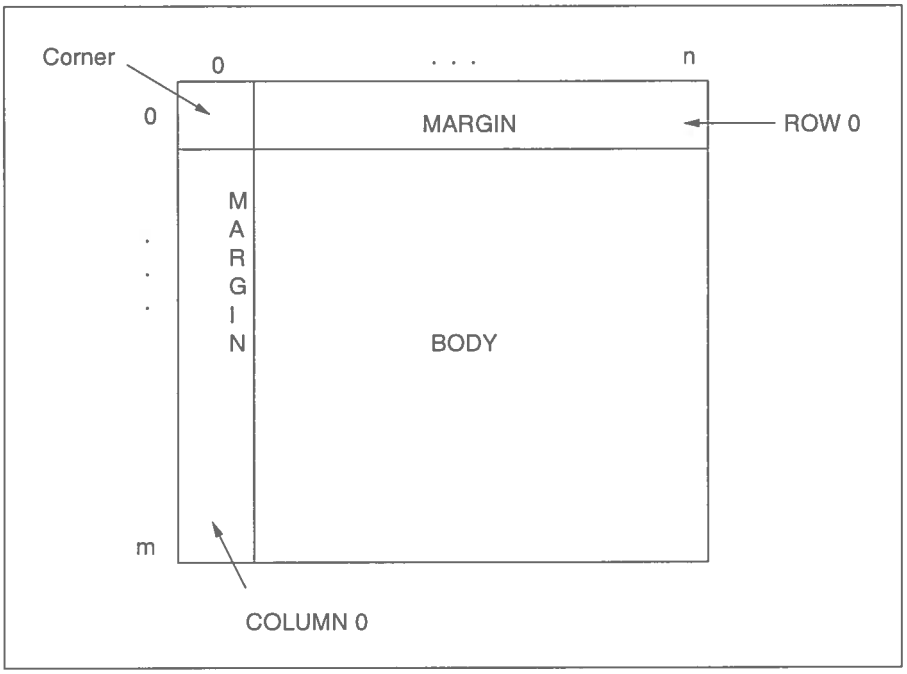


Figure 3.2: Array used to perform dynamic programming algorithm

is required as well as the best score, then the alignment path may be determined by tracing back through the H matrix, usually this is done by using pointers which were generated when the alignment was done. These pointers show which of the three choices was taken at each cell in the matrix.

3.2.2 Local alignment

The above algorithm is a global alignment algorithm, because it aligns the whole of sequences A and B. In practice this is not very useful, since many proteins with similar structures sometimes have poor global alignments. It is better to test whether a portion of sequence A, is similar to a portion of sequence B. This is called *local alignment*, and yields matches which potentially have similar structures, because components of their sequence match.

An algorithm which searches for common subsequences in proteins was devised by Smith and Waterman [Smith & Waterman 81]. This algorithm is

essentially the same as the global alignment algorithm given above, except a zero is added to the recurrence equation.

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i-1,j} + w_{A_i,\Delta} \\ H_{i,j-1} + w_{\Delta,B_j} \\ 0 \end{cases}$$

Thus all $H_{i,j}$ must have a value $>= 0$. The score for the best local alignment is simply the largest value of H and the corresponding alignment is obtained by tracing back from this cell. You can also find the next most optimal alignment by taking the second largest value of H and tracing back from this cell. In this way you can generate the n most optimal local alignments from the same matrix H.

3.3 Methods for deriving scoring matrices

In this section we discuss the different types of scoring matrix available, and how they were derived. We also discuss how scoring systems can be compared and evaluated, since this is an important part of our optimisation process. We have already mentioned the simple identity scoring scheme, we now go on to more sophisticated schemes.

3.3.1 Genetic code scoring

The genetic code scoring scheme was introduced by Fitch [Fitch 66]. It weights substitution by considering 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[1]. The scheme has been used in the construction of phylo-

[1]Each amino acid is coded in the DNA/RNA by three bases, these are known as *codons*. There is more than one possible combination of the three codons to make the same amino

genetic trees (see Section 3.3.3 below) and in the determination of homology between protein sequences having similar three dimensional structures. However today it is rarely the first choice for scoring alignments of protein sequences.

3.3.2 Chemical similarity scoring

The aim with chemical similarity scoring schemes [McLachlan 72],[Feng *et al.* 84] is to give greater weight to the alignment of amino-acids with similar physicochemical properties. This is desirable because 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. For example some amino acids are hydrophilic which means they are attracted to water molecules, and some are hydrophobic which is the opposite. Now if an amino acid in one protein is hydrophilic, and the same residue in the other protein is hydrophobic, then this substitution should get a low score, since it would not be selected by evolution.

3.3.3 Observed substitution scoring schemes

Scoring schemes based on observed substitutions are derived by analysing the substitution frequencies seen in alignments of known sequences. In [Altschul 91], it is argued that all scoring schemes can be represented as implicit substitution frequencies, so by taking the inverse of this we can generate a scoring scheme by looking at the probabilities of observed substitutions. This is something of a chicken and egg problem, since in order to generate the alignments, one really needs a scoring scheme but in order to derive a scoring scheme one needs the alignments! Early schemes based on observed substitutions worked from closely related sequences that could easily be aligned

acid.

by eye. More recent schemes have had the benefit of the earlier substitution matrices to generate alignments on which to build. Long experience with scoring schemes based on observed substitutions suggests that they are superior to simple identity, genetic code, or intuitive physico-chemical property schemes.

The Dayhoff matrix

Possibly the most widely used scheme for scoring amino acid pairs is that developed by Dayhoff and co-workers [Dayhoff *et al.* 72] and [Dayhoff *et al.* 78]. The system arose out of a general model for the evolution of proteins. The model is based on accepted point mutations (or PAMs). An accepted point mutation is an exchange of one amino-acid for another, accepted by natural selection. 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. These mutations were observed using *phylogenetic trees* (see Figure 3.3), and a few pairs of related sequences. 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 ratio is called the *relative mutability*. 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, for example after 2 PAM (Percentage of Acceptable point Mutations per 10^8 years).

When used for the comparison of protein sequences, the mutation probability matrix is normalised by dividing each element $M_{i,j}$ by the relative

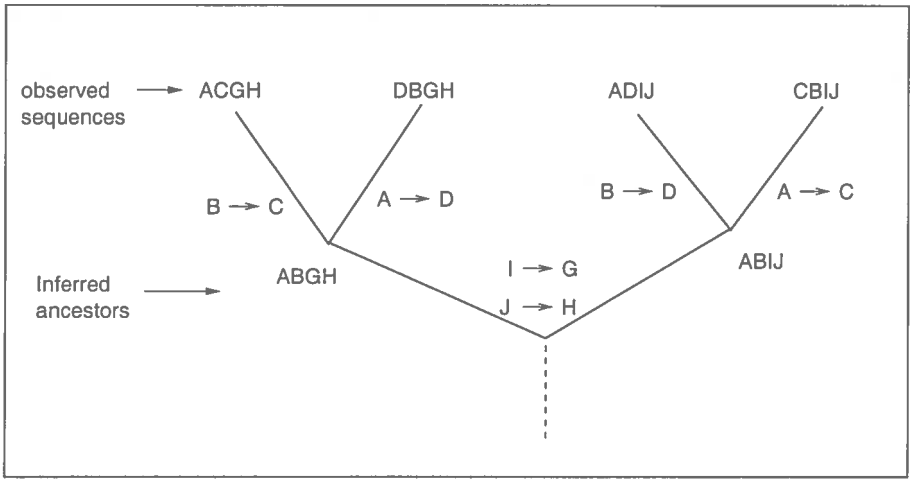


Figure 3.3: A Phylogenetic tree.

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 occurence of i per occurence 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 the limit of detection of distant relationships where approximately 80% of the amino acid positions are observed to have changed. Others have argued that other PAM distances are better, e.g. Altschul in his 1991 paper [Altschul 91] stipulated that the PAM 120 matrix was the best for detecting homologues to unknown proteins.

Limitations of the Dayhoff model

Since the Dayhoff matrix is based on an evolutionary model, the matrix is limited in that it is based on a model which makes certain assumptions. Assumptions made in the derivation model of the PAM series of matrices

include :

1. Replacement at any site depends only on the amino acid at that site and the probability given by the table.

2. Sequences that are being compared have average amino acid composition. This is an ideal which does not exist in the real world.

Sources of error in the evolutionary model include :

1. Many sequences depart from average composition.

2. Rare replacements were observed too infrequently to resolve relative probabilities accurately.

3. Errors in 1 PAM are magnified in the extrapolation to 250 PAM.

4. The Markovian process is an imperfect representation of evolution : distantly related sequences usually have islands (blocks) of conserved residues. This implies that replacement is not equally probable over the entire sequence.

5. In the Dayhoff model evolutionary distance is used as a proxy for structural relatedness, however these two may not always correspond.

These are some of the weaknesses we hope to overcome when we use direct optimisation to produce a scoring matrix.

Other matrices based on observed substitutions

There have been several other matrices which are also based on observed substitutions. A popular matrix is the BLOSUM matrix [Henikoff & Henikoff 92], which is based on multiple alignments of several ungapped sequences. The sequences were more distantly related to each other, and 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 groups and this was 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.

Another matrix of interest is the PET91 matrix, which was derived by Jones et. al. [Jones *et al.* 92]. This matrix was derived using the same methods as the Dayhoff Matrix, but with different data. The original Dayhoff matrix was constructed using a relatively small set of sequences. Many of the 190 possible substitutions were not observed at all and so suitable weights were determined indirectly. The PET91 matrix was derived by examining 2,621 families of sequences in the SWISSPROT database release 15.0. The principal differences between the PET91 matrix 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 principal properties of the amino acids.

3.4 Comparison of matrices

Various studies have been conducted to find which of the different scoring systems is the best. The consensus with all studies is that there is no one matrix which can be used to solve all problems.

Vogt, Etzold and Argos [Vogt *et al.* 95] did an extensive comparison of various matrices. They used a total of 80 matrices, which were applied to 37 protein families, containing 204 sequences, which allowed 1064 distinct pairwise alignments. For each matrix, gap penalties were optimised to achieve the best percentage of correctly aligned residues relative to standards-of-truth

derived from protein tertiary structural superposition[2]. They found that the best performing matrix was that derived by Gonnet et al. [Gonnet *et al.* 92]. It was built from a sequence database with 8,344,353 amino acid residues. Each sequence was compared against the entire database, such that 1.7×10^6 subsequence matches resulted for the significant alignments. These matches were subsequently used to generate a matrix with a PAM distance of 250.

Henikoff and Henikoff [Henikoff & Henikoff 93] evaluated several substitution matrices in their ability to recognise distantly related proteins. They evaluated sequence and structure based matrices as well as extrapolated matrices based on evolutionary models such as the PAM series. They focused on recognition of distantly related proteins (and not sequence alignment accuracy) by searching large sequence databases with a distant query sequence, and checking how many true and false positives occurred within the top listed sequences with the most significant scores. They found the BLOSUM62 matrix to perform best. The BLOSUM50 matrix was the second best performer.

Altschul [Altschul 91] studied amino acid substitution matrices from an information theoretic point of view. In this study the properties of matrices were analysed in the context of *local* alignments *lacking gaps*. He concluded that for protein databases of typical current size (about 1.7×10^7 residues), the most broadly sensitive matrix should be a log-odds matrix of distance PAM120. In order to detect short but strong homologies or long but weak ones, this matrix should be complemented by the PAM40 and PAM250 matrices respectively. However this may not necessarily be the optimum for local alignments *with* gaps (which a lot of database search algorithms use).

"Of course, many database search methods, such as the FASTA

[2]This is where structures are compared by eye using molecular graphics, and other tools.

programs, seek local alignments with gaps, and such measures are potentially more sensitive to distant homologies. Unfortunately if gaps with associated scores are allowed, the specific quantitative discussion above is no longer correct." [Altschul 91]

3.4.1 What makes a good matrix?

From the work of those cited above we can glean that there is no one matrix that is suitable for all alignments. But how do we define a good matrix? A good matrix is one that finds results which are statistically relevant, i.e. the alignment cannot be by mere chance. For instance if we have a large database (say 5,000 proteins), we need to determine whether the results of a search are not just coincidences. This can be done by establishing a random protein model, which generates random sequences, and aligning them with the database. This will then tell us the level at which *hits* occur by chance. If our results are beyond this level, then we can say they are *true* hits. For the purposes of structure prediction, it is important to make this distinction, since we do not know which protein we are searching for, and thus cannot always decide whether it is a true hit or not just by looking at it.

So a good matrix is that which discriminates most accurately between true and significant hits and random hits. We can think of this graphically, since the results of a database search can be plotted, and are usually in the shape of a poisson distribution (see Figure 3.4). The true hits lie in the *tail* of the distribution, while the random hits (or noise) lie in the main body. Usually the two overlap as shown in 3.4(a) and 3.4(b). Our task is to find a matrix which better separates the tail from the body to achieve a distribution like that of 3.4(c).

3.4.2 Different matrices for different tasks?

It has been suggested by some [Altschul *et al.* 94], that you should use different matrices for different tasks. It would be favourable to have just one *magic* matrix which finds all homologues, because then you would only need to make one database search. However in reality this is probably not possible, so it might be better to have several matrices, which are adapted to look for certain classes of protein. Then you could use these one at a time, until you found a hit, this would at least tell you what class of protein you were dealing with. Further searches could be performed using more specialised matrices, and so on. This is one avenue we must consider if our efforts to create a good general matrix fail.

3.5 Summary

To summarise, the main method to compare two protein sequences is to align them using dynamic programming. This algorithm makes use of a scoring matrix which contains values for each unique pair of amino acids. There are many ways in which these scoring matrices can be derived, and the most useful is by observing substitutions in alignments of proteins which are known to be structurally similar. These observed substitutions are then used to calculate the probabilities of two amino acids being mutated into each other. The most popular matrix derived in this way is the Dayhoff matrix, which is widely used. A good matrix is that which allows us to distinguish true 'hits', i.e. the proteins which are actually similar in structure, from random hits, which occur by chance. It has been suggested that in order to better achieve this form of matrix, one should use specific matrices for specific tasks.

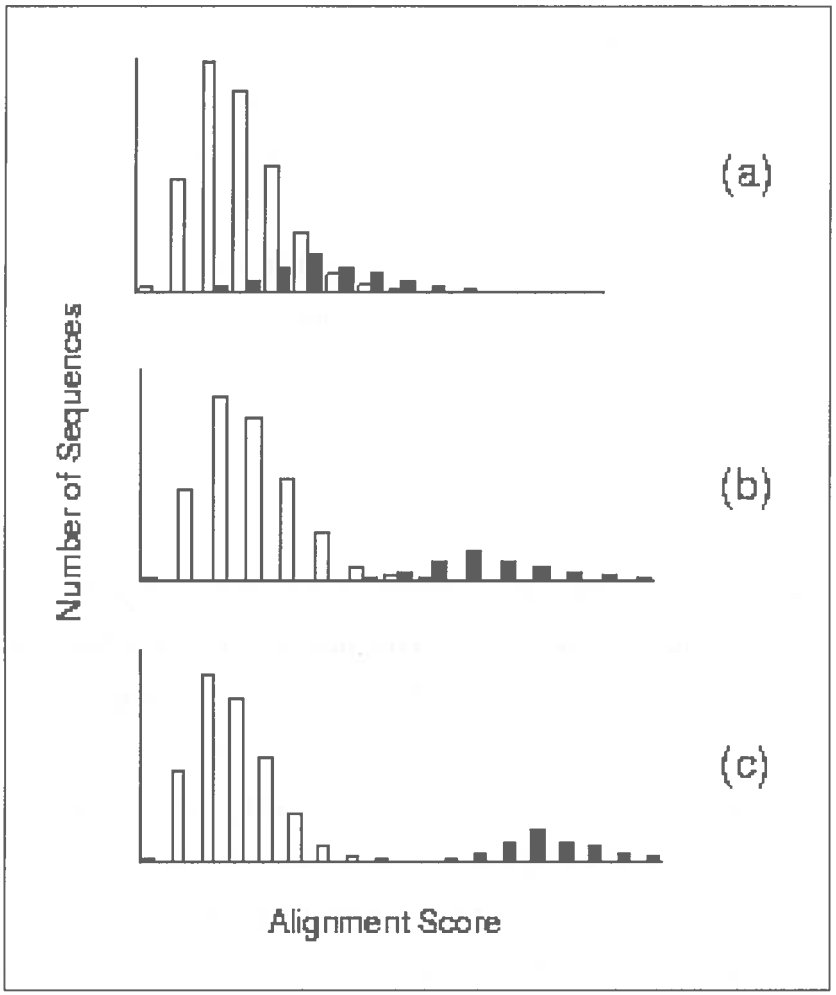


Figure 3.4: Plots of the database results, the dark bars represent 'true' hits, the white bars represent noise. We would like to achieve complete seperation as shown in (c).

Chapter 4

Design of the optimiser

In this chapter we formulate our problem in Section 4.1 and decide on a type of evolutionary algorithm which is described in Section 4.3. The design of representation and operators is discussed in Section 4.4, and finally the details of the implementation are described in Section 4.7. This is followed by a brief summary.

4.1 Problem formulation

Many optimisation problems can be cast as a search through a set of vectors of real or integer valued parameters to find the vector which will maximise some function. In our case the parameters are the matrix values, and the function is the performance of such a matrix on finding proteins with low sequence identity, but that have similar structure to our query protein. A large variety of both local and global techniques with varying degrees of specialisation have been proposed for tackling such problems [Tuson 98]. Evolutionary algorithms have been regularly applied to such problems, due to their generic nature (requiring only the ability to evaluate the function at any encoded point).

For our problem the search space can be characterised as a search for a

set of integer values in a specific range (since the program we will be using only allows matrices to be specified in integers). The set of values are the components of a matrix M, which is symmetrical about its diagonal (see Table 2.2). Each value in this matrix should be an integer $X_{i,j} \in [-60, +20]$, where $i, j \in [1, 20]$. The range restriction on $X_{i,j}$ is due to the program `sss_align`, which will make use of the matrix. This program only allows matrix values within this range and also only allows integer values. This constraint also narrows the search space to consider sensible values for $X_{i,j}$.

There is also a constraint on the diagonal elements of M, X_{ii}. Since this is the score for a direct match in amino acids, it cannot be negative. So all elements X_{ii} must be positive. Any operators on the matrix must obey this constraint, since if matrices have non-positive diagonal values they will be useless.

This optimisation task can be recast as a minimisation problem $minz(x)$ where z(x) is the objective function which tells us how effective the matrix is, and is a combination of cost functions. Of course you may just have one cost function in which case that would be z(x). In the case of multiple cost functions (for example if we had two fitness functions for the matrix), the different cost functions can be combined as follows:

$$z(x) = \sum_{m=1}^{T} w_m . f_m(x)$$

The weighting factors w_m are used to define the importance of the different objectives $f_m()$. The minimisation of z(x) is thus the overall task for the algorithm we choose to employ.

4.2 Choosing a suitable algorithm

Having decided to take the evolutionary optimisation approach, one has to decide on an algorithm which will be suitable for our problem. There are

several different flavours of evolutionary program, and they have all been tried on a variety of problems. Since evolutionary algorithms are a form of heuristic search, this leaves many choices at our disposal. One of the most popular algorithm is the Genetic Algorithm (or GA) [Holland 75] and there are several different flavours of this algorithm. The canonical GA uses binary string representations of the problem and maintains a population of these. Every generation, members of the population are made to 'reproduce' and mutate and then they are evaluated. The *fitness* of each member is then used to determine which members go on to the next generation and which do not. As in real evolution, survival of the fittest applies and eventually a 'good' solution will hopefully emerge. The advantage of using binary strings is that this representation lends itself well to theoretical analysis and allows the construction of elegant genetic operators. Thus the bit string representation of solutions has dominated genetic algorithm research. For some optimisation problems however (such as multidimensional, high-precision numerical problems), genetic algorithms perform poorly. However the power of the evolutionary algorithm does not depend on using bit strings and so the use of real-coded or floating-point genes has been developed [Michalewicz 96]. Since our problem is to evolve a matrix of integer values, it would be better to use an algorithm which manipulates members which are not binary strings.

Fortunately there are many algorithms which cope with real or integer valued population members. Evolution strategies [Bäck *et al.* 91] were among the first algorithms which used a real valued representation. One of the more recent algorithms is Differential Evolution (or DE) [Storn & Price 95], and this has proven a very efficient algorithm for continuous search space optimisation problems [Storn & Price 96]. It has proven to more efficient than other state of the art heuristic search techniques such as Adaptive Simulated Annealing [Ingber & Rosen 92], on the general test suite of De Jong func-

tions. This algorithm is a variant of GA's and is very simple. The advantage is that as long as we have a suitable objective function, any sort of real or integer valued representation can be used (in our case a 20 by 20 matrix of integers).

4.3 Differential evolution

Differential Evolution is similar to the original GA in that the general process is a simulation of the evolutionary process. A set of population members is maintained and operators are applied to mutate these members. The population is then ranked according to fitness and 'bad' members are discarded. This is where the similarity ends, however. DE uses a non binary representation, and uses a new form of mutation, and no crossover (although crossover can be included). At each generation a random member of the population is mutated, and then the mutated member is compared against the current member of the population. If the mutant is better then it is inserted into the next generation; if not then the old member is inserted. This happens for each population member. Over many generations the best solution emerges.

4.3.1 The DE algorithm

The basic DE strategy employs the difference of two randomly selected vectors (in our case matrices) as the source of random variations for a third vector. So initially one starts off with a population of random vectors, or in case a preliminary solution is available, the initial population is generated by adding random gaussian noise to the nominal solution $X_{nom,0}$. DE then generates new vectors by adding the weighted difference vector between two population members to a third member. If the resulting vector yields a lower objective function value than a predetermined population member, then the newly generated vector replaces the one to which it was compared. There are

several variants of DE, but the one that was used in our experiments works as follows : for each member $x_{i,G}, i = 1, 2, 3, NP$, a trial vector v is generated according to

$$v = x_{r_1,G} + F.(x_{r_2,G} - x_{r_3,G}),$$

The integers r_1, r_2 and r_3 are chosen randomly from the interval $[1, NP]$ and are different from the running index i. NP is the number of population members and G is the generation number. F is a real and constant factor which controls the amplification of the differential variation $(x_{r_2,G} - x_{r_3,G})$. This vector v is then compared to $x_{i,G}$ and whichever is better is placed into the next generation $G + 1$. For our problem the population vectors will need to be represented in such a way as to allow the above operations. The complete algorithm used in the experiments is give in Algorithm 1. Since we are evolving a matrix of integers, the values for the mutant vector v must also be integers. However the parameter F is a real number (less than 1), so the results of the calculations need to be rounded to the nearest whole number.

Algorithm 1 Differential Evolution

```
for G = 0 to Genmax do
  for i = 0 to NP do
    Select r1, r2 and r3 randomly
    Generate vector v = x_{r1,G} + F.(x_{r2,G} - x_{r3,G})
    if eval(v) ≤ eval(x_{i,G}) then
      Insert v into generation G + 1
    else
      Re-insert x_{i,G} into generation G + 1
    end if
  end for
end for
```

4.4 Design of representation and operators

The structure of a solution vector in any search and optimisation problem depends on the underlying problem. Our problem involves manipulating a matrix of integer values. The representation must obey the constraints of the problem, and also allow adequate manipulation of the population members depending on which move operators are being used.

When designing a chromosome representation, we must consider how the different parts of a population member interact, i.e. any dependencies which they have on each other. For example when using binary string representations one needs to be careful that what happens to the string actually reflects a proper and valid change to the actual solution after it is decoded from the string representation. Mutation and crossover on the string may result in invalid solutions which are useless. So it is beneficial to model these *constraints* as part of the representation. That way all vectors will be valid, and we do not need to worry about getting ambiguous results.

The issue of which representation to use has been widely discussed (see [Baeck *et al.* 97]), and there have been formal studies on the principles of how one should choose a representation [Surry & Radcliffe 97]. However these studies are still relatively new in the GA community and so the representation given here is not based on any formal theory. Rather we will use our problem and search strategy as a guide to designing the representation. Given that we are using DE as our search strategy, a direct integer valued representation would seem most appropriate. The authors of the system originally designed it for the purpose of real parameter optimisation problems, however we can modify the algorithm easily to cope with integer values. The basic representation can be described as simply a 20 by 20 array of integer values. The matrix has specific constraints and to cope with these the operators that manipulate the array (and generate it) will need to take these

constraints into account.

4.4.1 Encoding of matrix features

The main feature of the matrix is that it is symmetrical about its diagonal, and this is easy to encode into the representation. Since a square array is being used to represent the matrix, all we need to make sure is that element $X_{i,j}$ equals element $X_{j,i}$ when we are generating the initial matrices and also if we are reading any matrices in from an external file. We could use a non-square multi-dimensional array instead, but this is more complex to manipulate and it is a trivial constraint anyway.

The other constraint is to make sure that all diagonal elements are positive. This is easy to do when generating the matrix initially, but we also need to consider the fact that the matrices will be constantly changing due to the genetic operators. Therefore this constraint was built into the move operators, so that a check is made to see if any of the trial vectors disobeyed this rule. If they did, the value of the offending element was changed to be brought in line with the constraint. Also we use the same technique for all elements in the array to obey the range constraint. The integer value in each element must be between -60 and +20, so we need to check for this when doing the addition. If the value is out of range, then it is capped at either -60 or +20 so that the matrix will be valid.

4.4.2 Move operators

For the DE algorithm, the two main move operators are vector subtraction and vector addition. Since we are manipulating matrices, these were implemented using matrix subtraction and addition respectively. So for example to subtract one matrix from another, each element was subtracted component by component. This results in a third matrix which is the difference matrix,

and then this is added to the matrix which is being mutated.

A crossover operator was also constructed, although the original DE algorithm does not use it. This was implemented using UNBLOX crossover, where a square block of one matrix is subtracted from a similar square block in another matrix. This difference block is then added to the matrix which is being mutated.

To cope with the diagonal constraint, the addition operator checked when adding to a diagonal element to make sure the result was valid. If the result was negative, then it would replace the result with 1 instead (since the diagonal elements must be positive). A similar check was added to the add block operator. The range constraint was also dealt with in this way, the values being capped if they were out of range.

4.5 Evaluation of weight matrices

The evaluation function should indicate how good a matrix is at detecting homologous proteins. In our case we should make sure that the evaluation tests the ability to detect structurally similar proteins with low sequence identity. The evaluation function should take as its input a matrix, and produce as its output a value which is a measure of how good that matrix is for our problem. This value is then used to compare two matrices during evolution. The method of evaluation is discussed fully in Chapter 5.

4.6 Parameters

The DE algorithm and indeed many such algorithms have a number of parameters that must be chosen in order to run the optimisation experiments. For example the value **F** which determines the proportion of the difference vector which gets added to produce the trial vector. How is the initial population

generated? There is also the population size and number of generations to consider. These parameters were chosen by performing a few preliminary experiments and these are described in Chapter 6. Due to the fact that the evaluation function was very computationally expensive (see Chapter 5), the experiments were not run for hundreds of generations. Thus no stopping criteria was implemented since convergence in a small number of generations seemed unlikely. Instead the experiments were run for the full number of generations, and then the results examined.

4.7 Implementation

The DE algorithm was implemented in C++ [Stroustrup 91], as this allows one to define data structures and methods which manipulate those data structures collectively as objects. Thus the matrix which is the data object that our optimiser manipulates, was defined as an object, with methods for the operators of addition and subtraction. The DE algorithm was implemented as an object which maintains the population of matrices using an array of pointers..

4.7.1 Hardware

All software was developed on a DEC Alpha (533MHz SX 21164PC processor), under the LINUX operating system, and the GNU C++ compiler. All subsequent experiments were also conducted on the above platform.

4.7.2 Software

The software was developed in stages. First the matrix class was written, and tested independently, to make sure all the operators worked properly. The operator overloading feature of C++ was used to implement the matrix

addition and subtraction operators. Next the differential evolution algorithm was implemented and tested. A simple regression problem was used to test the DE program. The solution vector was a set of real parameters, and the program had to evolve a function which passed through all the points given. The fitness function was the sum of the squared errors at each point. The matrix class was then integrated with the DE program, and tested.

4.8 Summary

To summarise, the differential evolution algorithm was chosen to do the optimisation. The population members are 20 by 20 arrays of integers, and have a few constraints. Each element of the array must be between -60 and +20, and all diagonal elements must be positive. The move operators used by DE were programmed to manipulate these arrays. Addition and subtraction were implemented as matrix addition and matrix subtraction, where elements are added or subtracted component by component. The system was implemented in stages each stage being tested independently before integration.

Chapter 5

Design of the evaluation function

This chapter gives a description of the design of the evaluation function used to score the matrices. Possible approaches are discussed in Section 5.1. The final evaluation function is described in Section 5.2. A method to minimise evaluation time is described in Section 5.3, and the selection of training data is discussed in Section 5.4. This is followed by a brief summary.

5.1 Possible approaches to evaluating matrices

Evaluating a protein scoring system is a tricky task since to evaluate it properly one should use it to search for a variety of proteins from more than one database. However since we do not have time to do this for each matrix, we must come up with a fast way of assessing whether a matrix is good for the problem we are addressing.

There are two possible approaches to this problem. One is to use a direct database search, using the matrix to be evaluated. Then the results of this search would be used to score the matrix. This would probably have to be done more than once, since only one protein can be searched for at a time.

So a good representative sample of proteins would need to be used, in order to make sure that the matrix is adequate for all classes of proteins.

The other option is to use an analytical method for assessing the quality of a matrix. Several studies have been done on how to assess if a result from a search is statistically significant [Karlin & Altschul 90], [Karlin & Altschul 93], [Altschul & Gish 96]. These techniques use a random model for proteins to predict how many 'hits' can occur by chance, when searching large databases. Given a non-random protein model (i.e. one that represents the proteins that we are looking for), and a set of scores, we can use the same technique to determine how good those scores are in the context of searching for that particular set of proteins.

5.1.1 Using statistical models

The statistical method given in [Karlin & Altschul 90] is for ungapped alignments only. It shows how, given a protein model, you can predict the number of *Maximal Segment Pairs* (local alignments which give a high alignment score) with a score of at least S, that are likely to occur by chance alone.

Consider two independent sequences with letter probabilities $\{p_1,\ldots,p_r\}$ and $\{p'_1,\ldots,p'_r\}$, respectively. The pair of letters a_i of the first sequence and a_j of the second sequence occurs with probability $p_i p_j$. These probabilities are chosen to reflect the observed frequencies of the amino acids in actual proteins (see [McCaldon & Argos 88] for how these probabilities are derived). The theory that follows makes two assumptions :

1. At least one positive score must be possible.

2. The expected score $\sum_{i,j} p_i p_j s_{ij}$ must be negative.

where s_{ij} is the score for those two letters. Given a substitution matrix and a random protein model, one can calculate the amino acid substitution

frequencies $\{q_{i,j}\}$ by using the formula :

$$q_{i,j} = p_i p_j e^{\lambda s_{i,j}}$$

where λ is the unique positive solution to the equation:

$$\sum_{i,j} p_i p_j e^{\lambda s_{ij}} = 1$$

Given a random protein model any scoring matrix can be specified up to a constant factor by its implicit target distribution for paired amino acids. For our purpose we could reverse this and calculate the target frequencies from the matrix to be evaluated and then compare these to a model of 'ideal' target frequencies.

There are problems with this approach however. First, the theory only holds for ungapped alignments. Since we are interested in gapped alignments which are the most widely used, we need a theory which works for these. The extra complexity of gap penalties has been analysed experimentally [Karlin & Altschul 93] [Waterman & Vingron 94], and the results do seem to concur with the theory above. However there are no tried and tested formulas to calculate the values that we need [Altschul 98].

5.1.2 Using results of database searches

The 'brute force' method for assessing scoring matrices has always involved using the matrices in real database searches. The advantage of this is that one can be sure that the results are going to work in a real search environment, as the matrix has already been trained and tested on real databases. The disadvantage is that it takes time to do a database search, and since we are going to be evaluating a lot of matrices we need to economise the amount of time evaluating each one.

There are several points to consider when using this approach. For example which algorithm do we use to do the matching, there are several algorithms available. A popular alternative to dynamic programming is the

BLAST family of search tools [Altschul *et al.* 90] and there are also variants of the original DP algorithms [Pearson & Miller 92]. The dynamic programming method has been proved to be the most thorough [Sellers 80] and so it would be most appropriate to use this as we can then say that for the proteins that the matrix is trained on, a thorough search was performed and the results were as we expected.

The final consideration is the choice of training data. If we are to use a database search to evaluate the matrix, we need a query protein sequence for which the search must produce good results. We need more than one of these query sequences, otherwise the matrix we evolve will be very specific, and only look for that type of protein sequence. If we want a general matrix, then we need to have several query sequences which represent a good sample of all the proteins. The evaluation function will then consist of a series of searches, using the matrix to be evaluated, and the 'scores' for these searches will need to be totalled to get an overall measure of the fitness of that matrix.

5.2 Final choice of evaluation function

After careful consideration it was decided to use the database search method of evaluation. This was due to the fact that the analytical technique was not properly defined for gapped alignments and this is what the matrix will eventually be used for. Also if the theory was properly developed we would not need to evolve a matrix, as we could then derive the matrix from the frequencies given in our 'ideal' model.

> "Of course things are a little more complicated ...for target frequencies vary with evolutionary distance, and one therefore needs a sequence of matrices adapted to these distances. This, of course is what the PAM and BLOSUM series of matrices provide." [Altschul 98]

So in effect what we would be doing is deriving another form of Dayhoff's Matrix, in a slightly different way. However we are attempting to circumvent the assumptions made by this method by using direct optimisation of the matrix.

Another point to consider is that the database search is what the final matrix is going to be used for. So by using this as our evaluation function, we are really evaluating the performance of the scoring system under real world conditions. If a theoretical model were used it would need to be derived and, since new proteins are being discovered everyday, the model would soon be out of date and so would any matrix derived from it.

5.2.1 How it works

The evaluation function works as follows. Firstly the matrix to be evaluated is used to search a database for a small set of proteins. Each protein which is used to query the database, will have a set of homologues that the search is expected to find. The search will be done for each query sequence and the results will be parsed to see how well they match to the expected output. The search program returns a list of the matches which it finds. This list is in order of significance. The highest match is not important because it corresponds to the same protein as the query sequence. So the first match will always be the same protein used for the search. We are more interested in later matches which correspond to the next most similar protein (according to the scoring matrix), and so on.

The search will be done using the `sss_align` program. This program ranks the matches in order of significance, the most significant match being at the top of the list. The program returns the top 200 matches from the search (this value can be changed, but was kept at 200 for all our experiments). The best result would be to have all known homologues appear immediately after the highest match, but in practice they are often lower down in the list.

By counting the positions between the highest match and the lowest match, we can score a matrix on its performance for that protein. This is then done for all the proteins in the training set, and the results are summed. If there are no matches then that protein is given a score of 400. If there is only one match then this is the same as the query protein. So it is still a bad result, but it is not given a score of 400. Instead a score of 200 + the position of this highest match is given. This is because the `sss_align` program returns the top 200 matches, so if there is only one match and it is in position 200, then the score will be 400 (as if there were no matches). The scoring algorithm is given in Algorithm 2 below.

Algorithm 2 Scoring algorithm for evaluation function

```
main score = 0
for i = 0 to Numberoftestproteins do
    component score = 0
    parse results and mark any proteins p which match targets with a '1'
    mark any that do not match with a '0'
    start counting from highest match
    count from highest match h to lowest match l
    for j = h to l do
        component score = component score + 1 if p_i = 0
    end for
    if No matches at all then
        component score = 400
    else if one match then
        component score = 200 + position of match
    end if
    main score = main score + component score
end for
```

Example

The scoring algorithm can be illustrated with a simple example. Suppose the database search resulted in the output below.

1	515	5.50e-56	ELECTRON TRANSPORT	1ETP 1
2	102	3.33e-03	ELECTRON TRANSPORT	1DVH 0
3	95	1.75e-02	ELECTRON TRANSPORT(FLAVOCYTOCHROME)	1FCD 1
4	82	3.36e-01	ELECTRON TRANSPORT	1COR 0
5	71	3.46e+00	NUCLEOTIDYLTRANSFERASE	1TAQ 0
6	71	3.46e+00	NUCLEOTIDYLTRANSFERASE	1JXE 0
7	68	6.32e+00	NUCLEAR RECEPTOR	2LBD 1
8	66	9.36e+00	THIOLASE	1AFW 0
9	65	1.14e+01	IMMUNOGLOBULIN	1YUH 0
10	65	1.14e+01	PHOSPHATE TRANSPORT	1OIB 0

Here only the top ten results are shown, in a proper search there would be 200 of these. Our query protein was 1ETP, so we start counting from this (highest) match. Suppose the homologues of 1ETP are 1FCD and 2LBD. We now count all the zero's from 1ETP down to the lowest match which is 2LBD. Thus this protein would get a score of 4. Note the order of the homologues is not important, only their position in the list. So in fact 2LBD and 1FCD could have been the other way around and they still would have achieved the same score. Furthermore the number of matches found does not affect the score. For example one protein has two matches close together, and another protein has 10 matches but with a space between the first and the second match, the first protein will get a lower score. This was done to ensure no bias between examples, because some proteins have one homologue whereas others might have a lot more.

5.2.2 The sss_align program

The `sss_align` program [Sturrock 97] was developed to search databases for proteins whose secondary structure had already been predicted. By using this additional information the program could sometimes detect homologues where previous programs could not. The program has a number of features, but the main reason for using it was that it allows one to specify a scoring matrix given as a file, and also implements the Smith and Waterman algorithm efficiently. So, when using it as part of the evaluation function, one need only specify the correct command line parameters and the program runs and returns the results in an output file which can then be parsed by the evaluation function. The program uses its own database which was created for the program (see [Sturrock 97]) and this contains around 3,200 entries. This database was used as it contains enough representative entries, and is also small enough so that searches remain fairly fast.

5.2.3 Evaluation time

The complete evaluation process is illustrated in Figure 5.1. A database search using `sss_align` takes approximately ten seconds. If our training set of proteins was ten proteins then a run of 100 generations for a population of ten matrices would take around 28 hours. This is why a selection scheme was used to select subsets of proteins for evaluation, rather than evaluate the whole training set in every generation (see Section 5.3).

5.3 Subset selection methods

A database search takes time and there has to be several proteins in our training set in order to evolve a good general matrix. One way to reduce the time evaluating each matrix is by using a technique called *subset selection.*

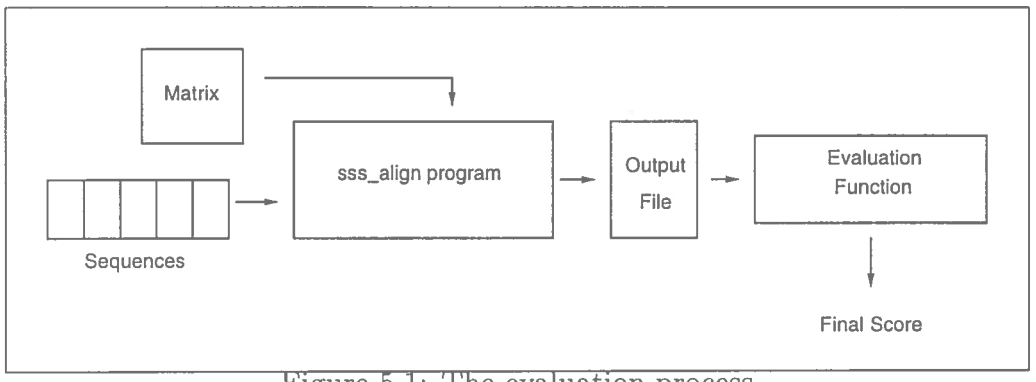


Figure 5.1: The evaluation process

This technique allows one to have a large training set, but then only use a subset of that training set when evaluating a solution vector. The subset is changed every generation, so that over a run of evolution the entire training set is used. There are various methods of picking the subset from the training set.

In *random subset selection* the subset is picked from the training set entirely at random. Normally the probability is scaled so that the subset size is the same as the target size (i.e. the number of members we can actually evaluate). However the algorithm was modified in this case so that the subset was picked at random, but was always a fixed size.

Dynamic subset selection [Gathercole & Ross 94] is based upon the fact that the algorithm must concentrate on the difficult cases and also on the cases which haven't been selected in a long time. In each generation, the subset is selected by taking two passes through the full training set.

In the first pass of the entire training set, of size $\mathbf{T}$, in a generation $\mathbf{g}$, each training case $\mathbf{i}$, is given a weight, $\mathbf{W}$. This is the sum of its current 'difficulty', $\mathbf{D}$, exponentiated to a certain power, $\mathbf{d}$, and the number of generations since it was last selected (or age), $\mathbf{A}$, also exponentiated to a certain power, $\mathbf{a}$:

$$\forall i : 1 \leq i \leq T, \quad W_i(g) = D_i(g)^d + A_i(g)^a$$

The sum of all cases' weights is also calculated during this first pass.

In the second pass of the entire training set, each case in turn has a probability, **P**, of being selected to be in the subset. A case's probability is given by its weight divided by the sum of all the cases' weights (found in the first pass) and multiplied by the target subset size, **S**:

$$\forall i : 1 \leq i \leq T, \quad P_i(g) = \frac{W_i(g) * S}{\sum_{j=1}^{T} W_j(g)}$$

This scaled probability ensures that the average selected subset size is the target subset size.

If a case, i, is selected to be in the subset, then its difficulty, D_i, and age, A_i, are set to 0, otherwise its difficulty remains unchanged, and its age, A_i is incremented. In this work the difficulty **D** was chosen to be the evaluation score for a particular case. With our evaluation function the higher the score, the more difficult that protein is to detect for the given matrix.

5.3.1 Subset size

In order to keep evaluations down to a minimum the subset size was fixed at **S=6**. This number was chosen because it also represented the number of different classes of protein which were in the training set (see Section 5.4). The training set was subdivided into sets of each class, and DSS was used to select one protein from each of these subsets. So in any one generation all six proteins in the subset had a representative from each class. This was done to ensure that the matrix evolved was as general as possible. If the subset was derived from the entire training set without restrictions, then a subset which contained proteins all from one class could result. This would bias the evolution of the matrix to a particular class, which was not the aim.

5.4 Training data

The selection of the training data must reflect all the different known classes of proteins. Also all classes of protein included in the training set should be equally represented. This is particularly important for those protein families that only have a few examples in them.

5.4.1 Which proteins should we include?

Proteins are classified under various umbrellas. For example there are protein families of which there are several hundred, above this there are superfamilies and folds, which still have hundreds of types. At the highest level there are protein classes, there are around seven of these (there are more but these represent the main classes of naturally occurring proteins). Our data set must represent all of these classes equally, and include a few examples from each class. Fortunately a database has been constructed which orders proteins by their class, and can be used to pick out examples for our training set.

5.4.2 The SCOP database

The SCOP (Structural Classification of Proteins) database [Murzin *et al.* 95], was created to facilitate understanding of, and access to, information which is available regarding protein structures. This database provides a detailed and comprehensive description of the structural and evolutionary relationships of the proteins of known structure. The method used to construct the protein classification in SCOP is essentially the visual inspection and comparison of structures. The unit of classification in SCOP is the *protein domain.* Small proteins, and most of those of medium size have one domain, and so have one entry in the database. A large protein may consist of several domains, and thus may have several entries in the database referring to these different domains.

Each SCOP entry is identified with a 5-part number, which gives information on its classification. SCOP entries with the same 5-part number have homologous domains. These entries usually represent different domains from the same large protein. Entries with the first 4 parts the same are also homologous, but are from different proteins. Two example SCOP entries are shown below:

```
>d1hstb_ 1.4.3.7.1 Histone H5, globular domain [chicken]
SHPTYSEMIAAAIRAEKSRGGSSRQSIQKYIKSHYKVGHNADLQIKLSIRRLLAAGVLKQ
TKGVGASGSFRLAK
>d1ghc__ 1.4.3.7.2 Histone H1, globular domain [chicken]
MAGPSVTELITKAVSASKERKGLSLAALKKALAAGGYDVEKNNSRIKLGLKSLVSKGTLV
QTKGTGASGSFRLSK
```

These two domains are homologous, but we can see that they come from different proteins, because they have a different fifth number. As shown in the example above, SCOP contains proteins sequenced from many different organisms.

The different folds[1] found among proteins in nature have been grouped into classes for convenience. This provides an easy way to select examples for our evaluation function. The seven main classes that are listed in SCOP are as follows.

1. All alpha (for proteins whose structure is essentially formed by α-helices).

2. All beta (for those whose structure is essentially formed by β-sheets).

3. Alpha and beta (for proteins with α-helices and β-strands that are largely interspersed).

[1]Proteins are classified using the term *folds* because the natural folding process is the way in which proteins of different structure have come about.

4. Alpha plus beta (for those in which α-helices and β-strands are largely segregated).

5. Multi-domain (for those with domains of different fold and for which no homologues are known at present).

6. Membrane and cell surface proteins and peptides (this class was not used in the experiments as it contained very few entries).

7. Small Proteins.

For the training set, four proteins were selected from each class, giving a total of 24 proteins in the training set. The subset size was set at six (as this was reasonable with the time taken for an evaluation), and the experiments were then conducted using subset selection as described above. Class six was not used as it only contained around 10 entries in the database.

5.5 Preparation of training data

The database used for the experiments was the `sss_align` database. The searches produce an output such as that shown in Section 5.2.1. In order to specify which proteins the evaluation function should look for, the unique ID for each protein must be known. This is known as a PDB (Protein Data Bank) identifier. Unfortunately the entries in SCOP do not have their PDB identifiers listed. So the data picked from SCOP was prepared, by searching for each sequence using the `sss_align` database, and finding the PDB identifier. These were then put into a file, which was loaded into memory once by the main program. This procedure was performed for all the sets of data used for training and testing. It also served as a useful check to make sure that the sequences picked from SCOP were actually in the `sss_align` database.

5.6 Problems with training data

The SCOP database is organised by protein domains (see Section 5.4.2), which means that there is often more than one entry in the database representing the different domains *of the same protein.* This presents a problem when we use a different database to do the searching, such as the `sss_align` database. This database has one entry for each whole protein, and so when matching, it will only find one match for each domain which is in a *seperate protein.* This means that only cases which have homologues from at least two different whole proteins can be considered for the training data. Fortunately there are many such examples in SCOP, which have different 5-part numbers, but the same 4-part number (for an example see Section 5.4.2). All the data used for the experiments can be found in Appendix B.

5.7 Summary

To summarise, database searches will be used to evaluate the scoring matrices. Each matrix will be used to search a database for a number of proteins, and will be scored on whether it can detect their homologues. To keep evaluations to a minimum, a subset selection scheme will be used to select subsets of the training set, rather than evaluate the matrix using the whole training set. The training data was selected using a database organised by structure. The training set includes examples from 6 major classes of protein. The subset selection picks examples so that each class has an example represented within the subset.

Chapter 6

Results

This chapter presents the results of the optimisation experiments. The approach used is described in Section 6.1, followed by a discussion on the choice of experimental parameters in Section 6.2. The results are presented in Section 6.3. Finally an analysis of the results is given in Section 6.4, this is followed by a summary.

6.1 Experimental approach

The experiments whose results are described in this chapter were all carried out in a similar fashion. First some preliminary experiments were done to see what sort of parameters were worth considering. The reason for doing this is due to the fact that each of the experiments take a substantial amount of time (about 12 hours) and so it was not possible to do a thorough parametric search to determine the most optimum parameter settings. Instead the recommended parameters for differential evolution were used and a few test runs were done to make sure that the system worked.

After the parameters were chosen, a variety of runs were performed. The population size and number of generations was varied. In order to keep the number of evaluations down, there was a trade off between the number of

generations and population size, so a large population would only be run for a short number of generations and vice versa.

At the end of each run the best matrix was saved, and then tested against the entire training set of proteins. The results of these tests can be found in Sections 6.3.1 and 6.3.5. To validate these results the matrices were then tested against another set of unseen proteins, these results can be found in Sections 6.3.3 and 6.3.7.

6.2 Choice of parameters

There are a few variable parameters which need to be set in order to use differential evolution. The main parameter is **F** and determines what proportion of the difference vector gets added onto the vector which is being mutated. The recommended setting was F=0.8, and so this was tried in the preliminary experiment. Another thing which needed to be decided upon was whether or not to use an initial seed matrix in order to help the optimiser in the right direction. If no seed matrix was used then DE might take a long time to get to the current level of 'good' matrices. So the PAM250 matrix was inserted into the initial population to help the optimiser get onto the right track. It was also decided at this stage that the original DE algorithm would be used, which does not use crossover. If this failed to get a result, then crossover would be included.

6.2.1 Preliminary experiments

The preliminary experiments were test runs and were performed with a population of ten matrices over 100 generations. One experiment did not use a seed matrix and the other did. The experiment which used a seed matrix performed better than the one without, see Figure 6.1. From the numbers on these two graphs it is clear that a seed matrix makes a big difference.

Without it the mean cost at the end of the run is around 1640, and with the seed matrix the mean cost converges to a value of around 200.

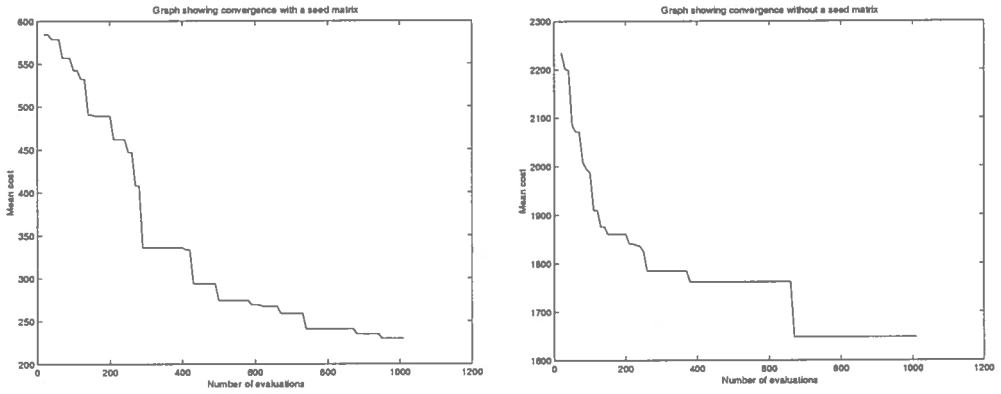


Figure 6.1: Graphs showing convergence with and without a seed matrix

For the remaining experiments, it was decided to keep the parameter settings as they were, and to use a seed matrix. Thus the parameter settings used were:

- The value for **F** was 0.8.
- The settings for the dynamic subset selection exponents were **d=1.0** and **a=3.5** (as recommended in [Gathercole & Ross 94]). With these exponents, the most difficult cases and cases around 15 generations old would have roughly equivalent weights.
- For the seed matrix, the PAM250 matrix was used.
- Default gap penalties were used (the `sss_align` program chooses appropriate gap penalties for the scoring matrix being used).

Furthermore the starting values for elements in the randomly generated matrices were restricted to be in the range [-30,10]. This was to ensure that the original population did not start with values which deviated too much from those found in most matrices (e.g the PAM250 matrix). The population

Experiment	Random seed	Population size	Number of generations
run1	3	10	100
run2	5	20	100
run3	7	10	100
run4	10	50	10
run5	2	20	200
run6	13	20	100

Table 6.1: Parameters for the first set of experiments

Experiment	SCOP class	Random seed	Population size	Number of generations
run1	1	9	10	100
run2	1	13	20	100
run3	1	52	30	100
run4	3	3	20	100
run5	3	15	10	100

Table 6.2: Parameters for the second set of experiments

size and number of generations were varied with each experiment and can be found in Tables 6.1 and 6.2 along with the random number seed used for each experiment.

6.3 Results

Two sets of experiments were conducted. In the first set of experiments the aim was to evolve a *general* matrix capable of finding homologues to all proteins. When referring to matrices which were not evolved, the term *derived matrices* is used, and refers to the PAM and BLOSUM matrices used in the tests. In the second set of experiments the aim was to evolve a matrix

which would be specific for a particular class of protein. The parameters for all experiments were kept the same (see Section 6.2).

6.3.1 Results for first set of experiments

The results for the first set of runs are shown in Table 6.3. These runs were seeded with the PAM250 matrix, and used the parameters given in Section 6.2 above. These results show the performance of the matrices with the training set of 24 proteins. The mean score is shown for the entire training set. This was derived by totalling the individual component scores for each protein and dividing by 24. Furthermore a breakdown by SCOP class is also given. This was computed by totalling the scores for the proteins in each class and dividing by the number of proteins in that class.

Matrix	Mean score	Mean score for each SCOP Class					
		Class 1	Class 2	Class 3	Class 4	Class 5	Class 7
PAM250	49.58	50	41.5	21.25	57.75	82.5	44.5
PAM120	42.5	83.5	75.25	1	53	4.75	37.5
BLOSUM60	28.58	53.5	38	1.5	54.25	9	15.5
run1	39.08	54.5	61.5	0.25	56	4	58.25
run2	67.79	86	91.5	41.5	87.25	0.25	100.25
run4	74.92	51.25	82.75	51.75	97	93	73.75
run5	28.13	25	39	1	52	3.5	23.25
run6	176.29	157.75	199	100.5	129.5	135.5	335.5

Table 6.3: Results for the first set of experiments using the training data

6.3.2 Discussion

The results for run3 are not shown as this run resulted in the best matrix being the PAM250 matrix. It seems strange that the matrix generated from a run is not always the best matrix. This is due to the subset selection.

A matrix may be really good for a particular subset, and for this subset it may be better than the PAM250 matrix, thus replacing that matrix in the population. Only when the matrix is tested against the entire training set does this 'sampling error' become clear.

However there are some interesting results in runs 1 and 5. In particular run5 scores a better overall score than the PAM250 matrix, and also scores better mean scores for each of the SCOP classes. From the results it is clear that some matrices are good at certain classes and not so good at others. For example run2 has a worse overall score than the PAM250 matrix, but scores considerably better for SCOP class 5 than the PAM250 matrix. The best candidate for an improved general matrix is that generated by run5, since it dominates the PAM120 and PAM250 matrices under each SCOP class, and is better than the BLOSUM60 matrix in all classes except classes 2 and 7.

6.3.3 Validation tests for first set of experiments

In order to test the best matrices generated in the first set of experiments a test set of proteins was produced. For the general type of matrix, this was exactly the same as the training set, a set of 24 proteins with 4 examples from each class. The tests were performed using the same evaluation function as the previous experiments. The results for these tests can be seen in Table 6.4. The scores shown in these tables were computed in the same way as those in Section 6.3.1.

6.3.4 Discussion of test results

The test results show that only the matrix generated from run5 is truly competitive with the PAM and BLOSUM matrices. This matrix performed better than the PAM120 and PAM250 matrices, but was outperformed by the BLOSUM60 matrix. The run5 matrix does however achieve better average

Matrix	Mean score	Mean score for each SCOP Class					
		Class 1	Class 2	Class 3	Class 4	Class 5	Class 7
PAM250	82.63	18.5	96	92.5	199	74	15.75
PAM120	79.88	49.75	115.5	94.5	151	24.5	44
BLOSUM60	73.21	50	90	57.75	120.5	95.5	25.5
run1	90.58	99.5	118.25	100.5	152	54.25	19
run2	97.63	115	120.5	118.5	156	67	8.75
run4	84.58	99.75	99.5	68	152.5	75.75	12
run5	76.25	53.25	90	71	160	60.5	22.75
run6	169.29	189.25	149.25	196.75	199	183	98.5

Table 6.4: Test results for the general set of matrices.

scores than the BLOSUM60 matrix for SCOP classes 5 and 7.

6.3.5 Results for second set of experiments

In the second set of experiments, matrices were evolved for specific classes of protein. This was done to investigate whether a matrix trained for a particular class of protein could outperform the general type of matrix (when used to detect proteins from that class).

The results for generating matrices specific to SCOP classes 1 and 3 are shown in Tables 6.5 and 6.6. The parameters were the same as those used in the first set of experiments, except that the training set was reduced to 18 proteins all from a particular SCOP class. The component scores for each protein are given, as well as the overall score which was computed by adding all the component scores together. The convergence graph for run2 is shown in Figure 6.2.

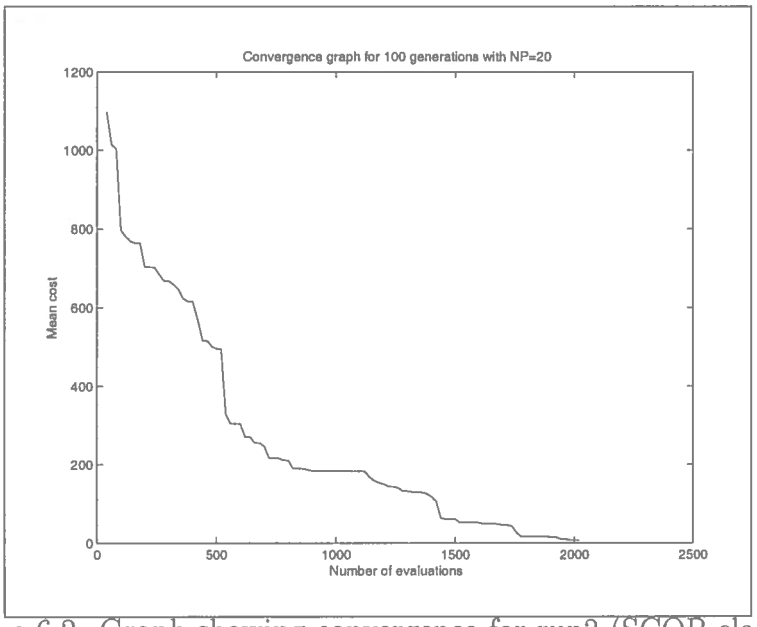


Figure 6.2: Graph showing convergence for run2 (SCOP class 1)

6.3.6 Discussion

For matrices derived to detect proteins from SCOP class 1, the results for run1 and run2 are better than the PAM250, PAM120 and BLOSUM60 matrices. The matrix generated in run2 beats the three general types of matrix in most components. Especially component 11, where run1 and run2 do much better than the general matrices.

The results for matrices evolved specifically for SCOP class 3 are not as good as those for class 1. The matrix generated by run4 is the best candidate, and it is not as good as the BLOSUM60 matrix. However, it is competitive with the PAM120 and PAM250 matrices.

6.3.7 Validation tests for second set of experiments

For the matrices evolved for a specific task, the test set was a set of 12 different proteins picked from the same class. The test set was reduced to 12 because of a shortage of suitable examples in the SCOP database. The

Matrix		PAM250	PAM120	BLOSUM60	run1	run2	run3
Overall Score		563	657	656	483	370	1423
Component Score	1	199	96	199	199	199	199
	2	1	135	14	2	0	168
	3	16	9	7	18	27	4
	4	74	199	199	199	72	199
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0
	7	0	0	0	0	0	0
	8	0	0	1	21	0	199
	9	75	21	62	38	48	46
	10	21	26	10	4	23	199
	11	176	170	161	0	0	11
	12	0	0	0	0	0	0
	13	0	0	0	0	0	0
	14	0	0	0	0	0	199
	15	0	0	0	0	0	0
	16	1	1	3	2	1	199
	17	0	0	0	0	0	0
	18	0	0	0	0	0	0

Table 6.5: Results for SCOP Class 1 experiments on training data

tests were performed using the same evaluation function as the previous experiments. The results for these tests can be seen in Tables 6.7 and 6.8. The scores shown in these tables were computed in the same way as those in Section 6.3.5.

6.3.8 Discussion of test results

For the class 1 matrix the test results show that the matrices from run1 and run2 are better than the PAM and BLOSUM matrices. In particular the score for component 11 is much better when using the evolved matrices.

Matrix		PAM250	PAM120	BLOSUM60	run4	run5
Overall Score		1170	1183	605	1167	1455
Component Score	1	2	400	5	2	400
	2	1	2	1	1	1
	3	0	0	0	199	0
	4	82	0	0	34	0
	5	171	179	32	53	10
	6	0	0	0	11	0
	7	0	0	0	62	0
	8	199	199	199	34	21
	9	0	0	0	113	2
	10	2	0	0	0	3
	11	84	160	48	118	199
	12	199	14	191	199	181
	13	95	9	62	0	0
	14	57	16	7	3	29
	15	1	1	1	32	400
	16	199	199	59	199	199
	17	53	0	0	107	10
	18	25	4	0	0	0

Table 6.6: Results for SCOP Class 3 experiments on training data

With this test data the run1 matrix outperformed the run2 matrix, but if the results over the training data (see Table 6.5) are added to these test results, then run2 is better overall.

The class 3 matrices failed to beat the PAM and BLOSUM matrices, confirming the result found in the training tests (Table 6.6). However, with this data set, the matrix for run5 outperformed the matrix for run4. This shows that the results are very much dependent on the data set used.

Matrix		PAM250	PAM120	BLOSUM60	run1	run2	run3
Overall Score		552	601	564	519	525	774
Component Score	1	0	54	0	22	0	199
	2	0	0	0	0	0	0
	3	0	0	0	0	0	22
	4	1	4	1	4	44	12
	5	0	0	0	0	0	16
	6	199	199	199	199	199	199
	7	22	0	73	83	0	96
	8	0	0	0	0	0	2
	9	6	7	6	6	6	2
	10	175	144	199	199	199	199
	11	149	109	86	6	12	27
	12	0	84	0	0	65	0

Table 6.7: Results for SCOP Class 1 experiments on the test data

6.4 Analysis of results

For the general matrix most runs were not as good as the PAM and BLOSUM matrices. The matrix generated by run5 is the only candidate which beats all the 'derived' matrices. This result was as expected, most matrices evolved for a general protein search task would probably fall within the same surface as the PAM and BLOSUM matrices.

> "One problem appears to be that these scoring systems have a very broad plateau, so that once one has gotten a pretty good system it is difficult to improve it a great deal, and what one ends up with may depend on the idiosyncrasies of one's test data." [Altschul 98]

Matrix		PAM250	PAM120	BLOSUM60	run4	run5
Overall Score		601	612	656	1419	690
Component Score	1	0	54	85	140	0
	2	186	59	133	199	188
	3	104	80	190	199	18
	4	6	9	34	131	8
	5	126	199	183	34	199
	6	50	186	10	82	0
	7	2	2	2	0	4
	8	4	2	17	199	84
	9	0	0	0	199	0
	10	122	1	1	37	68
	11	0	0	0	199	0
	12	1	20	1	0	121

Table 6.8: Results for SCOP Class 3 experiments on the test data

The results from the validation tests show that the run5 matrix is not unbeatable. These results show that the BLOSUM matrix is the best performer overall, however the run5 matrix is still competitive with the PAM120 and PAM250 matrices. A more extensive study (such as that shown in [McCaldon & Argos 88] or [Henikoff & Henikoff 93]) would need to be performed to find the exact properties of the matrix, and whether it is good at finding proteins from (say) a particular evolutionary distance.

The matrices generated in run1 and run2 to detect homologues of class 1 proteins outperformed the PAM and BLOSUM matrices on both the training and test data. This shows that it is possible to use an optimisation approach in generating good matrices for specific tasks. This matrix would also need to be tested more extensively, using a bigger test set of examples.

The matrices generated to detect class 3 examples did not perform as well as those for generated for class 1. None of these matrices outperformed the BLOSUM matrix. The run4 matrix however, did well against the training data set. A biological reason for the poor results for this class may be because of the complexity of proteins found within class 3. Class 3 proteins consist of alpha helices and beta sheets which are interspersed, where as class 1 proteins only consist of alpha helices. Thus this result would seem to confirm that some protein structures are much more harder to reliably score (and thus predict) than others.

6.4.1 Significance of results

One thing that can be said from the above results is that the optimisation of scoring matrices is possible. From the graphs shown in Figures 6.1 and 6.2, one can see that the matrix is being steadily improved. This at least shows that the optimisation is working. They also suggest that at least a little more improvement might be possible with longer runs and larger populations.

The results show that it is possible to evolve a matrix which is as good as some of the derived matrices, but not better (so far), for detecting all classes of protein. It is possible to evolve a matrix however which is specific to a particular class of protein, and which does outperform the derived matrices for that particular class. The matrix generated for detecting proteins from SCOP class 1 shows this to be true. The main significance of this result is that this matrix can detect some homologues which the general matrix cannot. For example in Table 6.5, the score for component 11 is much lower when using the evolved matrices. Also in the validation tests (Table 6.7), again component 11 gets a better score using the evolved matrices.

6.5 Summary

To summarise, the matrices evolved for general detection of protein homologues were competitive with some of the derived matrices, but not all. The results show that it is possible to evolve matrices which lie in the same ballpark as the PAM and BLOSUM matrices, but to improve upon these matrices is difficult. When evolving matrices which look for specific classes of protein the matrices evolved to look for class 1 proteins were more successful than those evolved to look for class 3 proteins, this highlights the fact that some types of protein are easier to detect than others. The matrix evolved to look for class 1 proteins outperformed the PAM and BLOSUM matrices, and this seems to indicate that the optimisation of scoring matrices is possible for specific types of protein. The best evolved matrices can be found in Appendix A. All of the results found however were only tested on a (relatively) small set of examples, and a more extensive testing study should be performed to corroborate these findings.

Chapter 7

Conclusion

This chapter concludes the research presented in this work. Section 7.1 restates the main aims of the project and what was achieved. This is followed by Section 7.2 which describes the most significant results. Section 7.3 discusses how one could follow up on the work presented here, and this is followed by some concluding remarks.

7.1 Aims and achievements

7.1.1 Objectives

The aim of this project was to investigate whether or not amino acid scoring matrices could be improved by an optimisation approach. The main task was to see if a scoring matrix could be evolved which would find homologous proteins that could not be detected by current state of the art matrices. It was decided to use an evolutionary approach to this task, and the differential evolution algorithm was chosen to perform the optimisation experiments. The function used to evaluate the weight matrices during optimisation was based on a direct database search using suitable training data.

7.1.2 Achievements

The results show that the optimisation approach used, does work, in that scoring matrices are steadily improved over a number of generations. This indicates that the evolutionary algorithm used is useful for this kind of problem and could be used for further investigations in this domain.

The experiments achieved mixed results. The best matrix evolved to detect homologues from all types of protein was as good as some of the derived matrices, but not better in all regards. In particular the BLOSUM matrix outperformed the best evolved matrix on the test data. However there were certain classes where the evolved matrix was better than the BLOSUM matrix on this set of data, so the BLOSUM matrix was not totally dominating.

The best matrix generated to detect proteins from SCOP class 1 was better at detecting examples from this class than the derived general matrices. However the matrix evolved specifically for class 3 did not perform as well. This matrix was beaten by the BLOSUM60 matrix on the first set of test data, and beaten by all the derived matrices on the second set of test data.

7.2 Significant results

The main significance of the results is that an optimisation approach for this problem is possible and credible results can be achieved. In particular it is encouraging that although the best general matrix did not totally outperform all the derived matrices it was still competitive with some of the best matrices in current use. This result also concurs with many people's suspicions that a good general matrix is very difficult to find. The search for an overall general matrix may be impractical since to derive such a matrix using statistical techniques requires a large set of data, and representative proteins from all known protein families. Even then, this sort of matrix may not

detect homologues, since some new proteins are from entirely new families which have never before been seen. However the question on whether or not there is a better general matrix remains unanswered.

The results show that it may be easier to evolve or derive matrices which are specific to a certain class of protein. This is not only easier in that there is less data involved, but also gives the optimisation a single goal to achieve. With a general matrix, the optimisation is dealing with multiple objectives, and if one objective is improved upon, then another may suffer as a consequence. This can be seen from the results shown in Table 6.3. Some matrices do better for some SCOP classes than for others, if one wished to have an unbiased matrix then the mean scores for each class would be equal. Of course this all depends on the training data used, but in the experiments conducted, the training data was chosen to be as unbiased as possible. Thus in every generation, the classes of protein selected for evaluation were equally represented.

The best matrix evolved for SCOP class 1 outperformed all the derived matrices, however the matrix evolved for class 3 failed to beat the derived matrices. This is probably due to the fact that class 1 proteins all consist of the same type of secondary structure (namely alpha helices), where as class 3 proteins consist of two types of secondary structure (alpha helices and beta sheets). Given hindsight it is easy to see that proteins which consist of only one type of secondary structure, will benefit from a table which is specifically dedicated to that secondary structure. The general matrices need to cope with all types of secondary structure and so represent some sort of average between these three types. Similarly the class 3 matrix had to average between these types and so was similar to a general type of matrix. Whether there better matrices for all classes remains unanswered, but more investigation into this area would clearly prove useful because the BLOSUM matrix (or one similar in characteristics) was not found, which we know is at

least better than some of the special class matrices that were found.

7.3 Further work

There are several approaches one could take to build upon the results presented here. First of all there are several improvements which could be made to the evolutionary process.

- Change the generation of the initial population so that matrices are generated by making random variations in a seed matrix. This would ensure that all starting matrices would have a similar composition to the seed matrix, this variation could be used when one has a good matrix but it needs to be fine tuned.
- Crossover was not used in any of the experiments, this could be tried to see if it achieves any improvements.
- Increase the size of the training set, and run for more generations perhaps with bigger population sizes.

This last point can be achieved fairly easily with the provision of more time and computing power. The DE algorithm used is fairly easy to parallelise (see [Storn & Price 95]). Thus it would be advantageous to port the algorithm to a fast parallel machine and run experiments using larger population sizes and more generations.

An obvious extension to this work would be to confirm the results found by doing an extensive test on the best of the evolved matrices. This could take the form of a statistical test where a large set of example proteins would be aligned using the matrix to be tested. Such large scale tests are described in [Henikoff & Henikoff 93] and [McCaldon & Argos 88].

The training data and experimental approach could also be changed. It was found that it is easier to evolve matrices for particular classes of protein

which contain one type of secondary structure. Thus it would be useful to try and evolve matrices for each of the three types of secondary structure. The data to train these proteins could easily be found in a database such as SCOP which is organised by structure. Once these tables are generated they could be used as a set. Using a program such as `sss_align` which gives data on secondary structure information, one could effectively *swap* scoring matrices mid-search to get the best alignment for a given protein. This would eliminate the problem of current scoring matrices which are an effective average between the three types of secondary structure.

One problem with this approach is that one does not know the secondary structure of an unknown protein. However there do exist reliable algorithms (such as the PHD algorithm [Rost & Sander 93]) which can predict the secondary structure of a protein with around 70% accuracy. Moreover the PHD algorithm also gives an estimate as to how good its prediction is for each secondary structure element in the protein. One could use these estimates as a weight to determine when to use a general matrix for a section of the protein or to use the special secondary structure matrix corresponding to that element.

One could also use the matrices evolved in this work as a building block, in order to evolve more refined matrices. The training data could be chosen to reflect very hard cases, where sequence identity is very low but the proteins are still structurally similar. Given a big set of such training data, the initial population could be seeded with one of the evolved matrices as described above. This would enable us to determine how successfully a matrix could be *fine tuned* for a particular task.

7.4 Concluding remarks

An unprecedented wealth of data is being generated by genome sequencing projects and other experimental efforts to determine the structure and function of biological molecules. In particular biotechnology, pharmacology, and medicine will be affected by the new results and increased understanding of life at the molecular level. This work explains one of the techniques which is important in this endeavour, namely the detection of protein homologues by searching databases. The results presented here, show that it is possible to use an optimisation approach, in improving the amino acid scoring matrices used in protein database searches. We have also found that it may be more effective to use a set of matrices which are suited to particular types of protein, rather than a single matrix suitable for all proteins. A possible avenue for extension was described and ways on how the optimisation process might be improved were also discussed.

Bibliography

[Altschul & Gish 96] S.F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[Altschul 91] S.F. Altschul. Amino acid substitution scoring matrices from an information theoretic perspective. *Journal of Molecular Biology*, 219:555–565, 1991.

[Altschul 98] S.F. Altschul, 1998. Personal Communication.

[Altschul *et al.* 90] S.F. Altschul, W. Gish, M. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[Altschul *et al.* 94] S.F. Altschul, M.S. Boguski, W. Gish, and J.C. Wootton. Issues in searching molecular sequence databases. *Nature Genetics*, 6:119–129, 1994.

[Bäck *et al.* 91] T. Bäck, F. Hoffmeister, and H. Schwefel. Extended selection mechanisms in genetic algorithms. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Con-*

ference on Genetic Algorithms, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[Baeck *et al.* 97] T. Baeck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, 1997.

[Branden & Tooze 91] C. Branden and J. Tooze. *Introduction to Protein Structure.* Garland Publishing Inc., 1991.

[CAS96] Critical assessment of techniques for protein structure prediction, 1996. Available on the web at `http://PredictionCenter.llnl.gov/`.

[Dayhoff *et al.* 72] M.O. Dayhoff, R.V. Eck, and C.M. Park. *Atlas of Protein Sequence and Structure*, volume 5, pages 89–99. National Biomedical Research Foundation, Washington, DC., 1972.

[Dayhoff *et al.* 78] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. *Atlas of Protein Sequence and Structure*, volume 5, suppl. 3, pages 345–352. National Biomedical Research Foundation, Washington, DC., 1978.

[Feng *et al.* 84] D.F. Feng, M.S. Johnson, and R.F. Doolittle. Aligning amino acid sequences: comparison of commonly used methods. *Journal of Molecular Evolution*, 21:112–125, 1984.

[Fitch 66] W.M. Fitch. An improved method for testing for evolutionary homology. *Journal of Molecular Biology*, 16:9–16, 1966.

[Gathercole & Ross 94] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. Technical report, Dept. of Artificial Intelligence, University of Edinburgh, February 1994.

[Gonnet *et al.* 92] G.H. Gonnet, M.A. Cohen, and S.A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[Henikoff & Henikoff 92] S. Henikoff and J.D. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science, USA*, 89:10915–10919, 1992.

[Henikoff & Henikoff 93] S. Henikoff and J.D. Henikoff. Performance evaluation of amino acid substitution matrices. *Proteins*, 17:49–61, 1993.

[Holland 75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[Ingber & Rosen 92] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated annealing - a comparison. *Mathematical and Computer Modeling*, 16(11):87–100, Nov 1992.

[Jones *et al.* 92] D.T. Jones, W.R. Taylor, and J.M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in Bioscience*, 8:275–282, 1992.

[Karlin & Altschul 90] S. Karlin and S.F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Science, USA*, 87:2264–2268, 1990.

[Karlin & Altschul 93] S. Karlin and S.F. Altschul. Applications and statistics for multiple high scoring segments in molecular sequences. *Proceedings of the National Academy of Science, USA*, 90:5873–5877, 1993.

[Kolodner 93] J. L. Kolodner. *Case-based reasoning.* Morgan Kaufmann Publishers, 1993.

[Krogh *et al.* 94] A. Krogh, M. Brown, I.S. Mian, K. Sjoelander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modelling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[McCaldon & Argos 88] P. McCaldon and P. Argos. Oligopeptide biases in protein sequences and their use in predicting protein coding regions in nucleotide sequences. *Proteins: Structure, Function and Genetics*, 4:99–122, 1988.

[McLachlan 72] A.D. McLachlan. Repeating sequences and gene duplication in proteins. *Journal of Molecular Biology*, 64:417–437, 1972.

[Michalewicz 96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1996.

[Murzin *et al.* 95] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. Scop: a structure classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[Needleman & Wunsch 70] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[Pearson & Miller 92] W.R. Pearson and W. Miller. Dynamic programming algorithms for biological sequence comparison. *Methods in Enzymology*, 210:575–601, 1992.

[Reeves 93] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications, 1993.

[Rost & Sander 93] B. Rost and C. Sander. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proceedings of the National Academy of Science, USA*, 90:7558–7562, 1993.

[Sankoff & Kruskal 83] D. Sankoff and J.B. Kruskal. *Time warps, String edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.

[Sellers 80] P.H. Sellers. The theory and computation of evolutionary distances pattern recognition. *Journal of Algorithms*, 1:359–373, 1980.

[Smith & Waterman 81] T.F. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[Storn & Price 95] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimisation over continuous spaces. Tr-95-012, International Computer Science Institute, Berkeley, CA, March 1995.

[Storn & Price 96] R. Storn and K. Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *Proceedings of the IEEE Conference on Evolutionary computation*, pages 842–844, Nagoya, Japan, 1996.

[Stroustrup 91] B. Stroustrup. *The C++ Programming Language: Second Edition.* Addison-Wesley Publishing Co., Reading, Mass., 1991.

[Sturrock 97] S.S. Sturrock. *Improved Tools for Protein Tertiary Structure Prediction.* Unpublished PhD thesis, University of Edinburgh, 1997.

[Surry & Radcliffe 97] Patrick Surry and N. Radcliffe. *A formalism for real-parameter evolutionary algorithms and directed recombination.* Morgan Kaufmann, San Mateo, CA, 1997.

[Tuson 98] A. Tuson. Optimisation with hillclimbing on steroids: An overview of neighbourhood search techniques. 1998.

[Voet & Voet 95] D. Voet and J.G. Voet. *Biochemistry (2nd Edition)*. John Wiley and Sons, 1995.

[Vogt *et al.* 95] G. Vogt, T. Etzold, and P. Argos. An assessment of amino acid exchange matrices in aligning protein sequences: The twilight zone revisited. *Journal of Molecular Biology*, 249:816–831, 1995.

[Waterman & Vingron 94] M.S. Waterman and M. Vingron. Rapid and accurate estimates of statistical significance for sequence data base searches. *Proceedings of the National Academy of Science, USA*, 91:4625–4628, 1994.

Appendix A

The best evolved matrices

On the following pages the two best evolved matrices are shown. Table A.1 shows the best *general* matrix evolved. Table A.2 shows the best matrix evolved to look for proteins from a particular class. Note that the scores for matches of two amino acids (e.g. A A) are positive.

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	3																			
C	-3	11																		
D	-3	-6	5																	
E	-2	-6	2	3																
F	-6	-7	-6	-6	8															
G	-1	-6	-2	1	-3	6														
H	-3	-2	2	0	-3	-5	6													
I	-3	-3	-3	-4	0	-2	-4	5												
K	-3	-4	1	-2	-4	-1	0	-3	4											
L	-3	-8	-6	-2	-1	-3	-3	1	-6	5										
M	0	-3	-2	-5	-3	-4	-4	0	-2	5	7									
N	1	-5	-1	0	-2	-2	-1	-4	2	-2	-3	3								
P	0	-3	-4	-2	-3	-2	-1	-4	0	-5	-3	-1	5							
Q	-3	-3	0	0	-6	-3	4	-1	0	-4	-2	2	1	3						
R	-3	-4	-2	-2	-5	-6	-1	-1	0	-6	-3	-1	1	1	5					
S	-1	1	-2	-1	-2	-2	-3	-2	0	-1	-3	-1	0	-3	1	3				
T	0	-3	1	-2	-5	-1	-3	-1	0	-4	0	-2	-3	0	-2	0	2			
V	0	-3	-5	-1	-4	-4	-2	1	-1	0	-1	-4	-4	-5	-1	0	1	5		
W	-9	-9	-6	-5	-1	-8	-5	-8	-3	-5	-6	-5	-7	-8	-1	0	-6	-5	15	
Y	-4	-1	-6	-5	4	-3	-3	0	-7	-2	-6	-4	-7	-3	-6	-4	-2	-2	0	9

Table A.1: The best general scoring matrix (evolved in run5)

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	3																			
C	-3	11																		
D	-1	-6	5																	
E	-1	-6	4	5																
F	-3	-3	-4	-5	9															
G	2	-5	0	-1	-5	5														
H	-2	-4	2	1	-2	-1	6													
I	-1	-3	-2	-3	0	-3	-3	5												
K	-1	-5	-1	0	-4	-3	1	-2	6											
L	-1	-5	-5	-2	1	-3	-3	3	-3	5										
M	-1	-5	-3	-3	1	-3	-1	2	1	4	7									
N	1	-5	3	0	-3	0	2	-3	0	-3	-1	3								
P	-1	-1	-1	1	-3	-1	1	-3	-2	-3	-3	0	7							
Q	-1	-5	1	1	-4	-1	3	-3	1	-3	-2	1	0	5						
R	-2	-4	-1	-2	-5	-4	1	-2	4	-4	0	0	-1	1	5					
S	0	-1	0	-1	-4	1	0	-2	1	-2	-3	1	0	-1	0	3				
T	1	-1	0	0	-3	0	-1	-1	-1	-3	0	-1	0	-1	-1	2	4			
V	-1	-2	-2	-3	-2	-2	-4	5	-3	2	3	-2	-2	-1	-3	0	0	5		
W	-7	-8	-8	-7	0	-6	-5	-5	-5	-5	-6	-5	-5	-5	3	-2	-6	-6	16	
Y	-2	-1	-3	-5	7	-5	1	0	-6	-2	-4	-1	-5	-4	-6	-4	-4	-3	0	9

Table A.2: The best scoring matrix for SCOP class 1

Appendix B

Training and test data

B.1 Training data for general matrix

class 1

>d1etpa1 1.3.1.3.1 (1-92) Cytochrome c4 [Pseudomonas stutzeri]
>d1hsta_ 1.4.3.7.1 Histone H5, globular domain [chicken (Gallus gallus)]
>d1phna_ 1.1.1.2.1 Phycocyanin [red alga (Cyanidium caldarium]
>d1hdj__ 1.2.2.1.1 HSP40 [Human (Homo sapiens)]

class 2

>d1vcaa2 2.1.1.4.1 (1-90) N-terminal domain of vascular cell adhesion molecule-1 (VCAM-1) [human (Homo sapiens)]
>d1ncia_ 2.1.5.1.1 N-cadherin (uvomorulin), domain 1 [Mouse (Mus musculus)]

>d2mcm__ 2.1.6.1.1 Macromycin [Streptomyces macromomyceticus]
>d1nbca_ 2.2.2.2.1 Cellusomal scaffolding protein A, scafoldin [Clostridium thermocellum]

class 3

>d1pkm_2 3.1.9.1.1 (1-104,207-384) Pyruvate kinase, N-terminal domain [cat muscle (Felis domestica)]
>d2myr__ 3.1.1.4.1 Myrosinase [white mustard (Sinapis alba)]
>d1akz__ 3.11.1.1.1 Uracil-DNA glycosylase [Human (Homo sapiens)]
>d1pii_1 3.1.8.1.1 (1-254) N-(5'phosphoribosyl)antranilate (PRA)isomerase [Escherichia coli]

class 4

>d2reb_2 4.27.1.1.1 (244-303) RecA protein, C-terminal domain [Escherichia coli]
>d1sso__ 4.9.1.1.1 DNA-binding protein [Sulfolobus solfataricus]
>d1esfa2 4.12.5.1.1 (117-229) Staphylococcal enterotoxin A, SEA [Staphylococcus aureus]
>d1vjw__ 4.33.1.4.1 Ferredoxin [Thermatoga maritima]

class 5

>d2hcka3 5.1.1.2.1 (167-437) Haemopoetic cell kinase Hck [human (Homo sapiens)]

>d7cata_ 5.6.1.1.1 Catalase [beef (Bos taurus)]
>d1buca2 5.7.1.1.1 (1-232) Butyryl-CoA dehydrogenase [Megasphaera elsdenii]
>d1hle.1 5.2.1.1.1 (a,b) Elastase inhibitor [horse (Equus caballus)]

class 7

>d1cnr__ 7.10.1.1.1 Crambin [abyssinian cabbage (Crambe abyssinica)]
>d4htci_ 7.18.1.1.3 Hirudin [leech (Hirudo medicinalis)]
>d1mcti_ 7.3.2.1.1 Trypsin inhibitor [bitter gourd (Momordica charantia, linn. cucurbitaceae) seed]
>d1danl3 7.24.1.1.1 (1-36) Coagulation factor VIIa [human (Homo sapiens)]

B.2 Test data for general matrix

class 1

>d1hcra_ 1.4.1.2.1 HIN recombinase (DNA-binding domain) [synthetic]
>d1flia_ 1.4.3.9.1 Fli-1 [Human (Homo sapiens)]
>d1hks__ 1.4.3.10.1 Heat-shock transcription factor [Drosophila melanogaster]
>d1ebdc_ 1.9.1.1.1 E3-binding domain of dihydrolipoamide acetyltransferase [Bacillus stearothermophilus]

class 2

>d1tupa_ 2.2.3.1.1 p53 tumor supressor, DNA-binding domain [Human (Homo sapiens)]
>d1hoe__ 2.4.1.1.1 HOE-467A [Streptomyces tendae 4158]
>d1cyx__ 2.5.1.2.1 Quinol oxidase (CYOA) [Escherichia coli]
>d2bpa1_ 2.8.1.1.1 Bacteriophage capsid proteins [bacteriophage phiX174]

class 3

>d1hvq__ 3.1.1.5.1 Hevamine A (chitinase/lysozyme) [Para rubber tree Hevea brasiliensis]
>d1muca1 3.1.6.2.1 (119-360) Muconate-lactonizing enzyme [Pseudomonas putida]
>d1zyma_ 3.5.1.2.1 N-terminal domain of enzyme I of the PEP:sugar phosphotransferase system [Escherichia coli]
>d1tml__ 3.2.1.1.1 Cellulase E2 [Thermomonospora fusca, strain yx]

class 4

>d153l__ 4.2.1.4.1 Lysozyme [GooseAnser anser anser]
>d4rhn__ 4.10.1.1.1 Histidine triad nucleotide-binding protein (HINT) [rabbit oryctolagus cuniculus]
>d1dar_3 4.11.1.1.1 (403-525) Elongation factor G (EF-G), domain IV [Thermus thermophilus]
>d1igd__ 4.12.1.1.1 Immunoglobulin-binding protein G, separate domains [group G streptococcus (Streptomyces griseus)]

class 5

>d1hcl__ 5.1.1.1.1 Cyclin-dependent PK [Human (Homo sapiens)]
>d3pte__ 5.4.1.1.1 D-ala carboxypeptidase/transpeptidase [streptomyces R61]
>d1kfd_2 5.9.1.1.1 (196-560) DNA polymerase I (Klenov fragment) [Escherichia coli]
>d1mml__ 5.9.1.2.1 MMLV reverse transcriptase [Moloney murine leukemia virus)]

class 7

>d1omc__ 7.3.5.1.1 omega-Conotoxin [sea snail Conus geographus G IVa]
>d1iva__ 7.3.5.2.1 omega-Agatoxin IV, IVa, IVb [funnel web spider (Agelenopsis aperta)]
>d2sn3__ 7.3.6.1.1 scorpion toxin [Centruroides sculpturatus ewing, variant 3]
>d1mtx__ 7.3.6.2.1 Margatoxin [scorpion (Centruroides margaritatus)]

B.3 Training data for specific matrix: SCOP class 1

>d1grj_1 1.2.1.1.1 (1-78) GreA transcript cleavage protein, N-terminal domain [Escherichia coli]
>d1etpa1 1.3.1.3.1 (1-92) Cytochrome c4 [Pseudomonas stutzeri]
>d1enh__ 1.4.1.1.1 engrailed Homeodomain [Drosophila melanogaster]

>d1hcra_ 1.4.1.2.1 HIN recombinase (DNA-binding domain) [synthetic]
>d1hsta_ 1.4.3.7.1 Histone H5, globular domain [chicken (Gallus gallus)]
>d1flia_ 1.4.3.9.1 Fli-1 [Human (Homo sapiens)]
>d1hks__ 1.4.3.10.1 Heat-shock transcription factor [Drosophila melanogaster]
>d1ebdc_ 1.9.1.1.1 E3-binding domain of dihydrolipoamide acetyltransferase [Bacillus stearothermophilus]
>d1erc__ 1.10.1.1.1 ER-1 [Euplotes raikovi]
>d1tafa_ 1.21.1.3.1 TAFii42 [Fruit fly (Drosophila melanogaster)]
>d1mmog_ 1.22.1.1.1 Methane monooxygenase hydrolase, gamma subunit [Methylococcus capsulatus]
>d1lpe__ 1.23.1.1.1 Apolipoprotein E3 [human (Homo sapiens)]
>d1vtmp_ 1.23.5.1.1 Tobacco mosaic virus coat protein [tobacco mosaic virus (Vulgare strain)]
>d1acp__ 1.26.1.1.1 Acyl carrier protein [Escherichia coli]
>d1cei__ 1.26.2.1.1 ImmE7 protein [Escherichia coli]
>d1an2a_ 1.33.1.1.1 Max protein [mouse (Mus musculus)]
>d1cnpa_ 1.34.1.2.1 Calcyclin (S100) [Rabbit (Oryctolagus cuniculus)]
>d1kjs__ 1.40.1.1.1 C5a anaphylotoxin [human (Homo sapiens)]

B.4 Testing data for specific matrix: SCOP class 1

>d1hyp__ 1.42.1.1.1 Soybean hydrophobic protein [soybean (Glycine max)]

>d1ihfa_ 1.46.1.1.1 Integration host factor (IHF) [Escherichia coli]
>d1vola1 1.59.1.2.1 (1-95) Transcription factor IIB (TFIIB), the core domain [Human (Homo sapiens)]
>d1lla_1 1.70.1.1.1 (1-359) Hemocyanin, N-terminal and middle domains [Limulus polyphemus]
>d1utg__ 1.72.1.1.1 Uteroglobin [Rabbit (Oryctolagus cuniculus)]
>d1csh__ 1.74.1.1.1 Citrate synthase [chicken (Gallus gallus)]
>d1rgp__ 1.83.1.1.1 p50 RhoGAP domain [human (Homo sapiens)]
>d2sblb1 1.85.1.1.1 (125-807) Lipoxigenase, C-terminal domain [Soybean (Glycine max), isozyme L1]
>d1poa__ 1.95.1.2.1 Snake phospholipase A2 [taiwan cobra (Naja naja atra)]
>d1rtm11 1.97.1.1.1 (1-32) Mannose-binding protein A, triple coiled-coil domain [rat (Rattus rattus)]
>d2ztaa_ 1.97.2.1.1 GCN4 [Yeast Saccharomyces cerevisiae]
>d1ifj__ 1.97.3.1.1 Inovirus (filamentous phage) major coat protein [strain fd]

B.5 Training data for specific matrix: SCOP class 3

>d1pkm_2 3.1.9.1.1 (1-104,207-384) Pyruvate kinase, N-terminal domain [cat muscle (Felis domestica)]
>d2myr__ 3.1.1.4.1 Myrosinase [white mustard (Sinapis alba)]
>d1akz__ 3.11.1.1.1 Uracil-DNA glycosylase [Human (Homo sapiens)]
>d1pii_1 3.1.8.1.1 (1-254) N-(5'phosphoribosyl)antranilate

(PRA)isomerase [Escherichia coli]
>d1hvq__ 3.1.1.5.1 Hevamine A (chitinase/lysozyme) [Para
rubber tree Hevea brasiliensis]
>d1muca1 3.1.6.2.1 (119-360) Muconate-lactonizing enzyme
[Pseudomonas putida]
>d1tml__ 3.2.1.1.1 Cellulase E2 [Thermomonospora fusca,
strain yx]
>d1zyma_ 3.5.1.2.1 N-terminal domain of enzyme I of the
PEP:sugar phosphotransferase system [Escherichia coli]
>d1nzya_ 3.8.1.1.1 4-Chlorobenzoyl-CoA dehalogenase
[Pseidomonas sp. strain CBS-3]
>d1pau.1 3.10.1.1.1 (a,b) Apopain [Human (Homo sapiens)]
>d1scua1 3.13.3.1.1 (122-288) Succinyl-CoA synthetase,
alpha-chain, C-terminal domain [Escherichia coli]
>d2naca1 3.13.9.1.1 (1-147,336-374) Formate dehydrogenase
[Pseudomonas sp. 101]
>d1fnb_2 3.14.1.1.1 (137-296) Ferredoxin reductase [spinach
(Spinacia oleracea)]
>d1gpma1 3.15.2.1.1 (206-380) GMP synthetase, central domain
[Escherichia coli]
>d2naca2 3.19.1.4.1 (148-335) Formate dehydrogenase
[Pseudomonas sp. 101]
>d1yvei2 3.19.1.6.1 (1-225) Acetohydroxy acid isomeroreductase,
ketoacid reductoisomerase (KARI) [spinach (Spinacia oleracea)]
>d1hrda1 3.19.1.7.1 (195-449) Glutamate dehydrogenase,
C-terminal domain [Clostridium symbiosum]
>d1pyda1 3.21.1.1.1 (173-341) Pyruvate decarboxylase [brewer's
yeast (Saccharomyces) uvarum strain]

B.6 Testing data for specific matrix: SCOP class 3

>d1pyda2 3.24.1.1.1 (1-172) Pyruvate decarboxylase [brewer's yeast (Saccharomyces) uvarum strain]
>d1gky__ 3.25.1.1.1 Guanylate kinase [baker's yeast (Saccharomyces cerevisiae)]
>d5p21__ 3.25.1.3.1 cH-p21 Ras protein [human (Homo sapiens)]
>d2mysa2 3.25.1.4.1 (1-28,70-669) Myosin S1, motor domain [chicken (Gallus gallus) pectoral muscle]
>d1dts__ 3.25.1.5.1 Dethiobiotin synthetase [Escherichia coli]
>d2reb_1 3.25.1.6.1 (1-243) RecA protein, ATPase-domain [Escherichia coli]
>d2sece_ 3.28.1.1.1 Subtilisin Carlsberg [Bacillus subtilis]
>d3cla__ 3.30.1.1.1 Chloramphenicol acetyltransferase [Escherichia coli]
>d2hnp__ 3.32.1.2.1 Tyrosine phosphatase [Human (Homo sapiens) 1B]
>d2trxa_ 3.33.1.1.1 Thioredoxin [Escherichia coli]
>d1dsba2 3.33.1.3.1 (1-64,129-188) Disulphide-bond formation facilitator (DSBA) [Escherichia coli]
>d2gsta2 3.33.1.5.1 (1-84) Glutathione S-transferase [rat (Rattus rattus)]

Appendix C

Glossary

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.

Base pair (bp): Two nitrogenous bases (adenine and thymine or guanine and cytosine) held together by weak bonds. Two strands of DNA are held together in the shape of a double helix by the bonds between base pairs.

Base sequence: The order of nucleotide bases in a DNA molecule.

Biotechnology: A set of biological techniques developed through basic research and now applied to research and product development. In particular, the use by industry of recombinant DNA, cell fusion, and new bioprocessing techniques.

Chromosomes: The self-replicating genetic structures of cells containing the cellular DNA that bears in its nucleotide sequence the linear array of genes. In prokaryotes, chromosomal DNA is circular, and the entire genome is carried on one chromosome. Eukaryotic genomes consist of a number of

chromosomes whose DNA is associated with different kinds of proteins.

Codon: See genetic code.

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): The molecule that encodes genetic information. DNA is a double-stranded molecule held together by weak bonds between base pairs of nucleotides. The four nucleotides in DNA contain the bases: adenine (A), guanine (G), cytosine (C), and thymine (T). In nature, base pairs form only between A and T and between G and C; thus the base sequence of each single strand can be deduced from that of its partner.

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

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

Double helix: The shape that two linear strands of DNA assume when bonded together.

Enzyme: A protein that acts as a catalyst, speeding the rate at which a biochemical reaction proceeds but not altering the direction or nature of the reaction.

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).

Gene expression: The process by which a genes 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).

Gene product: The biochemical material, either RNA or protein, resulting from expression of a gene. The amount of gene product is used to measure how active a gene is; abnormal amounts can be correlated with disease-causing alleles.

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.

Genome: All the genetic material in the chromosomes of a particular organism; its size is generally given as its total number of base pairs.

Homologies: Similarities in DNA or protein sequences between individuals of the same species or among different species.

In vitro: Outside a living organism (or in the test tube).

In vivo: Inside a living organism.

Nucleic acid: A large molecule composed of nucleotide subunits.

Nucleotide: A subunit of DNA or RNA consisting of a nitrogenous base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or

cytosine in RNA), a phosphate molecule, and a sugar molecule (deoxyribose in DNA and ribose in RNA). Thousands of nucleotides are linked to form a DNA or RNA molecule. See DNA, base pair, RNA.

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.

Ribonucleic acid (RNA): A chemical found in the nucleus and cytoplasm of cells; it plays an important role in protein synthesis and other chemical activities of the cell. The structure of RNA is similar to that of DNA. There are several classes of RNA molecules, including messenger RNA, transfer RNA, ribosomal RNA, and other small RNAs, each serving a different purpose.

Ribosomes: Small cellular components composed of specialised ribosomal RNA and protein; site of protein synthesis. See ribonucleic acid (RNA).

Sequencing: Determination of the order of nucleotides (base sequences) in a DNA or RNA molecule or the order of amino acids in a protein.

Transcription: The synthesis of an RNA copy from a sequence of DNA (a gene); the first step in gene expression. Compare translation.

Transfer RNA (tRNA): A class of RNA having structures with triplet nucleotide sequences that are complementary to the triplet nucleotide coding sequences of mRNA. The role of tRNAs in protein synthesis is to bond with amino acids and transfer them to the ribosomes, where proteins are assembled according to the genetic code carried by mRNA.

Translation: The process in which the genetic code carried by mRNA directs the synthesis of proteins from amino acids.

Virus: A noncellular biological entity that can reproduce only within a host cell. Viruses consist of nucleic acid covered by protein; some animal viruses are also surrounded by membrane. Inside the infected cell, the virus uses the synthetic capability of the host to produce progeny virus.