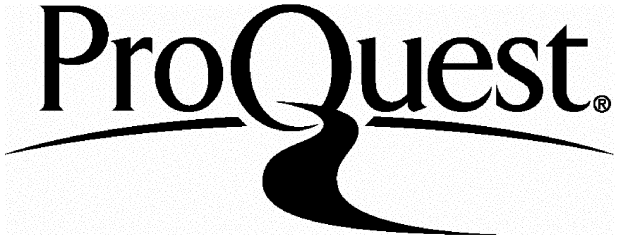
Sensitive and rapid methods for comparing and searching biological sequence data

John Norman Hatwell

September 2001

A thesis submitted in part fulfilment of the requirements of the University of London for the degree of Master of Philosophy

Department of Mathematical Biology
National Institute for Medical Research
The Ridgeway
Mill Hill
London
NW7 1AA



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

Sequence database searching is a key tool in current bioinformatics. To improve accuracy, sequence database searches are often performed iteratively: taking the results of one search as input for the next. The object of this approach being to progressively isolate increasingly distant relations of the original query sequence. In practice this method works well when it is supervised by an 'expert eye' which can determine when an alignment is good and when sequences should be excluded from it, but attempts to automate this process have proven difficult. At present PSI-BLAST is one of the few effective attempts, but a misalignment of sequences or the wrongful inclusion of a sequence will still rapidly destroy the specificity of the probe, making incorrect matches more likely.

By combining the search program Quest, which is capable of searching a database using full length multiple sequence alignments, with independent sequence alignment and assessment programs, we have been able to reduce the occurrence of this problem. We use a multiple alignment package to generate an accurate alignment of all hits generated by the Quest program. Sequences that do not appear to 'fit' with the rest of the alignment are automatically removed by the separate alignment assessment program Mulfil. The resulting alignment is fed back to Quest for the next iteration. This scheme has shown to generate results significantly better than those of PSI-BLAST. Whilst the

total number of correct homologues identified was not increased, the number of incorrect ones dropped significantly.

In addition, further work demonstrated that equally good quality results are possible without the use of multiple alignment or profile searching. The Cascade-and-Cluster scheme uses intermediate sequences and a simple clustering procedure and is able to produce a result almost equally sensitive and selective as our previous scheme, whilst running upto ten-fold faster.

Contents

Abstract 2

List of Figures 6

List of Tables 7

Abbreviations 8

Acknowledgements 9

1 Introduction 10
- 1.1 DNA and the Genetic Code . . . 11
  - 1.1.1 DNA . . . 11
  - 1.1.2 The Genetic Code . . . 13
- 1.2 The Journey from Gene to Protein . . . 14
  - 1.2.1 Transcription . . . 15
  - 1.2.2 Translation . . . 15
- 1.3 Evolution . . . 16
- 1.4 Sequence Databases . . . 19
- 1.5 Sequence Comparison & Database Searching . . . 22
  - 1.5.1 Sequence Alignment . . . 22
  - 1.5.2 Database Searching . . . 32
- 1.6 Aims . . . 44

2 Quest: A Profile Based Search Program 47
- 2.1 Quest: Method of Operation . . . 49
  - 2.1.1 Stage I: Input & Preprocessing . . . 49
  - 2.1.2 Stage II: Tripeptide Matches & Segment Extension . . . 56
  - 2.1.3 Stage III: Segment Assembly & Sequence Scoring . . . 59
  - 2.1.4 Stage IV: Hit Selection & Output . . . 63
- 2.2 Quest: Parameters, Modifiers and Run-Modes . . . 64
  - 2.2.1 Quest Parameters . . . 65
  - 2.2.2 Quest Modifiers . . . 69
  - 2.2.3 Quest Run-Modes . . . 70
- 2.3 Quest for Parallel Architectures and Workstation Clusters . . . 74

2.3.1 The Cluster 75
2.3.2 MPI 76
2.3.3 Simple Parallel Database Searching 77

3 The QUEST Scheme 81
3.1 The Search Phase 82
3.2 The Alignment Phase 83
3.2.1 MULTAL 83
3.2.2 CLUSTAL W 85
3.2.3 Praline 86
3.2.4 T-Coffee 87
3.2.5 Other Alternatives 88
3.2.6 The Final Decision 89
3.3 The Assessment Phase 91
3.3.1 T-Coffee 92
3.3.2 Mulfil 92
3.3.3 The Final Decision 95
3.4 The Method of Iteration 95
3.5 Cascade-and-Cluster - A New Scheme? 98

4 Benchmarks 105
4.1 Methods 108
4.2 Results & Analysis 110
4.2.1 Benchmarking Quest 110
4.2.2 Quest Parameter Effects 113
4.2.3 Benchmarking the QUEST Scheme 119
4.2.4 Cascade-and-Cluster 124
4.2.5 PSI-BLAST Version 2: An Improved Challenger 127

5 The Quest Server: Our Window on the World Wide Web 132

6 Discussion & Conclusions 139

Bibliography 151

List of Figures

1.1 Translation 16
1.2 Comparison of Gap Costs 26
1.3 Dynamic Programming 27
1.4 The QUEST Scheme 44

2.1 Input File Format 51
2.2 PSSM Construction 54
2.3 Segment Extension 58
2.4 Segment Assembly 61
2.5 An Example Quest Output 63
2.6 Topology of the Linux Cluster 75

3.1 The Real QUEST Scheme 97
3.2 A Sequence Cluster 100
3.3 Cascade-and-Cluster 102

4.1 Quest Benchmark Results 111
4.2 Score Cutoff Effects 114
4.3 Effects of the Strictness Parameter 115
4.4 Gap Penalty Effects 117
4.5 Benchmarking the QUEST Scheme 120
4.6 Parameter Effects on the QUEST Scheme 122
4.7 Effects of Alignment Phase on the QUEST Scheme 123
4.8 Cascade-and-Cluster Benchmark 125
4.9 PSI-BLAST Version 2 128
4.10 PSI-BLAST vs BLASTPGP 130

5.1 The Quest Web Server 134
5.2 Example Results Page: Part I 136
5.3 Example Results Page: Part II 137

List of Tables

1.1 The Universal Genetic Code 14
1.2 Amino Acids 14

2.1 Quest Command Line Options 65

Abbreviations

A	Adenine
BLAST	Basic Local Alignment Search Tool
C	Cytosine
DNA	Deoxyribonucleic acid
DDBJ	DNA DataBank of Japan
EBI	European Bioinformatics Institute
EMBL	European Molecular Biology Laboratory
EPQ	Errors Per Query
G	Guanine
HMM	Hidden Markov Model
HSP	High-scoring Segment Pair
HTML	HyperText Markup Language
ISS	Intermediate Sequence Search
mRNA	messenger RNA
NCBI	National Center for Biotechnology Information
NJ	Neighbour Joining
MPI	Message Parsing Interface
PAM	Point Accepted Mutation
PDB	Protein DataBank
PSSM	Position Specific Scoring Matrix
RNA	Ribonucleic acid
rRNA	ribosomal RNA
SCOP	Structural Classification Of Proteins
SYSTERS	SYSTEmatic Re-Searching
T	Thymine
tRNA	transfer RNA
U	Uracil
UPGMA	Un-weighted Pair Group Mean Arithmetic

Acknowledgements

I would Like to thank the following people for their help and assistance throughout the length of this project. My supervisors Dr. William Taylor and Dr. Jaap Heringa for their ideas, criticism and discussion. Nigel Douglas for his knowledge, patience and great help on all computing matters. Dr. Jens Kleinjung my co-worker on the Quest project, without whose discussion and input this project would not of been possible. Finally thanks go to all the remaining members of the Mathematical Biology department at the National Institute of Medical Research for all their help and support during my two year stay.

Chapter 1

Introduction

It was probably not anticipated how the development of rapid DNA sequencing technology in the 1970's would lead to such an explosion of freely available biological sequence information. The amount of sequence data being produced has increased exponentially year on year. What started as a trickle is now thanks to the various genome sequencing projects a torrent of information pouring in to the sequence databases. The entire genomes of several organisms have now been sequenced, these range from; The simple prokaryote, *Mycoplasma genitalium* (Fraser et al., 1995); the first multicellular eukaryote to be sequenced,*Caenorhabditis elegans* (The C.elegans Sequencing Consortium, 1998); the first plant genome, *Arabidopsis thaliana* (The Arabidopsis Genome Initiative, 2000). Within the last year even the draft copy of the Human genome has been published (International Human Genome Sequencing Consortium, 2001).

The huge amount of data available in the public sequence databases is an amazing resource. As the databases grow so does their potential use. More sequences have

probably been characterised by database searches than by any other technology. However, it is also true that the larger the database grows the more difficult it is to use them effectively. The non-redundant version of the Genbank database contains almost 750,000 protein sequences. It is obviously not possible to search through a dataset such as this manually and indeed several computer programs have been in general use for over a decade now, the most popular being BLAST (Altschul et al., 1990) and FASTA (Pearson, 1990) which are available at the NCBI and EBI websites respectively. However these are no longer always the best tools for the job. To understand the problems and pitfalls of searching these sequence databases we need to first understand some of the complexity of the biological systems that produce these sequences in the first place.

1.1 DNA and the Genetic Code

1.1.1 DNA

Deoxyribonucleic acid or DNA is the medium in which genetic information is encoded and recorded. It is through DNA that a parent passes genetic information to its offspring. This system is almost ubiquitous in nature with only a few organisms exceptions to this rule, even then most have a DNA stage in their life cycle. DNA is a linear polymer made up of deoxyribonucleotide monomers. Each of these subunits is made up of a base, phosphate group and a 2-deoxyribose sugar. These nucleotides are connected through phosophodiester bonds linking the sugar and phosphate groups of neighbouring nucleotides. There are four different nucleotides which go to make up the language

of the genetic code. They are defined by the one of four bases they carry, these are cytosine (C) and thymine(T) which are purines and adenine (A) and guanine(G) which are pyrimidines.

Structure

In eukaryotic organisms the nuclear DNA occurs as a duplex of two complementary antiparallel strands arranged in the classic right-handed double helix. This structure is stabilised by the hydrogen bonds between complementary base pairs, A binding to T and C to G. Within the DNA strand it is possible to define discrete units known as genes. A gene is a region of the DNA strand that codes for a particular product, usually a protein, also including any regulatory regions e.g. promoters and inhibitors that regulate the expression of the gene. The total sum of all the genetic information contained in an organism is known as its genome. A eukaryotic genome may be made up of more than one double helix of DNA, each of these strands are known as chromosomes. The human genome for example is made up of 23 pairs of chromosomes.

Replication

As the genetic material it is obviously important that DNA is capable of being duplicated so that parents may beget offspring. This procedure is known as replication. DNA is replicated by a semi-conservative mechanism, which reduced it to simplest terms involves the splitting of the two strands that make the duplex and synthesising their complementary strands to form two new duplexes. Each of these new DNA molecules contains one newly synthesised strand and one from the original parent molecule, hence

the term semi-conservative replication. This procedure is overseen by an enzymatic complex known as DNA polymerase, which is responsible for the assembly and formation of the complementary strands.

1.1.2 The Genetic Code

In general genes encode proteins. Like DNA proteins are polymers but instead of nucleotide the monomeric units are amino acids. There are 20 different amino acids commonly utilised in all biological systems (Table 1.2). With its alphabet of four letters one nucleotide obviously does not code for one amino acid. Dinucleotides would only be able to encode 16 amino acids, however trinucleotides could encode 64 amino acids. This turns out to be the correct answer; the 64 possible trinucleotides are known as codons and each has a particular meaning. The majority, 61 of them encode the 20 amino acids, with varying degrees of redundancy. The remaining 3 are stop codons they mark out the end of the protein coding sequences. The 64 possible codons make up the genetic code are detailed in Table 1.1.

Redundancy

Redundancy is implicit in the genetic code, there are $4^3 = 64$ possible codons and only 20 (Table 1.2) amino acids. This simple calculation could infer two things: either most of the codons do not code for anything or many of the amino acids are represented by more than one codon. This is obviously the case, however it is important to note that this redundancy is not a case of each amino acid just having three possible codons each,

1^{st} Residue	2^{nd} Residue	3^{rd} Residue			
		T	C	A	G
T	T	Phe	Phe	Leu	Leu
	C	Ser	Ser	Ser	Ser
	A	Tyr	Tyr	Stop[a]	Stop[a]
	G	Cys	Cys	Stop[a]	Trp
C	T	Leu	Leu	Leu	Leu
	C	Pro	Pro	Pro	Pro
	A	His	His	Gln	Gln
	G	Arg	Arg	Arg	Arg
A	T	Ile	Ile	Ile	Met[b]
	C	Thr	Thr	Thr	Thr
	A	Asn	Asn	Lys	Lys
	G	Ser	Ser	Arg	Arg
G	T	Val	Val	Val	Val[b]
	C	Ala	Ala	Ala	Ala
	A	Asp	Asp	Glu	Glu
	G	Gly	Gly	Gly	Gly

Table 1.1: **The Universal Genetic Code**
Throughout nature there are very few exceptions to this code; almost all protein coding genes are translated using this system. [a]Stop denotes a stop codon and does not encode an amino acid. [b]Both AUG and GUG may serve as initiation codons.

A	Ala	Alanine
C	Cys	Cysteine
D	Asp	Aspartic Acid
E	Glu	Glutamic Acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
Y	Tyr	Tyrosine

Table 1.2: **Amino Acids**
The 20 amino acids plus their respective one and three letter codes.

the distribution is far from even. As shown in Table 1.1 leucine, arginine and serine have six different codons, cysteine and tyrosine have two whilst methionine and Tryptophan have only one.

1.2 The Journey from Gene to Protein

The journey from a DNA sequence to a protein is not a direct one. There are many intermediate stages, which make up the two major processes involved, transcription and translation.

1.2.1 Transcription

Transcription generates a single stranded ribonucleic acid (RNA) molecule complementary in sequence to the template DNA strand. There are three types of RNA that are involved in protein synthesis: messenger RNA (mRNA), transfer RNA (tRNA) and ribosomal RNA (rRNA). Of these it is mRNA that encodes the protein sequence, whilst tRNA and rRNA are active parts of the translation machinery. As a result it is mRNA I will focus on here.

Transcription has many parallels with DNA replication. Firstly the DNA duplex is partially unwound. This allows the binding of an RNA polymerase; this would be RNA polymerase II in humans. This transcribes the DNA sequence into a complementary strand of RNA. RNA is very similar to DNA except that the sugar that makes up the backbone is ribose rather than deoxyribose. Secondly RNA has a slightly different base composition to DNA with uracil replacing thymine. This single strand of RNA has a cap of 5-methyl-cytosine and a poly-adenosine tail added to it to create the final messenger RNA molecule. The mRNA strand must then be exported out of the nucleus before it can undergo translation.

1.2.2 Translation

Once the mRNA is exported from the nucleus to the cytoplasm it will be bound by a ribosome and translated in to protein (Figure 1.1). It is at this stage that the other RNA types mentioned earlier play a role. The ribosome itself is a complex of both protein and ribosomal RNA; its binding to the mRNA initialises translation. There is a transfer RNA

for each codon in the genetic code, these molecules can reversibly bind to their specific amino acids. Each of these molecules also contain an anti-codon for the amino acid they carry, which is a trinucleotide with a complementary sequence to the codon. Via this anti-codon they base pair to the mRNA bringing together the amino acid monomers in the correct sequence order. As this occurs the ribosome catalyses the polymerisation of these monomer units and their release by the tRNA. Once complete the protein is released from the ribosome, after which it may undergo post-translational processing or may be immediately exported to where it is required.

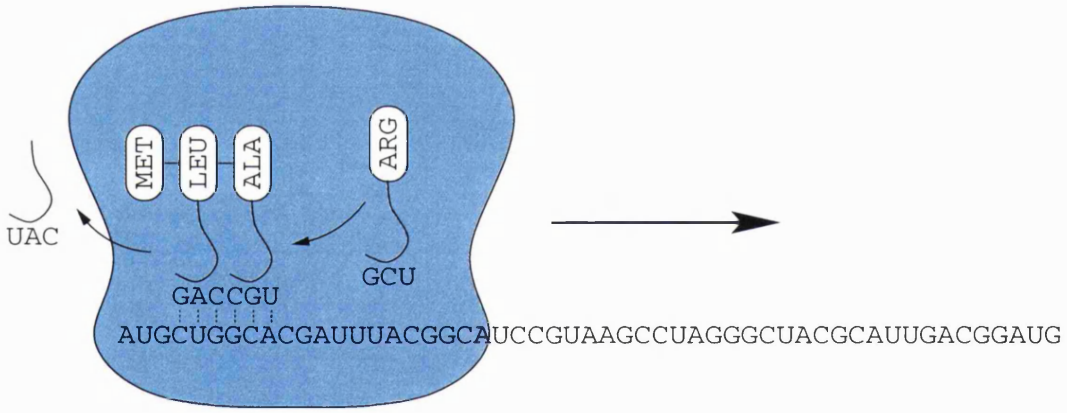


Figure 1.1: **Translation.** The ribosome moves along the mRNA strand in the direction indicated catalysing the polymerisation of the amino acid monomers, which are assembled into the correct positions by the influence of the respective tRNA molecules.

1.3 Evolution

Living is a dangerous thing to do, during every minute of every day our cells are under attack, often from the products of our own metabolism. There are many mechanisms through which a cell and more specifically it's DNA may become damaged. Ultra-violet

light, free radicals and a variety of chemicals can all damage and cause lesions on DNA. These substances are mutagens i.e. they can induce mutations, in humans they would often be referred to as carcinogens as many mutations can lead to the development of cancer. Most of the time damage to a DNA strand is repaired without problems, however occasionally a mistake in the repair mechanism will culminate in a mutation, causing a change in the genetic sequence. DNA damage can also result in a mutation during replication, when base pair mismatches and a host of other problems can occur. Even in the absence of damage the process of replication can lead to mutation, for example the DNA polymerase complex could slip leading to the duplication or deletion of a short region of the DNA strand. If these mutations occur in germ cells then they will be passed on to the progeny of the organism, becoming variants in the gene pool on which evolution can then act.

For the purposes of this work we could define evolution as a change in the DNA sequence of an organism that is passed onto it's progeny. This could be as simple as a single base change in an organisms entire genome. However it is probably more true to say that evolution has occurred when that change has become fixed in a population of organisms, be it via selection or other means. A single change in a single organism really only constitutes a new mutation or variant in the gene pool. There are a great many factors that can determine whether that change will become fixed in the population. It could be that the change is a synonymous one, i.e. it does not cause any change in the protein sequence. This can be brought about if the mutation occurs in a non-coding region of the genome or if a codon is changed in such a way that it still encodes the

same amino acid. This is possible due to the degeneracy of the genetic code; changes in the third base of many codons produce this result. If the mutation is synonymous then the forces that will lead to it being fixed in a population are essentially random. If it's frequency rises high enough by chance to become fixed then we can say this is a result of genetic drift (King and Jukes, 1969). These types of changes can make two DNA sequences that encode nearly identical protein sequences quite different at the nucleotide level.

The second type of changes are non-synonymous ones. These are mutations that cause a change in the protein sequence, for example a change in the second base of a codon will nearly always have this effect. If the effect of this change is very mild causing little or no change to the protein's properties or activity then the self same random effects can lead to loss or fixation, by a process known as neutral or nearly-neutral evolution (Kimura, 1983). However if the mutation has a stronger effect e.g. conferring an advantage of some kind to the carrier then natural selection may act to rapidly fix the change in the population or if the effect is deleterious to remove it.

It is the cumulative effect of all these changes and many others that lead to the differences observed between homologous sequences both within and between closely related organisms. This is why when we wish to search for related sequences in sequence databases it is essential to take account of evolution in one form or another.

1.4 Sequence Databases

With the advent of rapid DNA sequencing technology came the development of biological sequence databases. Since their inception they have multiplied to create a large variety of both specialised and general repositories. The largest and most well known is the Genbank/EMBL/DDBJ database, containing virtually all the DNA and protein sequences known to date. This is actually made up of three databases the Genbank database from the NCBI, the EMBL database maintained at the EBI and the DNA Databank of Japan. These three organisations share all data and submissions with each other to create a 'universally accessible' databank, for the remainder of this thesis I will refer to this simply as the Genbank database.

There are several other commonly used databases. SWISS-PROT (Bairoch and Apweiler, 2000) is a curated protein database with a minimum level of redundancy. Its intention is to provide a high level of annotated information as well as the protein sequence itself. This annotation can include details such as function, structure and post-translational modifications. TrEMBL (Bairoch and Apweiler, 2000) is a supplement to the SWISS-PROT database. It includes and attempts to automatically annotate all the translations of Genbank nucleotide sequence entries not yet integrated in SWISS-PROT. The protein data bank (PDB) (Berman et al., 2000) is a database of known three-dimensional protein structures. Whilst this is not solely a sequence resource it is extremely useful, structures can help us to judge when sequences are true homologues of one another even when the sequences in question do not appear to share any significant homology. This is useful not only in general biology but also in this project in particular

as it allows us to judge how well our database searching system is working by being able to identify and quantify correct and incorrect hits. Derivatives of the PDB are particularly useful for this purpose, specifically SCOP (Murzin et al., 1995) and CATH (Orengo et al., 1997). These data sets seek to hierarchically classify sequences according to their structure. SCOP is manually produced and focuses on reliability, only classifying proteins together if there is significant evidence that they are related. CATH on the other hand is largely automatically generated but still seeks to classify proteins and their folds into homologous families. These classifications can then be used for grading sequence searching systems or as a method of annotating new sequences (Park et al., 1997; Brenner et al., 1998; Park et al., 1998; Müller et al., 1999; Salamov et al., 1999; Pearl et al., 2000; Wallqvist et al., 2000).

In addition to these more established resources there are now a significant number of organism specific databases, which have resulted from various genome sequencing projects in addition to previous work. These include FlyBase (The FlyBase Consortium, 1999) for *Drosophila* and WormBase (Stein et al., 2001) for *C. elegans*. Probably the most well known to the world at large is the human genome sequencing project (International Human Genome Sequencing Consortium, 2001), whilst the results of this are still not fully complete, preliminary results are available in various forms. The impact of these various genome sequencing projects upon all the databases has been massive. Already growing at an alarming rate, the number of sequences being deposited has taken yet another leap over the past two years. As these databases get larger so do the problems in using them.

Whilst these large centralised data resources do have great potential use for scientists, they do also present a problem. The non-redundant Genbank database has 750,000 sequences; in this amount of data the problem is to locate the ones you are interested in. Ideally each sequences would be catalogued by organism, position in the genome, name of gene, function of protein and we could then simply search for and find our sequence of interest. Unfortunately things are not as simple as this. Whilst this information is often put into the sequence entries, it is not always available. Many genes have been sequenced without knowing what the protein they encode. This type of data is routinely produced by the various genome sequencing projects, in fact they produce large amounts of sequence data where it is not even known where the genes lie yet alone what they encode.

Furthermore this approach is fine if you have a simple query like: Find all globin gene sequences from the mouse *Mus musculus*. Effective systems to do this job already exist e.g. the *Entrez* interface (`http://www.ncbi.nlm.nih.gov/Entrez/`) at the NCBI website. However what if we have a DNA or protein sequence and we want to find any similar sequences in the database. For example we might have a fragment of a gene and wish to find out if the entire length has already been sequenced. Or we may wish to identify homologues in different species, or other members of the same family of genes. These types of problems require a different type of solution.

1.5 Sequence Comparison & Database Searching

The need to be able to compare sequences with one and other has always been apparent. Initially a simple question like how do two related sequences line up against each other i.e. which residues are conserved could be answered by aligning the two sequences by eye. However this is a time consuming process and requires a certain degree of expert knowledge. What was needed was some method of automating this process.

1.5.1 Sequence Alignment

Sequence alignment is a far from trivial process. When aligning by eye an expert can take many factors into account including the biological properties and the prevalence of the various amino acids and also perhaps their relationship to the surrounding sequence e.g. taking into account knowledge of the proteins structure. Also most importantly it has to be decided when and where to introduce gaps in the sequences to fully align them. It is therefore not as simple a task as many might think for a computer to automate.

When attempting to align sequences it is important to bear in mind which amino acids you consider similar and which dissimilar. The simplest approach would be to consider only identities, for example an amino acid scoring matrix based on identities only would give a score of 1 to an identical match and zero otherwise. Whilst this system would work for closely related proteins with very high sequence identities, it would be useless for more divergent ones. A relatively simple way around this problem is to consider which amino acids have similar physical and chemical properties for example we could consider arginine and lysine similar as then both contain basic groups and aspartic acid would be

dissimilar because of its acidic group. Qualitatively this method makes a lot of sense however it is difficult to quantify, for example how much more similar would we consider serine to threonine as opposed to tyrosine and vice versa. Another simple solution would be to use the genetic code (Table 1.1) itself to judge the relationships, it is possible to calculate the minimum number of base changes required to change any amino acid into any other amino acid. It is also possible to draw inference on how much we can read in to amino acid identities (when an amino acid matches itself). Methionine only has one codon whereas serine has six; should a serine-serine match only score a sixth of a methionine-methionine one, the genetic code would draw you to think so. As attractive as this concept of scoring may be it is far from prefect. It probably does give an indication of how often specific amino acid changes occur. However it does not indicate how often these changes become fixed and it does not take into account the effects of evolution, specifically those of selection. Therefore a way of judging how frequently one amino acid changes to another in real sequences is needed.

The first attempt to do this was the PAM (Point Accepted Mutations) matrices (Dayhoff et al., 1978). This series of matrices was based upon alignments of very closely related sequences sharing at least 85% identity. The amino acid substitutions observed in these alignments were recorded and converted to mutational probabilities according to 1% accepted mutations - one amino acid changed in 100. This matrix was then scaled by self-multiplication to produce the other matrices of the set which are applicable to more distantly related sequences e.g. PAM250 log odds matrix represents 250 substitutions in 100 amino acids. Despite the relatively small datasets these matrices

have proven to be remarkably effective. The work has been repeated with newer larger datasets (Jones et al., 1992) but the original set are still widely used. In fact it is only in the last decade that another popular substitution matrix set has arisen. The BLOSUM (Henikoff and Henikoff, 1992) matrices were derived from approximately 2000 blocks of aligned sequence segments from more than 500 groups of related proteins. Whilst both the BLOSUM and PAM matrices have proven very effective for sequence alignment and database searching there is a certain amount of empirical evidence (Henikoff and Henikoff, 1992; Henikoff and Henikoff, 1993) pointing to the BLOSUM set being slightly more reliable.

These matrices provide a reliable method of judging the probability that one amino acid will be substituted for another over evolutionary time. The next dilemma is how to use this information to compute the best alignment of two protein sequences.

Dynamic programming

The solution to this problem is most generally credited to Needleman & Wunsch (Needleman and Wunsch, 1970). The procedure they introduced is now commonly known as dynamic programming. The dynamic programming algorithm operates in two steps. Taking two sequences A and B firstly a search matrix is constructed of dimensions mn with m being the length of sequence A and n of sequence B. The matrix is filled from the top left to the bottom right corner, each cell $[i, j]$ receives the score of the exchange value of residue A_i and B_j plus the maximum score of row $i - 1$ and column $j - 1$ minus any incurred gap penalties. Cell $[i, j]$ therefore contains the maximum score for an alignment of the two sequences up to the point of cell $[i, j]$. This first step can be

simply written as:

$$S[i,j] = s[i,j] + \text{Max} \begin{cases} S[i-1, j-1] \\ \max_{1<x<i}(S[i-x, j-1] - P(x-1)) \\ \max_{1<y<j}(S[i-1, j-y] - P(y-1)) \end{cases} \quad (1.1)$$

Where $S[i,j]$ is the score of the best alignment of sequences A and B up to residues i and j respectively. The $s[i,j]$ term refers to the exchange value of the residues associated with cell $[i,j]$. The Max term denotes the maximum score of the three equations detailed in the brackets. The first term would be the maximum in the cases where no gap had to be inserted at this position in the alignment. The second would be true if a gap had to be inserted in sequence B for residues i and j to be aligned. The third term obviously corresponds to a gap insertion in sequence A. $P(x-1)$ is a non-negative penalty value for a gap of length $x-1$ with $1<x<i$ denoting the range of values x could possibly take.

In its original implementation (Needleman and Wunsch, 1970) this gap penalty was defined as a fixed value for the inclusion of a gap of any length. This idea was soon replaced by the concept of length-proportional gap costs (Sellers, 1974) where a gap of length x would have the penalty $P(x) = Gx$, where G is the cost of a gap of length 1. However over the years it was noticed that the highest scoring alignments produced using length-proportional gap costs often contained a large number of short gaps. Such a high number of short insertions or deletions was not thought to be very plausible. Because a single mutation may result in the insertion or deletion of a large number of residues e.g. the insertion or excision of a transposon, it was hypothesised that a long gap should

not cost sizably more than a short one. Affine gap costs use two gap penalties, G_o the penalty for opening a gap and G_e the penalty for extending a gap giving the formula $P(x) = G_o + G_e x$. This scheme is the predominant one employed by modern alignment packages, typically G_o will be a order of magnitude larger than G_e therefore making it difficult enough to open a gap but relatively easy to extend it where necessary. The difference between length-dependent and affine gap costs can be seen in Figure 1.2.

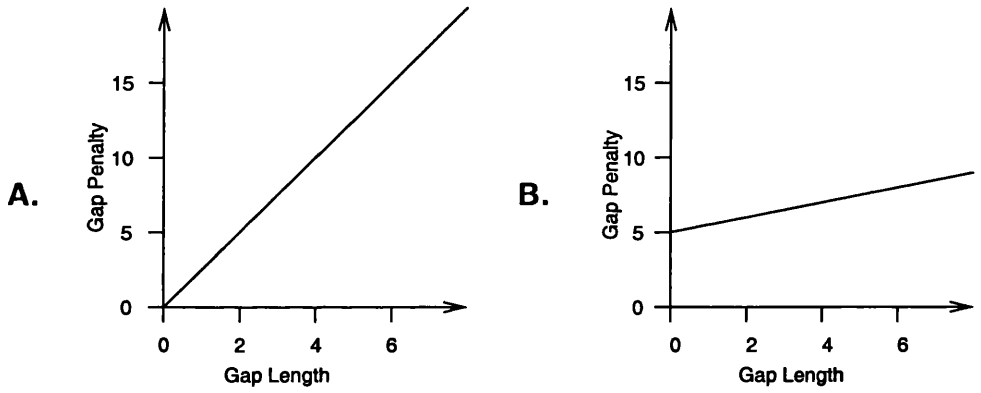


Figure 1.2: **A comparison of length dependent and affine gap costs** A. length dependent gap penalty of 5 per unit length. B. Affine gap score; gap opening penalty of 10, gap extension penalty of 1 per unit length.

The second stage of the dynamic programming algorithm is the traceback step. It is at this point that the actual optimal or highest scoring alignment is reconstructed. Starting from the cell with the highest alignment score which will lie in the final row or in on the final column. The path is traced back following successively lower scores, each time selecting the highest scoring cell from the previous column or row from the present position. This continues until both the sequences are fully aligned i.e. until the traceback reaches the first row or column. This procedure is graphically illustrated in Figure 1.3.

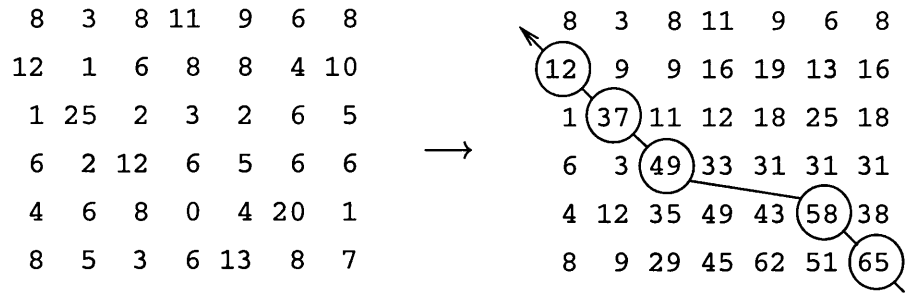


Figure 1.3: **The Dynamic Programming Algorithm.** The initial alignment matrix is shown on the left, the values are then summed (taking into account any gaps) to produce the matrix on the right. The final alignment is then realised by the traceback step starting from the highest scoring cell.

The dynamic programming algorithm will always find the optimal alignment between two sequences for a given scoring scheme. This is fine when the sequences share a certain degree of similarity e.g. 40% sequence identity. When the sequences are much more divergent perhaps only sharing 25% sequence identity a global alignment such as this, where the full lengths of both sequences are aligned, may be far less useful and more worryingly it may be completely wrong. If the two sequences share only short sections of homology then an optimal global alignment may miss these segments either in the noise generated by the divergent regions or the gap penalties may prevent these conserved segments from being successfully joined. Just because it is possible to align two unrelated sequences it does not mean that the sequences share any significant degree of similarity. Two random sequences will share approximately 20% sequence identity when aligned globally. In cases where the sequences are distantly related we are often more interested in whether they share have any homologous or more highly conserved segments.

Local Alignment

In addition to the problems already discussed many proteins are modular in structure, made up of discrete units known as domains. These domains which are mainly defined in a structural context can be present in a wide variety of proteins, although not always in the same order. In such examples a global alignment of two sequences would be meaningless. Consider a case where two proteins share a domain, in sequence A the domain makes up the final quarter of the sequence and in sequence B it makes up the first quarter. In most cases a global alignment method would not pay attention to an alignment of the first section of one sequence to the last of another. In fact in many programs to save time this section of the alignment matrix is not even calculated as a match is such a position is considered unlikely or wrong. There are also other possibilities that can limit the effectiveness of global alignment methods, for example the occurrence of internal sequence repeats.

In these circumstances the best strategy is to undertake a local (Smith and Waterman, 1981) as opposed to global alignment. A good way to describe the local alignment technique is that the most similar regions of two sequences are selected and aligned, paying no attention to dissimilar regions or linear ordering. The method is essentially the same as the global alignment algorithm, with a few additional conditions and stages. A dynamic programming algorithm is implemented but in this case the amino acid exchange values used must include negative values. Any score in the search matrix that would be negative is set as zero. The algorithm relies on dissimilar subsequences producing negative scores allowing them to be discarded by scoring the cells that make them up as

zero. Equation 1.1 thus becomes:

$$S[i,j] = s[i,j] + \text{Max} \begin{cases} S[i-1, j-1] \\ \max_{1<x<i}(S[i-x, j-1] - P(x-1)) \\ \max_{1<y<j}(S[i-1, j-y] - P(y-1)) \\ 0 \end{cases} \tag{1.2}$$

The consequence of these changes is that the highest alignment or more correctly sub-alignment score no longer has to be in the final row or column but can appear anywhere in the matrix. Once the highest scoring segment has been traced back it is but a simple procedure to find the next highest scoring sub-alignment. To do this the second highest scoring cell in the scoring matrix is located and traced back. However when this is done you must be careful to exclude the cells that make up the first sub-alignment, otherwise you will merely traceback along the same route. Following this procedure it is possible to locate all of the top scoring non-intersecting sub-alignments of the two sequences. The point at which the sub-alignments cease to be top scoring is obviously user defined.

Multiple Alignment

The mechanisms covered so far relate to pairwise alignments, i.e. the comparison and search for similarity between a pair of sequences. Whilst the number of available sequences was small this was an adequate solution. However as the number of sequences has grown and specifically when large sets of homologous sequences are available we often want to compare more than two sequences together and align many sequences at

once. In theory a multiple alignment should not be difficult to construct or calculate, however the speed of the dynamic programming algorithm is dependent on the product of the lengths of the sequences being aligned. In a pairwise alignment this is just mn the size of the search matrix, for three sequences we would have to construct a three-dimensional search matrix, for four it would be four-dimensional and so on. The number of calculations quickly become prohibitive. Imagine trying to simultaneously align four sequences each two-hundred amino acids in length, this would give a search matrix of size $200^4 = 1.6 \times 10^9$ this is a 40000 times larger computation space than a pairwise comparison of the same length. With the computation time being proportional to this it is easy to see that even with modern computers the time and resources needed to undertake such calculations is prohibitive. Methods have been developed that allow the multiple alignment of up to 10 sequences of a certain length (Lipman et al., 1989; Johnson and Doolittle, 1986). Although they work in different ways these algorithms make the problem tractable by reducing the search space effectively ruling out possible but unlikely alignment results. Being able to align ten sequences is not enough for most biologists, many of the sequence families in the databases have had a large number of their members sequenced. A simple search shows that well over 500 globins have been sequenced. Trying to complete a true multiple alignment of this number of sequences is frankly ridiculous.

The unfeasibility of a truly simultaneous multiple alignment algorithm that can work with large numbers of sequences has led to the development of various heuristic approaches. The most popular and successful of these has been the progressive alignment

strategy (Hogeweg and Hesper, 1984). The basis of this strategy is to align the closest sequences first and successively adding in the more distant ones until all the sequences are joined in a final multiple alignment. In essence this method reduces the multiple alignment to a serial sequence of pairwise alignments.

A good example of this method is the CLUSTAL alignment programs (Higgins and Sharp, 1988). This original program was designed specifically to run on desktop computers and thus had to be not too computationally demanding. The program firstly uses a fast pairwise alignment step to evaluate all the possible pairs; it then uses this information to construct a guide tree using the un-weighted pair group mean arithmetic (UPGMA) method (Sneath and Sokal, 1973). The sequences are then aligned following the branching order of the guide tree. When groups of sequences came to be aligned the original implementation used consensus sequences to represent the aligned subgroups. The CLUSTAL program has been developed and refined over the years: CLUSTAL V (Higgins et al., 1992) implemented a more efficient dynamic programming routine; CLUSTAL W (Thompson et al., 1994) made use of the Neighbour-Joining (NJ) algorithm (Saitou and Nei, 1987) to construct the guide tree and rather than consensus sequences, sequence blocks were represented using profiles. The success and speed of this method has made CLUSTAL W and CLUSTAL X (Thompson et al., 1997) its graphical counterpart probably the most widely used alignment package.

Other techniques have of course been developed, some such as MULTAL (Taylor, 1988) are variants of the progressive alignment strategy. More recently more novel methods have been developed. SAGA (Notredame and Higgins, 1996; Notredame et al.,

1998) makes use of a genetic algorithm to align the sequences, T-Coffee (Notredame et al., 2000) attempts to integrate information from various sources including local and global alignment information to construct a multiple alignment. Many of these new techniques have much to offer, however none of them match the speed of the progressive alignment approach, which will undoubtedly remain a favourite for some years to come.

1.5.2 Database Searching

Both pairwise and multiple sequence alignment have developed into commonly used tools. These tools are extremely useful for bioinformatics and bench biologists alike. They allow us to compare two or more sequences together and to putatively infer structure, function and evolutionary history. But given a sequence we are interested in how can we search the database for similar sequences. One solution would be to carry out global pairwise alignments of our query sequence against every other sequence in the database, it would then be possible select all the sequences with the highest alignment scores as probable homologues of our query. Unfortunately this is not really a feasible option. Firstly there are at present approximately 750,000 sequences in the non-redundant Genbank database. Doing the better part of a million alignments every time you want to search the database would be a very time intensive method. Secondly a global alignment would not be the best method for assessing similarity, as I have already mentioned global alignment methods can make mistakes when the sequences are distantly homologous, especially when only segments of the sequences share any degree of homology. In effect what we need to do is carry out a local alignment comparison of our query sequence to every member of the database.

Smith-Waterman search

The idea of a Smith-Waterman (Smith and Waterman, 1981) based search is a very attractive one. It should theoretically allow the accurate recognition of even quite remote sequences that share only short regions of homology. The same problem still exists, carrying out such a large number of pairwise local alignments is generally time prohibitive. This has not however prevented the development of such methods, of these the SSEARCH program (Pearson and Lipman, 1988) is probably the most well known. Because this approach is so exhaustive it has been frequently used as a yardstick to compare new approaches to database searching (Shpaer et al., 1996; Agarwal and States, 1998; Brenner et al., 1998). Attempts to accelerate this procedure have frequently relied on the use of expensive parallel computers. These are generally refered to as hardware implementations of Smith-Waterman, as well as being much faster have proven to be just as effective as their software counterparts at finding homologues (Shpaer et al., 1996). However the high cost of this machinery has meant that they are not in widespread use. Recent work has made use of special instruction sets present in common desktop computers to accelerate the search procedure (Rognes and Seeberg, 2000). This type of method holds a lot of promise, however the speed of searching still does not approach the heuristic methods that have been developed over the last decade.

Heuristic algorithms

Heuristic alignment algorithms were intentionally devised with database searching in mind, the goal being to drastically reduce the time needed to search an entire sequence

database for similarities to a given query sequence. There are two families of heuristic programs in general use, the FASTA (Pearson and Lipman, 1988; Pearson, 1990) and BLAST (Altschul et al., 1990) programs.

The first step in the FASTA program is to search for identical 'words' of a defined length (known as *k*-tuples) in both the query and target sequences. Generally for proteins a word length of two ($ktup = 2$) is sufficient for most searches, it combines good accuracy with high speed. A higher value increases the speed but at the risk of increasing inaccuracy. These *k*-tuples are then used to identify the ten most interesting diagonal regions in the alignment matrix. These regions are then re-scored using an amino acid substitution matrix, thus taking amino acid similarities into account as well as identities. Only regions with a score above a cutoff value are considered further. A gapped alignment score is estimated by joining together compatible regions using a joining penalty. FASTA also computes an optimal local alignment restricted to a band centred on the highest scoring region. Finally these scores are used to estimate the statistical significance of the matches.

The BLAST programs work in a similar way. However it is important to differentiate between the two BLAST programs that are commonly used for protein-protein sequence searches. BLASTP (Altschul et al., 1990) is the original program developed for searching protein sequence databases and is still in common use. BLASTPGP (Altschul et al., 1997) is a more advanced version and works in a significantly different manner. In the first step BLASTP uses words of length w. Unlike FASTA, BLAST allows the words to match similar rather than just identical amino acids. To be counted the words must

match at score greater than T when scored using the amino acid substitution matrix. By default BLASTP uses the parameter settings $w = 3$ and $T = 11$. In the second step BLASTP extends the initial words in both directions using the substitution matrix to form high-scoring segment pairs (HSPs). The extension is stopped when potential score of the extending segment drops below the the maximum score of the HSP within the segment. This version of BLAST does not consider gapped alignments at all but uses sum-statistics (Karlin and Altschul, 1990; Karlin and Altschul, 1993) to compute the significance of the matches from the highest scoring HSPs.

The BLASTPGP algorithm shares many similarities with the original BLASTP algorithm, however several key improvements have been made and to fully accommodate these the procedure has been changed somewhat. As before the program looks for all the words of length w scoring above T. However to reduce the number of words that are extended (in the original implementation this accounted for approximately 90% or more of the computation time) the refined algorithm looks for two non-overlapping words on the same diagonal of the scoring matrix, no further apart than A residues; for $w = 3$ and $T = 11$ it is recommended that $A = 40$. If two words are matched within the required distance then an ungapped extension of the second word is triggered. If the HSP generated has a normalised score above a cutoff then a gapped extension is triggered. This Smith-Waterman like extension requires 500 times the computation time of that of an ungapped one, however because extensions are triggered so much less frequently than in the BLASTP program one gapped extension will only be triggered for upto 4000 ungapped extensions in the original program. In addition because the number of ungapped

extensions are significantly reduced, the total time spent on the extension stage is cut by a factor of two. The significance of the gapped alignments is then evaluated (Altschul and Gish, 1996) before the results are reported.

Significance of matches

There one important problem that I have already alluded to that remains when undertaking a database search. How do you evaluate when a match is a true hit i.e. at what alignment score is a match no longer considered a true homolog. This is not a clean cut problem, it is not possible to define an arbitrary alignment score cut off value, longer sequences have a higher probability of producing a higher alignment score by chance alone. This problem has been dealt with by several methods but almost ubiquitously all search engines now evaluate the statistical confidence of their hits. Put most simply this is the probability that a hit of score score x would occur by chance.

If we assumed that the scores obtained by random sequences in database searches followed a normal distribution it would be a simple matter to calculate the probabilities, using the mean and standard deviation of all the scores it is a simple matter to assess the probability of achieving a score greater than x. However the scores do not follow a normal distribution and such a calculation would lead to gross errors in the estimates of confidence. The distribution of scores for an ungapped local alignment of random sequences has been shown to follow an extreme value distribution (Karlin and Altschul, 1990). For this to be true certain conditions have to be met, uppermost of these is that the expected score $\sum_{i,j=1}^{20} p_i p_j s[i,j]$ for a pair of randomly chosen residues is negative. Where p_i and p_j are the independent probabilities of selecting amino acids i and j.

This makes good sense, if the probability was positive then local matches would tend to extend to the full sequence length. Luckily scores based on likelihood ratios such as the PAM and BLOSUM matrices always satisfy this condition. As long as this condition is met and at least one of the scores has a positive value it is possible to calculate one of the parameters of the extreme value distribution. λ is the unique positive solution for x in Equation 1.3.

$$\sum_{i,j=1}^{20} p_i p_j e^{s[i,j]x} = 1 \tag{1.3}$$

The second parameter that is important to our calculations is K, this is a constant which is defined by a geometrically convergent series, which is dependent on the scoring scheme i.e. the values of p_i, p_j and $s[i,j]$. The formula to calculate K is explicitly defined but due to its complexity I have decided not to reproduce it here; it is well documented in the appendix of Karlin and Altschul (1990). Using these two parameters it is possible to calculate the probability of achieving a score S greater than or equal to x the observed score.

$$P(S \geq x) = 1 - \exp(-Kmne^{-\lambda x}) \tag{1.4}$$

The two parameters that have not been discussed yet, m and n, are the lengths of the query and target sequences respectively. For a database search the target sequence can be thought of as the database as a whole therefore n would be the length of the database in residues. It is on this general basis that all the statistical evaluations of sequence hits are based. Of course this formula only applies to ungapped alignments, which makes it ideal for use in simpler programs e.g. the original BLASTP. However the

newer implementation of both BLAST and FASTA produce gapped local alignments as rather than ungapped ones. An extension of this theory which is known as sum statistics (Altschul and Gish, 1996) allows the assessment of a set of top scoring local alignments rather than just the optimal segment via the sum of their scores. From this it has been shown empirically that the results from gapped local alignments seem to fit this function. However because the gap penalties alter the scoring scheme the values of λ and K for gapped and ungapped alignments are different. For gapped alignments these parameters cannot be estimated directly and instead have been estimated by fitting Equation 1.4 to scores from simulations. When gapped alignments are used a better result is obtained when the sequence lengths are corrected for edge effects. Because a subalignment does not exist at a single point it cannot start near the end of either sequence or it will run out of space before it can reach an optimal score. As a result to correct for this edge effect it is advisable to estimate the effective length of the sequences. Equation 1.5 shows how to calculate the effective length m' for sequence m, with the n' calculation following the same format. H is the relative entropy of the scoring system (Altschul, 1991; Altschul and Gish, 1996).

$$m' \approx m - \frac{\ln Kmn}{H} \tag{1.5}$$

Rather than P-values the new versions of BLAST utilise E-values (Altschul et al., 1997; Schaffer et al., 2001). These are the expected number of subalignments of random sequences with an optimal subalignment score greater than or equal to x.

$$E = Km'n'e^{-\lambda x} \tag{1.6}$$

The significance testing methods utilised by the FASTA suite of programs are also based on the extreme distribution theory. However the implementation of this is somewhat different (Pearson, 1998). Rather than using pre-computed values of λ and K, the distribution is itself calculated using all the similarity scores produced during the database search. The mean (μ) and standard deviation (σ) are related to Kmn and λ and are easy to calculate from these scores. This should mean that the correct distribution parameters are calculated each time, regardless of the scoring system or gap penalties used. However, the estimation is only accurate if all the sequences are unrelated e.g. if random sequences are used. However in a real search some sequences will be related, indeed these are the very sequences we are trying to locate, the homologues of our query sequence. When the dataset includes these related sequences this method of estimating the significance breaks down the effective values of K and λ that are implied by the distribution will be incorrect and will mean that the significance values reported will be incorrect, true hits would be ignored as false. To overcome this problem the authors of FASTA implemented a filtering system with the intent of removing any scores that might be from related sequences so that a correct estimate could be made.

$$z = \frac{(S - \mu}{\sigma} \tag{1.7}$$

Before the significance of a hit is calculated, its score is converted in to a z-value as shown in Equation 1.7. This z-value is then converted to a P-value using the extreme value distribution as shown in Equation 1.8. A more complete explanation of the procedure

can be found in Pearson (1998).

$$P(Z > z) = 1 - \exp(-e^{-\frac{-z\pi}{\sqrt{6}-\Gamma(1)}}) \tag{1.8}$$

The filtering method used splits all of the scores up into separate 'bins' according to the lengths of the sequences, the means and standard distribution of each of these bins is calculated and a simple linear regression line is calculated for the means of the bins. The z-values of all the scores are then calculated with the mean of each bin being taken from the regression line. Any scores with z-values <-3.0 or >5.0 are excluded before the procedure is repeated for a second time. Once complete are the remaining scores are used to estimate the extreme value distribution used to calculate the final P-values.

Advanced Search Schemes

As good as these heuristic programs have become and as accurate as the full Smith-Waterman search is, these methodologies still have limits to their success in finding homologues. The main limiting factor is that these are all pairwise approaches, they compare one query sequence to each sequence in the database. What they do not and can not do is take into account the information contained in any homologues that have already been identified to the query sequence i.e. the searches do not incorporate information on the protein family of the query sequence. The idea of doing such a thing is so that homologues that are much more remote can be isolated, which when compared to the query sequence alone show no definite homology.

The simplest approach to this problem is the Intermediate Sequence Search (ISS)

method (Park et al., 1997). This approach makes use of intermediate sequences to find remote homologues, for example if sequences one and two match each other with a high score and sequences two and three match with a high score, we can infer that one and three are homologous even though when compared directly they have a low score of similarity. This method and other similar ones (transitive sequence searching (Neuwald et al., 1997), systematic re-searching (Krause and Vingron, 1998) and multiple intermediate sequence searching (Salamov et al., 1999)) have proven to be very effective at recognising more remote homologues than the simple pairwise systems (Gerstein, 1998; Park et al., 1998).

Whilst ISS and related methods are simple and effective they are largely just a more complex way of carrying out a pairwise search. However methods have been developed that can use more than one sequence at once to search a database. These are the profile search methods, of these the most predominant is PSI-BLAST (Altschul et al., 1997; Schaffer et al., 2001). As it's name indicates PSI-BLAST is a member of the BLAST suite of programs available from the NCBI. It works in almost exactly the same way as BLASTPGP, in fact a single iteration of PSI-BLAST is just BLASTPGP. However the program is designed to iterate and sequences found in the first round are used to help find more distant homologues in the second round and so on. PSI-BLAST is not a true profile based database search engine. What it does is use the results of previous rounds or a user defined alignment to produce a position specific scoring matrix (PSSM) for the query sequence. The PSSM is a very powerful tool rather than relying on a substitution matrix like the BLOSUM or PAM series, the construction of the PSSM allows PSI-BLAST to

define which substitutions are most likely at each position in the query sequence on the basis of the hits already found. To form the PSSM PSI-BLAST uses a pseudo-multiple alignment of all the hits above a set threshold e.g. default is an E-value of 0.01. I refer to this as a pseudo-alignment because the hits are stacked on top of the original query sequence and no attempt is made at a full alignment, which would involve aligning all the hits to one another as well. Identical sequences are excluded from the alignment as are any sections that would involve the insertion of a gap in the query sequence, the aim of this is to make the PSSM the same length as the query sequence. One of the main reasons for not conducting a full multiple alignment is to keep the computation time to a minimum. In comparison to a database search a full multiple alignment is much more time intensive. The PSSM is then constructed from the pseudo-alignment using a modified version of the Henikoff and Henikoff weighting scheme (Henikoff and Henikoff, 1994; Altschul et al., 1997). PSI-BLAST continues iterating until the search has converged, finding no new hits over the set threshold, or until it reaches the maximum number of iterations set by the user. The power and success of this method has been demonstrated in several different benchmarks (Park et al., 1998; Müller et al., 1999).

A somewhat different approach is taken by the QUEST program (Taylor, 1998; Taylor and Brown, 1999). Unlike PSI-BLAST, QUEST uses an independent multiple alignment program MULTAL (Taylor, 1987; Taylor, 1988) to align the sequences between iterations, thereby hopefully improving the quality of the profile fed into the next stage. QUEST also incorporates two screening steps between iterations. The first removes sequences that are too divergent to align correctly, this is done with the intention of removing

any incorrect sequences so that the profile does not become 'polluted'. The second screening step is similar to one of the PSI-BLAST steps simply removing any sequences that are too similar thus preventing them becoming overrepresented and hijacking the profile. The search phase itself is not that dissimilar to the BLAST methods, the main difference comes from the automatic optimisation of several parameters on the fly. These parameters are the score cutoffs that will ultimately determine the speed, accuracy and success of the program. The idea of this automatic control of these parameters is to allow a relative novice to use the QUEST program and achieve very successful results without needing any specialist knowledge on judging alignment quality. The success of this approach has been evaluated and has been found to be quite effective (Taylor, 1998; Taylor and Brown, 1999).

A different approach is again used by hidden Markov model (HMM) based search systems. These approaches actually follow the profile based systems quite closely, but rather than a profile an HMM is used to model the sequence information. HMMs are a class of probabilistic models that are particularly applicable to linear sequences, such as biological sequence information. Much of their appeal derives from the strong mathematical and statistical theory that is their cornerstone, as opposed to the sometimes ad-hoc system used in generating profiles. There is not space here to explain the theory behind hidden Markov models, nor could I do it justice. There are however several good reviews of HMMs and their application in biological sequence analysis (Durbin et al., 1998; Eddy, 1998). Good examples of HMM based approaches are SAM-T98 (Karplus et al., 1998) and HMMER (`http://hmmer.wustl.edu`), which are proving to be effective search tools (Karplus et al., 1998; Park et al., 1998; Taylor and Brown, 1999).

1.6 Aims

The broad aim of this project are very simple - to design and devise more efficient systems for searching and accessing large sequence databases. The priority being to maximise both the sensitivity and selectivity, which in turn means identifying the maximum number of correct hits whilst picking up the bare minimum of incorrect sequences.

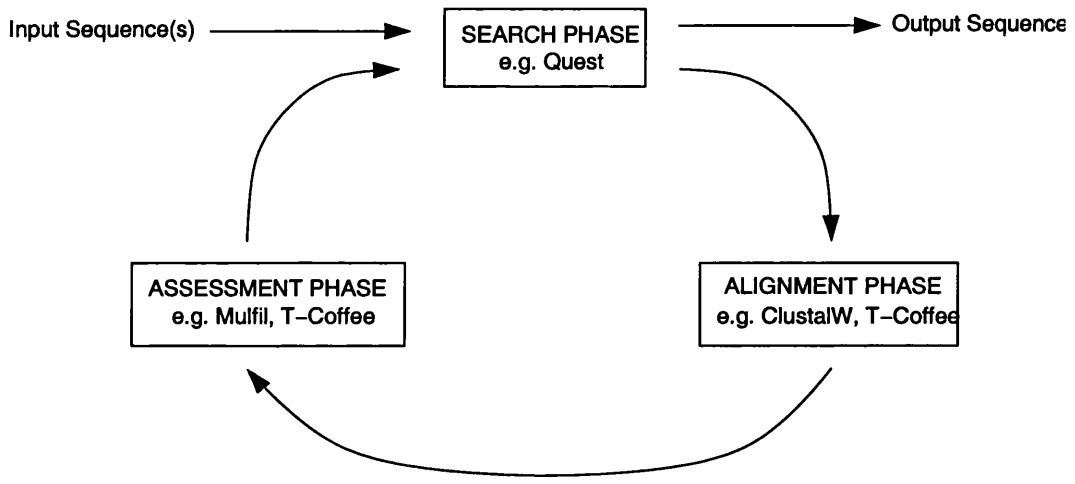


Figure 1.4: **The QUEST Scheme**

The basis for this work was the QUEST program (Taylor, 1998; Taylor and Brown, 1999). QUEST is actually a scheme made up of several stages and programs, one of these is Quest the database searching program. To distinguish between these two the search scheme will be refered to in capitals only i.e. QUEST whereas the search program will be largely lowercase i.e. Quest. This project aims to develop both the scheme and the program. With the QUEST scheme we intend to evaluate and optimise the various programs used to improve sensitivity whilst introducing advanced filtering steps between the various stages to maximise the selectivity. The basic form of the QUEST scheme

can be seen in Figure 1.4. It starts with a heuristic based searching step similar to that of PSI-BLAST but able to take a multiple alignment as a query. PSI-BLAST still only takes a single sequence as a query and uses the alignment just to construct a PSSM for the query. The second stage is the alignment phase. The hits from the first phase are aligned, unlike PSI-BLAST this is a true multiple alignment rather than a stack of pairwise similarities to a single query sequence. The third phase is the assessment phase where sequences that are too divergent or indeed too similar are filtered out. The resulting alignment is then fed back to the search phase as the query for the next iteration. In the original QUEST implementation the alignment program MULTAL was responsible for the alignment phase and most of the assessment phase. One of the intentions of this work was to more fully modularise this scheme so that these two phases can be separated allowing different programs to be mixed and matched in the various phases to maximise the overall success of the scheme.

At the same time the intention of the project is to improve the efficacy of the Quest program itself. Test runs of Quest proved it to work quite well when it is used iteratively with a multiple sequence input, however attempts to run it with a single query sequence gave extremely poor results. It is likely that in many cases a single sequence would be used as a query to start the iterative process, thereby it is hope that improving the efficiency of the program with a single sequence input should also improve the results from the multiple sequence iterations.

To sum up the aims of this project are to build on the foundations of previous work with ultimate aim being the same as that of the original project to develop a database

search system that maximises both sensitivity and selectivity. Whilst at the same time being simple to use allowing a bench biologists to produce an accurate set of results just as easily as a trained bioinformatician.

Chapter 2

Quest: A Profile Based Search Program

The original Quest program was written by Willie Taylor and Nigel Brown (Taylor, 1998; Taylor and Brown, 1999). As part of the QUEST scheme and under strict control its execution was quite effective. However preliminary work on the program showed it to have several key weaknesses. On the simplest level its interface and output was over complicated and confusing to new users. Much more seriously were problems inherent in its design. Quest had been written to use a multiple sequence alignment as a search query and in such circumstances this worked relatively well. But in many if not most real database searches we are naturally going to start with a single query sequence, which Quest could not read. It was possible to fool the program by presenting it with an alignment of two identical sequences as a query. However the method by which it constructed its scoring scheme meant that the result achieved in this manner extremely poor.

From these preliminary observations some of the key aims of this project were put together. Primarily the aim was to improve the stand alone efficiency of the Quest program, i.e. how well it works outside of the QUEST scheme. A core part of this aim was to make Quest work equally well with either a single sequence or a multiple alignment input. Both of these abilities would be important in an iterative scheme. The secondary aim was to improve the usability of Quest, whilst not as important to the core aims of the project this is still relevant if people are going to make use of the program. This goal incorporates as simple ideas as removing redundant parameters from the command line and using standard rather than proprietary file formats to presenting the results in a coherent easy to read manner. As well as making Quest easier to use this goal would also allow the QUEST scheme to be much more modular. The use of standard inputs and outputs would allow Quest to be paired with a much larger variety of sequence analysis programs.

Rather than starting again from scratch we decided to take the original program as our starting point and effectively go through all of the code line by line. This allowed us to understand totally how the program was working and how each function interacted. From this level it was then possible to simplify and improve the operation of the program. This included stripping out all the excess code that was no longer in use and updating the syntax allowing it to be compiled much more easily on most available platforms. This code was then taken as our starting point as we set about re-writing the key functions to improve the accuracy of the program.

2.1 Quest: Method of Operation

Because the Quest program has been so completely rewritten, I will go through the whole method of operation taking care to highlight the key differences between the two versions. The original methods have been well documented in the previous literature (Taylor, 1998; Taylor and Brown, 1999). The overall operation of Quest can be split into four distinct stages.

2.1.1 Stage I: Input & Preprocessing

The first stage of the program is the input of the necessary data and the preprocessing that has to be done before the search stages themselves can begin. This stage has three substages; data input, profile construction and tripeptide table construction.

Data Input

Data input does not just involve the reading of the input file by Quest, but also includes several other important pieces of data. The first thing Quest does is to read the information passed to it on the command line. There are a number of parameters and options that can be passed to the program is this way; these are fully detailed in Section 2.2 and I will not cover them fully again here. Of these parameters there are three that we are particularly interested in at this stage, the input, database and matrix parameters. These parameters respectively define which input, database and matrix file Quest should use in the current run. If a matrix file is specified that file is read by Quest and loaded into memory to use as the amino acid substitution matrix. If no file is specified a de-

fault BLOSUM62 matrix that is hard coded in the program is used. The database file information is stored until the search stages begin.

Once the matrix file has been evaluated, the input file is dealt with next. In the original implementation Quest expected a multiple alignment as an input file. In addition to this the multiple alignment had to be in MULTAL (Taylor, 1988) format. This is a quite unique format with the multiple alignment being in a vertical as opposed to horizontal orientation. Whilst easy to read by eye and computer, its weakness was its unusual nature. No other alignment programs produced this type of file format and none of the commonly available file conversion programs recognised it. It was thus decided to scrap this format and use a much more universally acceptable one. After evaluating several options we turned to the Fasta format (Figure 2.1). Although not ideal for representing multiple alignments, which are difficult for the human eye to make much sense of in this layout. The Fasta format had two distinct advantages; it could represent single sequences and multiple alignments in the same format, allowing Quest to take a single sequence input as easily as a multiple alignment, with no further rewriting needed. Secondly the Fasta sequence format is very widespread, the majority of sequence analysis programs are capable of understanding and using it. Because almost all database sequences are available in this format, they can be directly used in Quest with no previous alteration needed.

Profile Construction

After the initial sequence or alignment has been read into the program the preprocessing stages start. The first step is the production of the profile, which basically amounts to

A.

```
>SEQ1
ACDEFGHIKLMNPQRSTVWYACDEFGHIK
LMNPQRSTVWY
>SEQ2
ACD--GHIKLMNPQRS-------------
-MNPQRSTVWY
>SEQ3
TCDQYGYIKLMNPEGADI-----------
-MNPQRSTVWY
>SEQ4
TCNQYAYVELINPEGADL-----------
IINPQRSTVWY
```

B.

```
>SEQ1
>SEQ2
>SEQ3
>SEQ4
AATT
CCCC
DDDN
E-QQ
F-YY
GGGA
HHYY
IIIV
```

Figure 2.1: **Input File Formats:** A. Fasta: The present Quest input format. This format is defined by having a title line for each sequence starting with a '>' character, the actual sequence is then contained on the following lines. B. MULTAL: The previous Quest input format. The start of the same alignment is shown in MULTAL format. The vertical orientation is clearly visible as is its increased readability, divergent positions being more clearly visible in this format.

the construction of a PSSM from the input sequence(s). Before the PSSM is constructed the input alignment is pruned. The section of the alignment that is used to build the PSSM starts at the first position at which less than 25% of the sequences have gaps and ends at the last position where less than 25% of the sequences have gaps. The intention of this restriction is to prevent the frequently divergent tails of the multiple alignment from distorting the search specificity. Of course when the query is a single sequence, the whole sequence satisfies this condition and so no part of the sequence is excluded.

The actual method of the profile construction itself has changed radically from the previous version of Quest. It was in fact at this stage that a large amount of the errors and problems were previously introduced, especially with single sequence inputs. In the original version of Quest, this stage did not involve the production of a traditional PSSM, but instead the calculation of something known as the amino acid similarity transfer

function. Whilst the function did give a weight for each amino acid at each position much like a PSSM, the method of its construction was quite different and complex. First of all for a given position in the profile the 20 amino acids were given a weight (${}^{a}w$) of 1 if they are present and 0 if they are not. No attention was paid to the frequencies of the amino acid types, merely their presence or absence. A modified weight (${}^{b}w$) was then calculated according to the Equation 2.1. Where $s[i,j]$ is the amino acid exchange weight for residues i and j in the exchange matrix (PAM120 in the original program) and α is a constant between 0 and 10 (Taylor, 1998).

$$ {}^{b}w_i = {}^{a}w_i + \sum_{j=1}^{20} \frac{\alpha({}^{a}w_j + 1)s[i,j]}{100} \tag{2.1} $$

These weights were then normalised to have unit variance and a mean of zero. The normalised weights were shifted so that the mean was equal to the mean of ${}^{a}w$, the original weights. The resulting values were then mapped into the range -1 to +1 by a switch function formed by the tails of a Gaussian curve. A much fuller account of these transformations can be found in the original literature (Taylor, 1998). Finally, in an attempt to prevent the dissipation of the original signal, all the amino acids that were present at that position in the original profile were set to 1, the maximum value.

On closer examination we found this excessively complex weighting scheme to be responsible for some of the worst errors that Quest was producing. When a single sequence was used as an input, no matter which amino acid is present at a given position it would have a score of one for a self match. No discrimination was made against different amino acid types. In the BLOSUM62 matrix, tryptophan has a score of 11

for an identity match, whereas alanine has only 4. Scoring all of them the same means that a huge amount of information is lost. Essentially it converts a similarity matrix into a virtual identity matrix. Furthermore, if a multiple alignment is used as an input, highly conserved sites will not be made obvious unless they are conserved in every single sequence. For example an alignment of ten sequences has one highly conserved site; 9 of the sequences have the amino acid aspartic acid and 1 has lysine. Because only presence or absence is scored this will give the same result as a 50:50 split. As a result of these problems we moved to completely rethink the method of PSSM construction, with the intention of preserving as much information from the sequence or alignment as possible.

In the new implementation the PSSM is calculated in a more classical way (Gribskov et al., 1987), directly from the alignment. Where the input is a single sequence we can think of this as an alignment of only one sequence. For each position in the alignment, Quest scans through the amino acids present. For each occurrence of a residue its substitution values from the exchange matrix are multiplied by the weight of the sequence and added to those for the other sequences. When all of the sequences at that position have been added, the exchange values are divided by the sum of the sequence weights. This gives the final exchange scores for the present position of the PSSM. The idea of dividing by the sum of the sequence weights is to prevent the values of the scoring scheme increasing each time new sequences are added. When the input is a single sequence the weight of the sequence is obviously one and therefore the scoring scheme is essentially just the exchange matrix that was initially selected. For clarity a graphical representation of this system is detailed in Figure 2.2.

Sequence	1	2	3	4
Residue	A	A	T	T
Seq. Weight	1.0	0.5	1.0	1.5

A	4.0	2.0	0.0	0.0		6.0		1.5
C	0.0	0.0	-1.0	-1.5		-2.5		-0.625
D	-2.0	-1.0	-1.0	-1.5		-5.5		-1.375
E	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
F	-2.0	-1.0	-2.0	-3.0		-8.0		-2.0
G	0.0	0.0	-2.0	-3.0		-5.0		-1.25
H	-2.0	-1.0	-2.0	-3.0		-8.0		-2.0
I	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
K	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
L	-1.0	-0.5	-1.0	-1.5	⟶	-4.0	⟶	-1.0
M	-1.0	-0.5	-1.0	-1.5	sum up scores	-4.0	divide by sum	-1.0
N	-2.0	-1.0	0.0	0.0	of the 4 seqs.	-3.0	of seq. weights	-0.75
P	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
Q	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
R	-1.0	-0.5	-1.0	-1.5		-4.0		-1.0
S	1.0	0.5	1.0	1.5		-4.0		-1.0
T	0.0	0.0	5.0	7.5		12.5		3.125
V	0.0	0.0	0.0	0.0		0.0		0.0
W	-3.0	-1.5	-2.0	-3.0		-9.5		-2.375
Y	-2.0	-1.0	-2.0	3.0		-8.0		-2.0

Figure 2.2: **Construction of the PSSM.** The first 4 columns denote the amino exchange value for the four amino acids multiplied by their respective sequence weights. The effect of this can be seen by comparing columns 1 and 3 (which both have a weight of one) to 2 and 4 respectively (which do not). Each of the rows in these columns are then summed to produce the fifth column. Which is then divided by the sum of the sequence weights (4 in this example) to give the final exchange values for this position in column six.

Tripeptide Lookup Table

Once the PSSM has been constructed the next stage is to compile the tripeptide lookup table. Like many other database searching programs Quest uses a short initial segment match as a starting point, these are known as *k*-tuples in FASTA and words in BLAST. In Quest we do not allow their size to vary and so we refer to them simply as tripeptides. In

the original implementation Quest scanned through the profile and recorded the positions of approximately 25% of the top scoring tripeptides. It is important to note that these tripeptides did not necessarily occur in any one of the sequences in the input alignment, they can be formed from a combination of the sequences. The idea of the 25%/75% cutoff value was to save time by preventing Quest evaluating matches to the lower scoring tripeptides, which were less likely to be meaningful. Unusually the system used to score the tripeptides did not depend on the amino acid weights for the three positions, but instead was calculated as the sum of their positional conservation scores (see below Equation 2.2). This was a different method to that used to score segment extensions (Section 2.1.2). The consequence of which was that identical segments seeded from different but adjacent tripeptides could have quite different scores.

The new implementation uses a coherent scoring system throughout. The tripeptide lookup table is still constructed and is still categorised by score and position. However the tripeptide scores are now calculated from the PSSM in an identical way to the segment extension method. In addition to giving very reliable indications of how well conserved and informative the tripeptides were, this system also creates consistency, consequently identical segment matches will always have identical scores. Because Quest was being designed with the QUEST scheme still very much in mind, we made the decision to drop the tripeptide cutoff and accept any tripeptide with a score greater than zero. There was a specific reason for this decision, in the QUEST scheme it is the alignment phase rather than the search phase that is generally the rate limiting step. As a result time saving was no longer considered a good enough argument for the loss in sensitivity caused

by this cutoff. It is important to note that a restricted version of the cutoff is still in place. By defining the cutoff as zero we prevent negative scoring tripeptides from being considered. Examples of these would be tripeptides that have been included as a result of a low strictness setting (Section 2.2.1), i.e. a tripeptide made up of three amino acids similar to those found at those particular positions but not actually present themselves. So in short even though the cutoff is set to zero not all possible tripeptides will be found in the lookup table, unless of course the makeup of the input sequences is extremely diverse.

2.1.2 Stage II: Tripeptide Matches & Segment Extension

The second stage of the Quest program is responsible for the location and initial scoring of the local subalignments between the profile and the database sequences. Within the Quest program we refer to these subalignments as segment hits. There are two substages to the production of these segments; tripeptide matches and segment extension. I will deal with these together because the first does immediately trigger the second.

Before I go any further I must return to the subject of the database. In Section 2.1.1 it was indicated that the name of the database file is saved for later use. It is at this point that we put it to use. Quest works through the database by reading each sequence one at a time and evaluating the scores produced from stages II and III before moving on to the next sequence. The method of reading the database is not quite as simple as this sounds. Quest actually uses two files; the database file and a file known as the scanner file. The scanner file and the reasoning behind it is more fully explained in

Section 2.3.2. In short, it is a list of the databases attributes including the number of sequences, the residue length of the database and the file positions of all the sequences within the database.

Once the database sequence has been read in to the Quest program each of the overlapping tripeptides that make it up are read and analysed in turn. If the tripeptide is present in the lookup table the segment extension function is triggered. Segment extension is summarised in Figure 2.3 and basically involves Quest attempting to extend the tripeptide match into a larger subalignment known as a segment. The C- and N-terminal extensions of the tripeptide are dealt with identically but independently before being combined to give the final segment boundaries and score. Starting with the tripeptide the segment is extended by adding the exchange value from the PSSM of the next amino acid in the database sequence to the tripeptide score. If this score is higher than the tripeptide match score then it is set as the segment score and the segment coordinates are similarly extended. This is then repeated for the next residue and so on. If at any point the score is higher than the previous maximum obtained, the segment score and boundaries are set to the new values. However, if at any point the current score of the extension drops below zero, the extension is stopped and the segment score and boundaries are reported from the maximum scoring point. The process is then repeated in the other direction before the two are combined to give the final segment score and boundaries. This process may seem to have little in common with the Smith-Waterman local alignment procedure reported earlier (Section 1.5.1), but it is in fact a restricted version of it. Segments are essentially ungapped sub-alignments indicating local similarity between the probe profile and the database sequence.

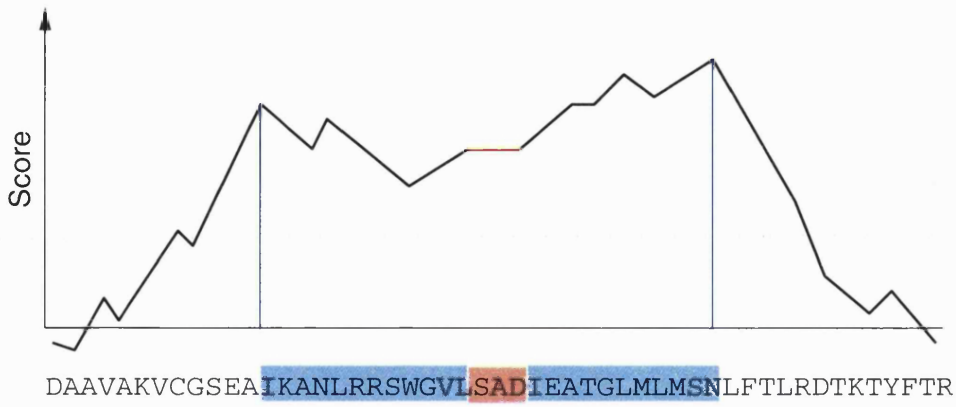


Figure 2.3: **Segment Extension.** The initial tripeptide match (red) is extended one amino acid at a time until the extension score (black line) reaches zero. The boundaries of the highest scoring segment are defined by the maximum scores (vertical blue lines) giving the segment shown (blue).

The original method of segment extension worked in a very similar manner. The one difference emanated from how the position specific scores, or weights were calculated in the original version. Because the amino acid weights took no account how many amino acids were present at a particular position, rather just their presence or absence. When the score at each position was added to the running total, the weight of that position was multiplied by P the positional conservation (Equation 2.2).

$$P = \frac{21}{Q} - 1 \tag{2.2}$$

Where Q was the number of positively weighted amino acids at that position. P only takes integer values, thus a completely conserved position would give $P = 20$, whilst a position with 11 or more positive residues would have $P = 0$. Because the new scoring system already takes into account the numbers and occurrences of amino acids this factor was redundant and hence removed.

2.1.3 Stage III: Segment Assembly & Sequence Scoring

Stage III builds on the second stage by attempting to turn the various segments and their scores into one continuous sequence hit. As before there are two substages to this process, this time quite distinct in their operation; segment assembly and P-value calculation. However, before either of these processes are initiated, the segments are sorted according to score with those scoring less than the cutoff being discarded. The cutoff score calculation is virtually unchanged from its original implementation. Once stage I is complete, Quest carries out a mini database search by reversing the input sequences and using them as a database. The purpose of his reversal is to generate sequences that have the same length and amino acid makeup as the original input sequences but with no homology to them. In effect this is a fast and effective method of producing a pseudo-random sequence. The segment cutoff is set so that 10% of the segments generated from the reverse sequences would be accepted and 90% rejected. In it's original incarnation this level was 95%, but experimental evidence indicated that a small drop produced considerably better results with only a very small trade off in computation time.

Segment Assembly

The new segment assembly function in Quest was designed to be as simple as possible. To save on excessive computing time it was designed as a 'greedy' algorithm. We assume the highest scoring segment hit to the current database sequence to be 'correct'. The definition of 'correct' obviously does not mean that the sequence is a match to the query,

but that any other segments that contradict this one will be ignored. From this highest scoring segment we then step down to the second highest scoring segment and evaluate whether it can be fitted to the first. For this to be true the segment must be linear to the highest scoring segment. This concept is easiest explained by an example. If the highest scoring segment is a match between the centre of the profile and the centre of the database sequence (dark blue line in Figure 2.4), then the second is rejected if it is a match between the start of the profile and the end of the database sequence. It is rejected because the segments cannot be laid out in a linear order. To be accepted the segment as well as following a linear sequence the segment cannot overlap with the highest scoring segment, it must be wholly in the top left or the bottom right corners of the alignment matrix (outside the shaded area in Figure 2.4). If these conditions are satisfied the two segments will be joined if their combined scores minus any gap penalty (Section 2.2.1) is greater than the present highest scoring segment. If the two segments are combined a quick check is done to see if any other lower scoring segments that fit the same conditions fall between the ends of the two joining segments. If these exist their scores are also added to the combined segment to form a new larger highest scoring segment. This process is then repeated with the next highest scoring segment and so on until all the segments have been assessed and either joined to the highest scoring segment or discarded from the evaluation. The final remaining extended segment is renamed as the sequence hit for the present database sequence.

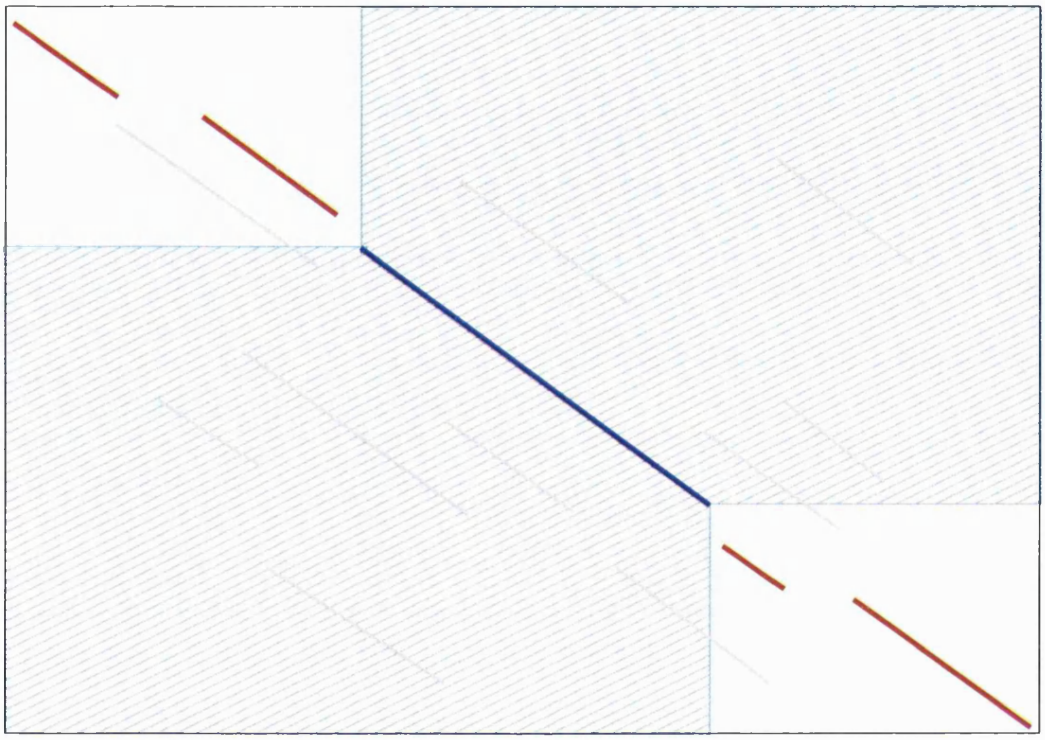


Figure 2.4: **Segment Assembly.** The large rectangle represents the alignment matrix. The blue line is the highest scoring segment. The light blue shaded area indicates the excluded region. Any segments within this area will either overlap or be non-continuous with the highest scoring segment. The red lines represent segments, that depending upon the gap penalty can be joined to the best segment. Whilst the grey lines are segments that fall on or in the excluded region and hence cannot.

P-Value Calculation

At this stage the score of the sequence hit is the sum of its constituent segment scores. The segments themselves are scored according to the pairwise similarity of the profile and the database sequence. On its own this scoring system has no inherent faults. However when we wish to compare the results of more than one database sequence, problems can occur. The score is length dependent, a longer hit is much more likely to have a high score than a shorter hit. By chance alone a long sequence be it in the input or database is more likely to generate a hit of score x than a shorter one.

In the original implementation any hits above a certain cutoff score were reported as true, with no attempt made to sort or return them on the basis of score. This was not an acceptable result, many researchers will wish to know which sequences are the best hits and what level of confidence to have in the various results. Consequently we decided to make Quest calculate and report P-values in a similar format to that of BLAST and FASTA. Rather than re-iterate the large amount of work that has been done in this field, we examined methods used by other authors and decided to implement a function that was kindly donated by S. Altschul (Karlin and Altschul, 1993; Altschul and Gish, 1996), which provided the exact functionality we required. The sum-statistics functions estimate the P-value of sequence hits made up of several smaller segments, making them ideal for our situation. The mathematics for these ideas has already been summarised in the previous chapter (Section 1.5.2) and I will not repeat them here. When Quest is given a single sequence input the scoring scheme is essentially just the BLOSUM62 matrix for which the values of λ and K are already known (Altschul and Gish, 1996). When multiple sequences are used as input, the scoring scheme although based on the BLOSUM62 matrix is unique to each alignment. As a result of this the values of λ and K may no longer be correct. However re-estimating the values of these parameters for each run of Quest or each iteration in the QUEST scheme would be excessive and very time consuming. As a result the values of the parameters are kept at the same values as for single sequences. Whilst not fully mathematically rigorous, this does save a large amount of time and does not appear to cause excessive errors in the selectivity of the program.

2.1.4 Stage IV: Hit Selection & Output

The final stage of the Quest program has one simple function, the output of results. The top five hundred hits of the database search are stored throughout the search procedure. This maximum is enforced purely to conserve run-time memory and it can easily be altered by changing a single definition during the compilation of the program. Whilst this maximum may seem restrictive, in practice it is not a problem. Any correct hits that do not get into the top set during the first iteration of the QUEST scheme should easily do so in the second.

```
p=============================================================q
|                          QUEST                              |
|                       Version 2.0                           |
|-------------------------------------------------------------|
|        J.Kleinjung, J.Hatwell, N.Brown, W.Taylor            |
|-------------------------------------------------------------|
|                       August 2001                           |
b=============================================================d

infile = seq1
dbfile = SCOP
strict = 100
cutoff = 1.000000e-02
matrix = Default:Blosum62

scanning SCOP

Signifcant Hits found to the Probe Sequence(s)
Name of Sequence                    Terminii   Score  P-value   nseg
D1SCTA_ 1.1.1.1.1                     1-150      755  1.07e-98   1
D1SCTB_ 1.1.1.1.1                    18-148      478  2.95e-60   1
D3SDHA_ 1.1.1.1.1                    12-145      358  6.01e-42   2
D1CQXA1 1.1.1.1.48                   56-126       94  7.69e-07   1
D1HBRA_ 1.1.1.1.21                    6-77        90  1.75e-05   2
D1OUTA_ 1.1.1.1.23                    6-62        84  2.08e-05   1
D1HLM__ 1.1.1.1.46                   65-149       81  2.18e-04   2
D1HLB__ 1.1.1.1.46                   89-150       74  4.87e-04   1
D1CG5A_ 1.1.1.1.25                   15-62        72  1.02e-03   1
D1BABA_ 1.1.1.1.16                    4-44        71  1.48e-03   1
D1A4FA_ 1.1.1.1.22                    2-42        68  3.85e-03   1

 scanned = 4133 matched = 11
```

Figure 2.5: **An example of a Quest output.** The top of the output is the Quest program information, followed by the parameter settings and results for this run.The five result columns respectively are: The name of the database sequence found; the termini of the sequence hit on the database sequence; score of the sequence hit; P-Value of the sequence hit; number of segments combined to produce the sequence hit.

When Quest has finished scanning all of the database sequences it prints out the results with a P-value lower than the cutoff (Section 2.2.1). By default Quest actually produces two outputs. The primary output is to the screen, an example of this can be seen in Figure 2.5. All of the hits that were accepted as true are printed out, the information includes the sequence name, size, score, P-value and the number of segments that make up the hit. Unlike the previous implementation this output is in the form of a sorted list with the sequence hit with the lowest P-value being printed first. It is important to note that P-values and scores are not the same thing. Most of the time the sequence with the lowest P-value will be the highest scoring sequence, but this will not always be true.

The second output is to a file, by default named 'hits.seq'. The information saved from each hit is used by Quest to grab the sequence information for each of these hits back from the database file. The sequences are then printed out in Fasta format to the 'hits.seq' file. This file can then be aligned with the original input sequences and fed back into Quest for the next iteration. To prevent the need to align non-homologous regions only the areas of the sequences that make up the sequence hit are written to the 'hits.seq'. Thus the sequences in this file will almost always be shorter than the database sequences themselves.

2.2 Quest: Parameters, Modifiers and Run-Modes

Despite our aim to make Quest easier to use it has accumulated no less than 16 different command line options (Table 2.1). It is not however necessary to use all of these options.

In fact only two are compulsory, the input file name and the database file name, all of the other options have default states. Many of the options would not be used at all when Quest is used on it's own; they have been added specifically for use as part of the QUEST scheme. The options can actually be separated in to the following three groups; parameters, modifiers and run-modes.

Option	Short Description
-c <cutoff>	Specify P-Value cutoff. Default = 0.01
-d <database>	Specify the database you wish to search.
-D <scanner file>	Specify the scanner file
-e	Exclude mode.
-f	Return full length hits.
-g <open>:<extend>	Specify gap penalty. Default = 10:1
-h	Print out the help information
-i <input file>	Specify input file name.
-m <matrix>	Specify the substitution matrix. Default = Blosum62
-o <output file>	Specify output filename. Default = hits.seq
-P	Generate a pseudo alignment.
-r	Recursive mode.
-s <strictness>	Specify the strictness parameter. Default = 100
-T	T-Coffee compatability mode.
-v	Verbose output.
-w	Very Verbose output.

Table 2.1: **Command Line Options for Quest.** A short explanation of the sixteen command line options available to Quest. The '-h' option simply causes quest to print out the information contained in this table.

2.2.1 Quest Parameters

There are three true parameters for the Quest program, they all work by changing key settings within the program . Alteration of their values will lead to a change in the reported results.

Score Cutoff

The score cutoff is defined by the '-c' option on the command line. By default if no value is given the cutoff is set to 0.01. The score cutoff is the simplest of the three parameters in both its operation and effects. The score cutoff sets the level at which sequence hits are accepted or rejected. Any sequence hit that has a P-value less than the score cutoff is accepted as a sequence hit and any value that is greater is rejected. Therefore, if no sequence hits are found during a Quest run, the score cutoff can be raised until one is reported. However just because it is possible to do this it does not mean that that sequence hit will be homologous to the query sequence. Probably more useful is the ability to set the cutoff at a very low level, allowing only very significant hits to be reported. This reduces the chance of returning an incorrect sequence as a hit.

Strictness Parameter

When undefined the strictness parameter ('-s') has a default setting of 100. This is the maximum value the parameter can take and effectively switches off the match softening function. The effects of changing this parameter are much less predictable and more complex in their action than those of the score cutoff.

The strictness parameter takes a value in the range of 0 to 100, with 100 denoting a completely strict match set. Any value lower than this leads to a softening of the match set. As the strictness level drops the procedure for selecting amino acids that make up the tripeptides in the lookup table gets softer. At 100% all of the tripeptides are made up of amino acids that are present at those positions in the input alignment. As the level

drops other highly similar amino acids will be included. Similarity is of course defined by the score of the amino acids in the PSSM. As the strictness drops lower still less similar ones will be included. Until at 0% all amino acids will be included at every position. However, many of these tripeptides will have scores lower than zero and hence will be excluded by the tripeptide cutoff. The strictness parameter works by calculating the highest and lowest amino acid exchange score at each position and turning the distance between them into a percentage. Any amino acids that score equal or greater than the strictness parameter will be allowed to form tripeptides in the lookup table, if those tripeptides have a score greater than zero.

Gap Penalty

The gap penalty used in Quest is similar to the affine gap penalties used in alignment algorithms. As such it has two parts; a gap opening penalty G_o and a gap extension penalty G_e. By default Quest operates with $G_o = 10$ and $G_e = 1$. As already described the gap penalty does not act on the alignment algorithm itself but rather on the segment assembly function. The function of the penalty is to prevent segments which would require too large a gap in either the database or profile sequences from being joined. If S_a and S_b are the scores of the highest scoring segment and the present segment respectively and n is the length of gap required to join the two segments. Then for a

single sequence query, the two segments will be joined when Equation 2.3 is true.

$$S_a < \begin{cases} S_a + S_b & \text{if } n = 0 \\ S_a + S_b - (G_o + G_e n) & \text{if } n > 0 \end{cases} \tag{2.3}$$

However, when a multiple alignment is used as an input the assessment is more complicated. If any of the sequences in the alignment have a gap between the positions of the two segments, then the gap penalty is altered. If q_n is the maximum gap length in the alignment between the two segments, then the segments will be joined if Equation 2.4 is true.

$$S_a < \begin{cases} S_a + S_b & \text{if } n < q_n \text{ or } n = 0 \\ S_a + S_b - (G_o + G_e n) & \text{if } n > 0 \text{ and } q_n = 0 \\ S_a + S_b - G_e(n - q_n) & \text{if } n > q_n \end{cases} \tag{2.4}$$

In essence, when $n < q_n$ the gap is not penalised because a gap of this size or greater is already present in the alignment. When $n > q_n$ the increased size of the gap is penalised, but because gaps have been recorded in this position previously no gap opening penalty is charged. This procedure allows the gap penalty to be somewhat tailored to the input alignment without actually calculating position dependent gap penalties for its entire length. It also offers the opportunity to control the type of gaps we allow. Charging an extremely high opening penalty but low extension penalty would prevent new gaps from being opened but would happily allow gaps to be introduced in regions they are already present. A high extension penalty would discourage new gaps larger than one residue

from forming, whilst allowing gaps up to the previously observed size in regions where they already exist.

2.2.2 Quest Modifiers

There are a variety of options that do not actually act on the searching or assessment of sequences but can none the less be essential for the operation of the program. Some of these options are essential and others may only be used very rarely, however they all fit into roughly two groups. Those that modify the inputs of the program and those that modify the outputs.

Input Modifiers

The input modifying options allow the user to specify the filenames of the input ('-i'), exchange matrix ('-m'), database ('-d') and the scanner ('-D') files. Of these only the input and database file names are compulsory. Quest has a built in copy of the BLOSUM62 matrix which it will utilise if no matrix is specified or if the '-m' option is used with the argument 'blosum62'. Whilst the scanner file is compulsory for Quest to operate, if it has the standard filename and path that Quest expects there is no need to specify it explicitly. Quest expects the scanner file to be in the same directory as the database file and to have the exactly the same name as the database but with the file extension '.scan'. So for example a database named 'genbank' would need a scanner file named 'genbank.scan'. If the scanner file does not fit the default then the '-D' option must be used to explicitly define its name and path.

Output Modifiers

Unlike the previous set most of the output modifiers have quite different effects. The output file option '-o' like the input modifiers allows the output file name to be changed from the default 'hits.seq'. The verbose output option '-v' prints out a little extra information to the screen, most significantly it shows the effect of the strictness parameter on extending the match set. The very verbose option '-w' sends extreme amounts of output to the screen and should not be used apart from debugging purposes. The other two output modifiers can be more useful. The pseudo-alignment option '-P' causes the output of a file named 'hits.aln' containing a pseudo-alignment of the output and query sequences. This is somewhat akin to the method that PSI-BLAST uses and was introduced to allow fast independent iteration. However it's efficiency has not yet been fully assessed. The full sequences option '-f' is probably the most useful output modifier. When specified it causes Quest to output the full length hit sequences to the output file, rather than just the hit regions. This saves the user from having to extract them manually from the database and can allow the more divergent non-hit sections of the sequences to be used to locate more distant hits in subsequent iterations.

2.2.3 Quest Run-Modes

The three run-mode options could easily be grouped together with the input and output modifiers specified above. The reason I have separated them is twofold. Firstly they do actually alter the internal operation of the program rather than just the inputs or outputs. Secondly, their functions were specifically designed with the QUEST scheme in mind.

Recursive Mode

The recursive mode is undoubtedly the most important of these three modes. It was written into Quest to allow the program to be more easily used in an iterative context. The obvious effect during the first iteration is that the query sequences as well as the hits have been printed to the output file. This is so that they can all be aligned before being fed back to the next iteration of the program. The second difference is in the format of the sequence names in the output file. Rather than the standard Fasta format sequence name, which would have the format:

```
>Sequence name
```

Quest tags on a small amount of additional information so that all of the sequence names follow one of two possible formats:

```
>?S|Sequence name
```

```
>?H|13527|Sequence name
```

The '?' simply indicates that these sequences have already been through at least one Quest iteration. The first of these formats relates only to the initial query sequences of the first iteration. These are known as the seed sequences and hence have been marked with an 'S'. There is no other extra information needed for these sequences so after the vertical bar spacer the original sequence name is added. The second format relates to sequence hits from this or previous iterations of Quest, with the 'H' obviously representing this fact. The number inside the vertical bars is unique for each hit, it represents the file position of that sequence inside the database. Once these sequences

have been aligned and fed back into Quest, the recursive mode becomes responsible for reading this extra information back into the program. When the program comes across a previous hit it records the database file position information. This knowledge is then used during the search stages to prevent the program hitting the same sequence again. The final results of the Quest iterations are therefore the sum of all the iteration results and unlike PSI-BLAST not just the final iteration result. This means that without any outside intervention once a sequence is assigned as a hit it is always a hit. Because the QUEST scheme has a separate sequence assessment stage this is not a serious problem. This concept can also be advantageous. Previous work (Park et al., 1998) on PSI-BLAST has noted that strong hits from early iterations are often lost in the final rounds and have to be artificially added back. At the end of the iteration all of the hits from the present and previous rounds along with the initial seed sequences are printed in the output file ready for the next round. It is important to note that the recursive mode does not make Quest iterate on its own accord, it merely adds some features that may be useful when running the program iteratively.

Exclude Mode

The idea of the exclude mode grew very simply from the recursive mode and indeed it is intended to operate along side it. During iterative runs it may be decided that one or more of the sequence hits are incorrect. If these sequences are removed from the alignment then they may frequently be hit again in successive rounds. Rather than filtering them out from the results each time, the exclude mode works to exclude them from being hit again. When the exclude mode is active, any sequence names in the

'exclude.seq' file will be excluded from consideration in the database. This works in exactly the same way that the recursive mode prevents previous hits from being hit again. The sequence names in the exclude file must be in the same format as the hit names returned by the recursive mode above.

T-Coffee Compatibility Mode

The T-Coffee compatibility mode was added to allow Quest to produce a library file that is compatible with the T-Coffee (Notredame et al., 2000) alignment program. T-Coffee is capable of bringing together disparate sources of information to align a set of sequences. By default it uses global and local pairwise alignment information to align a set of sequences. The library produced by Quest contains information indicating how all the segments in the accepted sequence hits align to the query sequences. The scores of these hits are also included as an indication of how much we trust these sub-alignments to be true. The purpose of this library is not to improve the quality of the multiple alignment because the Quest alignment system is not expected to be fully accurate. Instead T-Coffee also has the ability to judge the quality of an alignment on the basis of the information it has received. Thus, if for a certain number of sequence hits the Quest library disagrees with the alignment suggested by the local and global alignments, T-Coffee will indicate that these sequences are likely to be incorrect and should be removed from the alignment. This would allow us to use T-Coffee for the assessment phase as well as the alignment phase.

In addition to the library file a sequence weight file is also produced by the T-Coffee mode. This is an extremely simple file consisting of all the sequence names in the output

file and the weights that should be given to them during alignment. At present this is set up so that the original seed sequences are weighted as five times more important than the hit sequences. This has the effect of partially forcing the other sequences to be aligned to the original query. This was implemented after we observed the seed sequences being badly broken up when aligned to larger numbers of hits, thus severely reducing the signal we were originally searching for.

2.3 Quest for Parallel Architectures and Workstation Clusters

Whilst one of the original goals of the Quest project was to develop a rapid as well as sensitive database searching program, Quest never was one of the fastest heuristic based search programs and the changes that have been made in the latest version have done nothing to improve this. On a 650 MHz Pentium III workstation with 256Mb of memory the latest version of Quest takes approximately 100 seconds to search a database of 500,000 sequences. Whilst BLASTPGP in comparison takes on average 18 seconds, over five and half times faster. Rather than introduce more heuristics to speed up the operation of Quest, we decided to make more efficient use of the computing resources available to us.

2.3.1 The Cluster

There are three main computing resources available for general use within the department. There are 15 user machines of various configurations and ages all running a standardised version of the Linux operating system (Linux is a freely available Unix-like operating system, suitable for use on most desktop computers; for further information see `http://www.linuxhq.org`, `http://www.linux.org` and for the particular distribution we use `http://www.suse.com`). We also have two Linux clusters (also known as compute farms) named Jura and Oban. Jura is made up of 27 dual processor computers, which because it was constructed over a significant period of time, vary from 400MHz Pentium II's with 512Mb to 600MHz Pentium III's with 1Gb of memory. Oban is much newer and is made up from 64 identical dual 733MHz Pentium III processor machines each with 1Gb of memory. The basic setup of the clusters is shown in Figure 2.6. Each processor in these clusters represents a single computing node that run an individual program, this makes 54 nodes on Jura and 128 on Oban. The two clusters are gen-

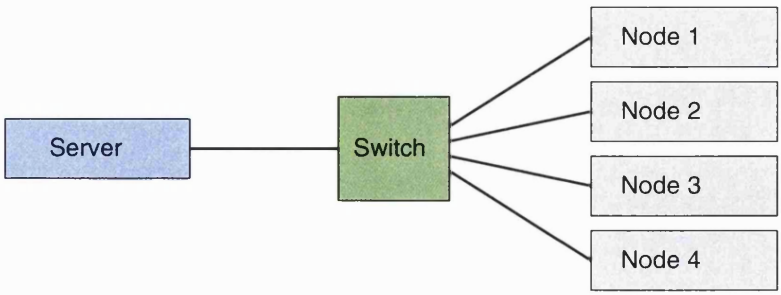


Figure 2.6: **Topology of the Linux Cluster.** This diagram shows a simple cluster made up of four nodes and one server. The server is a centralised computer that distributes jobs to the cluster nodes. It is connected to all the nodes by a switch. The switch also acts to connect all the nodes to one another, allowing a process running on node 1 to communicate directly with another on node 4. When a node has finished its assigned job it sends the results back to the server, from where it is delivered to the user.

erally used to run time consuming, computationally exhaustive or memory demanding programs. For example an all against all database search (all sequences in a database being compared to all other sequences) could run upto 128 times faster if it split across all the oban nodes.

As well as being possible to run multiple copies of one program at the same time, it is also possible to design a program so that it can run across many computers/processors at once to finish a single task more quickly. This technique is known as parallel programming. By implementing parallel programming in Quest it should be possible to significantly cut the time required for a database search.

2.3.2 MPI

For a program to utilise several computers/processors at once it must be capable of communicating between them. There are several communication systems available that provide libraries for C programming, but one of the most widely used is the Message-Parsing Interface (MPI). MPI is actually a standard that defines the syntax and semantics of various library routines. There are several MPI implementations available, both commercial and free. We used the freely available 'mpich' from Argonne National Lab and Mississippi State University (`http://www.mcs.anl.gov/mpi/mpich/`), but any other implementation should work just as well. The scope and use of MPI is too broad to cover here, but extensive literature is available both on the web and more traditional sources (Pacheco, 1997). A parallel program running on several computers at once is not a single program, but instead multiple copies of the original program communicating with

each other, working towards a common goal. MPI provides all the necessary functions to carry out this communication. But it is still up to the programmer to figure out how to spread out the work load and what information to communicate between the processes.

2.3.3 Simple Parallel Database Searching

The first task in converting Quest to run in a parallel fashion was to assess which functions and routines could be spread across several processors and which ones all of the processors would need knowledge of. Stage I of Quest would need to be completed by all the processes as it provides the information that is the basis of the search. It would however be quite possible to split the work done in stages II and III across many processes. These stages are responsible for comparing the search profile to all the database sequences. A simple approach to parallelise this process is to split the database into smaller segments and have each process search one of the segments. This is essentially the method we decided to implement. Rather than physically splitting the database in several files, which would fix the number of computation nodes we would be able to use. We internally assigned sets of sequences from the database to each node, so that no matter how many or how few nodes were used each one would have approximately the same amount of work to do. This assignment of sequences is done in a naive manner with no load balancing implemented. Each computation node is allocated an equal share of the sequences, for example if three nodes were used; node one would scan sequences *1,4,7,10...*; node two would take *2,5,8,11...*; whilst *3,6,9,12...* would go to node three. If one node finishes before the others it must wait, no attempt is made to allocate it some of the

unfinished sequences from other nodes. Whilst it is possible to implement load balancing we decided not to for the sake of simplicity. This will only become a problem if Quest is being run across machines that operate at vastly different speeds, the program will always run at the speed of the slowest machine.

Once all of the nodes have finished scanning their sections of the database they have to transmit their results back to a single node to sort and print out the results. Like the normal version of Quest each of the nodes store the top 500 results of their database search. Every node except the first one then uses MPI to package up these 500 results and transmit them to the first node. These slave Quest processes then exit having completed the parallel part of the program. The first node which can be thought of as the master process, receives all the results from the nodes and adds them to its own. The increased result set is then sorted according to P-value and the top 500 are retained. The program then deals with these as normal and prints out the final results.

Whilst the parallel processing scheme implemented in Quest is very naive, it is nonetheless very effective. The previously used example stated that Quest would take about 100 seconds to search a database of half a million sequences. When running across two processors this drops to 50 seconds, across 25 Quest takes just 6 seconds. There is not much further gain beyond this, the more processors that are used, the greater amount of communication that has to go back and forth between them. Eventually it is the speed of this communication that becomes the limiting factor, with each process spending more time waiting for data to arrive. Similarly this is why the speed does not double each time the number of processor used doubles. This limitation is not a severe

problem for us, using parallel processing we are able to accelerate Quest upto 16 times faster. At the same time we are still using less than a fifth of the computing power available to us.

The scanner file

The scanner file was introduced in order for each computation node to rapidly move between it allocated sequence. The scanner file comprises three important pieces of information: the number of sequences in the database, the length of the database in amino acids and the file positions of all the sequences in the database. The first and third of these are used by each node to scan its required sequences. Using the file position information to directly access the database file at the correct positions turned out to be much simpler and faster than making each node read through the file from start to end, whilst only assessing the sequences it was allocated. Accordingly the 'scanner' file was generated. A program available with Quest, appropriately named scanner will generate the file automatically for the given database. The reason that the scanner file has been made part of the normal Quest program is twofold. Firstly it allows both versions of Quest to work in identical ways. This prevents the need for duplicate functions that do the same job. In fact by programming all of the MPI dependent code within 'IFDEF' statements it was possible to code both versions of the program in the same files, with the desired version being specified at the time of compilation. This had the advantage that when a change was made to the program it affected both the MPI and normal versions without having to remember to update two file sets. Secondly the scanner file allows additional information on the database to be passed to Quest. For example, one of

the standard pieces of information the scanner file contains is the length of the database. This is essential in the P-value calculations and its presence in this file prevents the need for it to be passed to or calculated by Quest during each run. One future goal for the development of Quest is to expand the use of the scanner file, perhaps use it to pre-filter the database and thus accelerate the search procedure, in much the same way as the formatdb program does for the BLAST programs.

Chapter 3

The QUEST Scheme

In the original literature (Taylor, 1998; Taylor and Brown, 1999) very little distinction was made between the QUEST scheme and the Quest program. Nor indeed was there any need to be, only two programs were responsible for all of the phases of the scheme; Quest and MULTAL (Taylor, 1988). In addition these two programs were very interdependent, each produced output in a format the other could use, but incompatible to most other programs. The two programs together were fairly inseparable and comprised the entire scheme. Throughout this thesis I have been very careful to distinguish between the scheme and the program. With good reason, when this project was started we believed that this search scheme could be much enhanced by increasing its modularity. This would allow different programs to be slotted into the different roles depending on which was the most efficient for the job in hand.

In its original conception the QUEST scheme only really had two phases; the search phase and the alignment/assessment stage. When redefining the scheme we separated

the alignment and assessment into separate phases to give the scheme shown in Figure 1.4. The reason for this is simple, most alignment packages make no comment on the quality of the sequences they are aligning. They will attempt to produce the best alignment even if the sequences are not homologous. Hence a separate stage would be needed to assess the quality of the alignment and whether any sequences should be rejected. For any alignment packages that are capable of filtering out bad sequences the separate assessment phase offers a simple and convenient double-check.

3.1 The Search Phase

The search phase has one requirement of whichever program is used. The program must be able to take a multiple alignment as an input and it must return the sequences that it considers as true hits in the database as an output. There were two main contenders available to us for use in this phase, one obviously was Quest, the other was PSI-BLAST. We chose to use Quest for a variety of reasons; firstly because it was our program and we had done a lot of work on it to offer extra functionality to the QUEST scheme. Secondly PSI-BLAST did not offer some of the functionality we were looking for. Although it is possible to feed PSI-BLAST an externally produced multiple alignment, it still requires a single query sequence as an input as well. Whilst it is true that many if not most searches would start with a single sequence and PSI-BLAST would work well in these cases, it will not always be so and in such circumstances PSI-BLAST would require us to choose one of the input sequences as the query. This generates further problems because no gaps are allowed in the query sequence, which would mean a loss of information in the other

sequences. Quest was therefore chosen because it could take a full multiple alignment as a query, with each sequence contributing equally and gaps allowed in all. The method of Quest's operation has been well covered previously and will not be repeated here.

3.2 The Alignment Phase

As previously stated the function of the alignment phase was originally undertaken by the program MULTAL (Taylor, 1988). Now that Quest had been changed so as not to be dependent on the MULTAL file format we decided to evaluate various packages for the alignment phase. We looked into four main possibilities: T-Coffee, Praline, CLUSTAL W and the original program MULTAL.

3.2.1 MULTAL

MULTAL is a implementation of the progressive alignment strategy (Hogeweg and Hesper, 1984). It makes use of a clustering algorithm to align the most related sequence first, followed progressively by more distant sequences. To represent multiple sequences during the rounds of pairwise alignments MULTAL makes use of consensus sequences. What makes MULTAL different from many other alignment packages is that it will not force all the sequences given to it into an alignment. If the score for adding a sequence to an aligned block or joining two blocks is too low then they will not be joined. Any sequence blocks that are not joined to the main alignment (which from the point of view of the QUEST scheme is defined as the alignment block containing the probe sequences) will be reported as separate alignments. It is this ability to reject distant or unrelated

sequences combined with its speed of operation that led to its use in the original scheme.

Whilst these attributes are useful, they are no longer essential. The separate assessment phase has made the ability to reject sequences somewhat redundant. This is now a useful feature as opposed to a required one. In addition to this change in circumstances we also ran into some other problems with the program. To produce an alignment MULTAL requires a parameter file in addition to the sequence file. The parameter file controls the behaviour of MULTAL during its iterative alignment procedure. Quite small changes in the file can lead to large differences in the reported alignment. Before we began to evaluate other alignment packages to carry out this phase of the scheme, we made an attempt to optimise the operation of MULTAL. To do this we needed to identify the parameters that produced the optimal output for the QUEST scheme. This turned out to be no easy task. Each iteration of the MULTAL alignment procedure depends upon a minimum of 5 parameters. In the initial QUEST paper (Taylor, 1998) MULTAL was given up to 14 iterations, this made a grand total of 70 parameters to find the optimum value for. An attempt towards this goal was made using the BAliBASE (Thompson et al., 1999a) alignment set as an assessment criteria. However this attempt was soon abandoned because the time needed to achieve this goal was not affordable for the importance of the task and the fact that the work done did not yield any appreciable improvement in results. In addition to this, work done using BAliBASE to benchmark various alignment programs (Thompson et al., 1999b) showed MULTAL to be one of the worst performing in terms of alignment accuracy.

3.2.2 CLUSTAL W

The CLUSTAL W (Thompson et al., 1994) alignment program has already been touched upon in Section 1.5.1. Like MULTAL it is a progressive alignment strategy. The original CLUSTAL (Higgins and Sharp, 1988) program was a contemporary of MULTAL, but the program was revamped several times leading to CLUSTAL W. CLUSTAL W is many leaps and bounds ahead of it first incarnation. Firstly the input sequences are weighted to prevent near duplicate sequences causing distorted alignments. Secondly different amino acid substitution matrices are used at different stages depending on how divergent the sequences or blocks are that have to be aligned. Thirdly gap penalties are implemented in a residue specific type manner, giving much greater control over where gaps are inserted. This procedure also includes reducing the gap penalties for regions which already have gaps in them from the early alignment stages. This encourages new gaps to form in the same regions. Fourthly the guide tree is constructed by the Neighbour-Joining method (Saitou and Nei, 1987) rather than the UPGMA method (Sneath and Sokal, 1973). This host of improvements coupled with those made in earlier revisions has made CLUSTAL W one of the most reliable alignment programs available. This conclusion is borne out by its performance in the benchmark (Thompson et al., 1999b) carried out using the BAliBASE alignments. CLUSTAL X (the graphical user interface for CLUSTAL W) was found to be in the top four of the alignment packages tested. In addition to this it was the only one of these four that was based on the older progressive alignment method and as such ran approximately 80 times faster than it's nearest competitor. The speed accuracy and ease of use of CLUSTAL W have made it one of the most widely used packages and are very compelling arguments in favour of its use in the alignment phase.

3.2.3 Praline

Praline (Heringa, 1999) is a relatively new implementation of the progressive alignment system. Unlike CLUSTAL, Praline does not build a guide tree, its progressive strategy is more akin to that of MULTAL. However unlike MULTAL, Praline is profile based. All of the initial pairwise alignments are calculated, with the most similar pair of sequences being joined and replaced with a profile. The profile is then compared to all the remaining sequences and the highest scoring alignment is joined and replaced by a profile. It is important to note that at this stage the highest scoring alignment may be a pairwise one between two as yet unjoined sequences or between the first profile and another sequence. This procedure continues until all the sequences have been combined in single multiple sequence alignment. In addition to this well proven technique, Praline has the capacity to implement two additional novel methods to the alignment strategy.

The first is refered to as profile-preprocessed multiple alignment. The specific aim of this method is to reduce match errors during the early alignment rounds, specifically in relation to the placement of gaps. The idea being to combat one of the inherent errors of the progressive strategy that once a error has been made in a early round it will be propagated through all the remaining ones, the so called 'once a gap, always a gap' problem. This method works by creating preprocessed profiles for each of the input sequences before the alignment itself begins. To construct these profiles all of the pairwise alignments are calculated, then for each input sequence these alignments are used to produce a stacked alignment of sequences in much the same way that PSI-BLAST produces its alignment. Sequences that fall below a user defined cutoff are not

added to the stack. Any regions that would involve the insertion of a gap into the stack's 'seed' sequence are excised (again in the same way as PSI-BLAST operates). These stacks are then used to generate profiles for each of the starting sequences along with position specific gap penalties. These profiles are then used in place of the original input sequences to generate the final multiple alignment.

The second strategy Praline introduces is secondary structure-induced multiple alignment. This allows Praline to use secondary structure information to improve the quality of the alignment. Because many sequences have not yet had their structures determined, Praline is able to make use of secondary structure prediction programs such as SSPRED (Mehta et al., 1995) with iterative rounds of alignment until a final alignment is produced.

3.2.4 T-Coffee

T-Coffee (Notredame et al., 2000) is another recent alignment package that is loosely based on the progressive alignment strategy. The strategy itself is very similar to the one found in CLUSTAL W. The neighbour-joining method is used to create a guide tree which is then used to determine the order of sequence alignment. However the alignment procedure itself uses weights generate by T-Coffee from library files. These library files can be produced by a variety of programs e.g. Quest. They allow T-Coffee to integrate information from various sources when constructing an alignment. It would be possible to integrate information from structural alignments, threading, manual alignments, sequence motifs, different multiple alignment algorithms and a whole host of

others. By default T-Coffee produces and integrates information from global and local alignments of the input sequences to produce the final multiple alignment. Within the T-Coffee program are versions of CLUSTAL W (Thompson et al., 1994) and Lalign (Huang and Miller, 1991; Pearson and Lipman, 1988) that are responsible for generating global and local pairwise alignments respectively. The results of these alignments are combined together to form an extended library file which is then used in the final alignment process. Whilst these extra alignment rounds do cost T-Coffee in terms of computation time, the program still runs relatively fast and the cost is more than compensated by the result. When T-Coffee was compared against other high-scoring programs using the BAliBASE dataset it had a statistically significant higher average accuracy in each of the BAliBASE categories (Notredame et al., 2000).

3.2.5 Other Alternatives

Whilst these programs were the main four we were giving serious consideration there are a variety of very effective alignment packages available that operate in a diverse manner of methods. I have already mentioned that the BAliBASE benchmarking paper (Thompson et al., 1999b) recommended four programs that performed extremely well. I have already covered CLUSTAL X, but the other three also deserve a mention; DIALIGN (Morgenstern et al., 1996), PRRP (Gotoh, 1996) and SAGA (Notredame and Higgins, 1996). All three of these programs are iterative alignment strategies of one form or another. DIALIGN uses a local alignment approach, constructing a multiple alignment from segment to segment comparisons in an iterative manner, as opposed to the traditional residue to

residue comparison method. PRRP on the other hand is an iterative progressive global alignment strategy. SAGA uses a different approach altogether using a genetic algorithm to optimise an objective function (a measure of the quality of the alignment), this was further enhanced by the use of the consistency based objective function COFFEE (Notredame et al., 1998). Whilst these methods have all been shown to produce very accurate results (Notredame et al., 1998; Thompson et al., 1999b; Notredame et al., 2000), they do have one downfall as far as their use in the QUEST scheme goes and that is the computation time they require. An alignment of 89 histone proteins has been demonstrated to take only 161 seconds for CLUSTAL X, whereas DIALIGN and PRRP both took over 13000 seconds each (Thompson et al., 1999b). The run time required for SAGA to complete a multiple alignment can vary considerably between alignments but examples of over 110000 seconds have been quoted (Notredame et al., 1998). The alignment phase is already the rate limiting step of the QUEST scheme and thus there is always going to be a trade off between speed and accuracy. Even so we considered the time required for many of these programs to be excessive for our requirements and hence they were not seriously considered for the role.

3.2.6 The Final Decision

The final decision was only effectively between two of these programs. MULTAL was eliminated first for multiple reasons including its difficulty to use and optimise but mainly because it had scored so badly in measurements of its accuracy. Its ability to filter sequences was not a compelling enough reason to retain it. Praline was also rejected

fairly early on. Initial work with the program had shown it to be very accurate. Another compelling fact in its favour was that it was an 'in-house' program and as such we had direct access to the author and could get certain changes implemented if necessary. However the reason for the rejection was the same as that given above for not considering other alignment programs. The computation time required to align large numbers of sequences was too great when used in an iterative scheme. This left us with CLUSTAL W and T-Coffee. CLUSTAL W has an established pedigree and is well respected as a fast and accurate alignment program. There was no real issue on which we could fault it. Compared to many other algorithms it is very fast even when aligning very large numbers of sequences. It is already well benchmarked and well optimised. But in the end we chose to use T-Coffee, not because of any failing in CLUSTAL W but because of a few additional features in T-Coffee that could be of use to us. T-Coffee is not as fast as CLUSTAL W by any means, but it is not excessively slow either, it will align large numbers of sequences significantly faster than Praline. Even in the relatively short space of time that T-Coffee has been available it has established itself as a strong alignment strategy, outperforming many of the best packages available including CLUSTAL W itself (Notredame et al., 2000). But there are three features that really tipped the scale. Firstly T-Coffee uses local and well as global alignment information to construct it's alignment. Local alignments are often much better when the sequences that are being compared are very distant and this will be a common occurrence in the results of a database search. Secondly T-Coffee easily allows the user to assign weights to the sequences which are to be aligned. This allows us to increase the weight of the original seed sequences to

prevent their signal being dissipated too much. Finally T-Coffee also has the ability to judge how well sequences fit/belong to an alignment and as such it could be used to filter out incorrect hits. This particular function will be more fully covered in the next section. So for these reasons we judged the extra time that T-Coffee requires over CLUSTAL W to be an acceptable tradeoff, and hence chose T-Coffee for use in our alignment phase.

3.3 The Assessment Phase

Unlike the alignment phase we did not consider a large variety of options for the assessment phase. We considered two main options T-Coffee and a program we have developed named Mulfil. MULTAL was not considered as it would only reject sequences on the basis of its own alignment and not one fed to it from another program. It is important to note that for the assessment phase it is not really a choice of either/or, it is quite possible to use both T-Coffee and Mulfil to filter the sequences, or indeed any combination of programs. It would be possible to tailor QUEST to very specific queries by enforcing a set of equally specific filters at this stage. The filters do not have to act just to accept or reject sequences, they could also alter the alignment itself e.g. filter out low or high complexity regions. As far as this project goes we have kept the assessment phase very simple, using a single filtering program to simply reject any sequences that it deems to be incorrect sequence hits.

3.3.1 T-Coffee

As described in the previous section T-Coffee brings together disparate sources of information to produce an optimal multiple alignment. By default the sources it uses are local and global pairwise alignment results. These sources of information do not always agree, they will often indicate that the same residue of one sequence should be aligned to completely different positions in a second. It is possible to force T-Coffee to output a file that give a score to each residue in the alignment according to how consistent its positioning is to all the alignment methods used. Very distant or or unrelated sequences are likely to have a large number of low scoring residues, as it is likely that the different alignment methods will disagree fairly heavily on how best to deal with them. It is therefore a simple procedure to analyse the contents of the score file to exclude incorrect sequences. In addition, the use of the T-Coffee library produced by Quest will highlight any hit sequences that are aligned to different regions of the query sequences than they were in the original database search, thus flagging them for removal. This filtering system has another advantage, although it is possible to exclude whole sequence on the basis of their consistency scores. Because the scores are instituted on a residue by residue basis it would be possible to just filter out the poorly fitting regions. Thereby keeping the high scoring sections to extend the profile in the next QUEST round.

3.3.2 Mulfil

Mulfil is a purpose written program for the assessment of sequences within the QUEST scheme. It is written in 'C' and shares a large amount of its code with the Quest program

itself (sequence input/output etc.). It reads in a multiple alignment file in Fasta format and then uses 3 separate filters to remove or exclude sequences, before writing the remaining sequences to its output.

Filter 1: Sequence identity

The first of the three filters is by far the simplest. From the alignment it has been given, Mulfil calculates the sequence identity between the sequence hits and the seed sequences. Any sequence hit that has an identity higher than a user specified level (95% by default) is removed from the alignment. This is done to prevent identical sequences from masking the signals of more distant sequences in the Quest profile. Any sequences that are rejected by this method are obviously hits and as such can be recovered at the end of the QUEST iterations. They are just excluded from the remaining search by entering their details into Quest's exclude file. This is the only of the three filters that intentionally excludes correct hits.

Filter 2: Sequence weights

The second filter is responsible for cutting out most of the incorrect sequence hits from the alignment. It is based on the same scheme that Quest uses to weight sequences when constructing its PSSM (Henikoff and Henikoff, 1994). Every sequence at each column of the alignment is given a positional weight (W_p) according to Equation 3.1.

$$W_p = \frac{1}{af} \tag{3.1}$$

With a being the number of different amino acids that occur at the given position in the alignment and f being the frequency of the observed amino acid in the present alignment column. Unlike the original weighting scheme of Henikoff and Henikoff we include gaps (as a 21st amino acid) when calculating the value of a. The sequence weights (W_s) are then calculated by summing all the positional weights of the given sequence, dividing this by the length l of the alignment and finally multiplying by the total number n of sequences (Equation 3.2).

$$W_s = \frac{\sum W_p}{l} \times n \tag{3.2}$$

By definition we know that the mean of the sequence weights is exactly one. From these values we can also calculate the standard deviation (σ_w). Using these two pieces of data we implement our second sequence filter. Any extremely divergent sequence will have an anomalously high sequence weight. Hence we exclude any sequence that has a weight greater than five standard deviations above the mean i.e. if Equation 3.3 is true then the sequence will be excluded.

$$W_s > 1 + 5\sigma_w \tag{3.3}$$

Filter 3: Sequence connectivity

The pairwise similarity scores of all the sequences in the alignment are converted into P-values using exactly the same method that Quest employs. Any pairs that have a P-value lower than a specified cutoff are said to be 'linked'. If a sequence is 'linked' directly to the seed sequence(s) or is linked to more than 25% of the sequences that are directly linked to the seed sequence(s), then the sequence is accepted. Otherwise

the sequence is rejected and deleted from the alignment. The idea of this filter is to reject sequences that form distinct groups within the alignment set which are not well related/connected to the other sequences and the original query sequences in particular.

3.3.3 The Final Decision

In the end the choice of filtering program was made for us. During much of the development of both Quest and the QUEST scheme we had not decided finally on the best alignment program to use for Quest. As a result for the assessment phase we needed a program that could work with any alignment package, this was Mulfil. For T-Coffee to be easily used it is much simpler if it is the alignment package as well as the assessment package. Now that we have settled on T-Coffee for the alignment phase we would also like to implement its filtering abilities alongside those of Mulfil. Unfortunately, T-Coffee does not remove sequences from the alignment itself, another program would be required to read the score file and then cut out the sequences accordingly. This functionality could easily be added as a fourth filter in the Mulfil program. However time constraints have meant that at the time of writing this has not yet been implemented.

3.4 The Method of Iteration

The QUEST scheme itself and its method of iteration is controlled by a Perl script. This script is responsible for triggering the programs required during the search, alignment and assessment stages. However, the method of iteration is not just a matter of running cycles of Quest, T-Coffee and Mulfil until no new hits are found. There are further

complexities, the most obvious of which is that the simplistic scheme shown in Figure 1.4 is not entirely correct. Once the search phase is over the results are passed to the alignment phase. The subsequent sequence alignment is then used in the assessment phase to determine whether any of the sequence hits are incorrect. If this turns out to be so, those sequences are removed from the alignment before it is passed back to the search phase. This is where the problem exists, if sequences are removed from the alignment then the remaining alignment is unlikely to be the optimal one for those that remain. As such the sequences from the assessment phase must re-enter the alignment phase. The resulting alignment may indicate that other sequences are incorrect, and so it must be fed back to the assessment phase. Thus the real QUEST scheme actually has a loop between these two stages within the loop of the scheme itself (Figure 3.1).The scheme will exit from the inner assessment/alignment loop when the assessment phase does not remove any new sequences. The alignment then finally being passed back to the search phase for the next iteration.

This is not the only complexity in the iteration scheme. The QUEST script must also control how the parameters of the three programs alter during their iterations. This process starts with the first iteration. In many circumstances the initial query will be a single sequence, therefore the first run is responsible for adding sequences to form the first profile for the next run. As a result it is possible in the QUEST scheme to set the parameters of the first run independently of the other iterations. This allows us to set stringent parameters for the initial run so that only the closest homologues are added to the profile that the remaining iterations build upon. There are two other

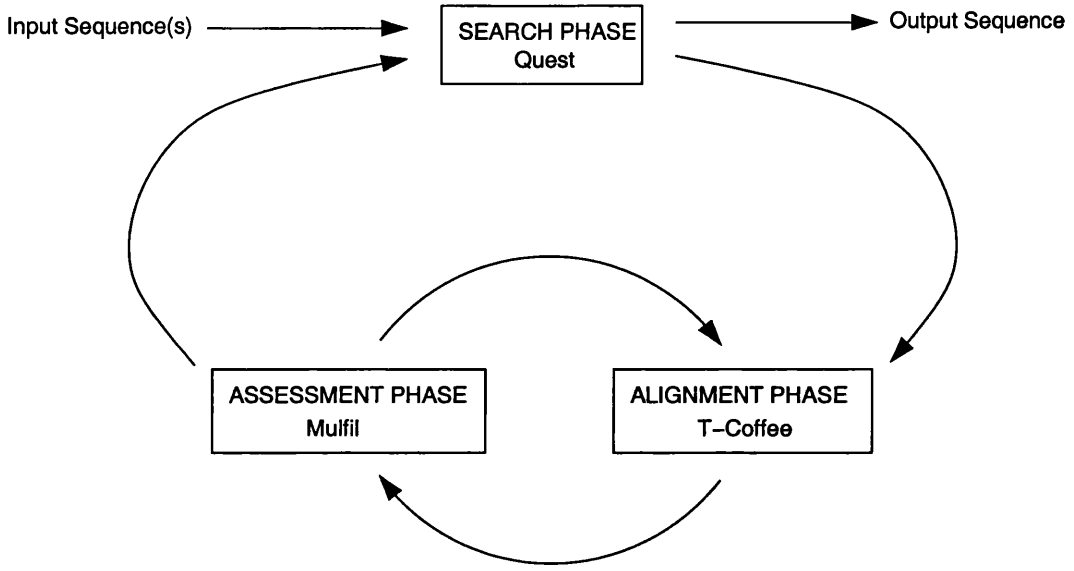


Figure 3.1: **The Real QUEST Scheme**

points at which the parameters may be modified, the first is the divergence checkpoint which occurs after the search phase and the second is the convergence checkpoint which occurs after the assessment phase. The divergence checkpoint put quite simply acts to prevent Quest hitting too many new sequences from the second iteration onwards. If Quest returns more than double the number of hits it found in the previous iteration then the score cutoff is reduced by a factor of ten and the search phase is run again. This is done because casual observation indicated that the worst errors were creeping into the results when large numbers of sequences were suddenly incorporated in one iteration. We believe this was an indication that the query sequences included divergent sequences which had caused the profile to match to new unrelated sequence groups. In addition to reducing the Quest cutoff, the Mulfil cutoff was also reduced by an equal amount, in the hope of filtering out any of these incorrect sequences in the next assessment phase. The second checkpoint is for convergence. This occurs after the assessment-alignment

loop has completed. It has a different role in later rounds, but if no new hits are located in the first two iterations then the convergence checkpoint reduces the strictness setting for the next Quest run and increases the score cutoff (upto a maximum of 0.01 - no increase is made beyond this). This has the effect of widening the search and reporting slightly weaker results, making it more likely that some new hits will be found in the next iteration for the rest of the QUEST scheme to act upon. From the third iteration onwards it does actually act as a convergence checkpoint. It is responsible for stopping the QUEST cycles and reporting the final results when no new sequence hits have been found during the present iteration.

3.5 Cascade-and-Cluster - A New Scheme?

The various problems and errors we encountered during the development of Quest and the QUEST scheme led to the development of a second scheme that I refer to as Cascade-and-Cluster. Profile and alignment based search schemes like PSI-BLAST and QUEST have one intrinsic problem that can not be overcome. By combining several sequences together to search a database we are effectively always searching with a centroid of a group of sequences. Taking a set of homologous sequences in the database, some of those sequences will be outliers i.e. much more distant members of the group. When the results of database searches are combined to search for more sequences we are likely to end up focusing on the 'average' group members, those that lie in the canter of the group. These centroid sequences are likely to be well connected, by which I mean that a database search using them will find a large number of the remaining sequences of the

set. But they may well not hit the outlier sequences, which by definition are not well connected. When used as a query outlying sequences may hit only a few members of the set. The use of a profile to search a database will usually have the effect of searching with a centroid sequence, after all using a profile is a method of searching with the average sequence of those that make up the profile.

There was a second problem with the QUEST scheme that I wished to avoid. The aim of this project was to develop a rapid and sensitive database searching system. Whilst we may well have achieved sensitivity, the QUEST scheme is not rapid. Part of this reason is that the Quest program has not been written with speed as a primary consideration. This could easily be rectified and I am sure that given time it would be possible to rewrite and optimise its functions so as to increase its speed substantially without reducing its sensitivity. The main reason for the slow pace of QUEST is the alignment phase, which as I have already indicated is generally the rate limiting step. As a result if I wanted to use Quest in a much faster scheme I would either have to abandon the alignment phase or at least radically simplify it.

It was in response to these two problems that I developed the cascade and cluster scheme. To approach the first problem it is easiest to think of a set of homologous sequences in a database as forming a sequence cluster or web as shown in Figure 3.2. The letters A-K represent homologous sequences; each possible pair of sequences are connected by a line if the two sequences are recognised as homologous by a sequence search. The length of the lines represents approximately how similar the two sequences are. Sequences A-G represent the central sequences of this cluster, they are well con-

nected to each other by quite short lines. Each of these sequences would be able to locate most of the rest of the group if used as a query. Equally if a number of these sequences were in the profile used to search the database then the majority of the cluster would be identified. Sequences H-K are outlying sequences, they are poorly connected. Using these sequences as a query will only allow us to find one or two other members of the set. If they are part of the profile they may prevent other outliers from being found, e.g. if sequence J was part of a profile with sequences A-K then it's signal may prevent sequences H,I and K from being recognised.

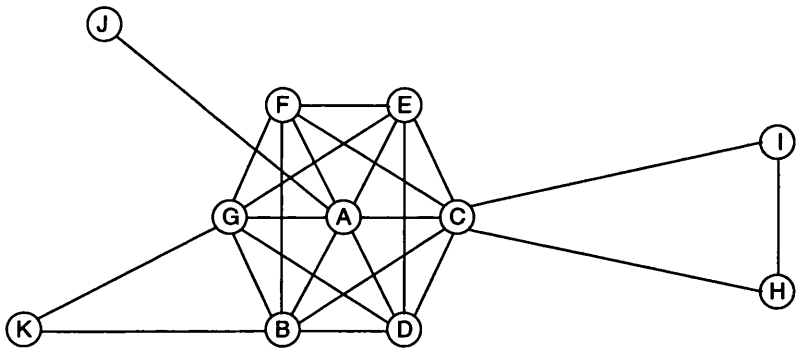


Figure 3.2: **A Sequence Cluster.** Letters A-K represent a set of homologous sequences. The lines connecting them indicate that those sequences are sufficiently similar to be identified as homologues by a database search. The length of the lines is an indication of how similar the sequences are, a long line indicates two distantly related sequences.

However, because these sequences form an interconnected web it is possible to locate all of the by searching the database sequentially with each of the sequences that were hit in the previous round. In effect triggering a cascade of database searches from the original query allowing us to identify all of the sequences contained in the cluster. Unfortunately it is not difficult to see how this cascade could easily run out of control. If all the homologous sequence sets formed discrete units like the one shown in Figure 3.2 there

would be no problem. But in reality these sets will by chance or evolution frequently be interlinked. On a single database search these interlinks will just appear as incorrect hits in the output. During a cascade we would follow these links to other sequence sets, which could then cascade to further sets. Eventually it is possible that this simple technique would involve querying the database with every single sequence contained within it. This is obviously not a feasible concept. To prevent this run away cascade I implemented a very simple limiter, restricting the search to only running two levels, i.e. searching with the original query and the hits produced by it only. Within this subset there is still ample scope for incorrect results, hence an assessment phase of some description is still required. My solution to this problem was to use the Quest results themselves to determine which sequences were likely to be correct hits and which were not. To do this I implemented a simple clustering algorithm (hence Cascade-and-Cluster). The initial Quest run with the original input sequence is referred to as the primary run and the sequences it matched as primary hits. The Quest runs using the primary hits as queries are secondary runs and any new sequences that are matched (i.e. non-primary hits) are secondary hits. It is the results of the secondary runs that I use for the clustering routine. Each time a primary hit matches another primary hit within a secondary run it is awarded a score of one. If two secondary runs share identical secondary hits they will be awarded a score of $\frac{1}{4}$. For example if a secondary run has 3 primary hits in its results and shares 3 secondary hits with other secondary runs, the query sequence of that run will be given a score of 3.75. When all of the primary hits have thus been awarded scores I reject any sequences that have a score of less than two i.e. any sequences that do not link directly back to

the query sequence by two or more primary hits (or eight secondary hits). For those sequences that score above this cutoff, all secondary hits that were instrumental in their score are also reported as results. A graphical representation of this process is illustrated in Figure 3.3. Primary hits 1-3 would be accepted as true hits whilst number 4 would be rejected. In addition to these three sequences the secondary hits coloured blue in runs 1-3 would also be reported as probable true hits. All of the secondary hits from run 4 would be rejected as would any secondary hits in the other runs that are coloured black.

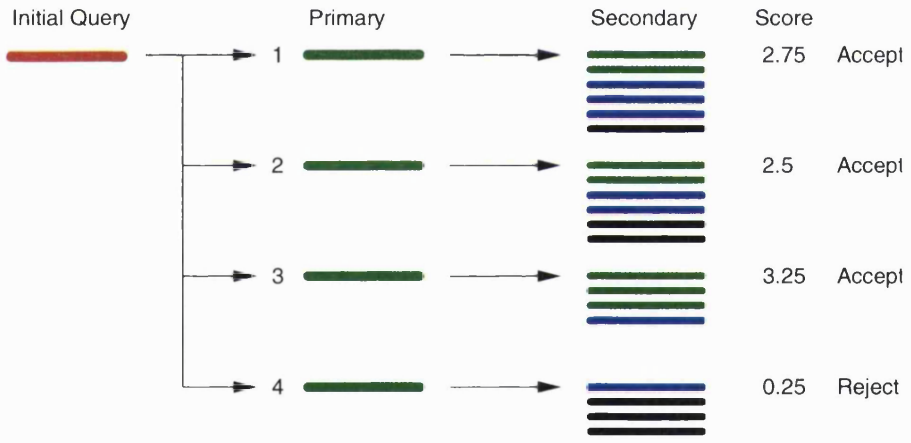


Figure 3.3: **Cascade-and-Cluster** The red bar represents the initial query sequence; the green bars are the four primary hits. The blue bars are secondary hits that are shared by more than one of the secondary runs, whilst the black bars are hits that are unique to that run.

It was only after the initial benchmarking of this technique (detailed in Section 4) that I realised that similar techniques had previously described. The Cascade-and-Cluster method has many similarities with the transitive sequence search (Neuwald et al., 1997; Gerstein, 1998), ISS (Park et al., 1997), MISS (Salamov et al., 1999) and SYSTERS (Krause and Vingron, 1998) methods. In the transitive and intermediate sequence search

(ISS) methods, if sequence a matches sequence b and sequence b matches sequence c, both with high degrees of confidence. Then sequences a and c are inferred to be homologues even if they share no significant similarity when directly compared. In the original ISS presentation the query sequence is used to search the OWL database (Bleasby et al., 1994), thus locating the intermediate sequences. These intermediate sequences were then used to query the database of interest, in this case the SCOP dataset (Murzin et al., 1995). The multiple intermediate sequence search (MISS) method is very similar to ISS the main difference being that as the name suggests multiple intermediate sequences may be employed to extend the search further. These methods allow a search to be extended much further than the traditional single sequence based methods. But at the same time they have the added bonus that they do not require any new programs or algorithms, the original BLAST and FASTA programs are ideal for the job. However the ISS based methods do not make any inherent judgements on the quality of the additional sequences they locate. In this respect the SYSTERS (SYSTEmatic ReSearching) method is much more similar to my Cascade-and-Cluster system. Again an initial query is used to search the database, all highly similar hits are retained, i.e. those with a P-value of 10^{-30} or less. The lowest scoring sequence of this set is used as a query for the next round. This procedure is repeated until either no new hits are found above the cut off or until the search result has no sequence in common with the set of accepted sequences from the first search. This second condition prevents the searches cascading out of control. All of the hits are then scored against the query sequence using Lalign (Huang and Miller, 1991). These scores are then used in a set theoretic clustering method (Krause

and Vingron, 1998). The idea of the SYSTERS method and the clustering procedure in specific is not purely as a database searching system but as a method of splitting a database up into discrete protein cluster units (Krause et al., 2000). As such the method and aim is different to that of the Cascade-and-Cluster system. The aim of my method is to use single sequences and all the diversity they contain to more effectively query a database, whilst at the same time using a simple clustering algorithm to assess and filter of the hits returned.

Chapter 4

Benchmarks

An important step in assessing the effectiveness of the methods and programs we have developed is to benchmark them against a known dataset or an established method. Previous benchmarking papers have explored various techniques for the comparison of database searching programs. One of the earlier papers (Shpaer et al., 1996) compared FASTA, BLAST and Smith-Waterman search methods. For the test set, they used the protein identification resource (PIR) database, or more specifically the super-family definitions of the PIR dataset were used to classify hits as correct and incorrect. A later benchmark (Agarwal and States, 1998), comparing 'probabilistic Smith-Waterman' (a hidden Markov model based method), WU-BLAST2 (a version of BLAST made available by Washington University), SSEARCH, FASTA and BLASTP also made use of the PIR classifications when sorting results. This method has some problems associated with it, the PIR classifications are not considered to be 100% reliable. In fact they can often be partly defined using the very tools that are being tested. More recent attempts (Brenner

et al., 1998; Park et al., 1998; Müller et al., 1999) have focused around the SCOP database (Murzin et al., 1995). SCOP or Structural Classification Of Proteins is compiled from the PDB structural database. Unusually for a structurally based dataset, the custodians of the SCOP database also classify the sequences according to the estimated evolutionary relationships of their structural domains. This combined with the very thorough manually controlled classification system make the SCOP dataset ideal for assessing sequence search methods.

The SCOP classification system is hierarchical, it indicates levels of relationship between the sequences/domains/structures. The SCOP database is organised according to four major levels: Class, Fold, Superfamily, and Family.

Class. For ease of use the folds have been grouped into very general classes on the basis of the secondary structural elements they are composed of. The five structural classes are:

1. all alpha, structure is essentially comprised of α-helices;
2. all beta, structure is essentially comprised of β-sheets;
3. alpha and beta, contain both α-helices and β-sheets;
4. alpha plus beta, α-helices and β-sheets are present but largely segregated;
5. multi-domain, proteins with domains of different folds and for those that have no known homologues.

Fold. Superfamilies and families are defined as having a common fold if the proteins have the same major secondary structures in the same arrangement with the same topological connections.

Superfamily. Families which have low sequence identity but whose structures and possibly functions suggest a common evolutionary origin are placed together in superfamilies.

Family. Sequences are clustered as families on the basis of two criteria. Firstly all proteins that have residue identities of 30% or greater or secondly proteins of lower sequence identity but whose structures and functions are very similar, e.g. the globin family has sequence identities as low as 15%.

It is this hierarchical classification that makes this dataset particularly interesting to us. By conducting searches within this database we can judge which hits are correct, which are incorrect and even how incorrect they are, e.g. if they are not in the same family but are still a member of the same superfamily. These factors combine to make a compelling argument in favour of the use of the SCOP database in benchmarking applications.

The benchmarking papers that make use of this dataset also present a simple but effective way of evaluating the results of different search programs. Using a shortened version of the database that only includes sequences that are less than 40% identical, previous benchmarks (Brenner et al., 1998; Park et al., 1998) ran all against all database searches for each program. Then for each program taking the combined results of the searches and sorting them by score, it is possible to plot a graph of true hits against false hits. This can easily be converted to a coverage (proportion of total true hits located) versus errors per query plot. Ideally the coverage should increase first to the maximum followed by any increase in the number of errors per query, denoting that all true hits are found first, followed by any false ones. In reality this perfect segregation is

not possible, but by comparing the lines of different programs it is possible to compare their sensitivities and selectivities simultaneously.

4.1 Methods

The methods we employed to benchmark our methods and programs are very similar to those mentioned above. For the reasons stated previously we have made use of the SCOP dataset. However rather than use the PDB40D set that contains only distantly related sequences we decided to make use of SCOP-70 (Astral-SCOP 1.53 (Brenner et al., 2000); `http://astral.stanford.edu`). This is made up of all the sequences in the SCOP database that share a sequence identity of 70% or less. The reason for using this dataset is twofold. The 40% filtered dataset is very small, containing approximately only 1000 sequences. The 70% set on the other hand has 4133 sequences, a significantly larger dataset and much easier to draw conclusions from. Secondly the SCOP-70 set is more realistic, when a database search is carried out there will frequently be highly similar sequences as well as a range of more distantly related ones. A database search program must be able to cope well over this entire range. This requirement was well demonstrated to us during development, when small scale tests proved that it is occasionally possible to improve the ability to detect very distant sequences at the same time making it more difficult to locate more similar ones. Therefore it is important to test a program against a whole range of conditions.

There are several other significant differences between the methods we employed and those that have been reported before (Park et al., 1998). Our approach throughout

was to compare the various programs as equally and realistically as possible. When running PSI-BLAST we did not carry out an initial search in a separate larger database to pick out useful intermediate sequences. All searches were carried out solely within the benchmarking dataset. This was done because not all programs are capable of using intermediate sequences making such comparisons unfair. Secondly, in a real database search users are likely to just be searching within one database, thus this provides a more realistic scenario. In the same paper any high scoring hits that PSI-BLAST lost during iterative rounds were artificially added back to the final results. This is not a fair way to conduct a benchmark, it could easily boost the PSI-BLAST results, making it appear to be better than it actually is. So to prevent these errors in our benchmarks all the programs were run in a simple manner with no outside intervention.

Each of the benchmark runs is assessed in an identical manner. For each program the database is queried in turn by each of the 4133 sequences it contains. Each of these 4133 results is then combined into a single list which is sorted by score. Each of the hits are graded as true or false depending on whether or not they are members of the same SCOP family. Cumulative scores are then calculated, so that for each sequence hit in the list the number of correct and incorrect hits identified so far is known. The data is transformed subsequently once more to give the coverage and errors per query (EPQ). Coverage is calculated by dividing the number of true hits located so far by the theoretical maximum number of true hits that could be located in an all against all database search, 44232 for the SCOP-70 dataset. The number of errors per query was calculated as the number of false hits identified so far divided by the total number of queries, in this case

4133. To generate the benchmarking graphs we plotted the coverage (y-axis) against the EPQ (x-axis). A perfect search algorithm would find all the correct hits before any incorrect ones and so the graph would consist of two straight lines, one following the y-axis from 0 to 100% coverage, the other would follow from the 100% coverage point running parallel to the x-axis. This ideal result will not generally be possible, there are typically some homologous sequences that are too distantly related to be distinguished from non-homologous sequences using sequence analysis alone. Nonetheless our aim and that of every database searching program is to approach this perfect result as closely as possible, thus maximising both our sensitivity and selectivity.

4.2 Results & Analysis

4.2.1 Benchmarking Quest

The first comparison we undertook was to measure the performance of the non-iterative Quest program on its own in comparison to other similar methods (Figure 4.1). The programs we chose to compare Quest against were two of the BLAST suite; BLASTP and BLASTPGP. BLASTP is the classic heuristically based protein database search program. It uses a single sequence input and conducts a pairwise search in a manner not dissimilar to Quest. It does not allow gaps in the high scoring segments it identifies, but the segments can be combined together to create a larger sequence hit. BLASTPGP is a much newer implementation and uses a significantly different method of operation (detailed in Section 1.5.2). It is capable of evaluating gapped segments and forms the

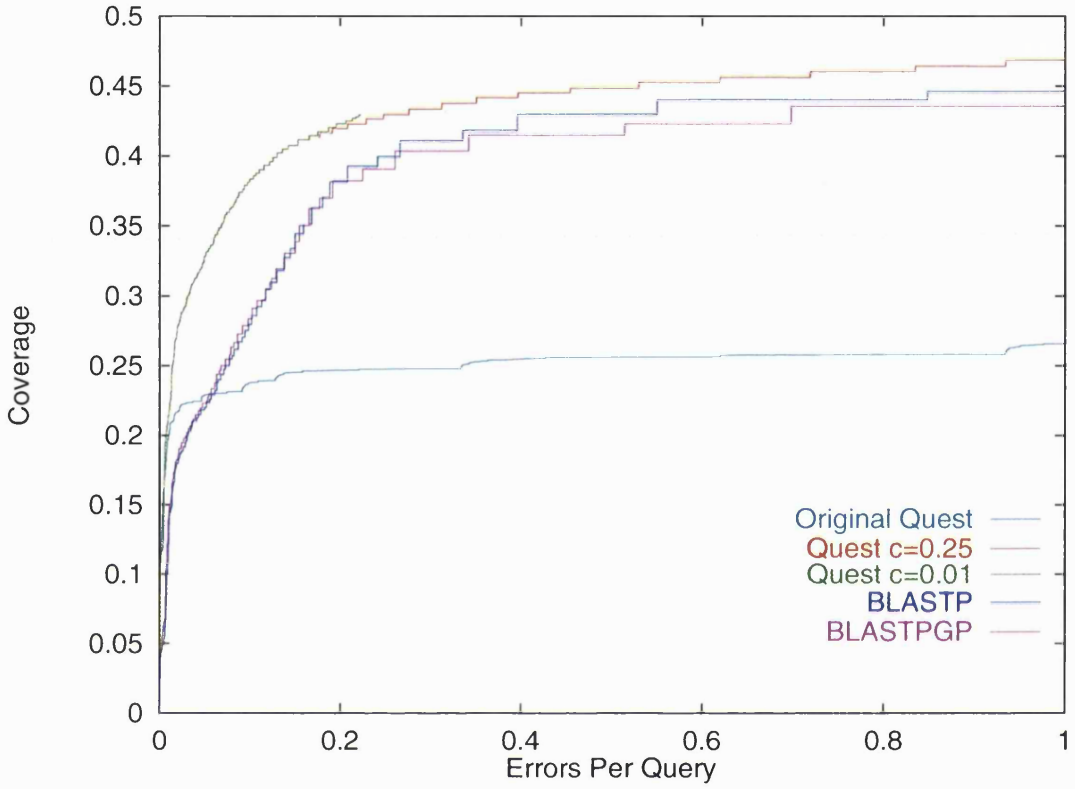


Figure 4.1: **Quest Benchmark.** Comparing non-iterative runs of the Quest program to BLASTP and BLASTPGP.

basis of the iterative PSI-BLAST scheme. All three programs were run with their default parameter settings. A second Quest run was also carried out with a considerably more lenient score cutoff ($c = 0.25$). This was done to give an extended Quest graph that can be more easily compared to the BLAST programs, which by default return hits to a lower significance level.

As can be seen in Figure 4.1 Quest outperforms both versions of BLAST, not only finding more correct hits in total, but also making less errors per query for equal levels of coverage. For example at a coverage of 35% both BLAST programs have an error rate of $\sim$0.2 EPQ, whereas Quest had less than 0.1 EPQ. The default Quest result ($c = 0.01$)

is very reassuring. Although it does not cause the program to stop when it reaches the highest coverage, it does stop Quest just as the Benchmark curve is levelling out, i.e. before the number of incorrect hits located increases dramatically.

Despite the ability to take a multiple alignment as an input, Quest actually shares far more similarity with the BLASTP method than the BLASTPGP method. Interestingly BLASTP does actually perform slightly better than BLASTPGP. The possible reasons for this are as numerous as the differences between these two programs. However, a possible source of this difference is that BLASTP carries out more segment extensions than BLASTPGP. Gapped sequence hits are more time costly to evaluate, as such BLASTPGP strives to save time in other areas. By carrying out less simple segment extensions it may be preventing itself from finding as many hits. This very simple explanation may also be the reason for Quest's out-performance of both the BLAST programs. Because in the present Quest implementation the tripeptide cutoff score has been dropped to zero, far more tripeptide matches and hence segment extensions are evaluated. This could be allowing Quest to recognise a few more homologous sequences than BLAST.

Also shown in Figure 4.1 is the result of the original Quest program. As already stated, the original program was extremely weak when dealing with a single sequence query, this is dramatically illustrated in it's benchmark curve. It achieves a little over half the coverage of either the BLAST programs or the updated Quest program. Whilst many changes have been made to the Quest program since the original implementation (Chapter 2), much of the improvement in this particular case comes from the alterations in the profile construction method.

4.2.2 Quest Parameter Effects

As well as allowing the easy comparison of Quest to other sequence database searching programs, the benchmarking system also allows us to observe how varying the parameter settings can alter the performance of Quest.

Score Cutoff

The score cutoff is a very simple parameter, it simply defines the level at which we consider a P-value to be significant. It's simple operation is reflected in the way it affects the Quest result. The benchmark curve in Figure 4.2 is the same as that in Figure 4.1 when Quest had a cutoff of 0.25. This is an extremely high level to set the cutoff at, the only reason it was chosen was to give an extended line that continues past 1 EPQ. The six vertical lines represent the positions that the Quest line would terminate at, if the cutoff was set to the specified level. For example, at a cutoff of 0.05 the line would end at a coverage of 45% and an error rate of 0.42 EPQ. In other words as the cutoff decreases the benchmark curve remains the same, it just terminates earlier. However, this shrinking of the line is not linear, as the origin is approached a larger reduction in the cutoff must be made to produce a similar drop in the error rate.

Strictness

The effect of the strictness parameter on the final Quest result is not quite as straight forward. Figure 4.3 shows the results of four Quest runs, each with a different strictness setting but otherwise identical default parameters. The strictness settings shown here

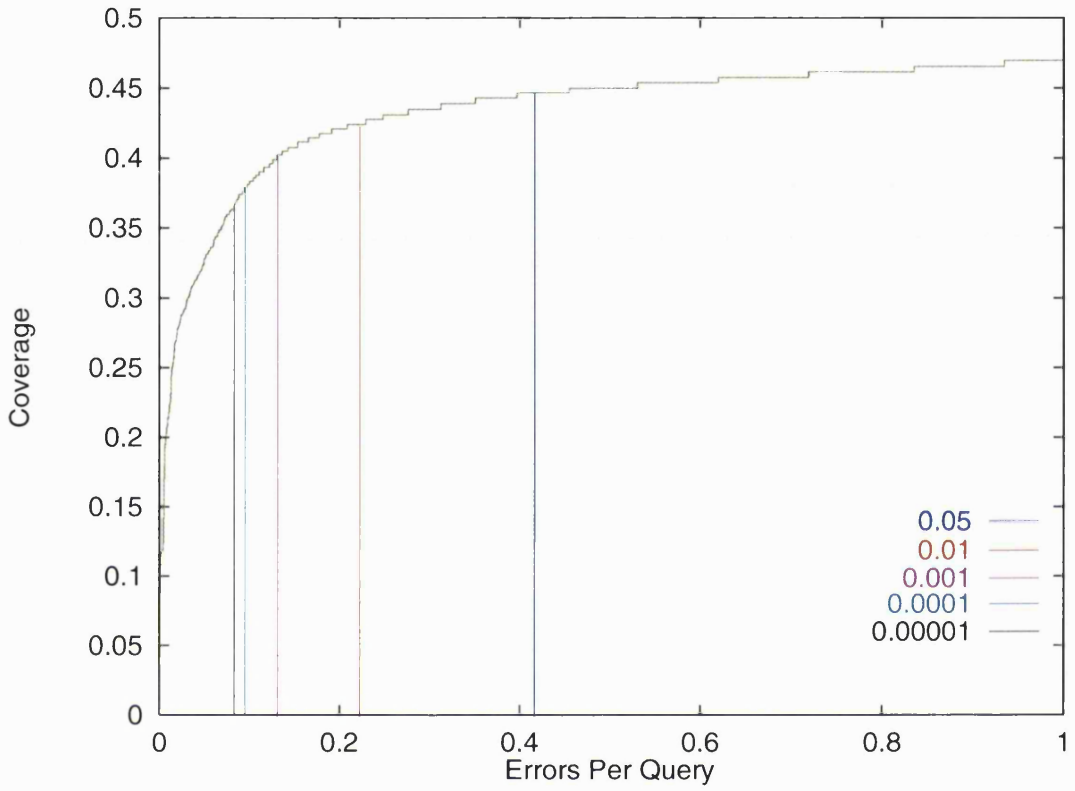


Figure 4.2: **The effects of the score cutoff parameter on the benchmarking curve.** The numbers listed in the key correspond to the score cutoff settings of the Quest program.

were picked for specific reasons. A setting of 100 is the maximum possible and also the default value, which basically turns off the softening function. However, when the input is a single sequence and the BLOSUM62 matrix is used a reduction in the strictness setting down to 50% has virtually no effect. This is purely because the scores for a self hit of a particular amino acid in this scoring matrix are almost always at least twice those for the next highest match. For this reason no other values in this range have been plotted on this graph. It is important to note that this is only the case with a single sequence input. With multiple sequences the scoring system changes and this fact may well not remain true. After this cutoff point each 10% drop in strictness is plotted

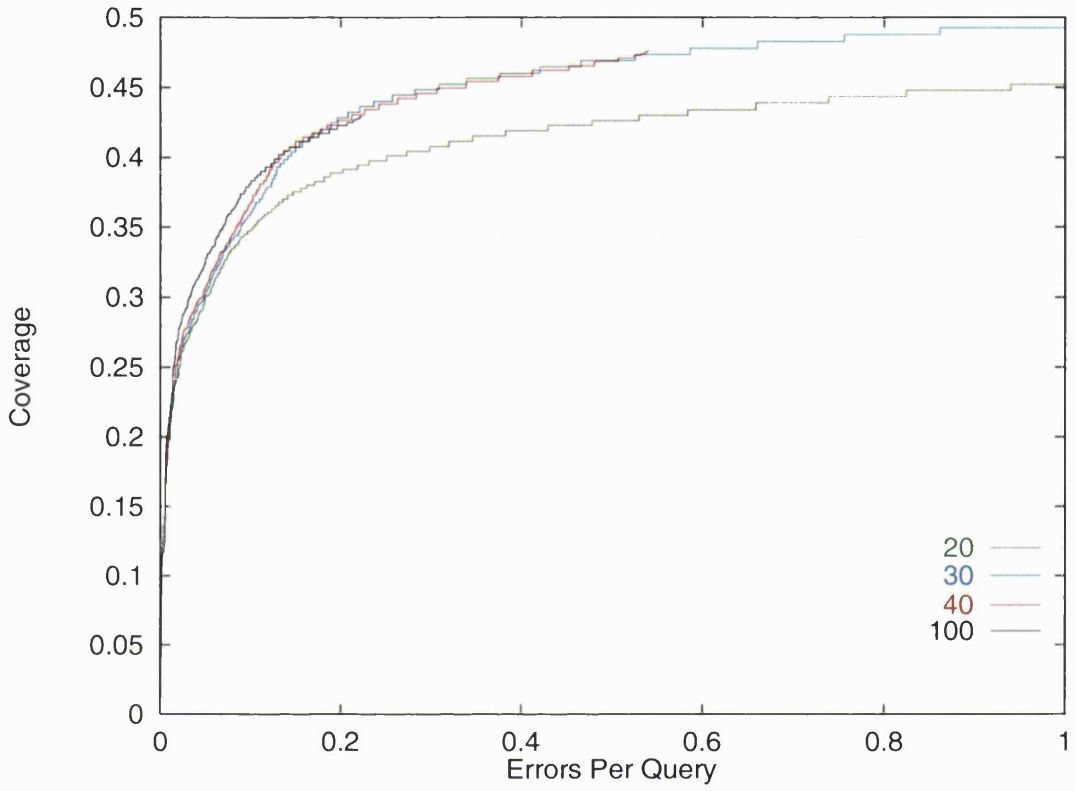


Figure 4.3: **The effects of the strictness parameter on the benchmarking curve.** The numbers listed in the key correspond to the strictness settings used in the various Quest runs.

until 20% is reached. The reason for not going further is purely for the sake of clarity, the 10% and 0% graphs are very similar to the 20% because at these levels the new tripeptides that are introduced are largely (but not wholly) negative scoring and so are prevented from joining the lookup table.

The results for strictness settings of 30, 40 and 100 superficially resemble those of the score cutoff above. It appears as though as the strictness increases the line gets shorter but otherwise stays the same. This is not so. As the line gets longer it is also deforming towards the right, with this being particularly visible in the coverage range 0.25 to 0.4. For equal levels of coverage the lower strictness levels are making slightly

more mistakes. Beyond this point where the lines begin to level out the lower strictness levels actually end up locating a larger number of correct hits, albeit at the cost of a substantial number of incorrect ones. The reason for this is that we are searching with an extended tripeptide match set. This means that we are more likely to find seed tripeptides on more distantly related sequences and as a result are more likely to get a positive result from them. Unfortunately this is a double-edged sword which means that we are also more likely to get chance hits from entirely unrelated sequences, hence the progressively longer tails on the graphs. This is particularly evident on the 20% strictness line which actually extends greatly beyond the bounds of Figure 4.3 to finally end at a coverage level of 58% but also an EPQ rating of 22.9.

Whilst this graph does give an indication of how the strictness parameter can affect the results of a Quest search, it is important not to read too much into it. The action of this parameter is very dependent on the query sequence and substitution matrix used. This is especially true when the input is a multiple alignment. In such cases the actual scoring matrix the parameter operates on (the PSSM) is virtually unique on each occasion and as such so is the effect of the strictness setting.

Gap Penalty

Like the strictness parameter above, the gap penalty does not affect the results in a simple manner. For clarity only a few of the possible settings are represented on the graph (Figure 4.4). The gap penalty has two parts, the gap opening and gap extension penalty. A setting of 0:0 means that there is no gap penalty, a gap of any size can be introduced anywhere at zero cost. With a single sequence query a setting of 100:100 means that no

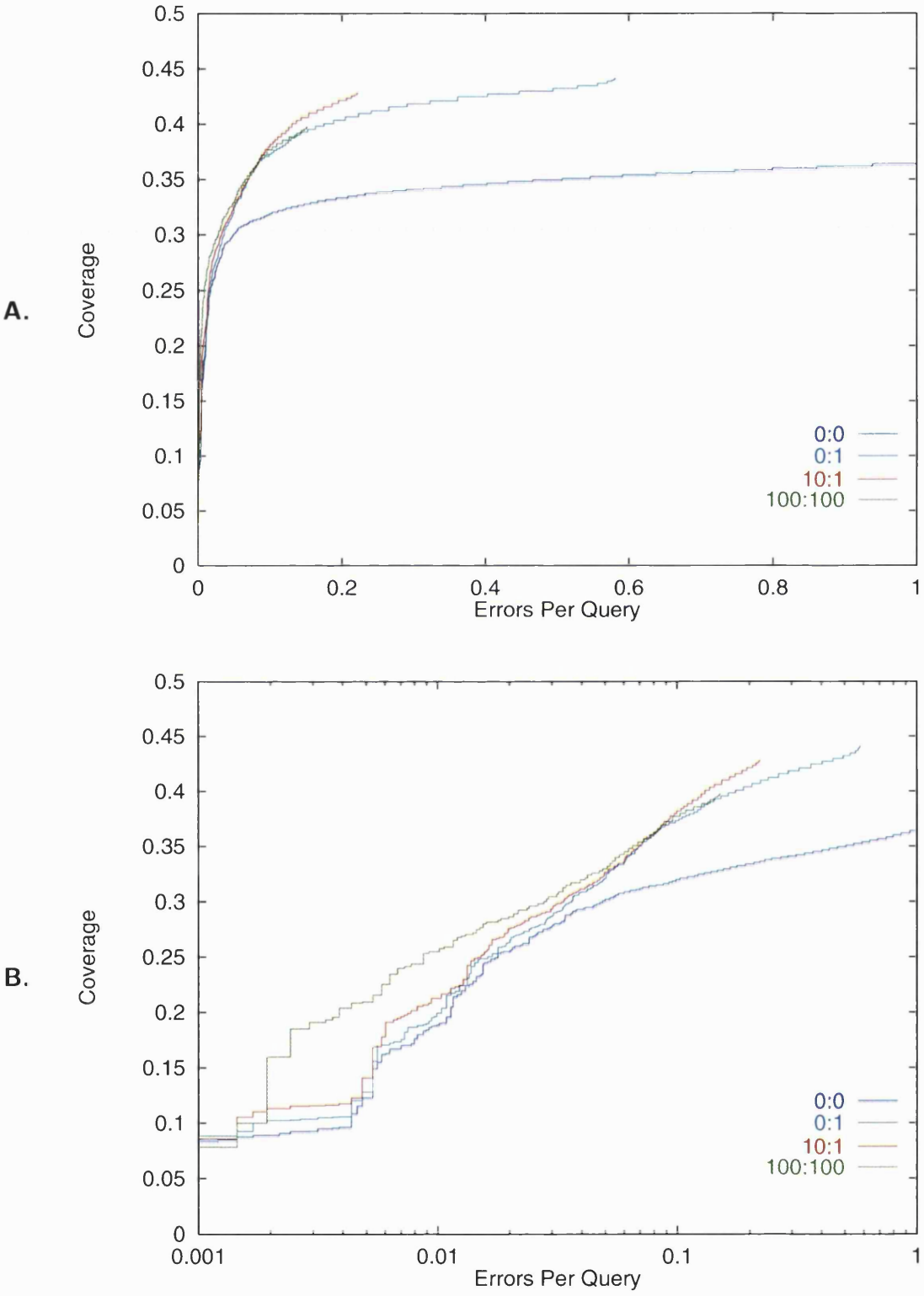


Figure 4.4: **The effects of the gap penalty on the benchmarking curve.** A. Graph shows the benchmarking curve for four specified gap penalties. B. Same graph as A. but with a logarithmic scale on the x-axis to exaggerate the observed differences.

gaps are allowed by the segment assembly function (unless the segments being joined are extremely high scoring). These two settings together represent the opposite extremes of gap penalties, the other two lines detailed on the graphs in Figure 4.4 represent far more sensible values. A gap opening penalty of 10 and an extension penalty of 1 is the default setting for Quest, whereas the 0:1 setting represents a simple length dependent gap penalty. Unsurprisingly the worst performing result is observed when there is no gap penalty imposed. This setting does not prevent any segments from being joined together no matter how distant, therefore the chance of getting a high scoring hit with an unrelated sequence increases significantly. The other extreme setting does much better, in fact until the coverage reaches ~30% it is the best performing setting. This is particularly visible in the log graph (B). At this level the sequences being assigned as homologues are very similar and are unlikely to need gaps between the segments. The sequences that are rejected are those that require gaps, which are more likely to be false hits. To extend significantly beyond this point, the ability to insert gaps becomes necessary to locate the more distant homologues. Somewhat surprisingly the very low length dependent gap penalty actually rises to equal the 100:100 setting at approximately 35% coverage, before continuing to rise to 44% coverage. This is surprising because this penalty is extremely low, in fact it is the smallest possible length dependent penalty available under this scheme. The 0:1 penalty is actually far closer to the zero setting than the strict 100:100 setting, but it performs better than both of these. This is because to find more distant homologues gaps will almost certainly have to be inserted between segment matches, hence a strict penalty will eventually fail. Equally even a very small penalty will prevent

low-scoring segments from joining together and hence will dramatically reduce the chance of unrelated sequences becoming hits. Another significant change is observed when we add a gap opening penalty of ten to this small gap extension penalty. Quest still manages to reach almost the same levels of coverage but at the same time finding two-thirds less incorrect sequences, only 0.2 EPQ instead of 0.6 EPQ.

4.2.3 Benchmarking the QUEST Scheme

Having established that Quest works well as a database searching program in its own right. The next focus of our benchmarking method was to compare the QUEST scheme to other similar methods. There are not a huge number of automated iterative search programs available, we chose to evaluate the performance of QUEST against the most prevalent and similar method; PSI-BLAST. The performance of PSI-BLAST has been evaluated in several previous works (Park et al., 1998; Müller et al., 1999). It has uniformly been found to be an effective search tool and a significant improvement on simpler pairwise search methods, finding approximately twice as many correct hits as FASTA and BLASTPGP for the same error rates (Park et al., 1998). The previous PSI-BLAST benchmarks do not use the default parameter settings, instead the E-value cutoff was set to 0.0005 and the maximum number of iterations was 20. Because these settings were observed to improve the performance of PSI-BLAST with the SCOP database, we also decided to use them and hence make our comparison more stringent. The optimal parameter settings for the QUEST scheme are still the subject of work and discussion, for the benchmark the settings we chose were as follows: For the initial QUEST iteration the

score cutoff for both Quest and Mulfil was set to 1×10^5, whilst all other parameters were left at the default. For all other iterations Quest ran with a strictness of 25% but default score cutoff of 0.01, whilst Mulfil used a score cutoff of 1×10^4. The maximum number of QUEST iterations was left as the default 15. The reason why the QUEST scheme does not let the constituent programs run with their default settings is that these defaults were defined as being optimal (or at least close to optimal) when the programs were run individually and non-iteratively. In effect the iterative context of the QUEST scheme alters what the optima for these various settings are. Figure 4.5 shows that the QUEST scheme does actually outperform PSI-BLAST. This success could be rooted in several

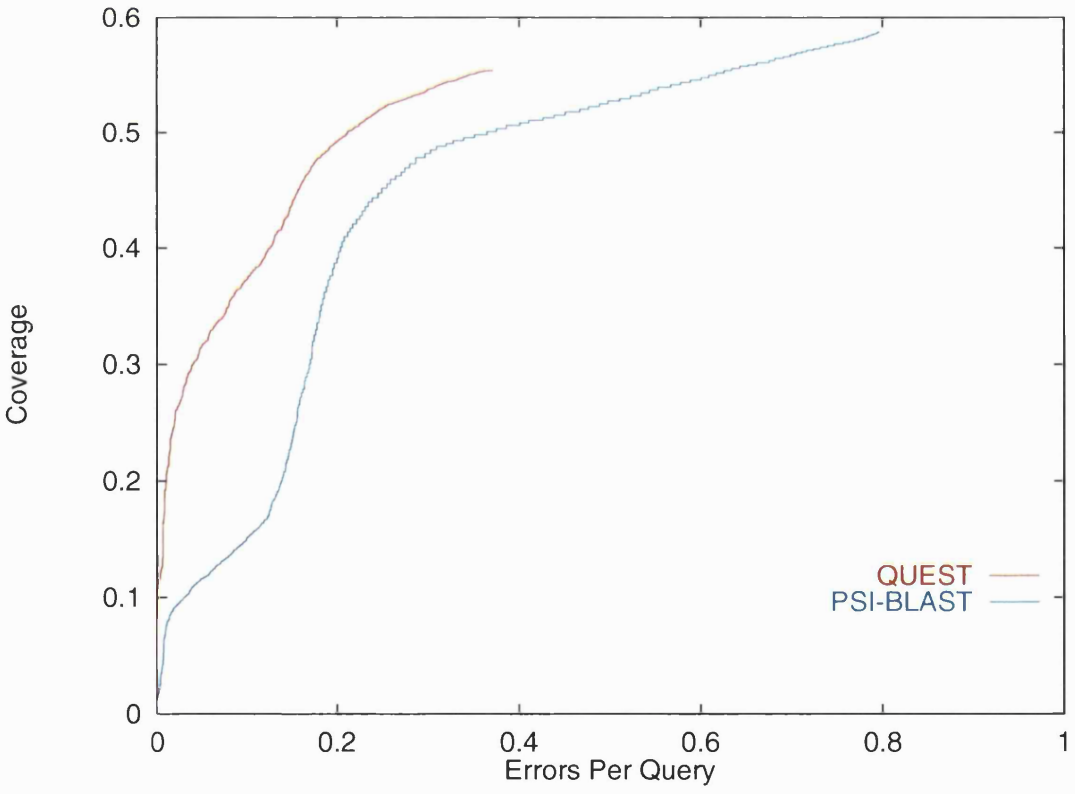


Figure 4.5: **Benchmark graph comparing the performance of the QUEST scheme to PSI-BLAST.**

factors. Firstly the Quest program outperforms the BLASTPGP program and both of these respectively form the basis of QUEST and PSI-BLAST schemes, what we are observing here could simply be the cumulative effect of this. Secondly QUEST uses more information than PSI-BLAST, the sequence hits are fully aligned during each iteration and no parts of the alignment are excluded for introducing gaps into the seed sequence. The alignment quality itself in QUEST as opposed to PSI-BLAST may have been enough to cause this disparity. Finally it is impossible to discount the effect of the assessment phase. The difference could be due to QUEST more efficiently excluding incorrect hits from the search profile. In reality it is likely to be a combination of these factors that are responsible for this difference. It is not easy to isolate which differences are the key ones in causing this result, relatively minor alterations to the various parameters controlled within the QUEST scheme can quite significantly change the result. Two examples of this are shown in Figure 3.1, the original QUEST and PSI-BLAST results are shown for comparison. The first new QUEST line was generated by changing the score cutoff parameter fed to Mulfil. The new setting (0.01) is more lenient than the original. As a result sequences were left in the profile that were previously removed and the QUEST iterations consequently picked up a higher number of incorrect sequences. Additionally during progressive iterations the search profile would be of lower quality and the total number of true hits discovered would have been reduced. The second new QUEST result came from altering the strictness parameter of the Quest search itself from 25 to 30. This change makes the search slightly more strict therfore less segments will be evaluated and similarly less hits will be considered significant. Hence, this line is very similar to

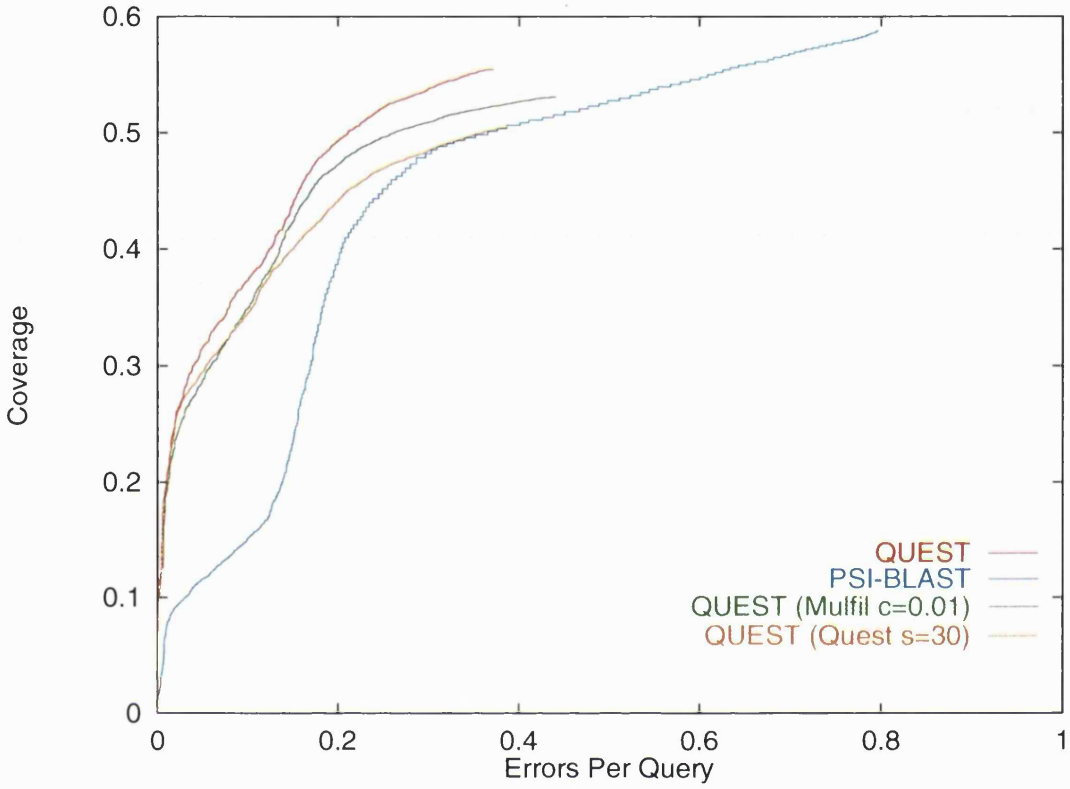


Figure 4.6: **An evaluation of the effects of different parameter settings on the QUEST scheme.**

the original QUEST result just with a lower overall coverage. The fact that the lower strictness setting was more efficient at finding correct sequences is a testament to the effectiveness of the assessment phase. A decreased strictness also greatly increases the chance of hitting inappropriate sequences, yet the error rate in the original run ($s = 25$) is not higher than the $s = 30$ run.

It is quite simple to look at the effect of the multiple alignment quality on the QUEST result. We decided to repeat this analysis using two different alignment methods in the alignment phase. Firstly we tested our second preference alignment program CLUSTAL W. As already stated, this program is not thought to produce quite as reliable

alignments as T-Coffee, it does nonetheless rapidly produce high quality alignments. The second program we tested was Quest itself. Using the pseudo-alignment option it is possible to force Quest to produce an already aligned output file. This shares several similarities to the PSI-BLAST alignment system, it does not allow gaps in the seed sequence and all sequences are only aligned to the seed sequence, not to each other. The effects of these methods on the benchmark results shown in Figure 4.7 speak for themselves. The speed of the CLUSTAL W and pseudo-alignment algorithms was very

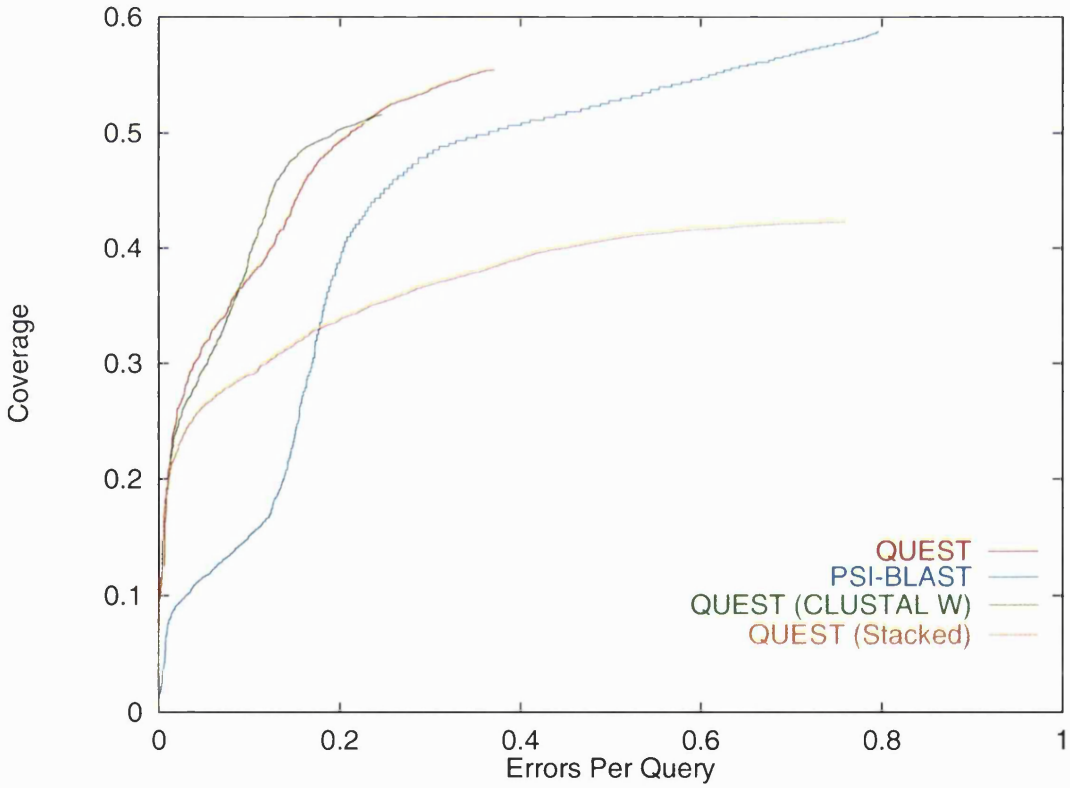


Figure 4.7: **An evaluation of the effects of different alignment systems in the QUEST scheme.** The QUEST and PSI-BLAST results are the same as those presented previously. The CLUSTAL W result uses the same settings as the previous QUEST run and default alignment parameters. The 'Stacked' result uses the Quest pseudo-alignment mode to align the sequence hits. It is so called because the sequences are simply stacked upon one another.

noticeable when running the benchmarks. The pseudo-alignment method negated the need for a true alignment program and as a result the QUEST scheme ran extremely rapidly. Unfortunately the QUEST alignment system is not meant to be completely reliable and it's inaccuracies are quite obvious, e.g. at 40% coverage the original scheme had an error rate of only 0.1 EPQ, whereas the pseudo-alignment scheme was five times higher at 0.5 EPQ. Part of these differences could be attributed to the decreased amount of information in the pseudo alignment compared to the normal alignment file. The pseudo-alignment only includes the regions of sequences that are contained in the segment hits that make up a sequence hit. In the normal Quest output file the intervening regions are also placed in the output file, allowing the search to be extended through them once they have been properly aligned. The CLUSTAL W program does very well, the reported difference in alignment quality between itself and T-Coffee is not large and this is a point which is well reflected in the results. At a coverage level of between 35% and 50% the CLUSTAL W based scheme actually outperforms the T-Coffee one. However, the result drops off and terminates after this stage whereas the T-Coffee one continues to climb. This result combined with the much reduced time required to run the QUEST scheme using CLUSTAL W makes it much more difficult to justify the use of T-Coffee in the present implementation for the small gain that it does make.

4.2.4 Cascade-and-Cluster

The Cascade-and-Cluster method was devised as a possible complement or alternative to the QUEST scheme. Whilst slower than PSI-BLAST, it runs considerably faster than

the QUEST scheme itself. This is because there is no alignment phase involved in this method, hence the slowest part of the QUEST scheme is not present. In addition to this the assessment of sequences is carried out on the Quest results themselves using a very fast and simple algorithm. In consequence the total time taken is just the sum of the required for the multiple Quest searches. Whilst this particular search scheme is still very much in development and little or no optimisation has been done on it we considered the theory behind it compelling enough to warrant further investigation. The results of the benchmark are seen in Figure 4.8, the QUEST and PSI-BLAST results are unchanged

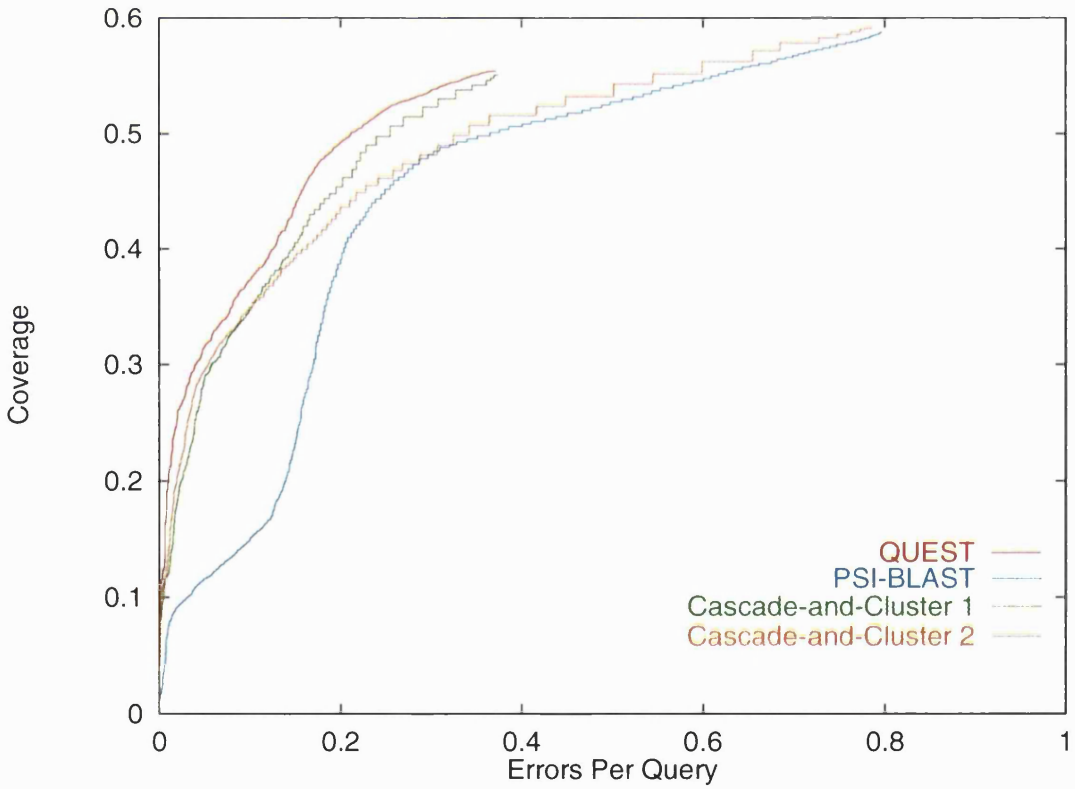


Figure 4.8: **Benchmark of the Cascade-and-Cluster scheme.** The QUEST and PSI-BLAST results are taken from Figure 4.5. The primary and secondary runs of 'Cascade-and-Cluster 1' use the Quest program running with default parameters (strictness of 100 and score cutoff of 0.01). 'Cascade-and-Cluster 2' is the same as the first except that for all the secondary runs the Quest strictness is set to 30.

from Figure 4.5, whilst the two Cascade-and-Cluster results correspond to two different parameter settings. The first result was obtained simply by using Quest with the default settings for both the primary and secondary searches. The second result shows how the coverage can be increased simply by lowering the strictness of the secondary search runs from 100% to 30%. Both of these results are very interesting. Despite the simplicity of the scheme and not having optimised the settings for either Quest or for the clustering mechanism, the first result still performs significantly better than PSI-BLAST and only slightly less well than our best QUEST result. The slightly less strict settings of the second run do significantly increase the number of incorrect sequences found, however at the same time the total number of correct hits identified is boosted, so that the overall result is slightly better than PSI-BLAST for the entire length of the graph.

More simplistic intermediate sequence search methods have been shown to perform at a level half way between the best pairwise and profile based search methods (Park et al., 1998). The Cascade-and-Cluster scheme is really only an extended form of intermediate sequence searching, yet with this particular dataset it has shown itself to perform better than the profile based search system of PSI-BLAST. At the same time it has performed on an almost equal par to the QUEST scheme. This poses questions as to whether combining sequence information to produce a profile is always the best method to use. Is the time spent aligning sequences in the QUEST scheme truly justified? PSI-BLAST does well with a much more approximate alignment and the Cascade-and-Cluster scheme uses no alignment whatsoever.

4.2.5 PSI-BLAST Version 2: An Improved Challenger

During the writing of this thesis a new improved version of PSI-BLAST has been made available (Schaffer et al., 2001). The changes made to produce this new version are generally quite small, none of them are of the magnitude of those that corresponded to the change from BLASTP to PSI-BLAST (Altschul et al., 1997). The most important of these improvements is the introduction of composition based statistics. The two parameters that are required to calculate P-vales and E-values are λ and K; their values are dependent on the scoring system used and the composition of the sequences being compared. It is not feasible to recalculate these values for the composition of the sequence based in each database search, however, it is possible to rescale the PSSM to alter the value of λ to fit the one used. Such an estimation should improve the accuracy of these statistical measures when the composition of the query and database sequences are significantly divergent from the expected values. Another small change was to incorporate the information provided by gaps in the sequences. Previously when gaps occurred in an alignment column, only the sequences that had amino acids present were counted, any with gaps were ignored. The new implementation treats gaps as a 21^{st} amino acid, allowing a little more information to be gleaned from the profile. The cumulative effect of these small changes has been quite significant. Figure 4.9 shows the best results from the QUEST and Cascade-and-Cluster schemes along with the original PSI-BLAST result (denoted as version 1) and the new PSI-BLAST result (denoted as version 2). The first graph (A) shows that the performance of PSI-BLAST has jumped upwards by quite a margin. Whilst both the QUEST and Cascade-and-Cluster

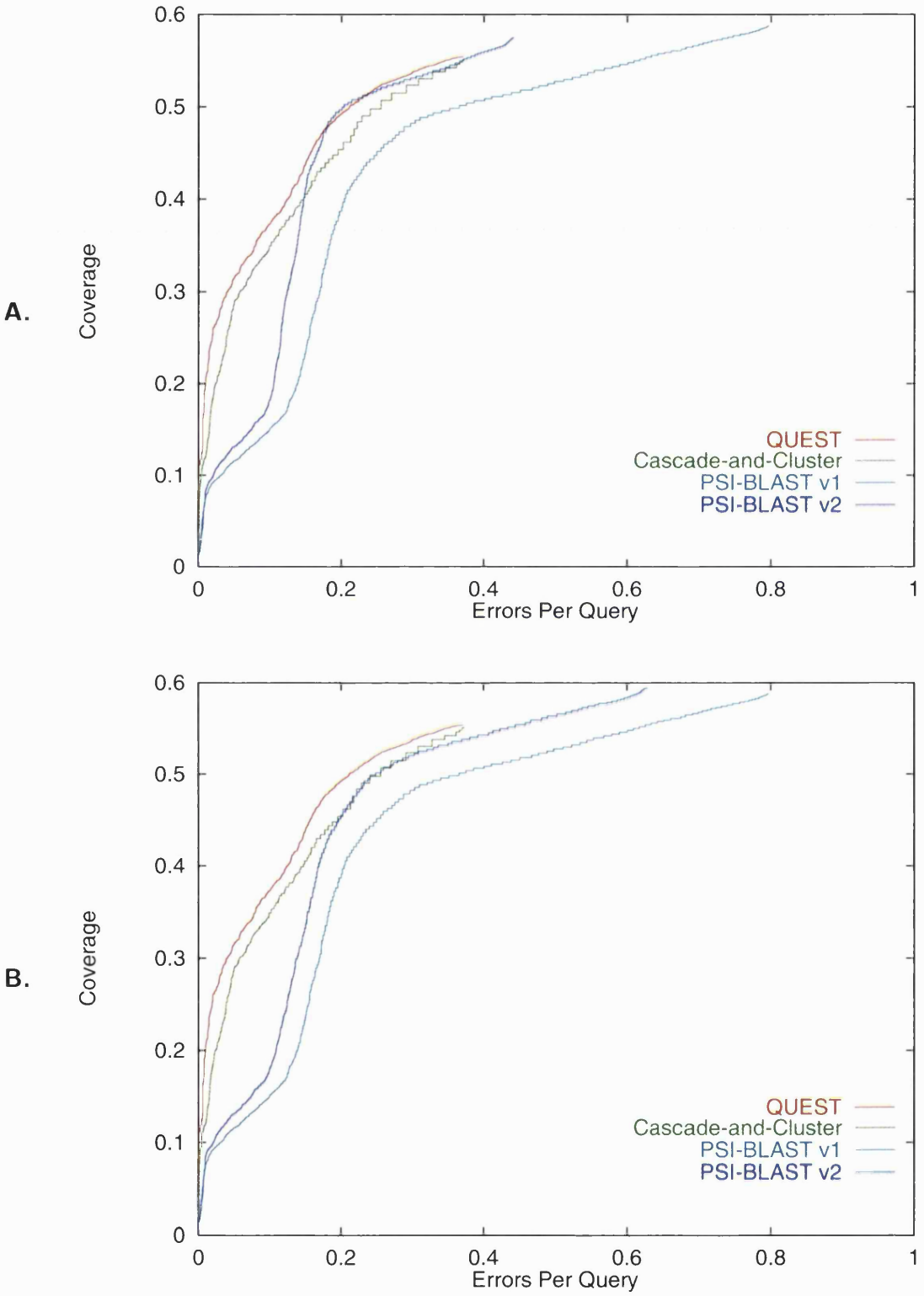


Figure 4.9: **Benchmark of the new version of PSI-BLAST** A. The result for PSI-BLAST version 2 was obtained using the same optimised parameters as the original PSI-BLAST run. B. PSI-BLAST version 2 running with its default parameter settings.

schemes still outperform PSI-BLAST up to the 40-45% coverage level, beyond this point the performance of PSI-BLAST is equal to QUEST and above Cascade-and-Cluster. Furthermore the PSI-BLAST result continues past the point that the QUEST result reaching a high of 57% coverage. The second graph (B) shows the same results, but this time the new version of PSI-BLAST was run with it's default parameters rather than the more optimal ones suggested previously for this dataset. In the original implementation the effects of these parameter settings was not large; they improved the result, but not by much. The new PSI-BLAST implementation is another matter entirely, the default settings do not do as well on this particular dataset. PSI-BLAST still rises above the final results of both the QUEST and Cascade-and-Cluster schemes, but it does not actually pass through their results and the increased number of errors per query shifts the line to the right. This result is obviously something of a disappointment to us as PSI-BLAST is still a much faster scheme than QUEST. It does not use the extra sequence information that QUEST can, yet it still manages to perform as well as QUEST in this new version. However, these results are not as bad as they may at first appear. PSI-BLAST does do extremely well, however a lot of work has been put into optimising its performance since the original release and by comparison there is still a lot of improvements to be done within the QUEST scheme. The Cascade-and-Cluster system on the other hand has had no optimisation whatsoever and yet still performs almost as well as PSI-BLAST, which is certainly one of the most powerful search programs in common use. In addition to this the graphs of both the old and new PSI-BLAST programs present an interesting result. When compared to QUEST, Cascade-and-Cluster or even just a single run of BLASTP

or Quest, the PSI-BLAST results are unusual in that for the early part of the benchmark curve they perform very badly. This point is emphasised in Figure 4.10. PSI-BLAST not only performs substantially worse than QUEST up to the 50% coverage level, it is also outperformed to the 30% level by a single BLASTPGP run (the basis of PSI-BLAST itself). This figure rises to 40% when PSI-BLAST is compared directly to a non-iterative Quest run. The form of the PSI-BLAST graph is almost sigmoidal; it rises well at first, but after finding about 10% of the possible correct hits it suddenly takes in quite a large number of errors. This causes PSI-BLAST to have an error rate of approximately 0.1 EPQ for only 20% coverage, compared to QUEST and Quest which find less than 0.01

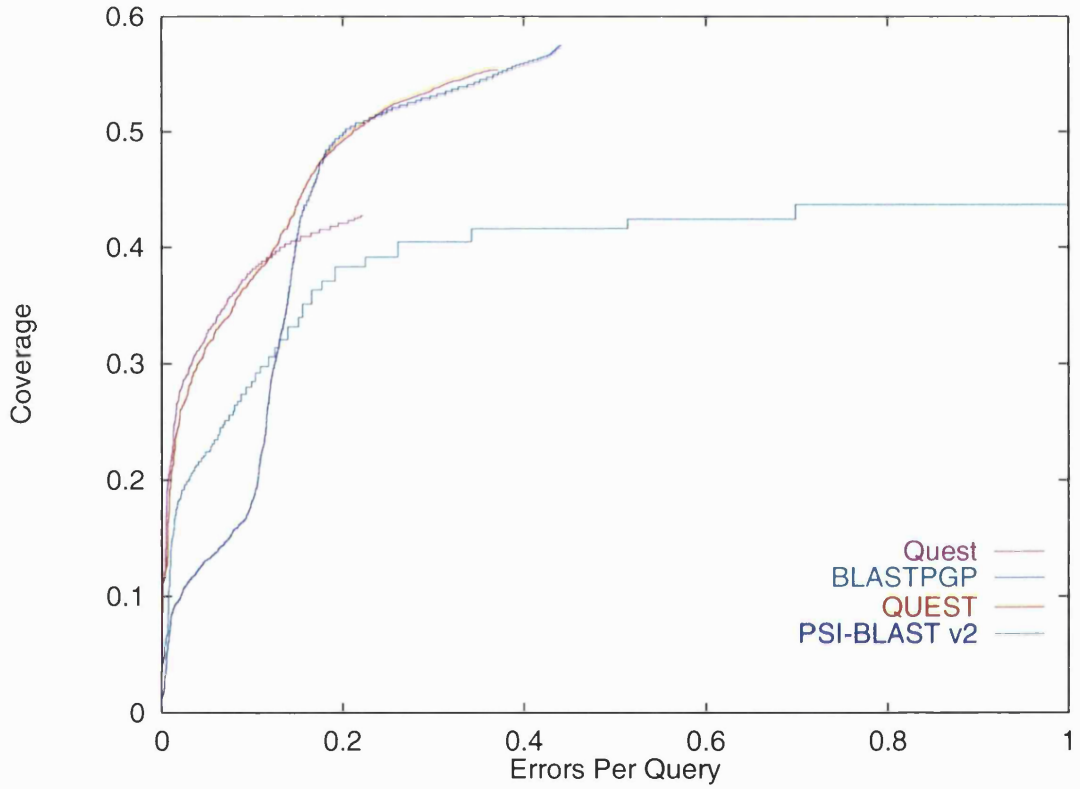


Figure 4.10: **Comparison of PSI-BLAST and QUEST to non-iterative schemes.** the PSI-BLAST (version 2) and QUEST results reported are the same as those in Figure 4.9A., the Quest and BLASTPGP results are taken from Figure 4.1.

errors per query, more than ten times less. From this point on PSI-BLAST picks up again, the gradient of the line becomes very steep and the coverage increases to 50% with very little increase in errors. The QUEST scheme in comparison closely follows the Quest results, rising steeply before gradually levelling out. The QUEST result simply continues on past this point before finally starting to level out itself at which point it meets the PSI-BLAST result coming up. For a blip to occur such as that seen in the PSI-BLAST curve means that there are a significant number of high scoring but incorrect hits in the result set. Because these are not seen in QUEST or even in a non-iterative run of BLASTPGP, this indicates that there are some intrinsic problems in the PSI-BLAST scheme.

Chapter 5

The Quest Server: Our Window on the World Wide Web

A major goal of this project was to make the tools developed easily available for the use of the research community as a whole. A large amount of the work in the furtherance of this goal went into improving the user friendliness of the Quest program. This ranged from altering the Quest program to accept and deliver sequences in a common file format, to making the command line act in a standard way, similar to most other text based Unix/Linux programs. However, a text based interface is not considered friendly by a large proportion of the more casual computer users that make up a large proportion of the scientific community. In addition some workers will not have access to the necessary machinery or computing power to run these programs effectively. In response to these problems we developed a web interface for the Quest program which has been made available at: `http://mathbio.nimr.mrc.ac.uk/quest`.

The web interface takes the format of a simple form (Figure 5.1) that is made up of six sections. Section one allows users to paste their sequences of interest into the form or specify a file on their local machine that contains the sequences. The second section asks for the user's email address so they can be notified when the job is complete. The 'Settings' section allows users to choose from specific parameters settings, including the database to search, the score cutoff and the search strictness. The 'Run mode' section gives a choice of three methods in which Quest can be run: 'Iterative' is the QUEST scheme specified in this thesis (although to save time and resources the alignment package used is CLUSTAL W). 'Cascade and Cluster' is unsurprisingly an implementation of the Cascade-and-Cluster scheme specified here. 'Single run' triggers a simple non-iterative Quest run, although when complete the user is asked whether they wish to use the results as a query for a second iteration. The fifth section, 'Output Options' allows the user to specify whether they wish to have full length sequence hits returned as results rather than just the high scoring regions. The second option specifies that the results page should use coloured output. This means that the amino acids in any sequence alignments returned in the results will be coloured according to their similarities (Taylor, 1997). This allows users to easily judge for themselves the quality of the alignment and the hits contained within it. This output is not returned by default because the colour format produces big HTML files that can take a long time to load when a slow connection is being used. The final section simply consists of the submission button labelled 'Run QUEST' and the reset button which clears the form.

When the necessary data has been entered and the submission button has been clicked

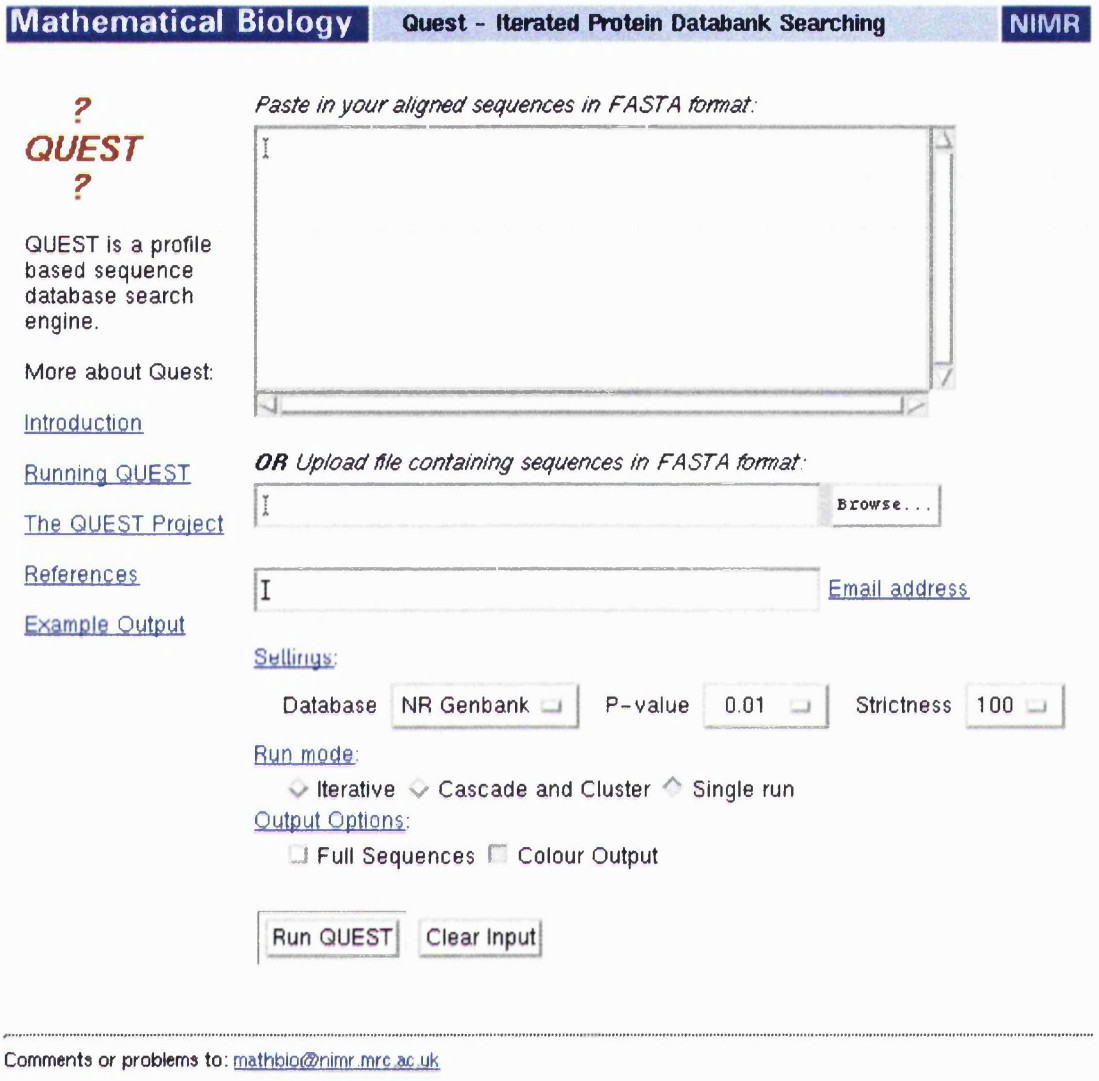
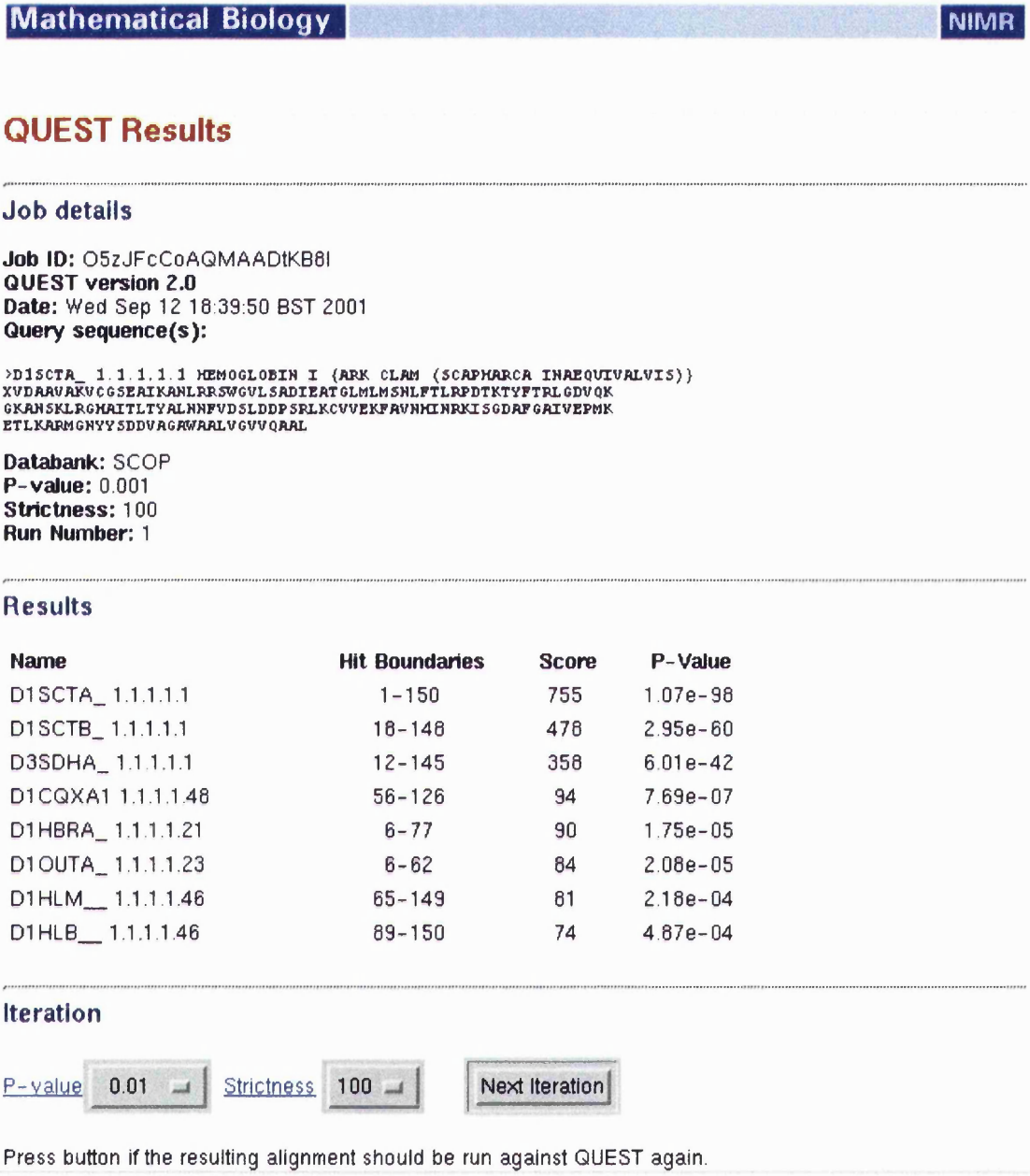


Figure 5.1: **The Quest Web Server**

on the information is sent across the web to our local server, which passes it on to the controlling script. This script simply checks which run-mode was specified and triggers one of three secondary scripts accordingly. The specified script is then sent to one of the nodes on our local Linux cluster which executes the job. When complete the results are processed into an HTML format and sent back to the web server, which publishes

them and simultaneously emails the user the information that the job is complete and where the results can be found. When the user checks their results they will find a web page similar to that shown in Figures 5.2 & 5.3. The results fall into three easy to interprete sections, the first two are shown in Figure 5.2 and the third in Figure 5.3. The first section contains the details of the job, including: the job ID number, date, query sequence(s), database searched and the Quest parameter settings used. This is all the information that would be required to re-run this search. The second section is the initial results section; it is a list of all the significant hits to the query sequence(s) found in the specified database, complete with their scores and associated P-values. The third section is shown in Figure 5.3, it contains the second half of the results; a multiple alignment of all the hits to the original query sequence(s). If the coloured output option was selected then the alignment will be coloured as shown, otherwise an identical monochromatic alignment will be presented. If a large number of hits are located, the size of the results page also becomes very large, this is especially true when coloured output is specified. For a user on a low bandwidth connection this will mean a long wait to download the file and view the results. To remedy this, if the total number of hits is larger than 20, the alignment is moved to a separate page and an addition link to it is added to this section. The other two links in this section are to non-HTML files, they allow users to simply download the hit sequences in either a plain or aligned format.

The example result shown here was generated by the 'Single run' mode of the web server. As such it actually has a fourth section not contained in the results for either the 'Iterative' or 'Cascade and Cluster' modes. This is shown at the base of Figure 5.2.



Mathematical Biology NIMR

QUEST Results

Job details

Job ID: O5zJFcCoAQMAADtKB8I
QUEST version 2.0
Date: Wed Sep 12 18:39:50 BST 2001
Query sequence(s):

```
>D1SCTA_ 1.1.1.1.1 HEMOGLOBIN I (ARK CLAM (SCAPHARCA INAEQUIVALVIS))
XVDAAVAKVCGSEAIKANLRRSWGVLSADIEATGLMLMSNLFTLRPDTKTYFTRLGDVQK
GKANSKLRGHAITLTYALNNFVDSLDDPSRLKCVVEKFAVNHINRKISGDRFGAIVEPMK
ETLKARMGNYYSDDVAGAWAALVGVVQAAL
```

Databank: SCOP
P-value: 0.001
Strictness: 100
Run Number: 1

Results

Name	Hit Boundaries	Score	P-Value
D1SCTA_ 1.1.1.1.1	1-150	755	1.07e-98
D1SCTB_ 1.1.1.1.1	18-148	478	2.95e-60
D3SDHA_ 1.1.1.1.1	12-145	358	6.01e-42
D1CQXA1 1.1.1.1.48	56-126	94	7.69e-07
D1HBRA_ 1.1.1.1.21	6-77	90	1.75e-05
D1OUTA_ 1.1.1.1.23	6-62	84	2.08e-05
D1HLM__ 1.1.1.1.46	65-149	81	2.18e-04
D1HLB__ 1.1.1.1.46	89-150	74	4.87e-04

Iteration

P-value 0.01 Strictness 100 Next Iteration

Press button if the resulting alignment should be run against QUEST again.

Figure 5.2: **An example results page: Part I.** Example results from the web server generated by a single sequence query of the SCOP database, with Quest running in 'Single run' mode. Continued in Figure 5.3

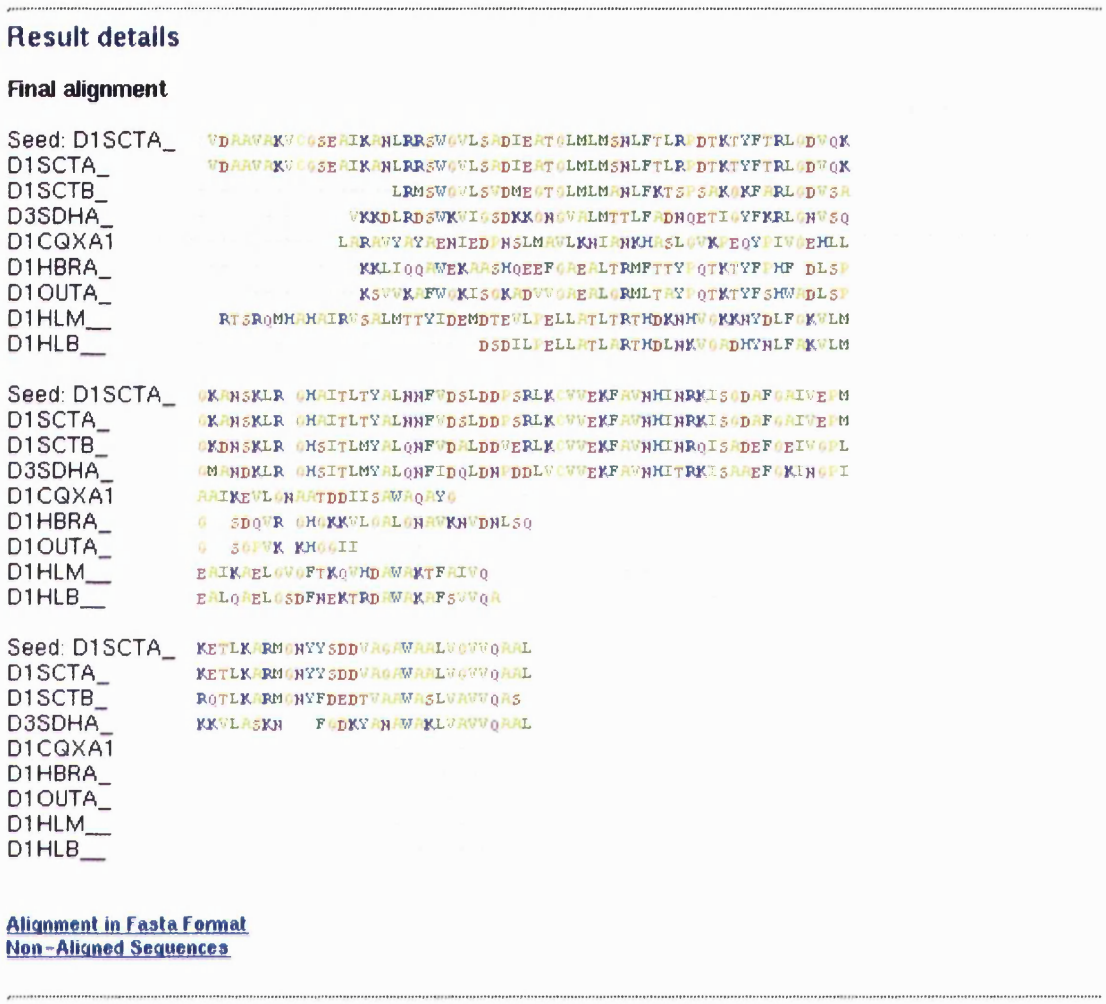


Figure 5.3: **An example results page: Part II.** Continuation of the Quest web server result started in Figure 5.2

The 'Iteration' section allows a subsequent single run of Quest to be initiated using the results of this run as a Query. This manual form of iteration can be tightly controlled by altering the strictness and the score cutoff of the subsequent searches, allowing confident users to tightly control the expansion of their search. Equally it is possible to download the sequence alignment result file and use this as a query for a new search, perhaps across a different database or using a different scheme.

The Quest web server allows anyone with an internet connection to make use of the sequence database searching methods detailed in this thesis. However, it does not provide the full functionality or controllability of the programs themselves. For users that require these abilities the programs and source code are available for download from the ftp server: `ftp://mathbio.nimr.mrc.ac.uk/pub/`.

Chapter 6

Discussion & Conclusions

The initial aims of this project were to design and produce more effective systems for searching and accessing large sequence databases. Rather than re-approaching this problem from scratch we used the Quest program and the QUEST scheme (Taylor, 1998; Taylor and Brown, 1999) as our initial starting point. We then proceeded to re-evaluate all the parts of these processes in an attempt to improve their operation in any way possible. This involved the substantial rewriting of the Quest program itself, less than 15% of the code in the present version has remained unchanged from the original implementation. Broadly speaking the drastic nature of these changes was successful. When benchmarked the Quest program (when used in an independent non-iterative context) has improved its sensitivity by ~100%, to the extent that it now capable of finding a larger number of correct sequence hits than either BLASTP or BLASTPGP. As well as improving the sensitivity we have maintained its strength of selectivity. For equal levels of coverage the Quest program also returned a lower number of errors per query than the two BLAST packages.

For the QUEST scheme the task of improvement was split into tree stages. Firstly, increasing the modularisation of the scheme, allowing a variety of different methods and programs to fulfil the requirements of the search, alignment and assessment phases. This task was completed without significant trouble. The second stage, involved the evaluation of the different possible methods to allow us to put together out 'preferred' QUEST scheme. We retained the newly improved Quest program for the search phase, but decided on the replacement of the MULTAL program for the alignment and assessment phases. The replacements chosen were T-Coffee, a relatively new but powerful alignment package and Mulfil, a purpose written (for the QUEST scheme) alignment assessment program. The third and final stage was to optimise the settings for these various programs and the automatic parameter control within the scheme itself. Whilst not fully completed, the current settings used within the QUEST scheme allowed it to outperform the most popular iterated search program, PSI-BLAST. Unfortunately this result is not quite as clear cut as that for the Quest program itself. Whilst the entire length of the QUEST benchmark curve is closer to the ideal than PSI-BLAST it does not quite reach the same level of sensitivity. PSI-BLAST eventually reached a coverage of 59% whereas QUEST managed only 56%. However, whilst the sensitivity is slightly lower the selectivity of QUEST is significantly higher, at these final coverage levels QUEST only finds 0.37 EPQ's in comparison to PSI-BLAST's 0.80. In fact even at lower coverage levels the selectivity of QUEST is noticeable, at a coverage level of 20% QUEST returns approximately ten-fold less errors than PSI-BLAST.

Part of the goals of this project were to develop methods that were rapid as well

as sensitive. Quest is a heuristic based search method and as such it is faster than a full Smith-Waterman search algorithm. However, it is significantly slower than other heuristic based methods. The reason for this is that speed of operation was not such a high priority for us as it was for the developers of the other heuristic based methods. When BLAST and FASTA were originally developed, computers were much slower and less widespread and the need to make a program run as fast as possible was much higher. Our situation has been quite different because the computing resources available to us and many others are far greater. As a result the Quest program was not as heavily optimised as perhaps it might have been, instead our focus was firmly on increasing both the sensitivity and selectivity of the system. However, when completed we still needed to address the issue of its execution speed. We approached this problem by optimising the program to make the best use of the computing resources available to us. As such we made use of the MPI libraries for the 'C' programming language to allow Quest to run in parallel across more than one processor or computer at a single time thereby drastically reducing its runtime. This acceleration of Quest was fine when it was being used as a simple database search program, when used as part of the QUEST scheme parallelisation is much less useful. The slowest step of the QUEST scheme is almost uniformly the alignment phase (except when very few sequences are involved), it is our rate limiting step. Unfortunately, it is far more difficult to parallelise a multiple alignment method than a database search procedure. So it was not possible to speed up the QUEST scheme in the same way as the Quest program itself. In effect, the only way to speed up the operation of the QUEST scheme in it's current form is to use a

faster alignment package. Trial runs with the CLUSTAL W program proved that a faster alignment package does speed up the scheme significantly, however the runtime can still not be called rapid. The Quest program has a pseudo-alignment option similar to the alignment system used in the PSI-BLAST program. When this is used the run time is very fast, but the results are extremely poor, too much information is lost through this approximate alignment system (PSI-BLAST has a better intrinsic alignment system than Quest and as a result seems to avoid this fate). Because the entire point of the QUEST scheme is to use as much information of the best quality possible, we decided to abandon any further attempts to accelerate its operation and instead merely focus on improving its effectiveness as a search tool.

The second search scheme developed during this project manages to completely avoid these problems by doing away with an alignment phase altogether. The Cascade-and-Cluster system can be thought of as an intermediate sequence search based method. Quest is used to search a database with a query. All of the significant hits are then used as individual queries themselves. The results generated by these Quest runs are then brought together and a simple clustering algorithm removes the sequence hits that it regards as incorrect. The final result is a scheme that runs relatively quickly (it is still held back somewhat by the speed of the Quest program itself), but that produces a result significantly better than PSI-BLAST and only very slightly worse than the QUEST scheme. This is a significant improvement over simpler intermediate sequence search methods that are documented as performing only 75% as well as PSI-BLAST.

Recently a new version of PSI-BLAST has been made available that provides a

greater challenge to our results. The many small enhancements that have been made to this program have improved its final performance so that it is capable of reaching a coverage of 57% for only 0.44 EPQ. This final performance is greater than that of the Cascade-and-Cluster scheme and approximately equal to that of the QUEST scheme. The new PSI-BLAST program does still have some problems, it's selectivity at the lower coverage levels still leaves a lot to be desired, upto a coverage level of 47% it still performs significantly worse than the QUEST scheme. This increases when the default parameters are used so that both QUEST and Cascade-and-Cluster equal or better PSI-BLAST. In these cases the PSI-BLAST result does eventually rise above these but this is at a cost of a large number of incorrect hits. When very low cutoff levels are used to isolate sequences that are virtually certain hits, PSI-BLAST is likely to make more errors than QUEST and Cascade-and-Cluster, in fact even Quest and BLASTP are likely to do better. As far as our methods go these results are not as bad news as they may initially seem. PSI-BLAST is now capable of equalling our best results and and doing so quicker than we can manage, but it is also a heavily developed and optimised program. Our methods in comparison are still relatively new and are still partially under construction. The next stage in the QUEST scheme development is to finalise the optimisation of the parameter settings within the scheme. The preliminary work of this goal led to a significant improvement in the results. It is difficult to believe that its finalisation will not improve them further. There is plenty of scope for improvement within the programs used in the scheme as well. The implementation of the T-Coffee based alignment filter in the Mulfil program may well improve the quality of the sequence alignment used in

each iteration and hence both the sensitivity and selectivity of the scheme overall. But it is Quest itself that offers many opportunities. Despite the aggressive rewrite of the Quest program undertaken in this project it's general mode of operation was unchanged. As such Quest shares more similarity with the much older database searching programs, e.g. BLASTP, than the newer ones, e.g. BLASTPGP. Whilst the older methods have been shown to operate very well in a simple pairwise based database search, they maybe leaving a little to be desired in an iterative scheme. Furthermore QUEST performed very badly when it was run in pseudo-alignment mode. This method of operation uses Quest's own alignment of the sequence hits as a seed for the next search round. This problem is not observed in PSI-BLAST, which suggests that Quest's alignment system and hence scoring system may not be as accurate and reliable as it could be. As well as possibly improving the accuracy of the QUEST program incorporating some of the types of changes seen in newer search programs would also accelerate it's speed of operation. A good example of this is the initial double word hit of the BLASTPGP and PSI-BLAST algorithms (Altschul et al., 1997). There are also less dramatic changes that could be made to the Quest program of which the most significant would be the overhaul of the segment assembly function. If the alignment system was left as it is, improving the assembly function is quite likely to improve the final results. The present method is a greedy algorithm that assumes the highest scoring segment to be 'correct', therefore any segments disagreeing with this are disallowed. In certain cases the highest scoring segment may well be incorrect, especially in scenarios where there are several similarly scoring but incompatible segments. The original segment assembly function actually

took into account all these various possibilities, however it was replaced with this much simpler system for good reasons. Firstly it was very slow and whilst speed of operation was not our primary concern we had to take it into consideration. But the primary reason why the original method was replaced was that it did not perform as well as the newer simpler method, it returned more errors and less correct sequences. However, this was probably a result of an error within this implementation rather than the concept itself. To sum up, the QUEST scheme and the programs within it offer significant scope for further improvement, allowing the real possibility that they could surpass rather than equal the results of the improved PSI-BLAST.

One aspect of the Quest program that probably offers the most scope for improvement is the P-value calculation. The system presently implemented is the same as that used in the earlier BLAST programs (Karlin and Altschul, 1993; Altschul and Gish, 1996) and has proven to be less reliable than the schemes used in other programs, most noticeably FASTA (Pearson, 1998; Brenner et al., 1998). The accuracy of the latest version of PSI-BLAST is noticeably greater than that of the original version. Much of this improvement is undoubtedly due to the enhanced methods used in calculating the E-values. Because Quest uses an even more primitive scheme it is quite possible that implementing one of the newer or more accurate strategies would lead to a significant improvement in performance.

In my opinion one of the most promising results from this project was that for the Cascade-and-Cluster scheme. The performance of this scheme was not quite as high as the QUEST scheme and its result was surpassed by the new version of PSI-BLAST.

However, this method may have more scope for improvement than either of the other two. With no optimisation whatsoever the first run of this scheme almost equalled the best QUEST scheme result and surpassed the original PSI-BLAST. This scheme uses no form of multiple sequence alignment and only a very simple form of iteration, yet it's results are equal to these profile based schemes and are far beyond those of simpler mechanisms, be they non-iterative runs of Quest, BLAST, FASTA or simple intermediate search schemes (Park et al., 1998). A simple parameter change for the secondary Quest runs has shown to lead to an increased number of correct hits being located. It is quite possible that a proper tuning of the parameters could achieve much more. Beyond this, there are a number of further possibilities for improvement. The reason the optimum running parameters for this scheme have not yet been determined is that this scheme was only developed during the last few weeks of this project. Equally because of this time restriction a number of other shortcuts were made. The number of search levels was limited to two; the initial search and the secondary searches using the primary results. The reason given for this earlier was to prevent the search cascade from spiralling out of control. This is a valid reason to put a limit on the number of search levels, but it does not mean that two is the right number for that limit. Three search levels may result in a much higher coverage level without necessarily increasing the error rate too significantly. Similarly the program put together to perform the clustering procedure was written from scratch and no attempt to optimise its performance was made. The scoring system it uses to cluster the sequences is quite arbitrary and testing different values for direct and indirect links may give a better final result. It could also be possible to take into account

the strength of the hits i.e. their P-values, when evaluating their linkage to one another. Put simply, the Cascade-and-Cluster method presented here was designed ad hoc as a test of concept, which happened to perform very well. However, it is unlikely that it managed to produce the optimal results on the first attempt, which is what the results discussed here truly represent.

The Cascade-and-Cluster method does have its problems to overcome. To date the method has only been tested on the small benchmarking database. On a large dataset such as the Genbank database it would come against several problems. Firstly there would be the problem of redundancy. It is pointless to cascade a search with identical or nearly identical sequences to the original query because they will just give identical results, which would probably overwhelm the clustering procedure. As a result the cascade searches would have to be triggered by sequences that are significantly different enough from the original query. The method worked well with the benchmarking database which had no sequences that were more than 70% identical. Consequently this may be a good cut off level for the triggering of the cascade. It is important to note that this cutoff should not just apply to the original query sequence. Sequences that are deemed too similar to any of the previous sequences (including but not solely the original query) used in the search should be excluded from triggering another search themselves. The reasoning behind this is the same as for the query sequence itself. A large number of extremely similar sequences will override the scoring mechanism in the clustering algorithm and may lead to incorrect sequences being accepted as correct. This sequence exclusion would also probably largely overcome the second problem with the Cascade-and-Cluster scheme on

large datasets and that is the time that would be required to complete the search. In a small database this is not a problem because the number of searches in the cascade will generally be low due to the small size of the sequence clusters. In a larger database the clusters are likely to be of equally larger sizes. The exclusion of redundant sequences is likely to contract the size of these sequence clusters significantly, but the number of searches required is still likely to be larger. There are two ways to overcome this time problem, either the number of sequence searches is further reduced, perhaps by searching with only 50% of the possible hits, or secondly by using a faster search program. It has already been stated that Quest was not designed for speed, which is not so for many of the other search programs. In a simple comparison BLASTP has proven to perform much faster and only slightly less well than Quest, FASTA in turn has been documented as being slightly slower but more accurate than BLASTP. Trials with these alternative programs may lead to a significant acceleration of operation without much if any loss in accuracy.

If time had been available, there are numerous additional pieces of work that could have been done on the benchmarking of the methods presented here. Of these one of the most interesting would have been to repeat the benchmark run of both QUEST and PSI-BLAST, this time using a larger database for example the non-redundant Genbank database for at least the initial iteration, if not all of them. The purpose of this is to build up a diverse profile by grabbing as many similar sequences as possible from the larger dataset and then using this to search the smaller test set for the remainder of the iterations. This should allow both QUEST and PSI-BLAST to play to their strengths;

both programs were designed to make use of profiles to find more distant sequences. The more data they are given to start with and the larger the profile they have the more successful the search should be in locating distant homologues. Indeed this was the method used in the previous PSI-BLAST benchmarks (Park et al., 1998). The reason this approach was not used here was as I have already stated so that I could directly compare programs that were not able to use intermediate sequences with those that could. However it is possible that because the dataset used was so small it may not have contained enough information to build up useful profiles. The result of which being that the profile based methods would not have had any appreciable advantage over the traditional pair-wise methods. The results presented in the benchmarking chapter do indicate that this may have been the case.

To sum up, the primary goals of this project have been fulfilled. We have managed to develop both rapid and sensitive methods for searching sequence databases, using both single and multiple sequence queries. The Quest program itself has been much enhanced, performing up to twice as well as the original implementation and significantly better than several of its main competitors. The performance of the QUEST scheme is also much enhanced. Through the use of iteration, sequence alignment and hit assessment we have shown it possible to combine both high levels of sensitivity and selectivity in a single search scheme. The results of the QUEST scheme are now equal to if not better than it's nearest competitor PSI-BLAST. During the duration of the project a second simpler search scheme refered to as Cascade-and-Cluster was developed. This scheme is loosely based on the theory of intermediate sequence searching and unlike QUEST and

PSI-BLAST, it does not rely on multiple sequence alignment to expand the range of its search. As such it operates much faster than the QUEST scheme whilst producing results of an almost equal quality. Both of these schemes and the programs that comprise them offer much room for further development, indicating that more improvement should be possible, allowing the construction of both faster and more accurate searching tools in the near future.

Bibliography

Agarwal, P. and States, D. (1998). Comparative accuracy of methods for protein sequence similarity search. *Bioinformatics*, 14(1):40–47.

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

Altschul, S. and Gish, W. (1996). Local alignment statistics. In Doolittle, R., editor, *Computer Methods for Macromolecular Sequence Analysis*, volume 266 of *Methods in Enzymology*, chapter 27, pages 460–480. Academic Press, Inc., 24-28 Oval Road, London NW1 7DX.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–10.

Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–402.

Bairoch, A. and Apweiler, R. (2000). The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research*, 28:45–48.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. (2000). The Protein Data Bank. *Nucleic Acids Research*, 28:235–242.

Bleasby, A., Akrigg, D., and Attwood, T. (1994). OWL - a non-redundant composite protein sequence database. *Nucleic Acids Research*, 22:3574–3577.

Brenner, S., Chothia, C., and Hubbard, T. (1998). Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences of the United States of America*, 95:6073–6078.

Brenner, S., P., K., and M., L. (2000). The astral compendium for protein structure and sequence analysis. *Nucleic Acids Research*, 28(1):254–256.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978). A model of evolutionary change in proteins. matrices for detecting distant relationships. In Dayhoff, M., editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–358. National Academic Research Foundation, Washington DC.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU,UK.

Eddy, S. (1998). Profile hidden Markov models. *Bioinformatics*, 14(9):755–763.

Fraser, C., Gocayne, J., White, O., Adams, M., Clayton, R., Fleischmann, R., Bult,

C., Kerlavage, A., Sutton, G., Kelley, J., Fritchman, J., Weidman, J., Small, K., Sandusky, M., Fuhrmann, J., Nguyen, D., Utterback, T., Saudek, D., Phillips, C., Merrick, J., Tomb, J., Dougherty, B., Bott, K., Hu, P., and Lucier, T. (1995). The minimal gene complement of *Mycoplasma genitalium*. *Science*, 270:397–404.

Gerstein, M. (1998). Measurement of the effectiveness of transitive sequence comparison, through a third 'intermediate' sequence. *Bioinformatics*, 14(8):707–14.

Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264(4):823–838.

Gribskov, M., McLachlan, A., and Eisenberg, D. (1987). Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 84(13):4355–4358.

Henikoff, S. and Henikoff, J. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89:10915–10919.

Henikoff, S. and Henikoff, J. (1993). Performance evaluation of amino acid substitution matrices. *Proteins*, 17(1):49–61.

Henikoff, S. and Henikoff, J. (1994). Position-based sequence weights. *Journal of Molecular Biology*, 243(4):574–8.

Heringa, J. (1999). Two strategies for sequence comparison: profile-preprocessed and

secondary structure-induced multiple alignment. *Computers & Chemistry*, 23(3-4):341–64.

Higgins, D., Bleasby, A., and Fuchs, R. (1992). CLUSTAL V: improved software for multiple sequence alignment. *CABIOS*, 8:189–191.

Higgins, D. and Sharp, P. (1988). CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244.

Hogeweg, P. and Hesper, B. (1984). The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of Molecular Evolution*, 20:175–186.

Huang, X. and Miller, W. (1991). A time-efficient linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12:337–357.

International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature*, 409:860–921.

Johnson, M. and Doolittle, R. (1986). A method for the simultaneous alignment of three or more amino acid sequences. *Journal of Molecular Evolution*, 23:267–278.

Jones, D., Taylor, W., and Thornton, J. (1992). The rapid generation of mutation matrices from protein sequences. *CABIOS*, 8:275–282.

Karlin, S. and Altschul, S. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences of the United States of America*, 87:2264–2268.

Karlin, S. and Altschul, S. (1993). Applications and statistics for multiple high-scoring segments in molecular sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 90(12):5873–5877.

Karplus, K., Barrett, C., and Hughey, R. (1998). Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856.

Kimura, M. (1983). *The Neutral Theory of Molecular Evolution.* Cambridge University Press, Cambridge.

King, J. and Jukes, T. (1969). Non-Darwinian evolution. *Science*, 164:788–798.

Krause, A., Stoye, J., and Vingron, M. (2000). The SYSTERS protein sequence cluster set. *Nucleic Acids Research*, 28(1):270–2.

Krause, A. and Vingron, M. (1998). A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14(5):430–438.

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

Mehta, P., Heringa, J., and Argos, P. (1995). A simple and fast approach to prediction of protein secondary structure from multiply aligned sequences with accuracy above 70%. *Protein Science*, 4:2517–2525.

Morgenstern, B., Dress, A., and Werner, T. (1996). Multiple DNA and protein sequence

alignment based on segment-to-segment comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 93(22):12098–12103.

Müller, A., MacCallum, R., and Sternberg, M. (1999). Benchmarking PSI-BLAST in genome annotation. *Journal of Molecular Biology*, 293(5):1257–71.

Murzin, A., Brenner, S., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–40.

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

Neuwald, A., Liu, J., Lipman, D., and Lawrence, C. (1997). Extracting protein alignment models from the sequence database. *Nucleic Acids Research*, 25(9):1665–77.

Notredame, C. and Higgins, D. (1996). SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524.

Notredame, C., Higgins, D., and Heringa, J. (2000). T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217.

Notredame, C., Holm, L., and Higgins, D. (1998). COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422.

Orengo, C., Michie, A., Jones, S., Jones, D., Swindells, M., and Thornton, J. (1997). CATH - A hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108.

Pacheco, P. (1997). *Parallel programming with MPI.* Morgan Kaufmann Publishers, Inc., San Francisco, California, USA.

Park, J., Karplus, K., Barrett, C., Hughey, R., Haussler, D., Hubbard, T., and Chothia, C. (1998). Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284:1201–1210.

Park, J., Teichmann, S., Hubbard, T., and Chothia, C. (1997). Intermediate sequences increase the detection of homology between sequences. *Journal of Molecular Biology*, 273:349–354.

Pearl, F., Todd, A., Bray, J., Martin, A., Salamov, A., Suwa, M., Swindells, M., Thornton, J., and Orengo, C. (2000). Using the CATH domain database to assign structures and functions to the genome sequences. *Biochemical Society Transactions*, 28(2):269–75.

Pearson, W. (1990). Rapid and sensitive sequence comparison with fastp and fasta. *Methods in Enzymology*, 183:63–98.

Pearson, W. (1998). Empirical statistical estimates for sequence similarity searches. *Journal of Molecular Biology*, 276(1):71–84.

Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 85(8):2444–2448.

Rognes, T. and Seeberg, E. (2000). Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706.

Saitou, N. and Nei, M. (1987). The neighbour-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425.

Salamov, A., Suwa, M., Orengo, C., and Swindells, M. (1999). Combining sensitive database searches with multiple intermediates to detect distant homologues. *Protein Engineering*, 12(2):95–100.

Schaffer, A., Aravind, L., Madden, T., Shavirin, S., Spouge, J., Wolf, Y., Koonin, E., and Altschul, S. (2001). Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Research*, 29(4):2994–3005.

Sellers, P. (1974). On the theory and computation of evolutionary distances. *SIAM journal of Applied Mathematics*, 26:787–793.

Shpaer, E., Robinson, M., Yee, D., Candlin, J., Mines, R., and Hunkapiller, T. (1996). Sensitivity and selectivity in protein similarity searches: A comparison of smith-waterman in hardware to BLAST and FASTA. *Genomics*, 38:179–191.

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

Sneath, P. and Sokal, R. (1973). *Numerical Taxonomy.* W.H.Freeman, San Francisco.

Stein, L., Sternberg, P., Durbin, R., Thierry-Mieg, J., and Spieth, J. (2001). Wormbase: network access to the genome and biology of caenorhabditis elegans. *Nucleic Acids Research*, 29(1):82–86.

Taylor, W. (1987). Multiple sequence alignment by a pairwise algorithm. *CABIOS*, 3:81–87.

Taylor, W. (1988). A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28:161–169.

Taylor, W. (1997). Residual colours: a proposal for aminochromography. *Protein Engineering*, 10(7):743–746.

Taylor, W. (1998). Dynamic sequence databank searching with templates and multiple alignment. *Journal of Molecular Biology*, 280(3):375–406.

Taylor, W. and Brown, N. (1999). Iterated sequence databank search methods. *Computers & Chemistry*, 23(3-4):365–85.

The Arabidopsis Genome Initiative (2000). Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana. Nature*, 408:796–815.

The C.elegans Sequencing Consortium (1998). Genome sequence of the nematode *Caenorhabditis elegans.* a platform for investigating biology. *Sci-*

ence, 282:2012–2018. The full list of authors can be found at http://www.sanger.ac.uk/Projects/C_elegans/Science98/.

The FlyBase Consortium (1999). The FlyBase database of the Drosophila genome projects and community literature. *Nucleic Acids Research*, 27(1):85–88.

Thompson, J., Gibson, T., Plewniak, F., Jeanmougin, F., and Higgins, D. (1997). The CLUSTAL X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Research*, 25(24):4876–4882.

Thompson, J., Higgins, D., and Gibson, T. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4690.

Thompson, J., Plewniak, F., and Poch, O. (1999a). BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–8.

Thompson, J., Plewniak, F., and Poch, O. (1999b). A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–90.

Wallqvist, A., Fukunishi, Y., Murphy, L., Fadel, A., and Levy, R. (2000). Iterative sequence/secondary structure search for protein homologs: comparison with amino acid sequence alignments and application to fold recognition in genome databases. *Bioinformatics*, 16(11):988–1002.