

The **McGraw·Hill** Companies

Bioinformatics

Principles and Applications



Author's Profile

Harshawardhan P. Bal is currently a management and strategy consultant at Booz Allen Hamilton Inc., Rockville, MD. He has more than a decade of experience in biotechnology and bioinformatics, both in academia and in the industry. Through his research, he has made direct contributions to the annotation of the important food crop—rice—and in mining the human genome to identify novel target proteins for new drug development.

At Millennium Pharmaceuticals Inc., Cambridge, MA, Harshawardhan acquired considerable experience in design and deployment of enterprise-wide knowledge management systems in the pharma industry.

Harshawardhan has a Master's degree in pharmaceutical sciences and was a formulation scientist in the pharma industry in Mumbai. He has a Ph.D. in molecular biology from the National Institute of Immunology, New Delhi. He pursued research on HIV/AIDS and gene therapy at the University of Rochester Medical Center, Rochester, NY and moved on to Cold Spring Harbor Laboratory, Cold Spring Harbor, NY. At Cold Spring, he worked on whole genome sequencing projects and received training from experts such as Prof. W. Richard McCombie, Dr. Andy Baxevanis, Dr. William Pearson, Dr. Randall Smith, and Dr. Stephen Altschul.

Harshawardhan is the author of several peer-reviewed publications in scientific journals and a book entitled *Perl Programming for Bioinformatics*.

The **McGraw-Hill** Companies

Bioinformatics

Principles and Applications

Harshawardhan P. Bal

*Management and Strategy Consultant
Booz Allen Hamilton Inc.,
Rockville, MD*



Tata McGraw-Hill Publishing Company Limited
NEW DELHI

McGraw-Hill Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto



Tata McGraw-Hill



Copyright © 2005, by Tata McGraw-Hill Publishing Company Limited.

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
Tata McGraw-Hill Publishing Company Limited.

ISBN 0-07-058320-X

Published by the Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008, typeset in PalmSprings at The Composers,
260, C.A. Apt., Pashchim Vihar, New Delhi 110 063 and printed at SDR Printers,
A-28, West Jyoti Nagar, Shahdara, Loni Road, Delhi 110 094

RZZCRDRIRYZLA

To
My parents, wife, and son





Preface



Modern science has been transformed in recent times. Our thinking, our ways of analyses, our tools, our experimental systems, and certainly our powers to probe living systems have fundamentally altered in ways that we never imagined. Bioinformatics is that one field of science which has admirably demonstrated what integration and knowledge sharing across different disciplines can achieve to advance our understanding of complex living systems.



This book is about those fundamental tools and devices that spearheaded swift changes, which revolutionized biomedical research and enabled us to perform biology *in silico*. Today, as a result of these tools (and despite their limitations), discovering novel coding regions, genes and gene products in a haystack of unknown sequences, searching for remote homologies between sequences, etc. are but routine tasks that biologists with little or no background in computer science can perform effortlessly at the flick of a button. The volume, the unstructured or the heterogeneous nature of data, is no longer a bottleneck to scientific research. Instead, scientists can now focus on the more important and fundamental questions of the molecular basis of disease, and find new cures for hitherto untreatable ailments.

Part I of the book focuses on a core set of tools that have become indispensable to scientific discoveries. Part II of the book focuses on how these tools can be integrated with BioPerl modules programmatically, to enable them in an enhanced—bioinformatics on steroids—manner.

The first book in the series, *Perl Programming for Bioinformatics*, introduced Unix and Perl programming for bioinformatics analysis. The intent of this

book is to supplement it with the knowledge of bioinformatics tools and BioPerl. Both books have been written with a grassroots approach based on real-life experiences from high throughput genome sequencing centers and the pharma industry. It is hoped that the two books will facilitate the transition a biologist needs to make into the intriguing and fast-paced world of bioinformatics.

Thank you and happy reading!

HARSHAWARDHAN P. BAL





Acknowledgements



My first words of appreciation are for those clairvoyant thought leaders who brought together modern biology, medicine, mathematics, and information technology, and laid the foundations for the advent of the new sciences of genomics and bioinformatics.



My transition from molecular biology to bioinformatics was an exciting and intellectually rewarding experience, and indeed, provided me with new ways to put my basic research skills to understanding genome research, complex disease pathology, and drug discovery. I would like to thank the many teachers who made this possible. Among these are my mentors, Neilay Dedhia and W. Richard McCombie, and my colleagues at the Lita Annenberg Hazen Genome Sequencing Center at the Cold Spring Harbor Laboratory, New York, who helped me make this transition.



I also thank my mentor Brian Osborne at OSI Pharmaceuticals, Melville, New York, who first gave me the opportunity to utilize my combination of molecular biology and bioinformatics knowledge to new target and drug discovery.

No experience in bioinformatics can be complete without an understanding of modern day software design and development techniques, and I thank my supervisor, Jeffrey Moore, for providing this at Millennium Pharmaceuticals, Inc., Cambridge, MA. It was also at Millennium that I applied Knowledge Management to large scale integration of heterogeneous data sets emanating from diverse sources in a typical pharma environment such as high throughput genome sequencing, genome annotation, target validation, transcriptional profiling, pathway analysis and proteomics, etc.

I also want to thank Wayne Marasco at the Division of Cancer Immunology and AIDS, at the Dana-Farber Cancer Institute, an NCI designated Comprehensive cancer center, and Harvard Medical School teaching affiliate, Boston, Massachusetts, for enabling me to come full circle and lead a full scale development effort in discovery research of Adult T-cell Leukemia and HIV/AIDS.

Finally, I would like to thank the readers of my first book for encouraging me with their enthusiasm and their faith in me—I hope this second book proves as enjoyable and useful as the first.

Of course, nothing would have been possible without the dedicated efforts of the Tata McGraw-Hill team who guided me through the entire publication process and kept me motivated to keep turning the pages till the book was complete.

HARSHAWARDHAN P. BAL





Contents

Preface

vii

Acknowledgements

ix

PART ONE: PRINCIPLES

MK	1. Web-based Sequence Analysis: BLAST I	3	MK
+	1.1 Basic Local Alignment Search Tool (BLAST)	3	+
	1.2 The Purpose of BLAST	3	
	1.3 Terminology	5	
	1.4 BLAST Analysis	9	
	1.5 BLAST 2	13	
	1.6 Automated Alignments with Perl	17	
	References	22	
	2. Web-based Sequence Analysis: BLAST II	23	
	2.1 Basic Local Alignment Search Tool (BLAST)	23	
	2.2 Scoring Matrices	23	
	2.3 PAM or Per cent Accepted Mutation Matrices	24	
	2.4 BLOSUM (Blocks Substitution Matrices)	25	
	2.5 The Relationship between BLOSUM and PAM Substitution Matrices	26	
	2.6 Working of the BLAST Algorithm	26	
	2.7 A Practical BLASTN Exercise	28	
	2.8 Explanation of the BLAST Output	31	
	2.9 Advanced BLASTN	35	
	2.10 Biological Analysis of BLASTN: Cystic Fibrosis	40	
	2.11 Automating BLAST Analyses with Perl	41	

3. Web-based Sequence Analysis: BLAST III	44
3.1 Standalone BLAST	44
3.2 Configuring blastall	49
3.3 Downloading Databases from NCBI	49
3.4 Formatting NCBI's Databases	51
3.5 Running blastall	55
3.6 Downloading Pre-formatted Databases	57
3.7 fastacmd	62
3.8 bl2seq	63
3.9 Performing Local BLAST Searches with Perl	64
3.10 Sequence Annotation	65
4. Web-based Sequence Analysis: Gene Prediction	69
4.1 Introduction	69
4.2 Terminology and Concepts	70
4.3 Gene Prediction Programs	73
4.4 GenScan	75
4.5 Running GenScan Analyses	77
4.6 Analyzing GenScan Output	78
4.7 GenScan Analysis with LWP::UserAgent	84
5. Web-based Sequence Analysis: HMMER	89
5.1 Introduction	89
5.2 Downloading HMMER	89
5.3 Why use HMMER?	92
5.4 Running HMMER Commands	94
5.5 HMMER: A Practical Example	95
5.6 HMMER Utilities	104
References	108
6. PSI-BLAST	109
6.1 Introduction	109
6.2 PSI-BLAST and Protein Analysis	109
6.3 When is PSI-BLAST better than BLASTP?	110
6.4 The Design of PSI-BLAST	110
6.5 Advantages of PSI-BLAST	111
6.6 Limitations of PSI-BLAST	112
6.7 Example of a PSI-BLAST Search	113

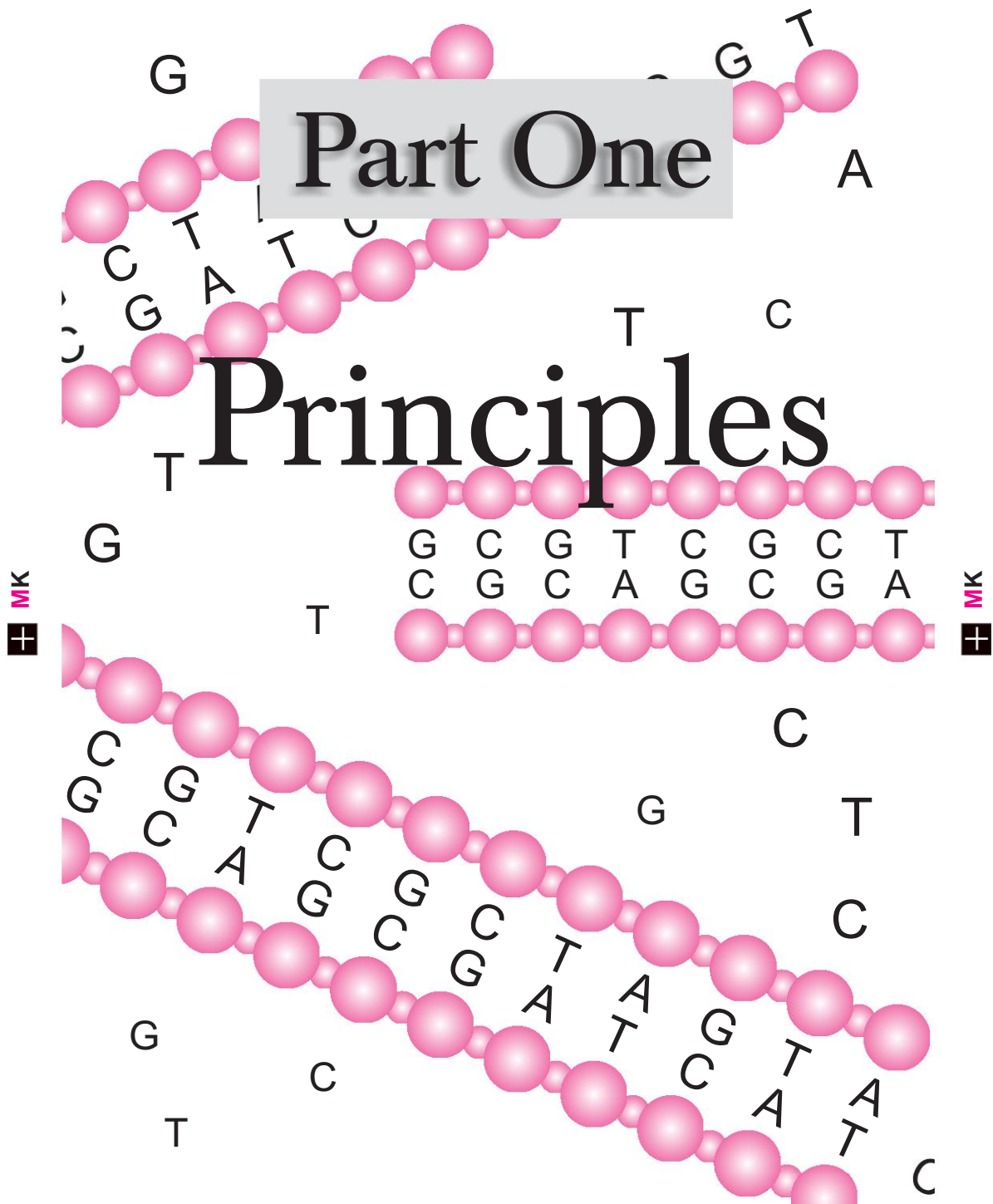


PART TWO: APPLICATIONS

7. Accessing Sequence Information Using BioPerl	127
7.1 BioPerl Installation	127
7.2 BioPerl Modules	133
7.3 Object Oriented Programming	136
7.4 Using BioPerl	138
7.5 The write_seq() Function	142
Appendix I: Installing External Modules	147
Appendix II: Upgrading BioPerl	147
Appendix III: Testing for Availability of Individual Modules	148
8. Bio::DB::GenBank	149
8.1 Introduction	149
8.2 Structure of a GenBank Record	150
8.3 The Bio::DB::GenBank Module	154
9. Accessing GenBank Data	161
9.1 Introduction	161
9.2 GenBank Tags	161
9.3 Extracting Tags and their Values	164
9.4 Sample Scripts	169
10. BioPerl BLAST Modules	175
10.1 Introduction	175
10.2 BLAST Programs	175
10.3 BLAST 2	177
10.4 Perl Modules for BLAST2	178
10.5 Using BioPerl for BLAST2	181
10.6 Standalone BLAST	183
10.7 Configuring blastall	185
10.8 Bio::Tools::Run::StandAloneBlast	187
10.9 Performing BLAST Searches	188
10.10 Formatting NCBI's Databases	189
10.11 Running blastall	190
10.12 Running BLAST with Bio::Tools::Run::StandAloneBlast	191
11. Parsing BLAST Output	194
11.1 Generating a Raw BLAST Report	194
11.2 The Bio::Tools::Blast Module	196
11.3 Parsing the HPR BLAST Report	203
11.4 Specifying a Filter Function	207
11.5 Formatting Parsing Results into a Table or HTML	209

Part One

Principles





Web-based Sequence Analysis: BLAST I



1.1 BASIC LOCAL ALIGNMENT SEARCH TOOL (BLAST)

BLAST is a database search tool, developed and maintained primarily by the National Center for Biotechnology Information (NCBI). The web-based tool is available at <http://www.ncbi.nlm.nih.gov/BLAST/>. The BLAST suite of programs has been designed to find high scoring local alignments between sequences, without compromising the speed of such searches. BLAST uses a heuristic algorithm which seeks local as opposed to global alignments and is, therefore, able to detect relationships among sequences which share only isolated regions of similarity (Altschul et al., 1990). The first version of BLAST was released in 1990 and allowed users to perform ungapped searches only. The second version of BLAST, released in 1997, allowed gapped searches.



1.2 THE PURPOSE OF BLAST

It is not uncommon nowadays, especially with the large number of genomes



being sequenced, that a researcher comes across a novel DNA or protein sequence for which no functional information is available. Some basic information on the sequence is necessary before a molecular biologist can even take the new sequence into the lab and perform meaningful experiments with it. It would, for example, make the job much easier if it were known that the new sequence encodes a Repetitive DNA Element (which would need an entirely different rationale and set of tools for analysis), a metabolic enzyme or, indeed, a protein that is a putative member of a known superfamily such as immunoglobulins, kinases, etc.

Note

The term protein superfamily was introduced by Margaret O. Dayhoff in 1974. The term was originally defined as a group of evolutionarily related proteins; it has also been used to refer to a group of structurally or functionally related proteins not necessarily of common evolutionary origin.



This is where database searching comes handy. Database searching, in general and with BLAST in particular, is mainly used to reveal the similarity between a test sequence (called 'query sequence') that a user wants to find more information about and other sequences (called 'target' sequences) in a biological database, which may be similar to the query sequence. This is the basis on which the whole premise of biological sequence analysis is built. Database searching is, therefore, one of the very first tools that a biologist uses to analyze a given sequence.



Database searches reveal sequences that have some degree of similarity to the query sequence. These sequences from the database are commonly referred to as 'hits'. Once such hits are found, users can draw inferences from the similarity about homology and molecular function. A thumb rule for drawing inferences is that two sequences that share more than 50 per cent sequence identity are usually similar in structure and function. Under such conditions, the major sequence features of the two sequences can be easily aligned and identified. If there is only a 25 per cent sequence identity, there will be some structure homology, although in such situations, the domain correspondence between the two proteins may not be easily apparent. It is also generally accepted that sequences that are important for function are generally conserved. We will illustrate this with some examples.

An example where a database search resulted in an important discovery was the finding reported by Doolittle et al. (1983) of the similarity between the oncogene, *v-sis*, of Simian sarcoma virus (an RNA tumor virus) and the gene encoding human platelet-derived growth factor (PDGF). The *v-sis* gene was the first oncogene to be identified with homology to a known cellular gene. This discovery provided an early insight into the critical role that growth factor signaling plays in the process of malignant transformation.

Another example of the value of database searching was the discovery that the defective gene that causes cystic fibrosis formed a protein that had similarity to a family of proteins that were involved in the transport of hydrophilic molecules across the cytoplasmic membrane. Cystic fibrosis is the most common inherited disease in the Caucasian population and affects the respiratory, digestive and reproductive systems. It is now known that mutations in the cystic fibrosis gene lead to loss of chloride transport across the cell membrane, which is the underlying cause of the disease.



1.3 TERMINOLOGY



Before we proceed with a detailed description of the BLAST algorithm and how it is used, it is important to understand a few terms that are used frequently during such analyses.

Identity: When two sequences are compared to each other, identity indicates the extent to which the two sequences have the exact same composition (i.e., nucleotide base or amino acid residue) at equivalent positions, usually expressed as a percentage.

Similarity: When two genes or proteins are compared with each other, similarity indicates the level of relatedness between the two on the basis of their primary sequences. For DNA sequences, this is the number of identical bases at equivalent positions, usually expressed as a percentage.

Note

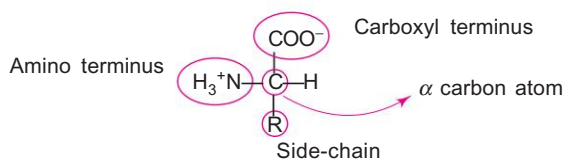
Depending on the ring structure of the bases, DNA is composed of two types of nucleotide bases: Purines—Adenine (A) and Guanine (G) are two-ring bases, and Pyrimidines—Cytosine (C) and Thymine (T) are single-ring bases.

Two closely related sequences such as the human pancreatic ribonuclease (HPR) gene and the bovine pancreatic ribonuclease (BPR) gene share a high degree of similarity when aligned with each other. The figure below shows the alignment of the first 15 codons of the two enzymes:

HPR: AAG-GAA-TCC-CGG-GCC-AAG-AAA-TTC-CAG-CGG-CAG-CAT-ATG-GAC-TCA
 ||| ||| -|- --- ||- --- ||- ||- -|| ||| ||| ||| -||| ||| ||| -
 BPR: AAG-GAA-ACT-GCA-GCA-GCC-AAG-TTT-GAG-CGG-CAG-CAC-ATG-GAC-TCC

The vertical bars and dashes indicate an exact match and a mismatch respectively. The similarity between the two sequences is fairly evident from the alignment.

Similarity in the context of protein sequences also means the number of amino acid residues that are identical at equivalent positions, usually expressed as a percentage. However, there is an additional parameter to consider when comparing proteins—the nature of the amino acid residues themselves. Remember that there are 20 amino acids and that each amino acid is a small chemical entity composed of a common backbone of an organic carboxylic acid (—COOH) and an amino group (—NH_2) attached to a saturated carbon atom:



Amino acids are divided into classes based on their chemical and functional properties. For example, both asparagine (Asn, single amino acid symbol: N) and glutamine (Gln, single amino acid symbol: Q) have uncharged polar side-chains, and differ only in the presence of an additional methyl group in glutamine. Both glycine (Gly, single amino acid symbol: G) and alanine (Ala, single amino acid symbol: A) are small and nonpolar amino acids. Refer to the single letter codes provided in Table A.1 in the Appendix.

A simple chemical classification of amino acids is as follows:

Based on the nature of side-chains:

- Aliphatic amino acids G, A, V, L, I, P
- Aromatic amino acids F, Y, W

- | | |
|---------------------------------|---------------|
| • Polar amino acids | S, T, N, Q |
| • Sulfur containing amino acids | C, M |
| • Charged amino acids | D, E, H, K, R |

Note

Aliphatic means that the protein side-chain is composed of only carbon or hydrogen atoms. Aromatic means that the side-chains contain an aromatic ring system. Polar amino acids have side-chains that are hydrophilic (i.e., water-loving).

Based on hydrophilicity:

- | | |
|--------------------------------------------|------------------|
| • Amino acids with hydrophilic side-chains | N, G, Q, R, H, K |
| • Amino acids with hydrophobic side-chains | V, I, L, M, P |

(The other amino acids have intermediate hydrophilicities.)

Based on charge:

- | | |
|----------------------|------|
| • Positively charged | K, R |
| • Negatively charged | D, E |

Note

Amino acids that are hydrophobic in nature are usually buried in the interior of the protein. Hydrophilic amino acids are more accessible on the surface of proteins to interact with solvent molecules and take part in electrostatic interactions with positively charged basic amino acids. Aspartate and glutamate can also take on catalytic roles in the active sites of enzymes and are well known for their metal ion binding abilities. A Venn diagram that summarizes these different ways of classification is shown in Figure 1.1.

Coming back to the problem of similarity in proteins, consider the alignment of HPR and BPR protein sequences:

```

HPR:  KESRAKKFQRQHMDSDSSPSSSTYCNQMMRRRNMTQGRCKPVNTFVHEPLVDVQNVCFQEK
      | |+ - | - | | + | | | | | - + | - + | | | | | + - | | + | - | | | | | | | - | - | | - | | + -
BPR:  KETAAAKFERQHMDSSTAASSSNYCNQMMKSRNLTKDRCKPVNTFVHESLADVQAVCSQKN
  
```

HPR: VTCKNGQGNCYKSNSSMHITDCRLTNGSRYPNCAYRTSPKERHIIIVACEGSPYVPVHFDASV
 | - | | | | - | | | + - | + | - | | | | - | - - | + | | | | | + | + - - - + | | | | | | + | | | | | | | |
 BPR: VACKNGQTNCYQSYSTMSITDCRETGSSKYPNCAYKTTQANKHIIIVACEGNPYVPVHFDASV
 HPR: EDST
 BPR: - - - -

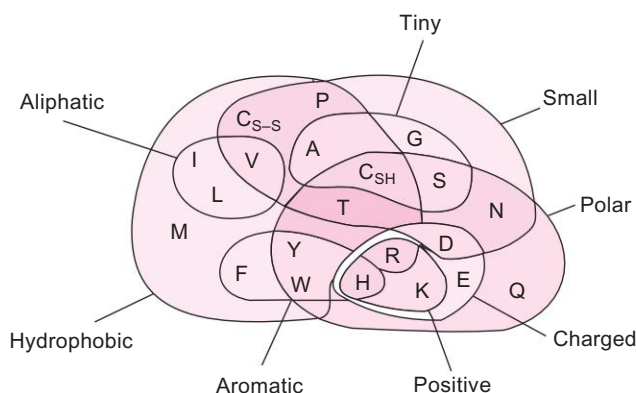


Fig. 1.1 Venn diagram: classification of amino acids

The vertical bars and dashes, as before, represent matches and mismatches respectively. The '+' sign indicates a conservative replacement: a substitution by an amino acid with similar properties, for example, serine (S) with threonine (T), arginine (R) with lysine (K), methionine (M) with leucine (L), etc. In such cases, similarity can be classified as identities (exact same residues at the equivalent positions) and positives (conservative changes at equivalent positions).

Homologs: Two sequences are said to be homologous if they are evolutionarily related. Orthologs and paralogs are two types of homologous sequences.

Orthologs: These are two genes in different species that derive from a common ancestor. They are derived as a result of vertical descent and typically have the same domain architecture. For example, mammalian α -hemoglobin and avian α -hemoglobin are orthologs. Orthologous genes may or may not have the same function.

Paralogs: These are two genes within a single species that diverged by gene duplication. (derived from *para* = *in parallel*). Paralogs are thus produced by gene duplication and subsequent divergence within an organismal lineage such as the individual members of a gene family.



1.4 BLAST ANALYSIS

To use BLAST, you need to select:

1. an input query sequence (this can be a nucleotide or protein)
2. the database to search against (this can be a nucleotide or protein database)
3. a database search program (any of the five available with BLAST)

The five search programs and their applications are as follows:

Table 1.1 *BLAST programs*

Program	Query sequence of type	Database of type	Comparison	Application
BlastN	DNA	DNA	DNA ↔ DNA. Compares a nucleotide query sequence against a nucleotide sequence database.	Find DNA sequences that match the query.
BlastP	Protein	Protein	Protein ↔ protein. Compares an amino acid query sequence against a protein sequence database.	Find identical (homologous) proteins.
BlastX	DNA	Protein	Protein ↔ protein. Compares a nucleotide query sequence translated in all reading frames against a protein sequence database.	Find what protein the query sequence codes for.
TBlastN	Protein	DNA	Protein ↔ protein. Compares a protein query sequence against a nucleotide sequence database dynamically translated in all reading frames.	Find genes in unknown DNA sequences.

(Contd.)

Table 1.1 (Contd.)

TBlastX	DNA	DNA	Protein ↔ protein. Compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.	Discover gene structure. (Find degree of homology between the coding region of the query sequence and known genes in the database.)
---------	-----	-----	----------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------

The query sequence can be a gene or a fragment of a gene that you have discovered experimentally or computationally, a piece of unfinished genomic DNA, a peptide sequence, or a repeat element (DNA).

The database can be any of the DNA or protein sequence databases supported by NCBI. These include both DNA and nucleotide sequences databases and several databases representing whole genomes of organisms. Some examples are described in Table 1.2.

Table 1.2 *BLASTable databases at NCBI*

nr*	non-redundant protein and nucleotide database of all sequences, excluding ESTs, STSs, GSSs and Phase 0, 1 or 2 HTG sequences.
est	expressed sequence tags. This database is available in three separate databases of human only, mouse only, and all non-human, non-mouse ESTs (called est_human, est_mouse, est_others respectively).
gss	genomic survey sequences, includes single-pass genomic data, exon-trapped** sequences, and Alu PCR sequences.
htgs	unfinished high throughput genomic sequences***: phases 0, 1 and 2.
pat	protein sequences derived from the patent division of GenBank.
yeast	<i>Saccharomyces cerevisiae</i> genome and protein sequences.
mito	database of mitochondrial sequences.
month	all new or revised nucleotide or protein sequences added to nr in the last 30 days (includes GenBank CDS translation + data from PDB, SwissProt, PIR and PRF).
pdb	sequences derived from the three-dimensional structure from Brookhaven Protein Data Bank.
dbsts	database of sequence tagged sites from GenBank+EMBL+DDBJ. STSs are short (~ 200–500 bp) genomic landmark sequences that are unique in a genome and, therefore, can be specifically detected in the presence of all other genomic sequences, and define a specific anchor position on a physical map.

(Contd.)

Table 1.2 (Contd.)

yeast	yeast (<i>Saccharomyces cerevisiae</i>) genomic CDS (coding sequence) translations.
ecoli	<i>Escherichia coli</i> genomic CDS translations.
drosophila	drosophila genome proteins provided by Celera and Berkeley Drosophila Genome Project (BDGP).

Notes:

*nr stands for “non-redundant”, which means that two or more sequences that are exactly identical in length and sequence composition (that is, amino acid or nucleotide base pair) at every position are considered the same and merged into one entry.

**Exon trapping uses splice acceptor sites as identifiers of candidate exons within cloned mammalian genomic DNA sequences and is a technique that allows for the rapid identification and cloning of coding regions from cloned eukaryotic genomic DNA.

***The HTG division of GenBank contains ‘unfinished’ DNA sequences generated by the high-throughput sequencing centers. These are generally first pass sequence data generated from a single cosmid, BAC, YAC, or P1 clone and together may comprise more than 2 kb of sequences with one or more gaps.

Phase 0 HTG sequences are usually one-to-few pass reads of a single clone, and so are not usually contigs.

Phase 1 HTG are unfinished sequences and may be unordered, unoriented contigs, with gaps.

Phase 2 HTG sequences are unfinished, ordered, oriented contigs, with or without gaps.

Phase 3 sequences are finished sequences with no gaps (with or without annotations).

Phase 3 HTG sequences are in nr.

Some specific terms relating to BLAST analysis are as follows:

Affine gap costs: A scoring system for gaps within alignments that charges a penalty for the existence of a gap and an additional per-residue penalty proportional to the gap’s length.

Alignment score: A numerical value that describes the overall quality of an alignment. Higher numbers correspond to higher similarity.

Bit score: A scaled version of an alignment’s raw score that accounts for the statistical properties of the scoring system used.

E value or Expectation value: The number of distinct alignments, with score equivalent to or better than the one of interest, that are expected to occur in a database search purely by chance. The lower the E value, the more signifi-

cant the score. During a BLAST search, hits with an E value less than 0.0001 are generally considered homologous to the query sequence. When a large number of hits are found, hits with E values significantly lower (that is, more closely related) than the other hits are most likely to be orthologs.

Gap: Within an alignment of two sequences, several adjacent null characters in one sequence aligned with adjacent letters in the other.

Gap score: The score assigned to a gap. A high penalty is used to initiate or open a gap and a lower penalty is used to extend a gap. These penalties are called gap opening and extension cost respectively.

Gapped alignment: An alignment in which gaps are permitted. A gapped alignment is an indication of an insertion or a deletion in one of the two sequences since their divergence. These are also referred to as indels.

Global alignment: The alignment of two complete nucleic acid or protein sequences over their entire length. In global alignments, typically, gaps are added whenever sequences do not match at identical positions. This provides a better indication of structures of the sequences being compared.

Local alignment: The alignment of segments from two nucleic acid or protein sequences. Local alignments highlight areas of sequence conservation and are ideal to locate motifs within sequences that may be important structurally or functionally.

Heuristics: A term in computer science that refers to 'guesses' made by a program to obtain approximately accurate results. Typically, these are used to increase the speed of a program at the cost of potentially yielding suboptimal results. BLAST uses heuristics based on knowledge of how sequences evolve.

High scoring pair (HSP): An HSP consists of two sequence fragments of arbitrary but equal length whose alignment is locally maximal and for which the alignment score meets or exceeds a threshold or cutoff score. Each HSP consists of a segment from the query sequence and one from a database sequence.

Substitution Matrices: In aligning two sequences, the method used to score the alignment of one residue against another is based on the use of substitution matrices. The choice of the scoring matrix is the most critical parameter in sequence comparison. The default matrix for BLASTP is BLOSUM62, developed by Henikoff & Henikoff (1992). Alternative choices include: PAM40,

PAM120, PAM250, etc. No alternate scoring matrices are available for BLASTN.

Maximal-scoring Segment Pair (MSP): This is defined by two sequences and a scoring system and is the highest scoring of all possible segment pairs that can be produced from the two sequences.

1.5 BLAST2

We will start with a simple BLAST exercise where we will calculate the level of similarity between two protein sequences. For this exercise, we will use

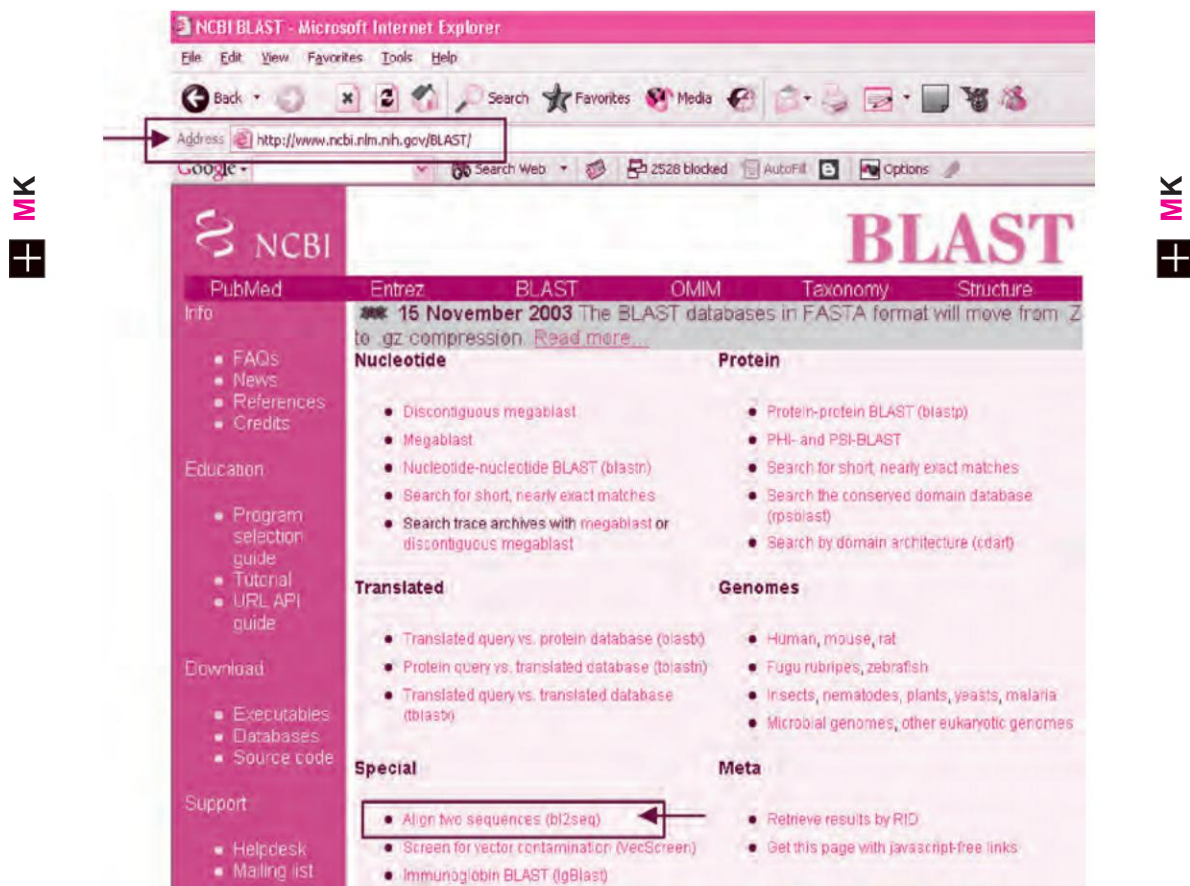


Fig. 1.2 The NCBI BLAST website

the Pair-wise BLAST tool at NCBI (<http://www.ncbi.nlm.nih.gov/BLAST/>). See Figure 1.2.

The screenshot shows the 'BLAST 2 SEQUENCES' web page in a Microsoft Internet Explorer browser. The page title is 'BLAST 2 SEQUENCES'. The address bar shows the URL: <http://www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html>.

The page content includes a description of the tool, a reference, and a form for sequence alignment. The form has several fields and buttons:

- Program:** A dropdown menu set to 'blastn'. A callout box labeled '1: Choose BLAST program & Substitution Matrix' points to this field.
- Matrix:** A dropdown menu set to 'NotApplicable'.
- Parameters used in BLASTN program only:**
 - Reward for a match:** A text box containing '1'.
 - Penalty for a mismatch:** A text box containing '-2'.
 - Use Mega BLAST:** A checkbox that is unchecked.
 - Strand option:** A dropdown menu set to 'Both strands'.
- Open gap:** A text box containing '5'.
- and extension gap:** A text box containing '2'.
- penalties:** A text box containing 'expect'.
- gap x_dropoff:** A text box containing '50'.
- word size:** A text box containing '11'.
- 2: Choose E value:** A callout box pointing to the 'expect' field.
- Sequence 1:** A text box for 'Enter accession or GI' or 'download from file'. A 'Browse...' button is next to it. A callout box labeled '3: Paste/download/Specify GI number' points to this field.
- Sequence 2:** A text box for 'Enter accession or GI' or 'download from file'. A 'Browse...' button is next to it.

Fig. 1.3 BLAST2

Navigate to the site and select the BLAST2 sequences link that will take you to the entry page for the tool: <http://www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html> (Figure 1.3).

The most important parameters to consider here are the BLAST program (box 1), the matrix (box 1), and the expect value (box 2). There are three choices to enter the two sequences for comparison (box 3)—pasting the se-

quences in the boxes, uploading from a local file or simply by specifying their GI numbers.

In the following examples, we will use the GI numbers for HPR and BPR protein sequences:

>GI:1350818

kesrakkfqrqhmdsdsspsststycnqmmrrrnmqtgrckpvntfvheplvdvqnvfcqekvtc
kngqgncyksnssmhitdcrltngsrypncayrtsperhiivacegspypvvhfdasvedst

>GI:133198

ketaaakferqhmdsstsaasssnycnqmmksrnltdrckpvntfvhesladvqavcsqknvac
kngqtncyqsytsitdcrtgsskypncaykttqankhiivacegpnypvvhfdasv

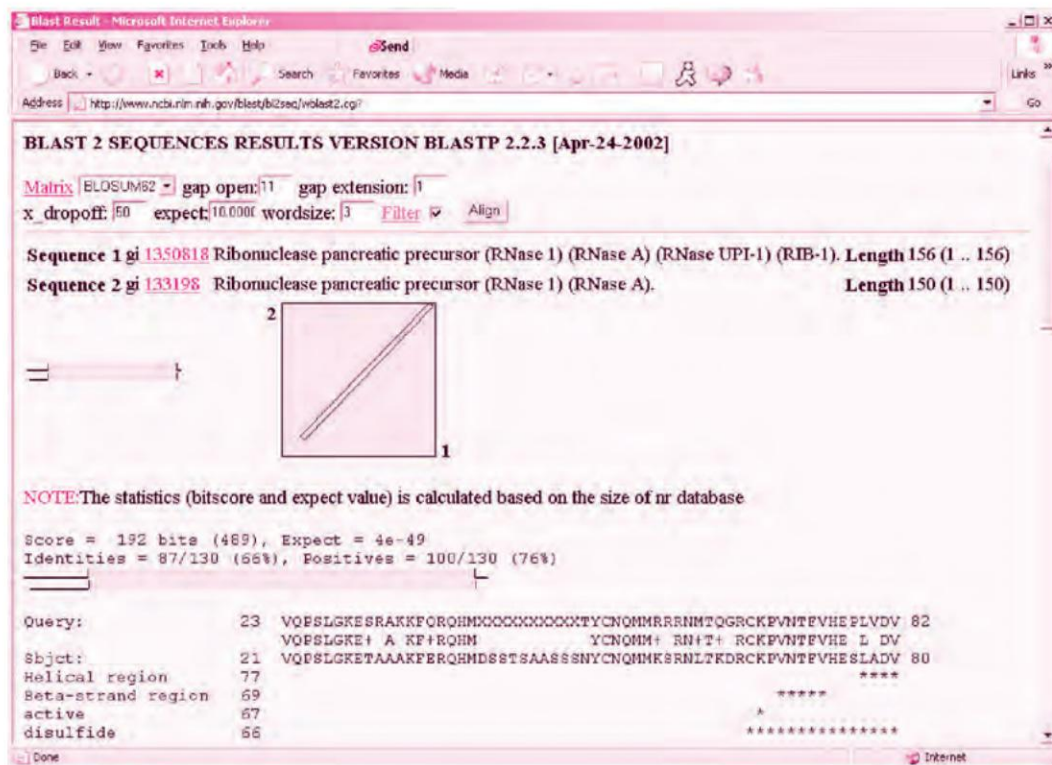


Fig. 1.4 BLAST2 output

```

Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 200
Number of Sequences: 0
Number of extensions: 8
Number of successful extensions: 1
Number of sequences better than 10.0: 1
Number of HSP's better than 10.0 without gapping: 1
Number of HSP's successfully gapped in prelim test: 0
Number of HSP's that attempted gapping in prelim test: 0
Number of HSP's gapped (non-prelim): 1
length of query: 156
length of database: 315,760,149
effective HSP length: 106
effective length of query: 50
effective length of database: 101,205,231
effective search space: 5060261550
effective search space used: 5060261550
T: 9
A: 40
X1: 16 ( 7.4 bits)
X2: 129 (49.7 bits)
X3: 129 (49.7 bits)
S1: 41 (21.8 bits)
S2: 64 (29.3 bits)

```

Fig. 1.5 BLAST2 output

Since these are protein sequences, we will use the BLASTP program which compares a protein sequence against another protein sequence. We will use the E value of 10 and leave the other parameters as their default values.

The output of the alignment is shown in Figures 1.4 and 1.5.

The output provides the following information about the analysis:

1. Values of parameters such as matrix (BLOSUM62), E value (10), penalties for gap opening (11) and extension (1), etc.
2. GI numbers of the sequences used for the analysis.
3. Hyperlinks for the sequences to the actual GenBank records.

4. A graphic of the alignment (the two blue bars on the left).
5. A plot of the alignment indicating the region of maximum alignment.
6. The score, E value, identities and positives for the alignment.
7. The actual alignment, and information on structural elements found—helices, beta-sheets, disulfide bonds, etc.

The bottom of the output provides information on the length of query sequence (156), the length of database (315,760,149), etc.



1.6 AUTOMATED ALIGNMENTS WITH PERL

In this section, we will learn how to use of the LWP::Simple module to generate automatic alignments of two protein sequences using the NCBI BLAST2 server.



Remember that the LWP is a collection of Perl modules, which provides a simple and consistent application programming interface to the World-Wide Web. We have used the LWP::Simple module earlier. We will now extend it to do sequence alignments.



The most important aspect of automating alignments using LWP is the URL that needs to be specified within the program. This is done through the URLAPI—a standardized application program interface (API) to access the NCBI BLAST web server. This system uses direct HTTP-encoded requests to the BLAST2 cgi-bin program at <http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi>.

Since these requests are performed directly over the web, users do not need to download on their local computers the BLAST2 program or the sequences they want to analyze. The URL can be directly used to specify all the values and parameters that need to be plugged into the appropriate places on the web form. For example, the name of the BLAST program, the GI numbers of the sequences, the E values and so on. If you look at the source of the BLAST2 page (Figures 1.6 and 1.7), this URL can be ascertained from the name-value pairs that are used to store information about the various parameters.

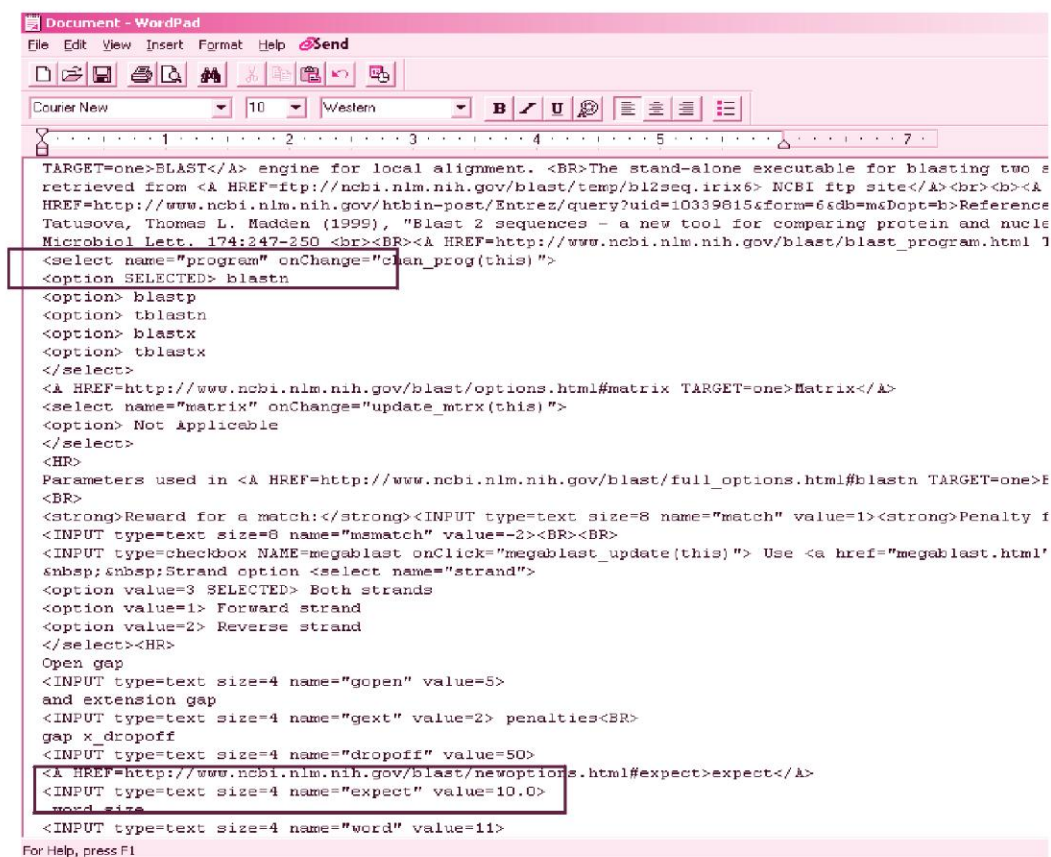


Fig. 1.6 BLAST2 parameters I

For a protein-protein BLAST2, the various parameters we are interested in are as follows:

Name of program	value = Blastp
Name of matrix	value = BLOSUM62
E value	value = 10
First sequence	name = one
Second sequence	name = two
Action (Command)	name = submit

The only true variables here are the two sequences (name = one and name = two) themselves. The rest of the parameters have fixed values. (For ex-

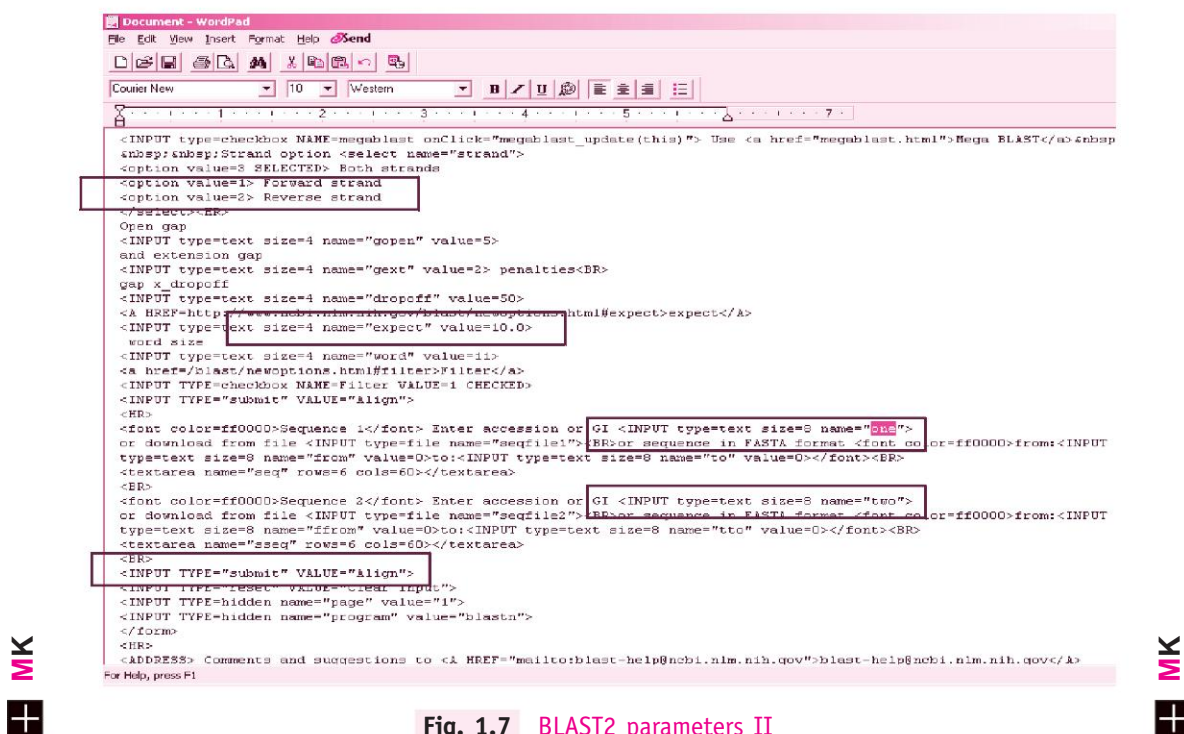


Fig. 1.7 BLAST2 parameters II

ample, the name of the program can be either BLASTN or BLASTP, the name of the matrix can be BLOSUM62, PAM30, etc.) Note that these matrices apply only to a protein-protein BLAST2. This information needs to be specified in a string and appended to the wblast2.cgi script that runs on the BLAST2 server. The general form of the URL is shown below:

<http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?parameters>

where, as mentioned earlier,

<http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi>.

is the URL for the BLAST2 service itself.

Parameters are a list of values that we want to feed to the URL. The question mark after the URL indicates the start of parameters, which are simply name-value pairs. Since there are multiple name-value pairs, the pa-

rameters are specified in the form of a string where the individual name-value pairs are separated by an ampersand ('&') sign.

To perform a BLAST2 analysis on protein sequences identified by GI numbers 1350818 (human pancreatic ribonuclease) and 133198 (bovine pancreatic ribonuclease) with an E value of 10, the program BLASTP and the matrix BLOSUM62, the parameter string can be constructed as shown below:

```
expect=10&program=blastp&matrix=BLOSUM62&one=1350818&two=133198
&Action=submit;
```

The last name-value pair "Action=submit" simply sends the information to the BLAST2 server for analysis.

The complete URL is

```
http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?expect=10&program=
blastp&matrix=BLOSUM62&one=1350818&two=133198&Action=submit
```



A more generic URL would be:

```
http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?expect=10&program=
blastp&matrix=BLOSUM62&one=$hprid&two=$bprid&Action=submit
```



where \$hprid and \$bprid are the GenBank IDs for HPR and BPR respectively.

The basic script using LWP::Simple is:

```
#!/usr/bin/perl

$/=undef;

use LWP::Simple;

$url = "http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?expect=$eva
lue&program=$program&matrix=$matrix&one=$hprid&two=$bprid&Action=submit";

$page = get($url);

print "$page";
```

The result of the analysis is shown in Figure 1.4.

Assignments

1. Read the papers (Altschul et al., 1997 and Henikoff and Henikoff, 1992) and answer the following questions:
 - (a) Describe the salient features of the BLAST algorithm. How does the ungapped version differ from the gapped version of BLAST?
 - (b) What is the rationale for scoring matrices? Describe the work of Henikoff and Henikoff on the development of matrices.
2. Write a script that generates a pair-wise alignment between a set of sequences in a multiple Fasta file and parses the output for the E values, identities and positives. The script should be run as follows:

```
blast2.pl -p blastp -e 10 -m BLOSUM62 -f filename
```

where

```
[ -p Program name (any of the five BLAST programs) ]
```

```
[ -e Expect value ]
```

```
[ -m substitution matrix, example, BLOSUM62, PAM30, PAM70 etc. ]
```

```
[ -f any multiple Fasta file containing protein sequences ]
```

and the output should be two tables:

Table 1

Alignment scores

Program used: Blastp

Matrix used: BLOSUM62/other

E value used: 10/other

ID1	ID2	Score	(bits)	Expect	Identities	Positives	Gaps
20560806	20542587	176	2e-42	108/285 (37%)	155/285 (53%)	6	0
285				(21%)			

...

...

Table 2: Protein sequence data

ID	Name	Length
20560806	similar to Kinesin-like protein KIF1C [H. sapiens]	1007
20542587	similar to kinesin-like protein GAKIN [H. sapiens]	1118
...		
...		

If no similarity is found, this should be stated as zero identities, zero positives, etc.

Appendix I

Amino acids and their three and single letter codes

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

References

- Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. (1997) *Nucleic Acids Research*, 25(17): 3389–3402.
- S. Henikoff, J. G. Henikoff (1992) Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*. 89(22): 10915–10919.



Web-based Sequence Analysis: BLAST II



2.1 BASIC LOCAL ALIGNMENT SEARCH TOOL (BLAST)

In the last chapter, we learnt how to use pair-wise BLAST to search for relationships between a large number of proteins. We used default parameters such as the BLOSUM62 substitution matrix, open gap (5) and extension gap (2) penalties, word size (11), etc. to run the program. We will now learn about these different variables and how they can be manipulated to alter the search. Finally, to illustrate the concepts we learned, we will take up a practical example with a nucleotide-protein and a protein-protein search using BLASTN and BLASTP.

A few definitions and concepts are in order before we proceed to study the details of the BLAST algorithm.



2.2 SCORING MATRICES

Detection of similarities between protein and DNA sequences are largely based

on score-based methods. In protein sequence comparisons, substitution scores based on models of amino acid conservation and properties are used. These scores describe the likelihood that an amino acid residue at a certain position was replaced or substituted by another amino acid by an evolutionary event. Stated simply, substitutions between residues that are identical or relatively similar to one another or substitutions that are observed frequently receive positive scores while substitutions between residues that are not similar or substitutions that are not frequently observed receive negative scores. Note that some amino acid substitutions are more tolerable than others due to similarity in their physicochemical properties. For example, as we saw in the previous chapter, the amino acids lysine (K) and arginine (R) are positively charged and the substitution of one with the other in the human and bovine pancreatic ribonuclease sequences was considered a conservative replacement (marked + in the alignment).

This is the basic function of scoring matrices—assigning scores to align any possible pair of residues from the sequences being compared. In doing so, substitution scores perform an important role—they provide a measure of the ‘trueness’ of a match, i.e., a measure of the probability that a match has not occurred by chance alone.

Scoring matrices are especially important when the relatedness of protein sequences that are distant in evolution is being studied. The use of better amino acid substitution weights contained in scoring matrices substantially improves the performance of such queries. Two commonly used matrices are the PAM series and the BLOSUM series. The features of these matrices are described below.



2.3 PAM OR PER CENT ACCEPTED MUTATION MATRICES

- Developed by Margaret Dayhoff (1978).
- Derived from global alignments of sequences that are at least 85 per cent identical.
- Is based on the observed rate of mutation during the predicted evolutionary changes in a smaller number of protein families.
- Uses a rough measure of how many generations of evolution it would take to mutate one sequence into another.

- PAM matrices are identified by numbers and the general notation is PAM(N). The number 'N' provides a measure of evolutionary distance between the proteins being compared. A bigger number indicates greater evolutionary (mutational) distance. For example, the PAM1 matrix is calculated from comparisons of sequences that have diverged only 1 per cent from each other.
- Matrices such as PAM40, PAM100, PAM250, etc. indicate greater evolutionary distances and are derived by extrapolation from those for lesser ones.
- PAM matrices are most sensitive for alignments of sequences with evolutionary related homologs.



2.4 BLOSUM (BLOCKS SUBSTITUTION MATRICES)

- Developed by Jorja Henikoff and Steven Henikoff (1992).
- They are derived from local, ungapped alignments of distantly related sequences
- All BLOSUM matrices are based on observed alignments; they are not extrapolated from comparisons of closely related proteins.
- They are based on the concept of 'blocks'—amino acid patterns derived from ungapped multiple alignments corresponding to the most conserved regions of a protein. These highly conserved sequences serve as protein signatures uniquely identifying them as a distinct protein family.
- They are derived from the observed amino acid substitutions in a large set of approximately 2000 such conserved patterns representing over 500 groups of related proteins.
- As with the PAM matrices, BLOSUM matrices are also identified by numbers. The number after the matrix (e.g., BLOSUM62) refers to the minimum per cent identity of the blocks used to construct the matrix; greater numbers mean lesser distances.
- BLOSUM62 is a matrix calculated from comparisons of sequences with no less than 62 per cent divergence.



2.5 THE RELATIONSHIP BETWEEN BLOSUM AND PAM SUBSTITUTION MATRICES

Remember that while the BLOSUM matrices are derived from alignments of distantly related sequences, the PAM matrices are derived from alignments of sequences that are closely related. Stated simply, the 'N' in BLOSUM is a measure of distance whereas the 'N' in PAM is a measure of closeness. The two matrices, therefore, are inversely related.

For closely related sequences, the matrices that would be used are BLOSUM(high N) and PAM(low N). Conversely, for distantly related proteins, BLOSUM(low N) and PAM(high N) matrices would be used. Though it is tailored for comparisons of moderately distant proteins (that is, for detecting weak protein similarities), BLOSUM62 performs well in detecting closer relationships. For long and weak alignments, the BLOSUM45 matrix may prove superior. The BLOSUM series of matrices generally perform better than PAM matrices for local similarity searches. Compared to the corresponding PAM60 matrix, the BLOSUM62 matrix was found to detect more distant relationships in a BLAST search. The BLOSUM62 matrix is, therefore, highly recommended for sequence alignment and database searching. It is the default matrix for BLASTP, BLASTX, TBLASTN and TBLASTX searches. The BLOSUM series, on the other hand, does not perform well for short queries, so the older PAM matrices may be used for such searches.



2.6 WORKING OF THE BLAST ALGORITHM

This is how the BLAST algorithm works:

1. Break the query sequence into words. The word size is typically three for peptides and 11 for nucleotides.
2. Select words that score above a threshold value when compared to words from the query sequence. These words serve as seed sequences.
3. Scan database for matches to seeds.
4. Extend all matches in both directions to seek high-scoring segment pairs.
5. Terminate extension when score falls below best score.
6. Return segment pairs scoring at least S (the raw alignment score), calculated from the scoring matrix and search parameters. The raw

score S is computed by adding the substitution and gap scores. BLAST also returns a bit score S' , which represent bit scores that have been normalized with respect to the scoring system, so they can be used to compare alignment scores from different searches. The BLAST Expectation (E) value is the number of alignments with an equal or better score that are estimated to occur by chance.

The threshold value in step 2 above determines the sensitivity of the BLAST search. If a small value for the threshold is chosen, the number of words that qualify as seeds is greater and BLAST has to expend more time searching for each of them in the database. However, since a larger subset of words is used, the search is more rigorous and sensitive. Conversely, if the threshold is set high, there are fewer words to be searched and the search is much faster, although this means that a lower sensitivity is obtained.

Table 2.1 *Peptide sequence databases*

nr:	all non-redundant GenBank CDS translations+PDB+SwissProt+PIR+PRF
month:	all new or revised GenBank CDS translation+PDB+SwissProt+PIR+PRF released in the last 30 days
swissprot:	last major release of the SWISS-PROT protein sequence database
yeast:	yeast (<i>Saccharomyces cerevisiae</i>) genomic CDS translations
E. coli:	<i>Escherichia coli</i> genomic CDS translations
mito:	mitochondrial sequences
pdb:	sequences derived from the three-dimensional structure from Brookhaven Protein Data Bank
patents:	protein sequences derived from the patent division of GenPept

Table 2.2 *Nucleotide sequence databases*

nr:	all GenBank+EMBL+DDBJ+PDB sequences (but no EST, STS, GSS, or phase 0, 1 or 2 HTGS sequences). No longer "non-redundant".
est:	database of GenBank+EMBL+DDBJ sequences from EST divisions
est_human:	human subset of GenBank+EMBL+DDBJ sequences from EST divisions
est_mouse:	mouse subset of GenBank+EMBL+DDBJ sequences from EST divisions
est_others:	non-mouse, non-human sequences of GenBank+EMBL+DDBJ sequences from EST divisions



2.7 A PRACTICAL BLASTN EXERCISE

We will now use an actual example to understand BLAST and its applications. Consider the case where routine sequencing of the human genome unearthed the DNA sequence shown below:

```
gctggatcca ctggagcagg caagacttca cttctaattg tgattatggg agaactggag
ccttcagagg gtaaaattaa gcacagtggg agaatttcat tctgtttctca gttttcctgg
attatgcctg gcaccattaa agaaaatata atctttgggtg tttcctatga tgaatataga
tacagaagcg tcataaaagc atgccaacta gaagaggaca tctccaagtt tgagagagaa
gacaatatag ttcttggaga aggtggaatc aactgagtg gaggtcaacg agcaagaatt
tctttagcaa gagcagtata caaagatgct gatttgtatt tattagactc tccttttggg
tacctagatg tttaacaga aaaagaaata ttgaaagct gtgtctgtaa actgatggct
```



You would like to find out what this sequence is, whether it codes for a protein and if so, what is its function. The first thing that you can do is a BLASTN (nucleotide-nucleotide) search. To do this, open the BLAST page at NCBI: <http://www.ncbi.nlm.nih.gov/blast/> and navigate to the BLASTN page (Figures 2.1 and 2.2).



To perform a BLASTN analysis, paste the sequence into the box called 'Search' (Fig. 2.2). The search form provides a number of options that you can choose from to tailor the search as per your requirements. To perform a basic BLASTN search:

- Set subsequence: Limit matches to a sub-string within the query sequence. This is useful if you want to analyze only a portion of the sequence that, for example, you know beforehand codes for a protein domain that you are interested in. We want to analyze the entire sequence and, therefore, will leave this option blank.
- Choose database: Select a database to search the query sequences against. This can be any of the databases mentioned in Tables 1.1 and 1.2. For our purposes, we will choose the nr database.

As seen in Figure 2.3, the sequence pasted in the search box need not be formatted and may contain gaps or numbers. BLAST will also remove any bad characters present in the sequence (Figure 2.4) before running the analysis.

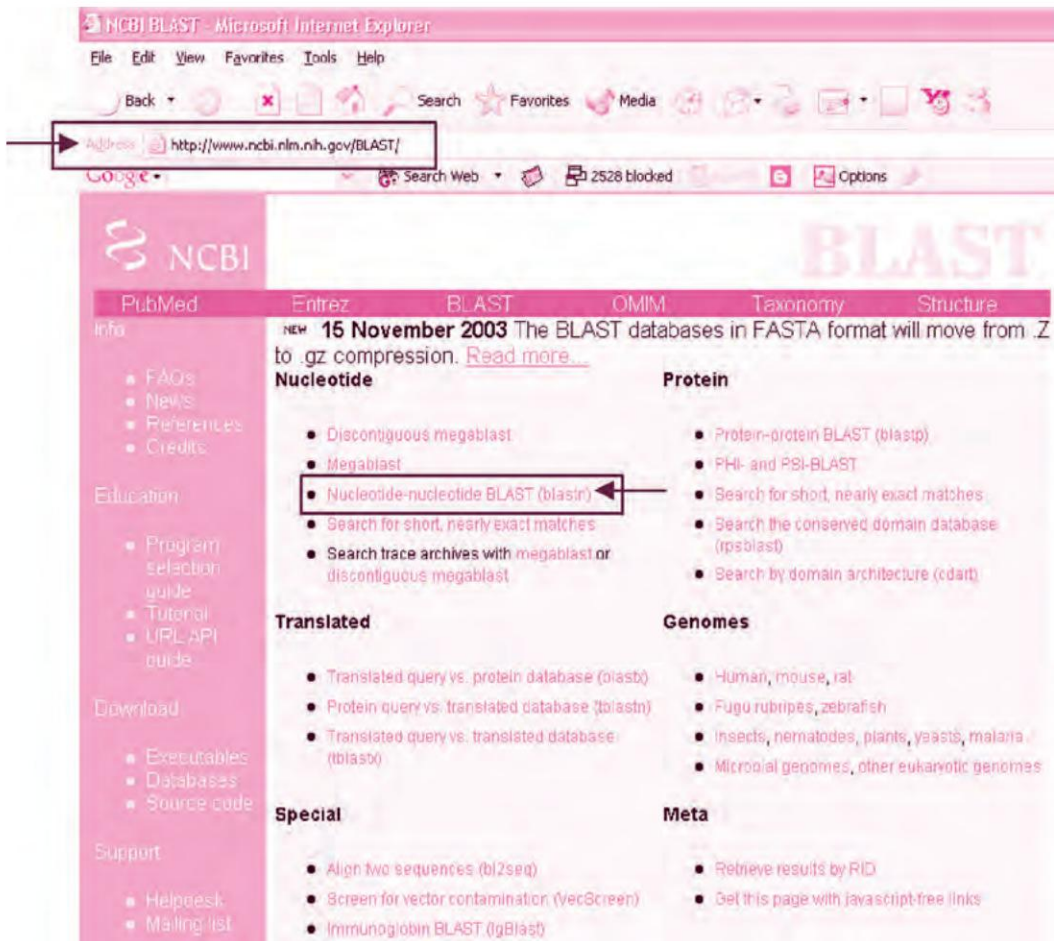


Fig. 2.1 NCBI BLASTN

Although BLAST is unaffected by such artifacts, to get accurate results, it is a good practice to ensure that the sequence is free of errors.

After you submit the search, BLAST responds with a message saying that the request has been successfully submitted and put into a queue and also provides an estimate of the time in which the results will become available (Figure 2.4).

BLAST also provides options to change the way the search results appear on the screen (Figure 2.5). We will use the default formatting to view the output.

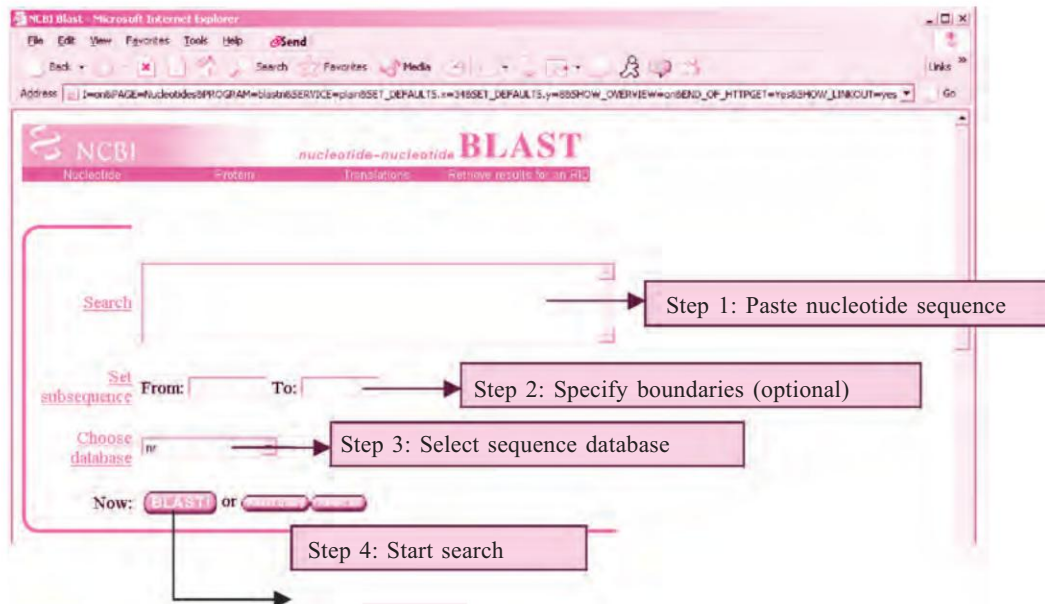


Fig. 2.2 BLASTN parameters

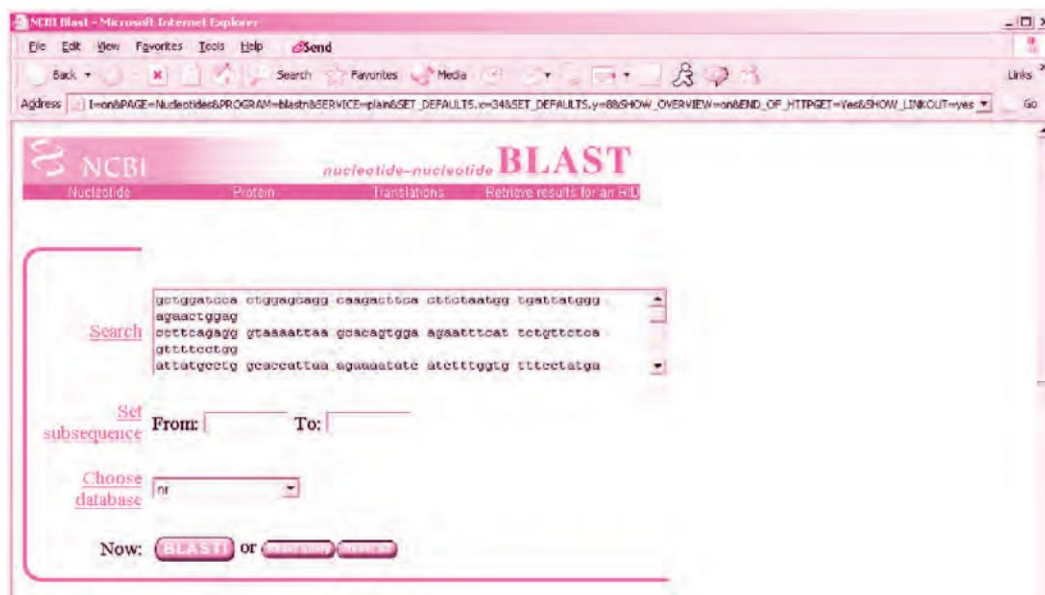


Fig. 2.3 Pasting a nucleotide sequence

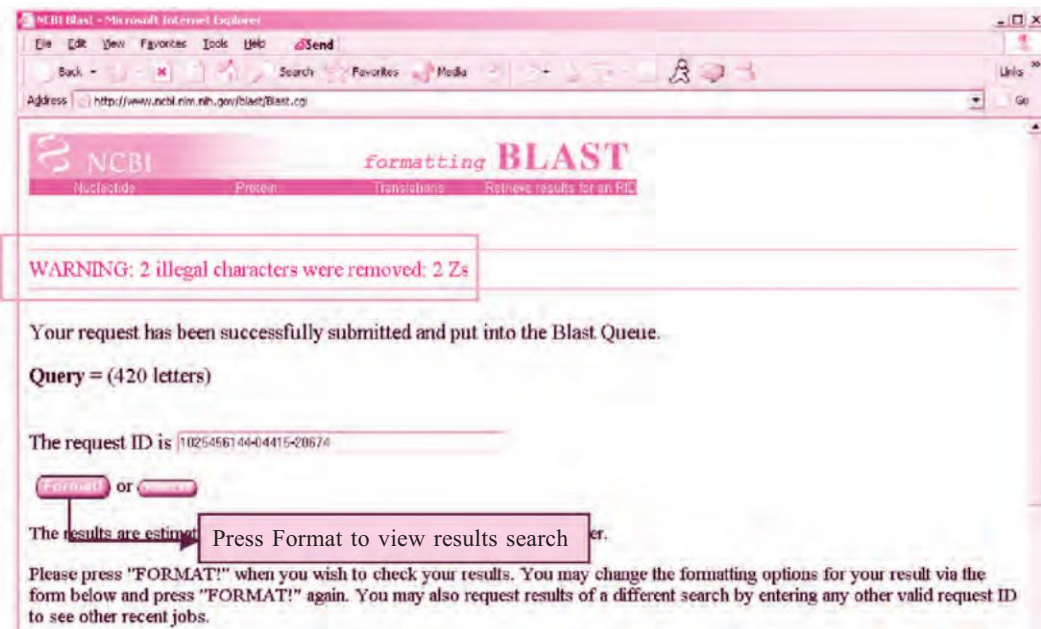


Fig. 2.4 Submitting a BLASTN search

To view the results, press Format. If the analysis is not complete, you may have to wait for the page to be updated until search is done. What follows is a series of screens that explain the different aspects of the output.

2.8 EXPLANATION OF THE BLAST OUTPUT

The general layout of the BLAST output is as follows:

- **Header:** This includes information on the BLASTN version used, a reference to the original publication on the BLAST algorithm (that the user should acknowledge in scientific communications), the request ID for the search, the length of the query sequence and information on the database used.
- **Graphical view of hits:** This is an interactive line up of sequences from the nr database that match the query sequence. The top of the page indicates the number of sequences found: 186 in this case. Each of the lines is color-coded and provides an indication of the score for the

NCBI Blast - Microsoft Internet Explorer

Address: <http://www.ncbi.nlm.nih.gov/blast/Blast.cgi>

Format or **Results**

The results are estimated to be ready in 46 seconds but may be done sooner.

Please press "FORMAT!" when you wish to check your results. You may change the formatting options for your result via the form below and press "FORMAT!" again. You may also request results of a different search by entering any other valid request ID to see other recent jobs.

Format

Show ☒ Graphical Overview ☒ Linkout ☒ NCBI-gen Alignment in HTML **Format**

Number of: Descriptions 100 Alignments 50

Alignment view: Pairwise

Limit results by: query or select from: (none)

Expect value range:

Fig. 2.5 Formatting results

match between the database sequence and the query sequence. Sequences on the top are more significant (have higher scores) than those below. Each of the lines carry information on the sequence that the search came up with. A mouse-over on the first line indicates that it is the *Homo sapiens* cystic fibrosis transmembrane conductance regulator (CFTR) mRNA with a score of 833 (which in this case is highly significant) (Figure 2.7).

Clicking on the line takes you to the actual alignment between the input sequence and the human CFTR gene (Figure 2.8). The alignment page shows more information on the human CFTR gene (called the subject sequence), such as the GenBank ID (14753226) and length (6128 nucleotides). This is hyperlinked to the actual GenBank record (Figure 2.9). The GenBank record lists out all the information that is known about the structure of the gene (source, functional domains, allelic variations if present and the sequence itself).

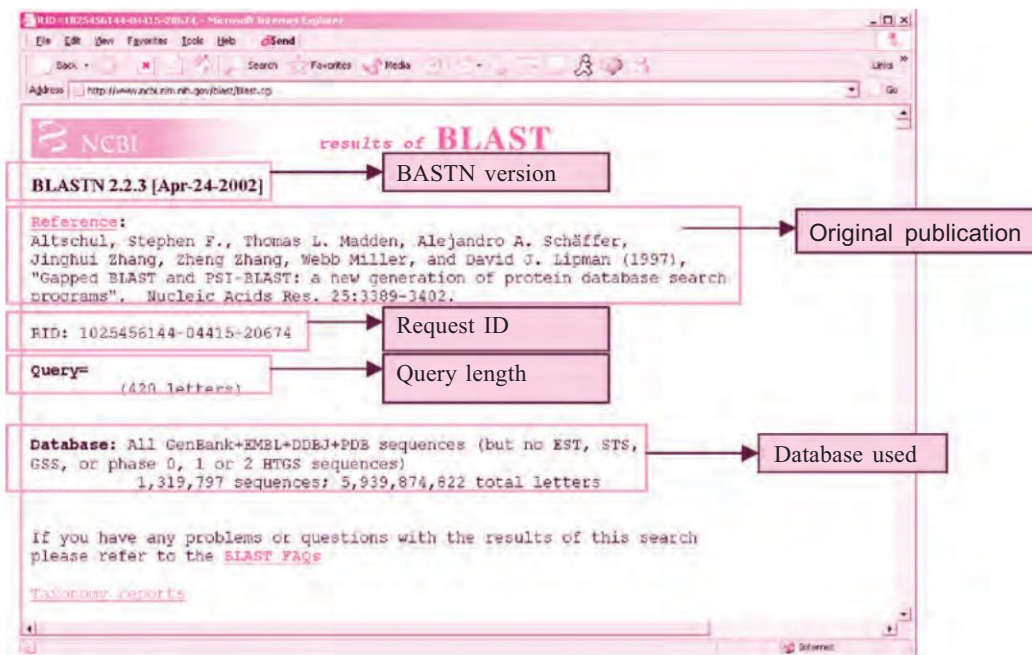


Fig. 2.6(A) BLAST output: Graphical view of hits

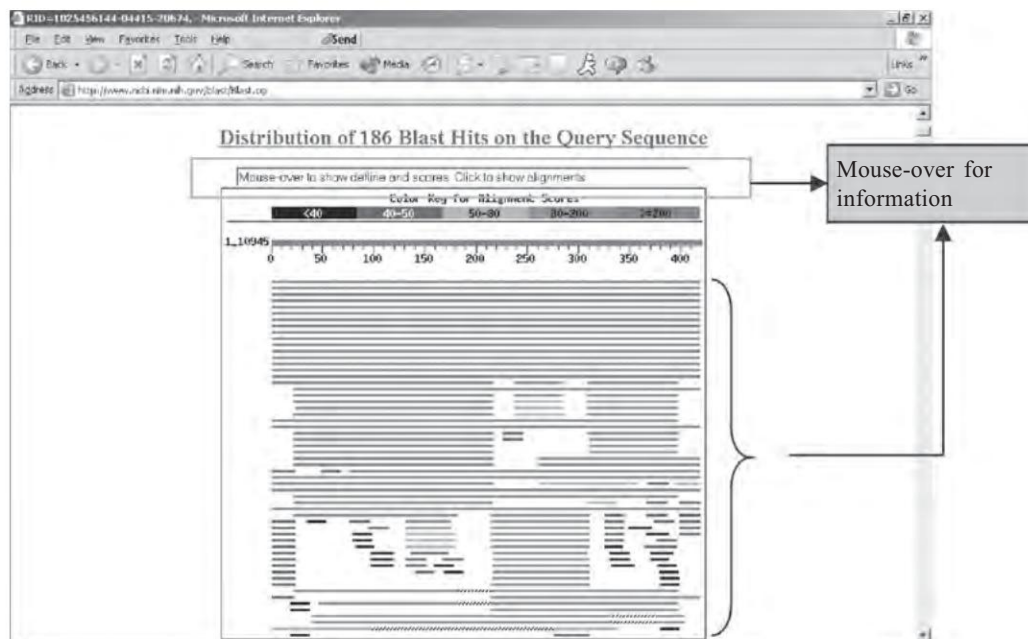


Fig. 2.7(B) BLAST output: Mouse-over for information

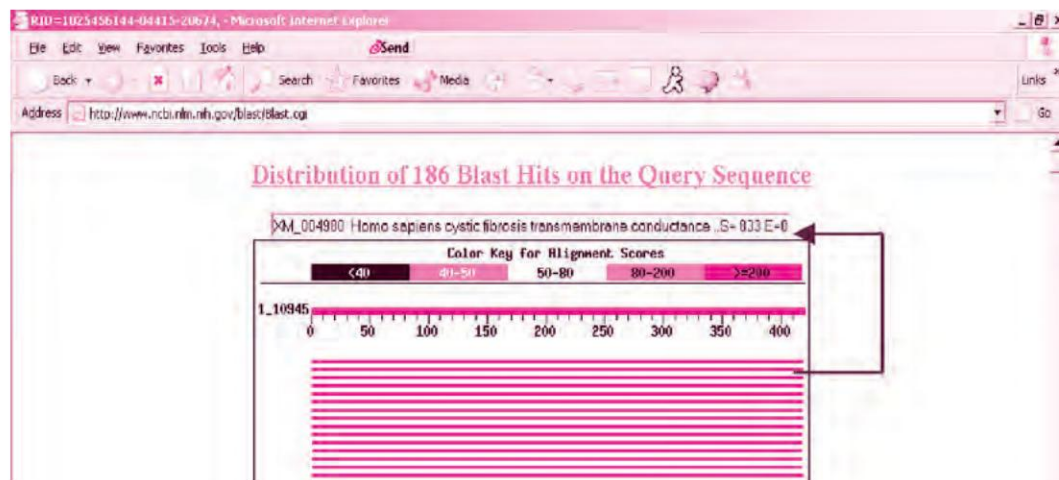


Fig. 2.7 BLAST output: Mouse-over for information

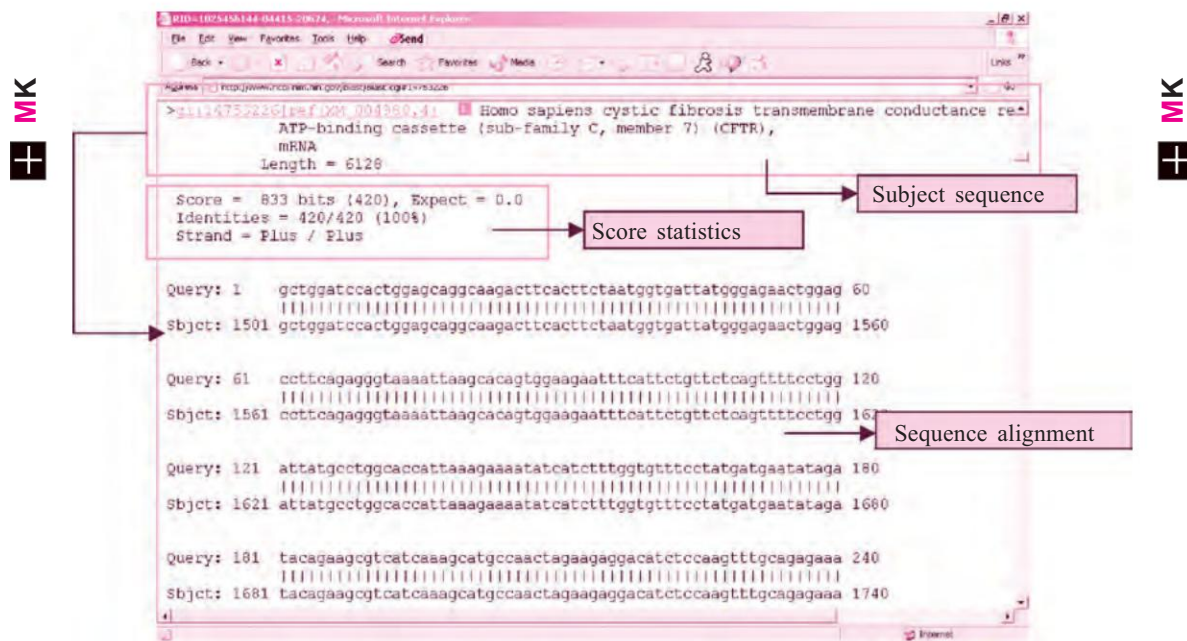


Fig. 2.8 BLAST output: Sequence alignments

Instead of looking at each alignment separately, you can also view the entire list of hits. The BLAST output provides such a list directly below the interactive output shown in Figure 2.6 (Figure 2.10). This page lists the name of the subject sequence and the score and E value for the match. Note that score and E value

1: XM_004980.1 Homo sapiens cystic fibrosis transmembrane conductance regulator, ATP-binding cassette (sub-family C, member 7) (CFTR), mRNA.

LOCUS XM_004980 6128 bp mRNA linear PRI 13-MAY-2002

DEFINITION Homo sapiens cystic fibrosis transmembrane conductance regulator, ATP-binding cassette (sub-family C, member 7) (CFTR), mRNA.

ACCESSION XM_004980

VERSION XM_004980.4

KEYWORDS .

SOURCE human.

ORGANISM Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.

REFERENCE 1 (bases 1 to 6128)

AUTHORS NCBI Annotation Project.

TITLE Direct Submission

JOURNAL Submitted (09-MAY-2002) National Center for Biotechnology Information, NIH, Bethesda, MD 20894, USA

COMMENT GENOME ANNOTATION : This model reference sequence was predicted from NCBI contig by automated computational analysis using gene prediction method: BLAST, supported by mRNA and EST

Fig. 2.9 GenBank record for human CFTR mRNA

are inversely related: higher scores and lower E values indicate more significant matches. The score (bits) is a value attributed to the alignment but is independent of the scoring matrix used. The higher this value, the better the match. The dark and light blue boxes called 'L' and 'U', to the right of the E value column, provide links to LocusLink (or Entrez Gene) and UniGene respectively (Fig. 2.10).

UniGene consists of a non-redundant set of gene-oriented clusters, each of which represent a unique gene. LocusLink provides a single query interface to curated sequences and descriptive information about genetic loci.



2.9 ADVANCED BLASTN

We will now submit the same sequence using the advanced BLASTN form (Fig. 2.11). The options that this form provides are as follows:

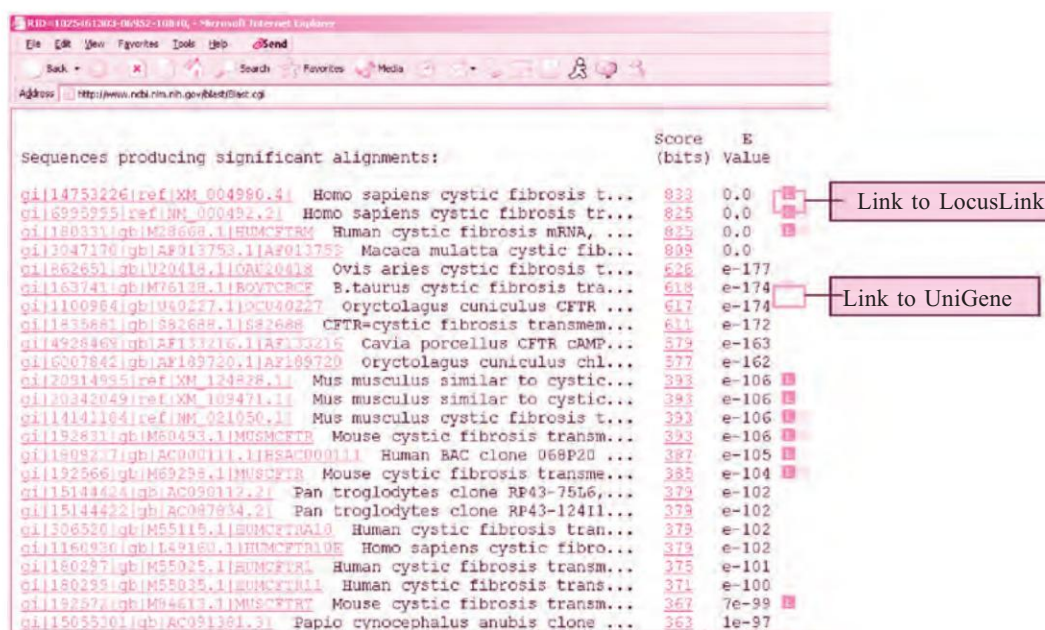


Fig. 2.10 BLAST output: List of significant alignments

Options for advanced blasting

Limit by or select from: (none)

Choose filter ☒ Low complexity ☐ Human repeats ☐ Mask for lookup table only ☐ Mask lower case

Expect

Word Size

Other advanced

Fig. 2.11 Advanced BLASTN

- Limit by Entrez Query: This option allows the user to limit searches by keyword to a certain protein or tissue, molecule or organism type, e.g., kinase NOT arabidopsis[Organism] will search all kinases, except

those from arabidopsis, and biomol_mrna[PROP] AND brain will limit search to all mRNAs from brain. Limits to a specific organism can be set using the pull-down menu and is the option that is more commonly used. The entries in this pull-down list cover a large number of organisms ranging from microbes to mammals. We will limit the search to human by selecting *Homo sapiens* from the list. Now the searches that show up will be limited to human CFTR genes.

- Choose filter: A filter is a tool that flags or masks regions of low compositional complexity and excludes them from the BLAST search. This usually eliminates regions that are uninteresting biologically. This feature is available with BLAST version 2.0. The effect of the masking is that low complexity regions in the query sequences are replaced with a string of 'N's (N means 'any DNA base'). Only the query sequence, and not the sequences in the database, is masked. Default filtering is done with the DUST program for BLASTN and SEG for other programs. Masking is commonly applied to sequences such as Alu sequences (a family of repetitive sequences approx. 300 bp in length), Poly A tails and proline rich sequences which are dispersed throughout the human genome in large numbers and can return artificially high scores and produce misleading results. We will use the default option (DUST) for our search.

- Other advanced options

The default values of these options are as follows:

Cost to open gap

default = 5 for nucleotides 11 proteins

Cost to extend gap

default = 2 nucleotides 1 proteins

Expect value

default = 10

W wordsize

default = 11 nucleotides 3 proteins

The descriptions of these options are as follows:

Gap: A gap is simply a space inserted into a sequence where there is a residue/base in the corresponding sequence in the alignment. A space is introduced into an alignment to compensate for insertions and deletions in

one sequence relative to another. Introduction of a gap results in the deduction of the gap opening score from the alignment score. In a similar fashion, extension of the gap to encompass additional nucleotides or amino acids also results in deduction of the gap extension score from the alignment score. The raw score of an alignment then is the sum of the scores for aligning pairs of residues and the scores for gaps. Gapped BLAST uses “affine gap costs” which charge the score- a for the existence of a gap, and the score- b for each residue in the gap. A gap of k residues receives a total score of $-(a+bk)$. We will use the default values for this option.

Some matrices and their open and extended gap penalties are provided in Table 2.3.

Table 2.3 *Matrices and their gap penalties*

Matrix	Open Gap	Extended Gap
BLOSUM45	15	2
BLOSUM62	11	1
BLOSUM80	10	1
PAM30	9	1
PAM70	10	1

Increasing the gap opening cost or the gap extension cost will impose a greater penalty on the alignment score and increase the stringency of the search. Fewer but better alignments will be reported.

- **Expect value:** This is a measure of the probability that a given match has occurred purely by chance. The default value is 10, meaning that 10 matches are expected to be found merely by chance. If the statistical significance ascribed to a match is greater (less significant) than the threshold, the match will not be reported. Increasing the E value has the effect of allowing less significant and more number of matches to be reported. A cutoff value of 0.00001 to 0.001 is usually chosen in most searches. Values higher than these are generally not considered significant. We will make our search stringent by choosing an E value of 0.001.
- **Word length:** As described earlier, BLAST uses ‘words’ to nucleate regions of similarity. The default word size for a protein sequence is

three residues and for nucleotide sequences it is 11 bp. Reducing the word size will increase the number of seed sequences and increase the time to complete the search. A BLASTN search will not work with a word size of less than seven. We will use the default values for this options.

There are no rules that describe how these different parameters need to be set in order to arrive at an optimal search strategy. This comes largely by experience and also depends, to a large extent, on the particular sequence being analyzed. As before, we submit the request by clicking the BLAST! button. The results of the search are shown below.

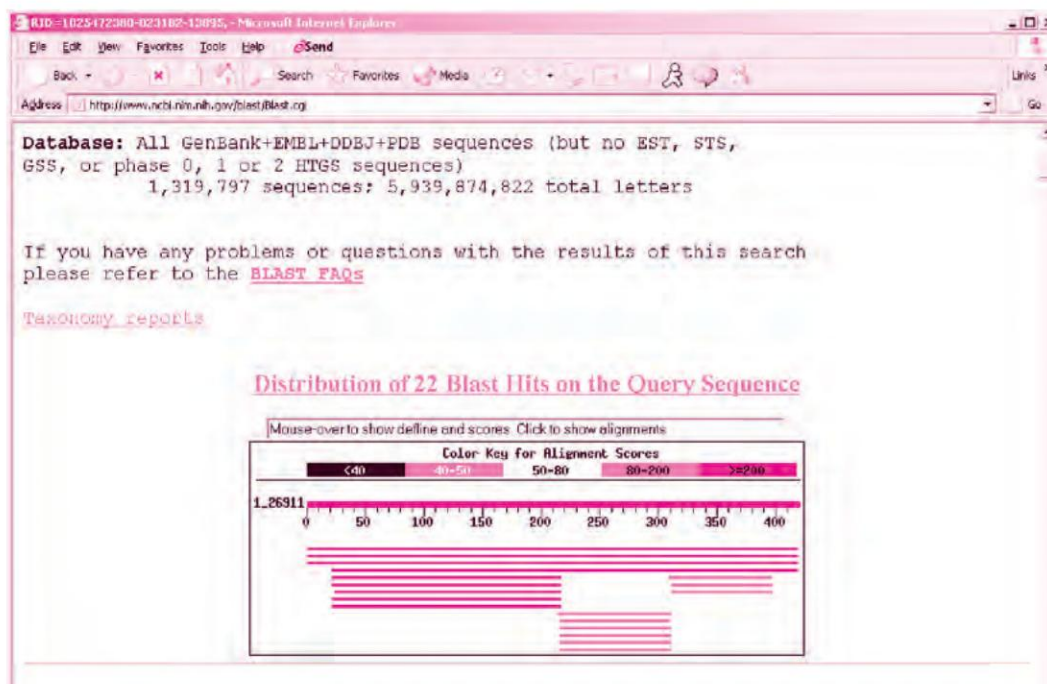
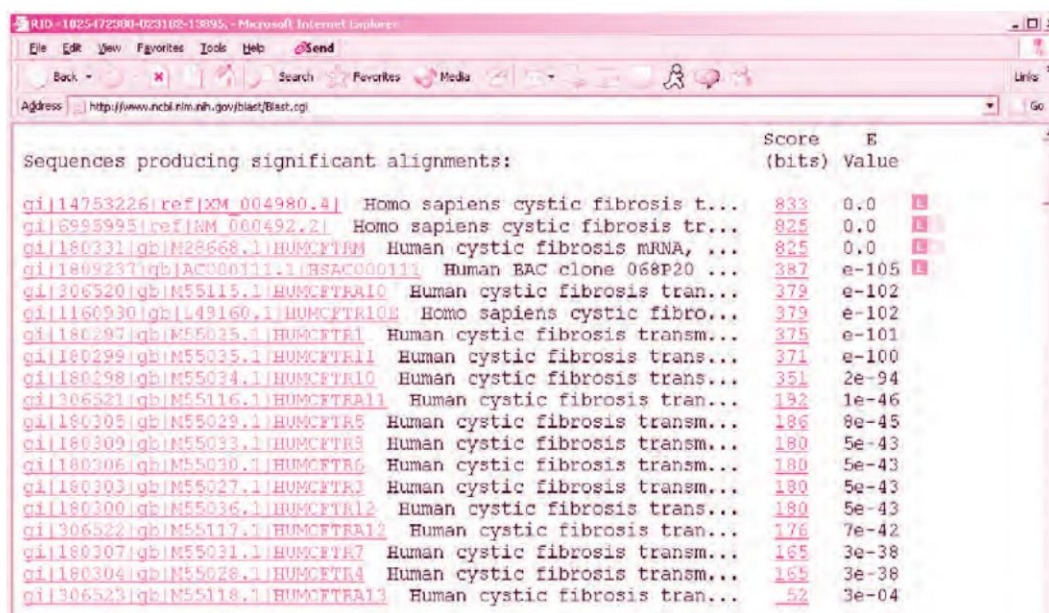


Fig. 2.12 Advanced BLASTN (E-value = 0.001)

Note that the stringent search resulted in fewer sequences being returned (189 at an E value of 10 versus 22 at an E value of 0.001). Note also that the subject sequences returned by the search are now exclusively human sequences (Figure 2.13).



Sequences producing significant alignments:	Score (bits)	E Value
gi 14753226 ref XM_004980.4 Homo sapiens cystic fibrosis t...	833	0.0
gi 6925995 ref NM_00492.2 Homo sapiens cystic fibrosis tr...	825	0.0
gi 180331 gb U28668.1 HUMCFTRM Human cystic fibrosis mRNA, ...	825	0.0
gi 1809237 gb AC000111.1 BSAC000111 Human BAC clone 068P20 ...	387	e-105
gi 306520 gb U55115.1 HUMCFTRA10 Human cystic fibrosis tran...	379	e-102
gi 1160930 gb U49160.1 HUMCFTR10B Homo sapiens cystic fibro...	379	e-102
gi 180297 gb U55035.1 HUMCFTR1 Human cystic fibrosis transm...	375	e-101
gi 180299 gb U55035.1 HUMCFTR11 Human cystic fibrosis transm...	371	e-100
gi 180298 gb U55034.1 HUMCFTR10 Human cystic fibrosis transm...	351	2e-94
gi 306521 gb U55116.1 HUMCFTRA11 Human cystic fibrosis tran...	192	1e-46
gi 180305 gb U55029.1 HUMCFTR5 Human cystic fibrosis transm...	186	9e-45
gi 180309 gb U55033.1 HUMCFTR3 Human cystic fibrosis transm...	180	5e-43
gi 180306 gb U55030.1 HUMCFTR6 Human cystic fibrosis transm...	180	5e-43
gi 180303 gb U55027.1 HUMCFTR3 Human cystic fibrosis transm...	180	5e-43
gi 180300 gb U55036.1 HUMCFTR12 Human cystic fibrosis transm...	180	5e-43
gi 306522 gb U55117.1 HUMCFTRA12 Human cystic fibrosis tran...	176	7e-42
gi 180307 gb U55031.1 HUMCFTR7 Human cystic fibrosis transm...	165	3e-38
gi 180304 gb U55028.1 HUMCFTR4 Human cystic fibrosis transm...	165	3e-38
gi 306523 gb U55118.1 HUMCFTRA13 Human cystic fibrosis tran...	52	3e-04

Fig. 2.13 Advanced BLASTN (Organism = Homo Sapiens)

2.10 BIOLOGICAL ANALYSIS OF BLASTN: CYSTIC FIBROSIS

What kind of inferences can we draw from the BLASTN searches? Almost every single hit from the BLASTN analysis was related to the cystic fibrosis transmembrane conductance regulator (CFTR) gene with very significant scores (high scores and very low E values). It is fairly evident, therefore, that the query sequence encodes the CFTR gene. Is the query sequence a partial sequence or the full-length gene? Considering the fact that the size of the CFTR mRNA is several thousand nucleotides long and our DNA sequence was only 420 bases long, it is obvious that the query sequence was only a fragment of the full-length gene.

The CFTR gene is important for several reasons. Mutations in the gene are responsible for the disease cystic fibrosis (CF), one of the most common inherited disorders in Caucasians, with as many as 1,000 affected individuals being in the United States each year. The disease is associated with pancreatic insufficiency, pulmonary infections, intestinal blockages, elevated sweat chloride levels and male infertility and remains a major health problem.

The genetic defect responsible for CF is a mutation in the CFTR gene that causes a deletion of three base pairs eliminating the amino acid phenylalanine and resulting in the expression of an aberrant form of the protein. The CFTR gene was discovered in 1989 and represents one of the most important triumphs in contemporary human genetics.

The human CFTR gene resides on the long arm of chromosome seven, consists of 27 exons, and encodes a 6,129-bp transcript that encodes a 1,480-aa protein shown to function as a chloride channel. CFTR belongs to the ATP-binding cassette (ABC) family of transporters, containing 12 predicted transmembrane helices and five cytoplasmic domains consisting of two nucleotide-binding domains and a regulatory domain. CFTR is a cAMP-dependent protein kinase-activated (PKA), ATP-gated Cl-channel whose channel function is defective in CF.



2.11 AUTOMATING BLAST ANALYSES WITH PERL



BLAST analyses can be automated with Perl and this is especially useful when a large number of sequences need to be searched against database at NCBI. A script that runs BLAST by sending sequences over the World-Wide Web proceeds in two steps:

1. Send the request to the NCBI BLAST server
2. Wait for the analysis to complete
3. Retrieve results from the server by using the Request ID

Remember, NCBI servers are used by researchers the world over and, therefore, every query is queued into the BLAST system before it can be analyzed. As a result, there may be a considerable amount of delay before you may be able to see the output.

A sample script that extracts a DNA sequence from a file and performs a BLASTN analysis is shown in Listing 1.1 below. The parameters used by the script are encoded in the URL string that is shown highlighted and are as follows:

Alignments:	50
Alignment view:	Pairwise



Database:	nr
Descriptions:	100
E value:	10
Program:	BLASTN
Query sequence:	\$seq (obtained from file)

Listing 2.1 **Sample script for remote BLAST**

```

$/ = undef;
use LWP::Simple;
$file = 'c:\perl\seq.txt';
open(IN, $file) or die "Error opening $file: $!\n";
$seq = <IN>;
$url =
"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=put&ALIGNMENTS=50&ALIGNMENT_VIEW=Pairwise&DATABASE=nr&DESCRIPTIONS=100&ENTREZ_QUERY=(none)&EXPECT=10&FILTER=L&FORMAT_OBJECT=Alignment&FORMAT_TYPE=HTML&HITLIST_SIZE=100&NCBI_GI=on&PAGE=Nucleotides&PROGRAM=blastn&SERVICE=plain&QUERY=$seq";
$page = get($url);

if ($page =~ /The request ID is.+value="\d+\-\d+\-\d+)/) {
    $rid = $1;
}

if ($page =~ /The results are estimated to be ready in (\d+) seconds/){
    $time = $1;
}

$resulturl =
"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&RID=$rid";

sleep($time); ##Wait till analysis completes. Counts time in seconds.

$result = get($resulturl);

print "<br>Result = $result<br>";

```

However, this may not be the most optimal method for database searching because it depends entirely on the availability of the BLAST server at NCBI. The script won't run, for example, when the server is down. In addition, the script may time-out for rigorous searches such as TBLASTX or TBLASTN. In the next chapter, we will see how to download a standalone version of BLAST that can be run locally on your machine.

Assignments

Assignment 1: Given the partial DNA sequence for an unknown gene:

```
aacccgaaaa tccttccttg caggaaacca gtctcagtgt ccaactctct aaccttggaa
ctgtgagaac tctgaggaca aagcagcgga tacaacctca aaagacgtct gtctacattg
aattgggatc tgattcttct gaagataccg ttaataaggc aacttattgc agtgtgggag
atcaagaatt gttacaaatc acccctcaag gaaccagggga tgaaatcagt ttggattctg
caaaaaaggc tgcttgtgaa ttttctgaga cggatgtaac aaatactgaa catcatcaac
ccagtaataa tgatttgaac accactgaga agcgtgcagc tgagaggcat ccagaaaagt
atcagggtag ttctgtttca aacttgcattg tggagccatg tggcacaaat actcatgcc
gctcattaca gcatgagaac agcagtttat tactcactaa agacagaaatg aatgtagaaa
aggctgaatt ctgtaataaa agcaaacagc ctggcttagc aaggagccaa cataacagat
```

List 3 possible BLAST programs that you can use to analyze this sequence. Perform each of these analyses separately and compare the first 10 hits from each of the outputs. Use your knowledge of E values, matrices, gap penalties, etc. to set parameters that may be optimal for the search. What is the effect of varying word length and gap penalties on the output? Identify the gene and describe its structure. What is the significance of this gene and its protein product?

Assignment 2: Write a Perl script to perform a BLASTP analysis using the nr database on protein sequences in a multiple Fasta file. Parse the BLAST output to extract only the top 10 hits for each protein along with the E values and scores.



Web-based Sequence Analysis: BLAST III



In this chapter, we will learn how to download BLAST from NCBI and run local queries on your own computer. The advantages of this method are that you do not have to be limited by the Internet connection that you have access to and do not have to depend on the availability of the BLAST server at NCBI at the time you submit your query. With 'local' BLAST, your queries are not queued on the NCBI server; they are performed on your computer and are, therefore, executed as soon as they are submitted. In addition, this method is secure—this aspect of a local BLAST is especially important for commercial Biotech firms, that do not wish to send out their proprietary sequence data over the World Wide Web for analysis. You need to make sure that you have the latest release of the various databases. To do this, you may need to download them periodically.



3.1 STANDALONE BLAST

The executables for standalone versions of BLAST are available from the NCBI ftp site (<ftp://ftp.ncbi.nih.gov>) and can be downloaded by anonymous ftp from

the <ftp://ftp.ncbi.nih.gov/blast/executables/> folder. Figure 3.1 shows the BLAST versions available for the various platforms—some of the common platforms such as Linux, Mac OS X and Solaris are indicated. The executable and supporting files for the Windows version of BLAST, called `blast-2.2.6-ia32-win32.exe`, is also available as a self-extracting archive from the ftp site.

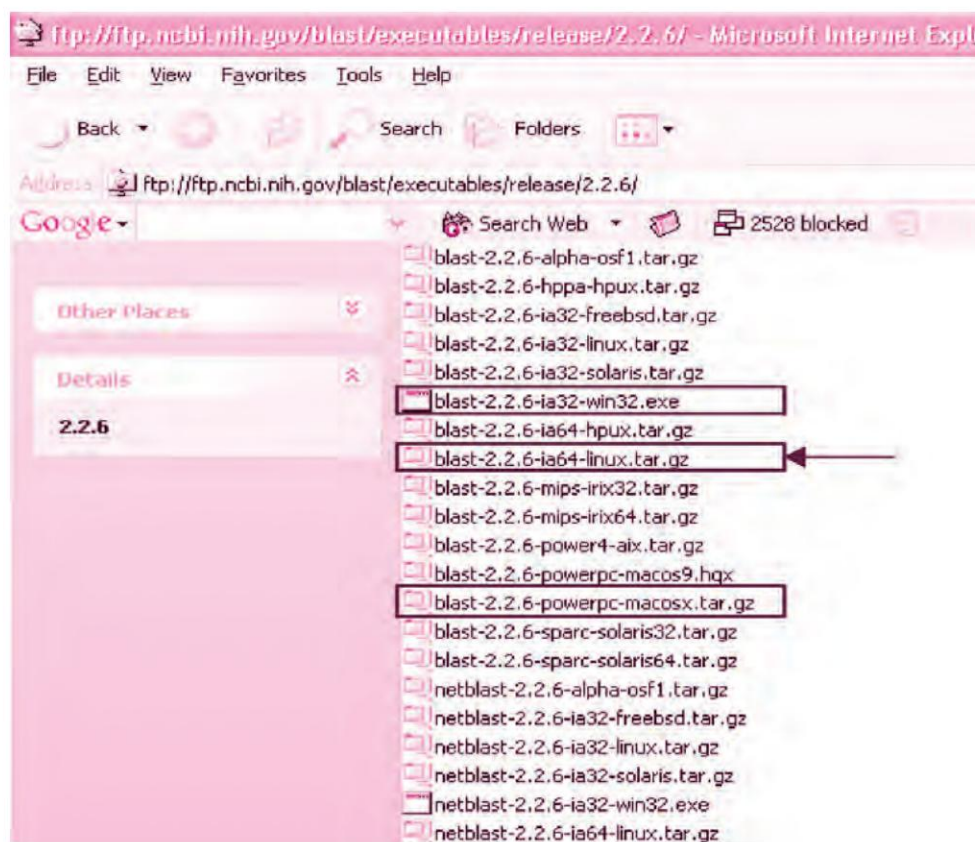


Fig. 3.1 BLAST standalone versions

Follow the steps shown in Figures 3.2–3.5 to install the BLAST application on your computer.

Figure 3.6 shows the various programs installed as part of the BLAST suite. Some of these are explained in Table 3.1.

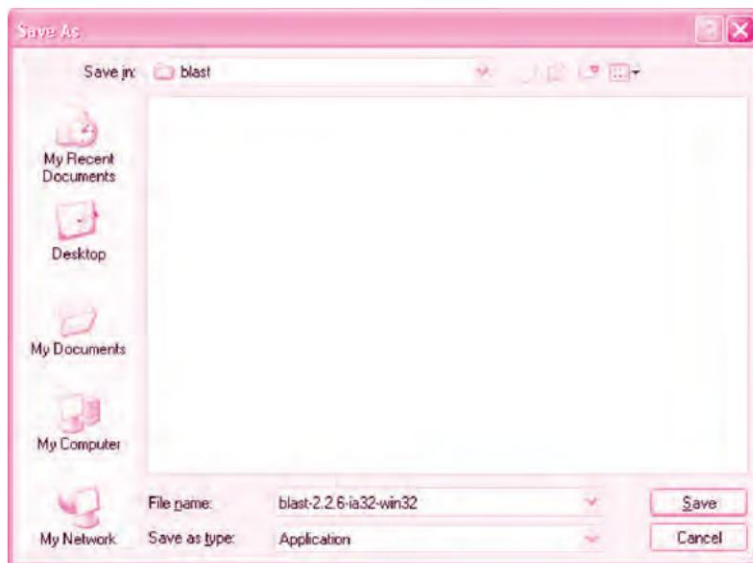


Fig. 3.2 Saving the BLAST executable

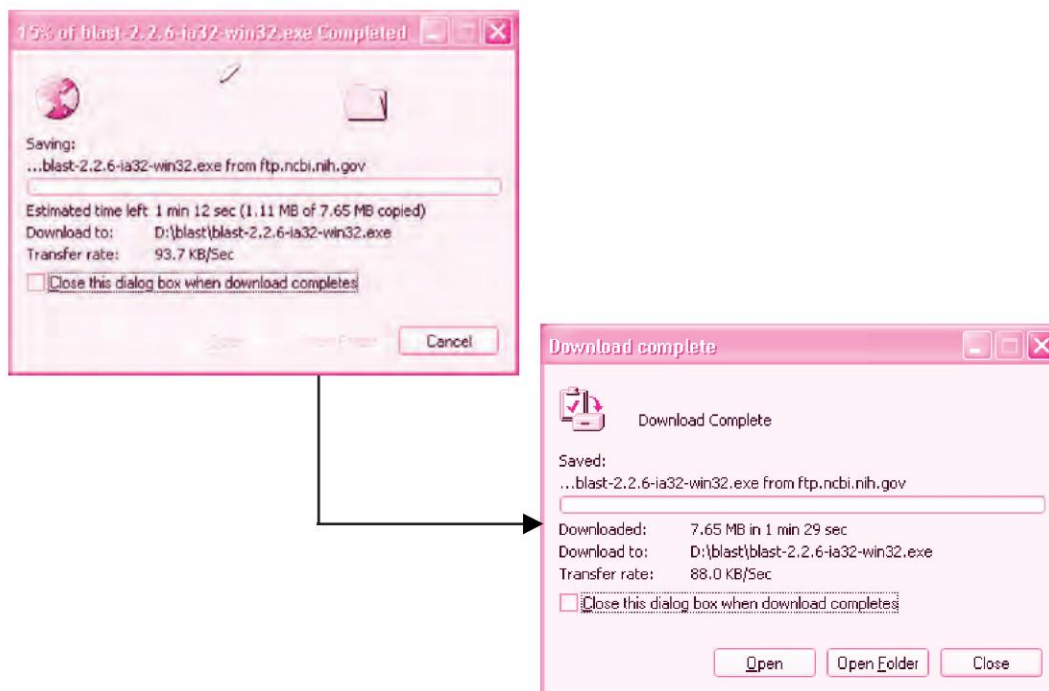


Fig. 3.3 BLAST installation after download

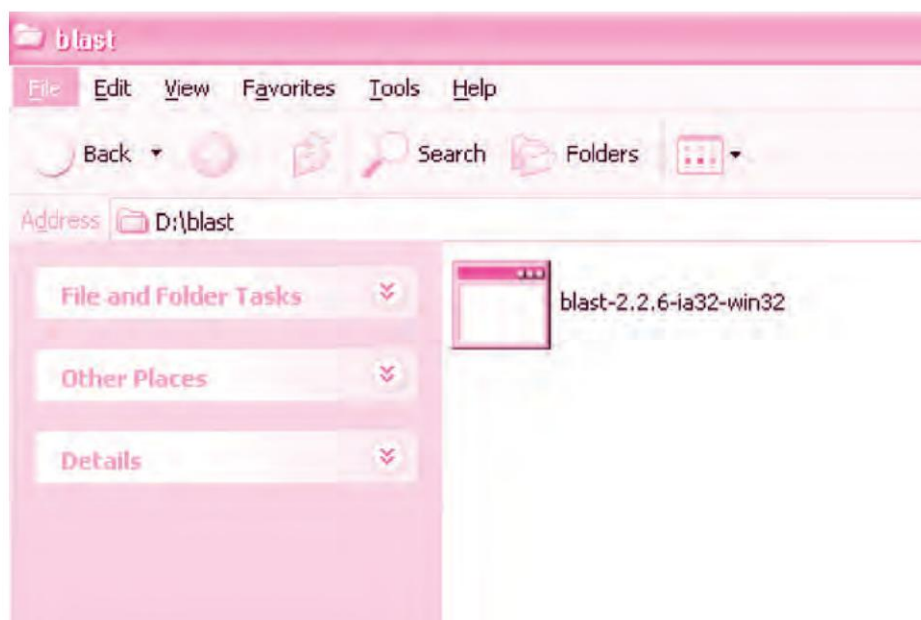


Fig. 3.4 The downloaded executable in D:\blast

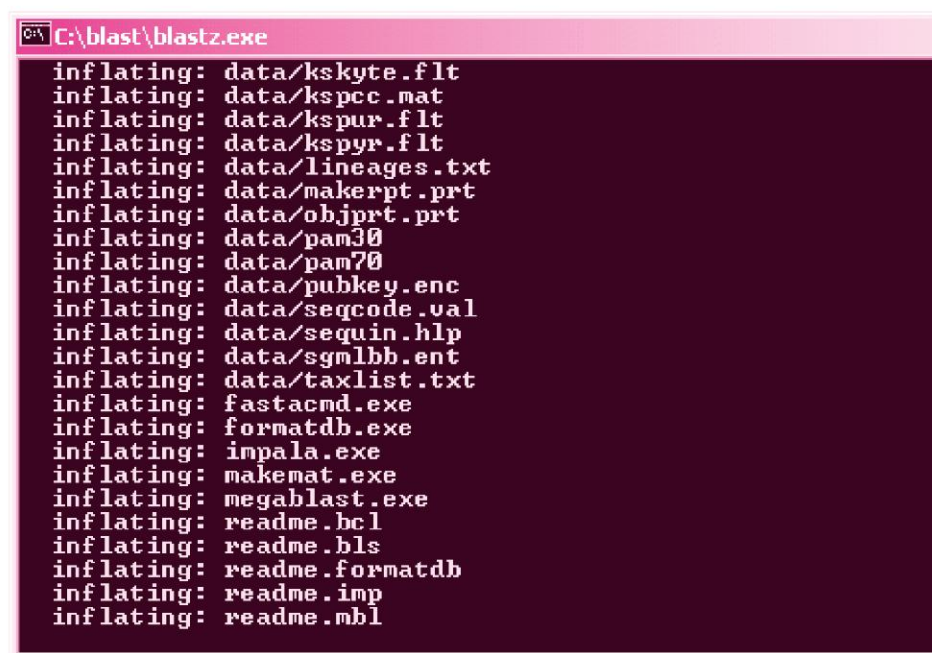


Fig. 3.5 Installing the executable



Table 3.1 List of programs installed with BLAST

Program	Function
blastall	Performs local BLAST searches using any of the five algorithms: BLASTN, BLASTP, BLASTX, TBLASTN or TBLASTX.
blastpgp	Performs gapped BLASTP searches and can be used to perform iterative searches using PSI-BLAST (position-specific iterative BLAST) and PHI-BLAST (pattern-hit iterative BLAST).
Megablast	Alignment program for nucleotide sequence where the sequences differ slightly as a result of sequencing or other similar errors. It is up to 10 times faster than more common sequence similarity programs and, therefore, can be used to swiftly compare two large sets of sequences against each other.
bl2seq	Performs a local alignment between two sequences using either BLASTN or BLASTP. Both sequences must be either nucleotides or proteins.

(Contd.)

Table 3.1 (Contd.)

blastclust	Clustering program for protein or DNA sequences. Based on pair-wise matches found using the BLAST algorithm in case of proteins or Mega BLAST algorithm for DNA.
rpsblast	Reversed Position Specific BLAST performs a BLAST search of a protein sequence vs. a database of conserved protein family domains. Used to derive putative protein family information for an unknown protein sequence.
seedtop	Performs a search between a sequence and a database of patterns and identifies which patterns occur in the sequence.
fastacmd	Program to retrieve FASTA formatted sequences from a BLAST database.

We have earlier used the `bl2seq` program at the NCBI site in Chapter 1. We will be focusing exclusively on the `blastall` program in this chapter.



3.2 CONFIGURING blastall



After the executable has been installed, create a file called “ncbi.ini” in the Windows or WINNT directory on your machine (C:\Windows or C:\WINNT etc. depending on the version of Windows you are running). The path to the file will be C:\Windows\ncbi.ini or C:\WINNT\ncbi.ini for the above two examples. Add the following lines to the ncbi.ini file (Figure 3.7):

```
[NCBI]
```

```
Data="C:\path\data\"
```

where,

```
C:\path\data\
```

is the path to the location of the standalone BLAST “data” subdirectory which should be present in the directory where the downloaded file was extracted.



3.3 DOWNLOADING DATABASES FROM NCBI

To check if the BLAST executable has been installed successfully, download one of the NCBI databases and do a test search against it. Figure 3.8 shows a list of databases available for download at NCBI.



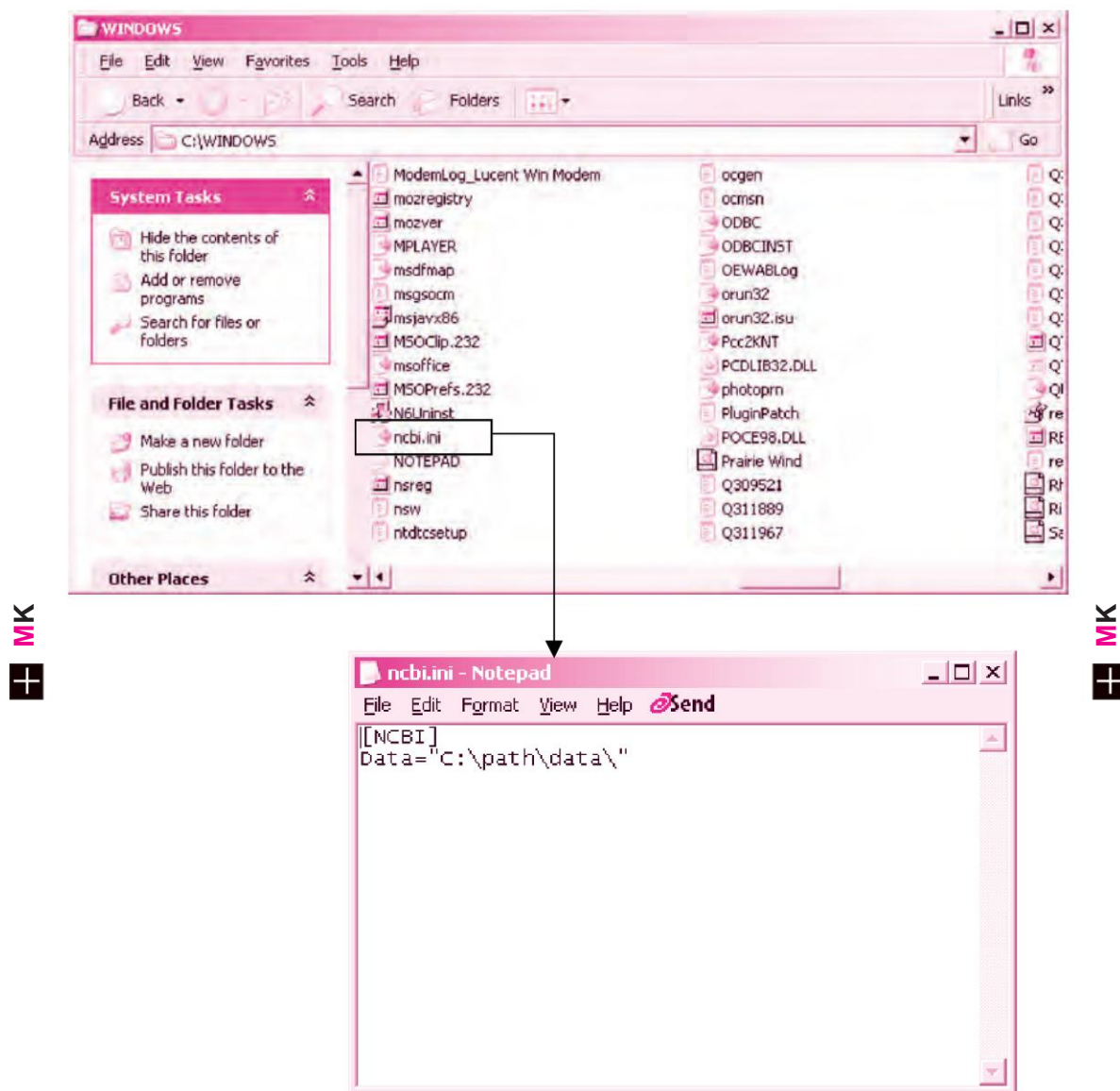
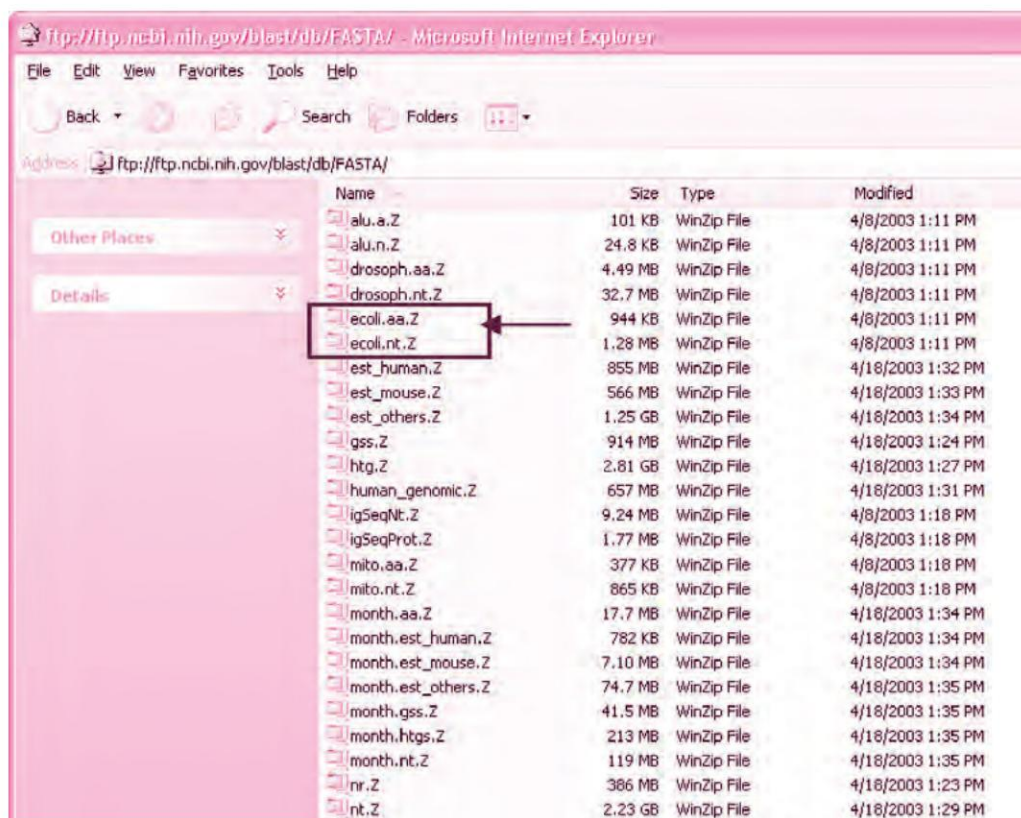


Fig. 3.7 Create the nebi-ini file in C:\WINDOWS

The ftp site is <ftp://ftp.ncbi.nih.gov/blast/db/FASTA/>. We advise installing a small database such as *ecoli.nt* or *ecoli.aa* (the nucleotide and amino acid databases of the bacterium *E. coli* respectively) to begin with. To do so, click on any of the *ecoli.nt.Z* or *ecoli.aa.Z* files and save it on your computer. Figures 3.9–3.11 illustrate how to download the *ecoli.nt* database.



Name	Size	Type	Modified
alu.a.Z	101 KB	WinZip File	4/8/2003 1:11 PM
alu.n.Z	24.8 KB	WinZip File	4/8/2003 1:11 PM
drosoph.aa.Z	4.49 MB	WinZip File	4/8/2003 1:11 PM
drosoph.nt.Z	32.7 MB	WinZip File	4/8/2003 1:11 PM
ecoli.aa.Z	944 KB	WinZip File	4/8/2003 1:11 PM
ecoli.nt.Z	1.28 MB	WinZip File	4/8/2003 1:11 PM
est_human.Z	855 MB	WinZip File	4/18/2003 1:32 PM
est_mouse.Z	566 MB	WinZip File	4/18/2003 1:33 PM
est_others.Z	1.25 GB	WinZip File	4/18/2003 1:34 PM
gss.Z	914 MB	WinZip File	4/18/2003 1:24 PM
htg.Z	2.81 GB	WinZip File	4/18/2003 1:27 PM
human_genomic.Z	657 MB	WinZip File	4/18/2003 1:31 PM
igSeqNt.Z	9.24 MB	WinZip File	4/8/2003 1:18 PM
igSeqProt.Z	1.77 MB	WinZip File	4/8/2003 1:18 PM
mito.aa.Z	377 KB	WinZip File	4/8/2003 1:18 PM
mito.nt.Z	865 KB	WinZip File	4/8/2003 1:18 PM
month.aa.Z	17.7 MB	WinZip File	4/18/2003 1:34 PM
month.est_human.Z	782 KB	WinZip File	4/18/2003 1:34 PM
month.est_mouse.Z	7.10 MB	WinZip File	4/18/2003 1:34 PM
month.est_others.Z	74.7 MB	WinZip File	4/18/2003 1:35 PM
month.gss.Z	41.5 MB	WinZip File	4/18/2003 1:35 PM
month.htgs.Z	213 MB	WinZip File	4/18/2003 1:35 PM
month.nt.Z	119 MB	WinZip File	4/18/2003 1:35 PM
nr.Z	386 MB	WinZip File	4/18/2003 1:23 PM
nt.Z	2.23 GB	WinZip File	4/18/2003 1:29 PM

Fig. 3.8 BLASTable databases at the NCBI ftp site

3.4 FORMATTING NCBI'S DATABASES

You need to format databases before you can run searches on them. NCBI provides a tool called `formatdb`, that is part of the BLAST suite of programs, to create your own BLAST-searchable database. To format a nucleotide database such as `ecoli.nt` database, run the following command from the DOS prompt (Figure 3.12):

```
C:\blast>formatdb -i ecoli.nt -p F -o T
```

This is illustrated in Figure 3.12.

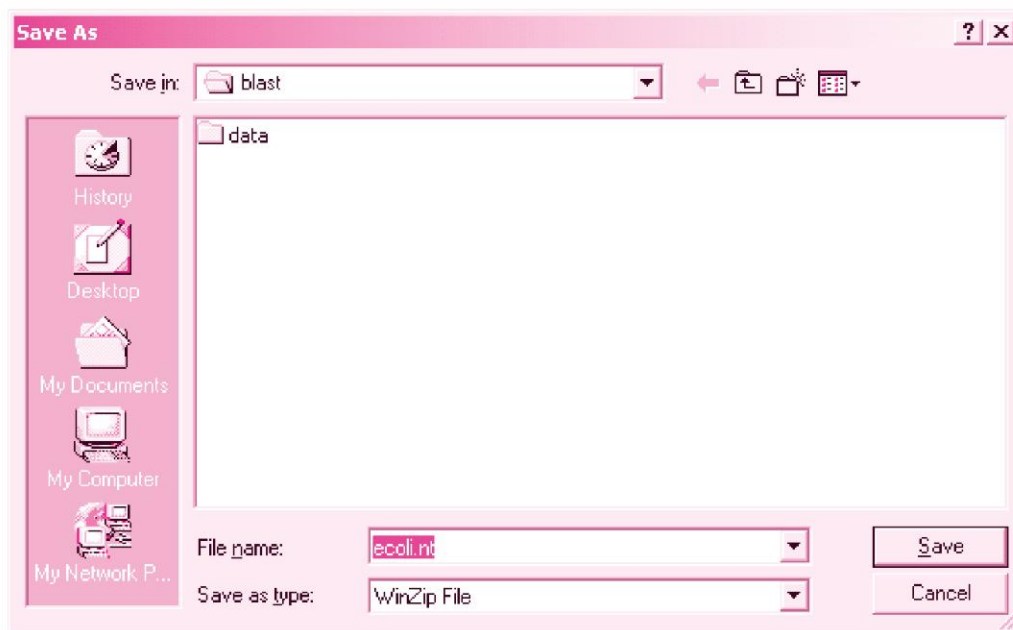


Fig. 3.9 Downloading a sample database: ecoli.nt

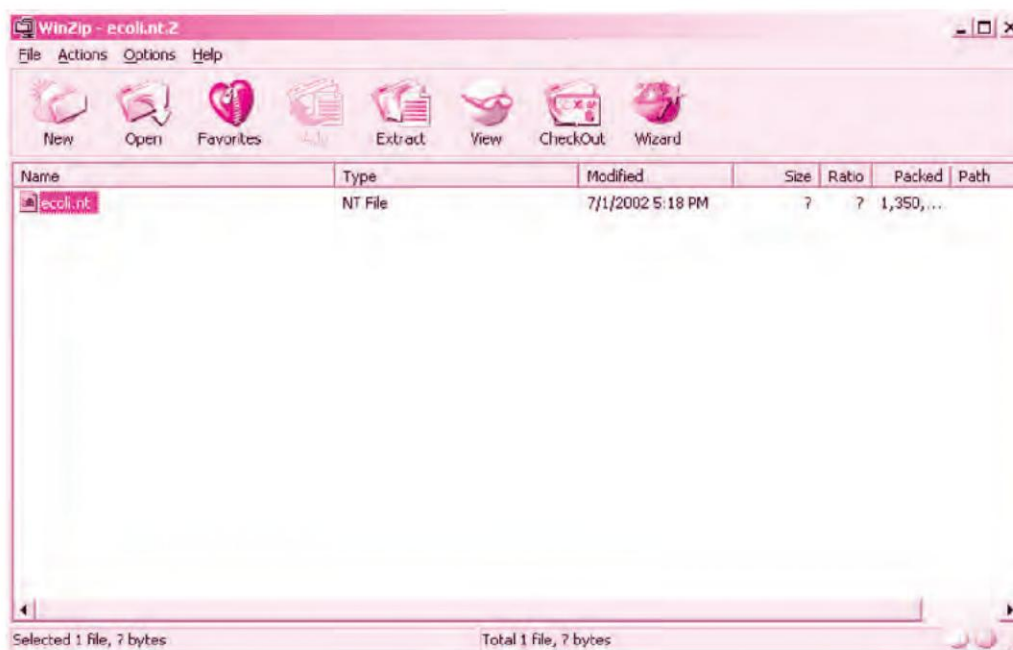


Fig. 3.10 Extracting ecoli.nt with WinZip

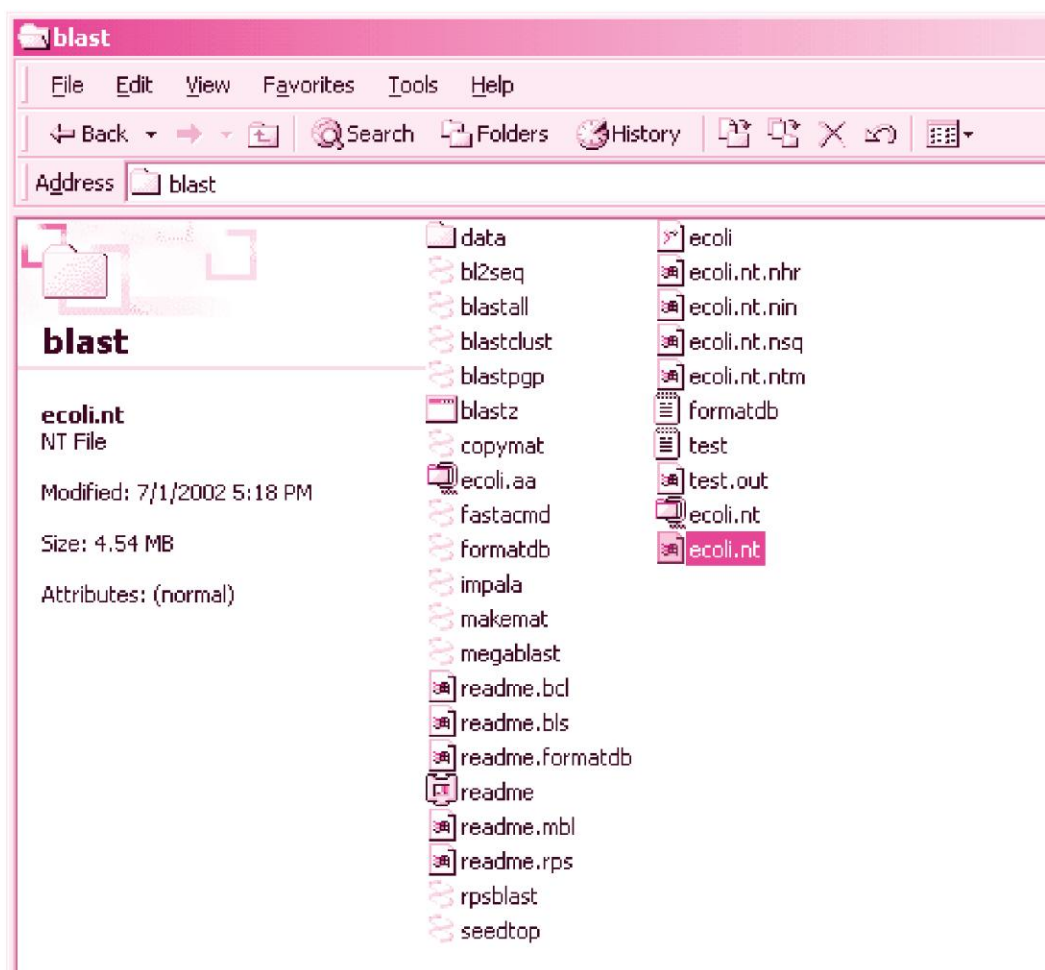
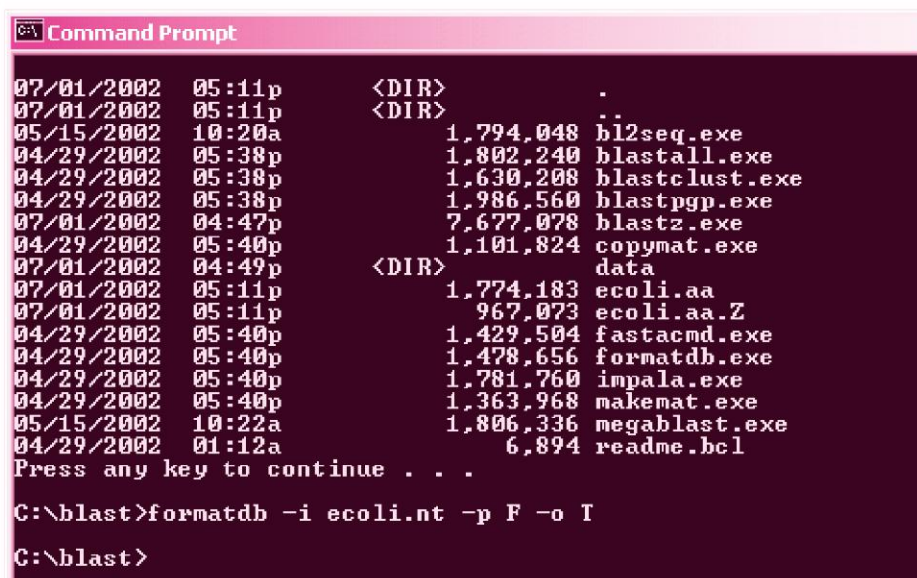


Fig. 3.11 D:\blast after downloading ecoli.nt

The corresponding command to format a protein sequence database such as ecoli.aa is:

```
C:\blast>formatdb -i ecoli.aa -p T -o T
```

The options -i, -p and -o used with formatdb are some of the most commonly used arguments. The individual options are explained in Table 3.2.



```

C:\blast>dir
07/01/2002  05:11p      <DIR>          .
07/01/2002  05:11p      <DIR>          ..
05/15/2002  10:20a             1,794,048  bl2seq.exe
04/29/2002  05:38p             1,802,240  blastall.exe
04/29/2002  05:38p             1,630,208  blastclust.exe
04/29/2002  05:38p             1,986,560  blastpgp.exe
07/01/2002  04:47p             7,677,078  blastz.exe
04/29/2002  05:40p             1,101,824  copymat.exe
07/01/2002  04:49p      <DIR>          data
07/01/2002  05:11p             1,774,183  ecoli.aa
07/01/2002  05:11p             967,073  ecoli.aa.Z
04/29/2002  05:40p             1,429,504  fastacmd.exe
04/29/2002  05:40p             1,478,656  formatdb.exe
04/29/2002  05:40p             1,781,760  impala.exe
04/29/2002  05:40p             1,363,968  makemat.exe
05/15/2002  10:22a             1,806,336  megablast.exe
04/29/2002  01:12a              6,894  readme.bcl
Press any key to continue . . .

C:\blast>formatdb -i ecoli.nt -p F -o T

C:\blast>

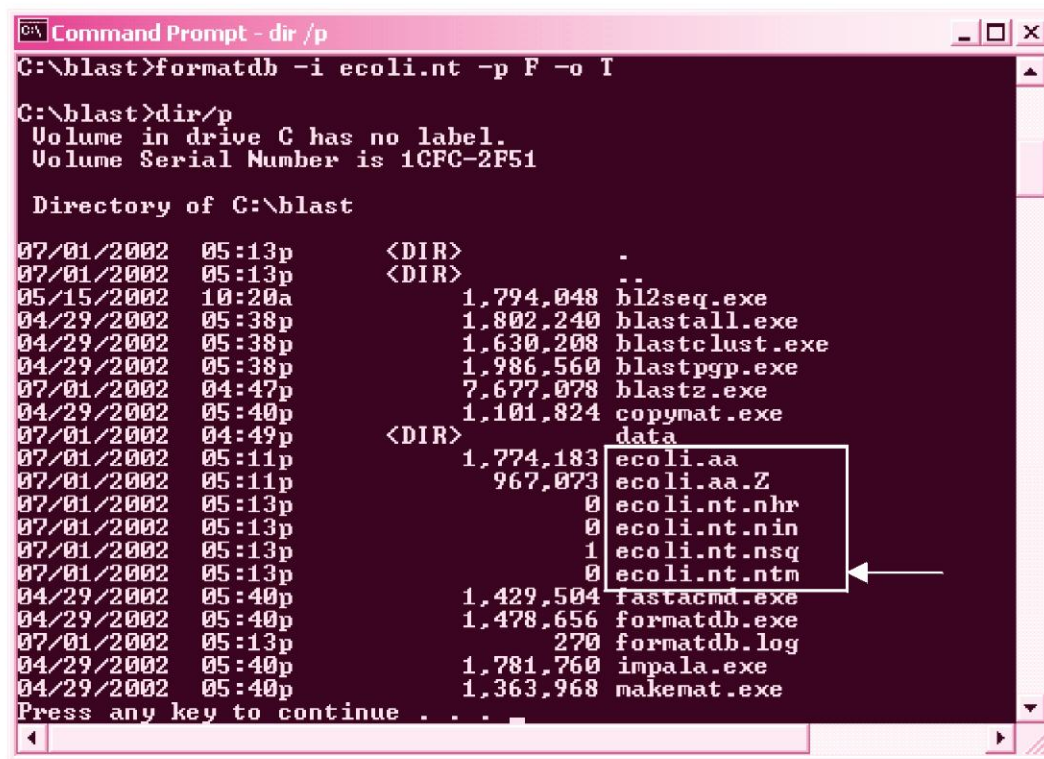
```

Fig. 3.12 Formatting the database with formatdb

Table 3.2 formatdb arguments

Option	Function
-i	Input file for formatting
-p	Type of file T — protein sequences (default) F — nucleotide sequences
-o	Parse options T — True: Parse SeqId and create indexes. F — False: Do not parse SeqId. Do not create indexes.
-t	Title for database file
-n	Base name for BLAST files. Produces a database with a different name than that of the original FASTA file. To create a database called myecoliDB from ecoli.nt, for example, type: formatdb -i ecoli.nt -p F -o T -n myecoliDB
-s	Create indexes limited only to accessions—sparse [T/F]. Default = F. This option limits the indices for the string identifiers used by formatdb to accessions (i.e., no locus names) and is especially useful for sequence sets like the ESTs where the accession and locus names are identical. formatdb runs faster and produces smaller temporary files if this option is used. It is strongly recommended for EST, STS, GSS and HTG sequences.

Some arguments such as title of database, base name of database, etc. are optional. When a BLAST-searchable database is created, a number of files are produced. Using formatdb, these files will have extensions .phr, .pin, .psq for protein databases and .nhr, .nin, .nsq for nucleotide databases (Figure 3.13). The *ecoli.nt* file can be removed once formatdb has been run.



```

C:\blast>formatdb -i ecoli.nt -p F -o I

C:\blast>dir/p
Volume in drive C has no label.
Volume Serial Number is 1CFC-2F51

Directory of C:\blast

07/01/2002  05:13p      <DIR>          .
07/01/2002  05:13p      <DIR>          ..
05/15/2002  10:20a      1,794,048      bl2seq.exe
04/29/2002  05:38p      1,802,240      blastall.exe
04/29/2002  05:38p      1,630,208      blastclust.exe
04/29/2002  05:38p      1,986,560      blastpgp.exe
07/01/2002  04:47p      7,677,078      blastz.exe
04/29/2002  05:40p      1,101,824      copymat.exe
07/01/2002  04:49p      <DIR>          data
07/01/2002  05:11p      1,774,183      ecoli.aa
07/01/2002  05:11p      967,073       ecoli.aa.Z
07/01/2002  05:13p      0             ecoli.nt.nhr
07/01/2002  05:13p      0             ecoli.nt.nin
07/01/2002  05:13p      1             ecoli.nt.nsq
07/01/2002  05:13p      0             ecoli.nt.ntm
04/29/2002  05:40p      1,429,504      fastacmd.exe
04/29/2002  05:40p      1,478,656      formatdb.exe
07/01/2002  05:13p      270           formatdb.log
04/29/2002  05:40p      1,781,760      impala.exe
04/29/2002  05:40p      1,363,968      makemat.exe

Press any key to continue . . .
  
```

Fig. 3.13 c:\blast after formatdb

3.5 RUNNING blastall

To run blastall against the *ecoli.nt* database, download a test *E. coli* sequence from NCBI (<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=nucleotide>) such as the *E. coli* beta-lactamase nucleotide sequence, save it on your computer and run the following command (Figure 3.14):

```
C:\blast>blastall -p blastn -d ecoli.nt -i lactamase.txt -o lactamase.out
```



Fig. 3.14 Running blastall against ecoli.nt

Note that you may get the “[NULL_Caption] WARNING: test: Could not find index files for database” error when blastall cannot find the database you have specified. If any of these databases or files are on a different directory than where BLAST is installed, you may need to specify the full path to the database. For example,

```
c:\blast\blastall -p blastp -d d:\blastdb\nr\nr -i kinase.txt
```

An explanation of common command-line flags used with the blastall command is provided in Table 3.3 and Figure 3.15.

Table 3.3 *blastall options*

Option	Function	Values
-p	Program name	blastn, blastp, blastx, tblastn or tblastx
-d	Database name	nr, swissprot, est, etc.
-l	Input (query) sequence file	cftr.txt, etc.
-o	BLAST results (output file)	cftrout.txt, etc.
-e	E value	0.1, 0.01, etc. Default = 10.
-F	Filter query sequence	T or F (for true or false)
-q	Penalty for a nucleotide mismatch	integer

(Contd.)

(Contd.)

-r	Reward for a nucleotide match	integer
-v	Number of one line descriptions	integer
-b	Number of alignments to show	integer
-g	Perform gapped alignment	T or F (for True or False)
-M	Matrix	matrix name
-W	Word size	integer
-T	Produce HTML output	T or F (for True or False)

```

C:\blast>blastall
blastall arguments:
  -p Program Name [String]
  -d Database [String]
    default = nr
  -i Query File [File In]
    default = stdin
  -e Expectation value <E> [Real]
    default = 10.0
  -m alignment view options:
0 = pairwise,
1 = query-anchored showing identities,
2 = query-anchored no identities,
3 = flat query-anchored, show identities,
4 = flat query-anchored, no identities,
5 = query-anchored no identities and blunt ends,
6 = flat query-anchored, no identities and blunt ends,
7 = XML Blast output,
8 = tabular,
9 = tabular with comment lines [Integer]
    default = 0
  -o BLAST report Output File [File Out] Optional
    default = stdout
  -F Filter query sequence <DUST with blastn, SEG with others> [String]
    default = 1
  -G Cost to open a gap <zero invokes default behavior> [Integer]
    default = 0
  -E Cost to extend a gap <zero invokes default behavior> [Integer]

```

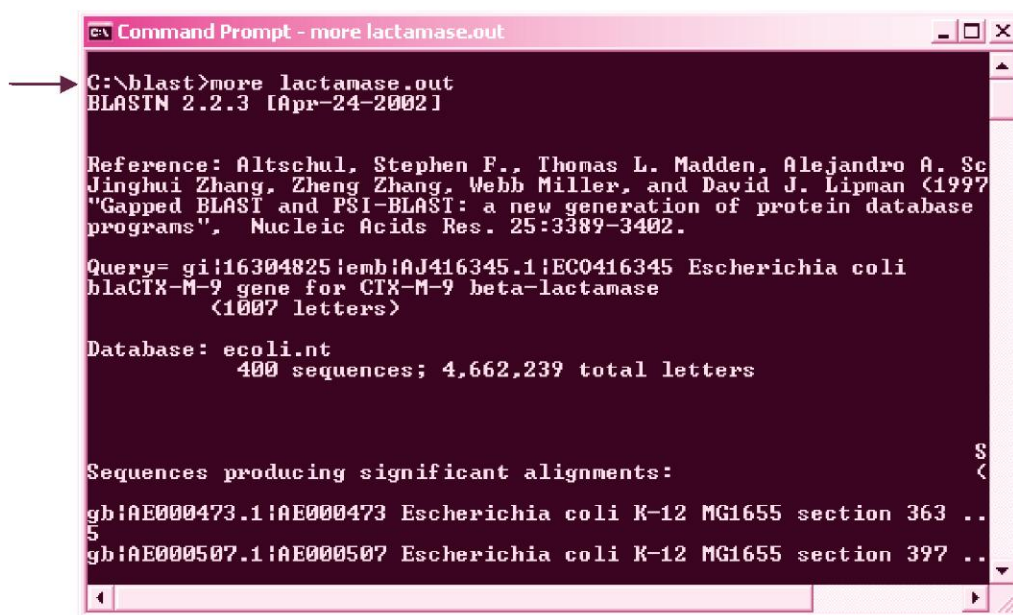
Fig. 3.15 Blastall comand-line options

To look at the contents of the BLAST results, open the lactamase.out file using the more command on the DOS command-line or on a text editor such as Notepad (Figure 3.16).



3.6 DOWNLOADING PRE-FORMATTED DATABASES

The NCBI ftp site also has a number of formatted databases. There is no need to run formatdb with such databases. Figures 3.17–3.22 show how to download



```
C:\ Command Prompt - more lactamase.out

C:\blast>more lactamase.out
BLASTN 2.2.3 [Apr-24-2002]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Sc
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997
"Gapped BLAST and PSI-BLAST: a new generation of protein database
programs", Nucleic Acids Res. 25:3389-3402.

Query= gii16304825|emb1AJ416345.1|EC0416345 Escherichia coli
blaCTX-M-9 gene for CTX-M-9 beta-lactamase
<1007 letters>

Database: ecoli.nt
400 sequences; 4,662,239 total letters

Sequences producing significant alignments:

gb|AE000473.1|AE000473 Escherichia coli K-12 MG1655 section 363 ..
gb|AE000507.1|AE000507 Escherichia coli K-12 MG1655 section 397 ..
```

Fig. 3.16 Checking the results of blastall

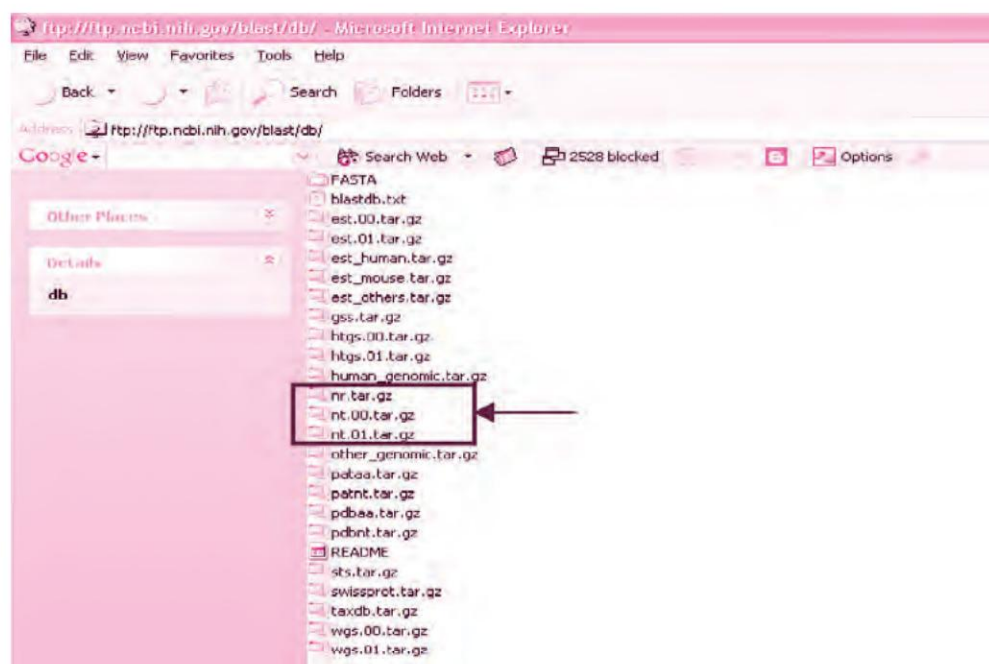


Fig. 3.17 The formatted nr and nt databases

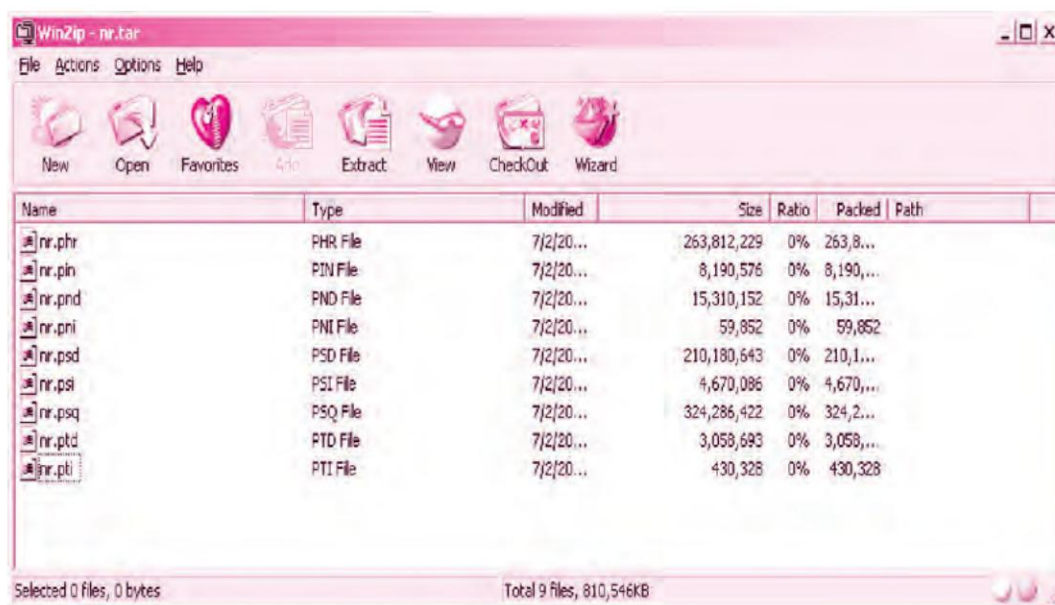


Fig. 3.18 Downloading the pre-formatted nr database

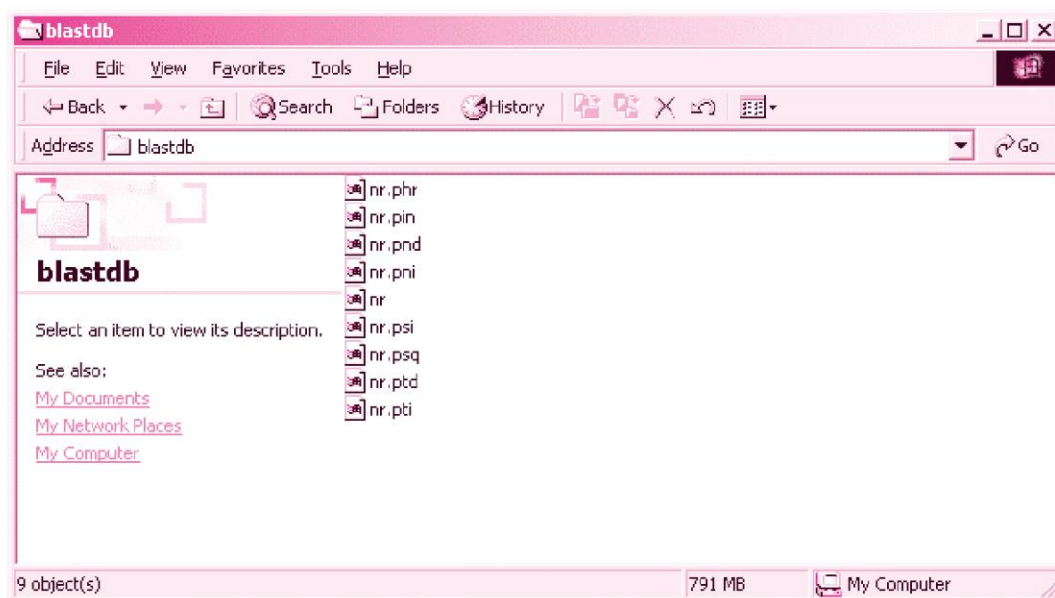


Fig. 3.19 Saving nr in D:\blastdb


```

>gi|1705762|sp|P13569|CFTR_HUMAN Cystic fibrosis transmembrane conductance
MQRSPLEKASVSVSKLFFSWTRPILRKGYRQRLSDIYQIPSVDSADNLSEKLEREWRELASKKNPKLI
NALRRCCFFWRFMFYGFILYLGVEVTKAVQPLLGRITIASYDPDNKEERSIAIYLGIGLCLLFIVRTLLHP
AIFGLHHIGMQMRIAMFSLIYKTKLKLSSRVLDKISIGQLVSLLSNNLNKFDEGLALAHFVWVIAPIQVAL
LMGLIWELLQASAFGLGLFIVLALFQAGLGRMMMKYRDQAGKISERLVITSEMENIQSVKAYCWEEA
MEKMENLRQTELKLRKAAVRYFNSSAFFSGFFVFLSVLPYALIKGIILRKIFTTISFCIVLRMAV
TROFPMAVQTVWYDSLGAINKIQDFLOKQEYKTLEYNLTTEVVMENVTAFWEEGFGEKFELFEKAKQNNMRK
TSNGDSDLFFSNFSLGTPVLKDINFKIERGQLLAVAGSTGAGKTSLLMMINGELEPSEGKIKHSGRISF
CSQFSWIMPGTIKENIIFGVSYDEYRYSVIKACQLEEDISKFAEKDNIVLGEAGITLSGGQRRARISLAR
AVYKDADLYLLDSPFGYLDVLTKEIFESCVCCKLMANKTRILVTSKMEHLKKADKILILHEGSSYFYGTF
SELQNLQPDFSSKLMGCDSDQFSAERNRSLTETLHRFSLEGDAPVSWTETKKQSFQGTGEFGEKRKNS
ILNPINSIRKFSIVQKTPLOMNGIEEDSDEPLERRLSLVPDSEQGEAILPRISVISTGPTLQARRRQSVL
NLMTHSVNQGNIRKTTASTRKVSLAPQANLTLDIYSRRLSQETGLEISEEINEEDLKECFDDMESI
PAVTTWNTYLRVITVHKSLIFVLIWCLVIFLAEVAASLVVLWLLGNTPLQDKNSTHSPNNSYAVIITST

```

Fig. 3.20 The CFTR protein sequence

```

C:\blast>blastall -p blastp -d d:\blastdb\nr -i cftr.txt -o cftr.out
C:\blast>more cftr.out
BLAST 2.2.3 [Apr-24-2002]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1990),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= gi|1705762|sp|P13569|CFTR_HUMAN Cystic fibrosis transmembrane
conductance regulator (CFTR) (cAMP-dependent chloride channel)
(1480 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF
1,023,806 sequences; 323,262,615 total letters

Sequences producing significant alignments:
Score E
<bits> Value
sp|P13569|CFTR_HUMAN Cystic fibrosis transmembrane conductance r... 2804 0.0
ref|XP_004980.4| <XM_004980> cystic fibrosis transmembrane condu... 2803 0.0
ref|NP_000483.2| <NM_000482> cystic fibrosis transmembrane condu... 2799 0.0

```

Fig. 3.21 BLAST of CFTR against nr

```

C:\blast>blastall -p blastp -e 0.00001 -d d:\blastdb\mr -i cftr.txt -o cftr.out
C:\blast>more cftr.out
BLASTP 2.2.3 [Apr-24-2002]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1990),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs". Nucleic Acids Res. 25:3389-3402.

Query= gii1705762!sp!P13569!CFTR_HUMAN Cystic fibrosis transmembrane
conductance regulator (CFTR) (cAMP-dependent chloride channel)
(1480 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF
1,023,806 sequences; 323,262,615 total letters

Sequences producing significant alignments:
                                     Score   E
                                     (bits) Value
sp!P13569!CFTR_HUMAN Cystic fibrosis transmembrane conductance r... 2804 0.0
ref!XP_004980.4! <XM_004980> cystic fibrosis transmembrane condu... 2803 0.0
ref!NP_000483.2! <NM_000492> cystic fibrosis transmembrane condu... 2799 0.0
gb!AAD46907.1!AF162427_1 <AF162427> cystic fibrosis transmembran... 2756 0.0

```

Fig. 3.22 Applying an Eval to BLAST



the pre-formatted nr database and run a query against it. You should preferably download the nr database in a separate folder (such as D:\blastdb\, if using a PC) because of its size (~ 350 MB). At the end of the download, you should see the following files in the folder:

nr.phr
nr.pin
nr.pnd
nr.pni
nr.psd
nr.psi
nr.psq
nr.ptd
nr.pti



3.7 fastacmd

fastacmd is a useful tool to retrieve sequences from BLAST databases using the sequence ID. fastacmd can be used with databases that have been formatted with the -o option. To retrieve the sequence with GenBank ID 1786181 from the ecoli.nt database, type:

```
c:\blast>fastacmd -d ecoli.nt -s 1786181
```

where,

-d name of database

-s sequence id

The output of the command is shown in Figure 3.23.

```

C:\blast>fastacmd -d ecoli.nt -s 1786181
>gi|1786181|gb|AE000111.1|AE000111 Escherichia coli K-12 MG1655 section 1 of 400
of the complete genome
AGCTTTTCATTCTGACTGCAACGGGCAATATGCTCTGTGTGGATTAAAAAAGAGTGTCTGATAGCAGCTTCGAACTG
GTTACCTGCGGTGAGTAAATTAAATTTTATTGACTTAGGTCACTAAATACITTTAACCAATATAGGCATAGCGCACAGAC
AGATAAAAATTACAGAGTACACAACTCCATGAACCGCATTAGCACACCATTACCAACCACCATCACCATTACCAAGGI
AACGGTGGGGCTGACGGGTACAGGAACACAGAAAAAGCCCGCACCTGACAGTGGGGCTTTTTTTTCGACCAAGG
TAACGAGGTAAACAACCATGCGAGTGTGAAGTTCGGCGGTACATCAGTGGCAATGCAGAACGTTTTCTGCGTGTGCCG
ATATTCGGAAGCAATGCCAGGCAGGGGCGAGGTGGCCACCGTCCTCTCGCCCCCGCAAAATCAGCAACACCTGGTG
GCGATGATTGAAAAAACCAITAGCGGCAGGATGCTTTACCCAAATATCAGCGATGCCGAACGTAATTTTCCGGAATTTT
GACGGGACTCGCGCGCGCCAGCGGGGTTCCGCTGGCGCAATTGAAACITTCGTCGATCAGGAATTTGCCAAATAA
AACATGTCCTGCAITGGCATTAGTTTGTGGGGCAGTCCCGGATAGCATCAACGCTGCGCTGATTTCGGTGGCGAGAAA
ATGTGCGATCGCCATTATGGCCGGCTATTAGAACCGCGCGGTACAAACGTTACTGTTATCGATCCGGTCGAAAACTGCT
GGCAGTGGGGCATTACCTCGAATCTACCGTCGATATTGCTGAGTCCACCCGCGGTAATGGGCAAGCCGCAATCCGGCTG
ATCACATGGTGTGATGGCAGGTTTACCGCGCGTAATGAAAAAGGCGAATGGTGGTCTTGACGCAACGGTTCGGAC
TACTCTGCTCGGGTGTGGCTGCCTGTTACGCGCGGATTGTTGCGAGATTGGACGGACGTTGACGGGGTCTATACCTG
CGACCCGGGTGAGGTGCCGATGCGAGGTTGTTGAAGTCGATGCTTACCAGGAAGCGATGGAGCTTTCCTACTTCGGCG
CTAAAGTTCCTACCCCGCACCAATTACCCCATCGCCAGTTCAGATCCCTTGCTGATTAAAAATACCGGAATCCT

```

Fig. 3.23 fastacmd to retrieve sequences

The search option (-s) can be GenBank Ids, accession and locus numbers. To retrieve multiple sequences, supply a file containing GI numbers (one on each line) with the -i flag:

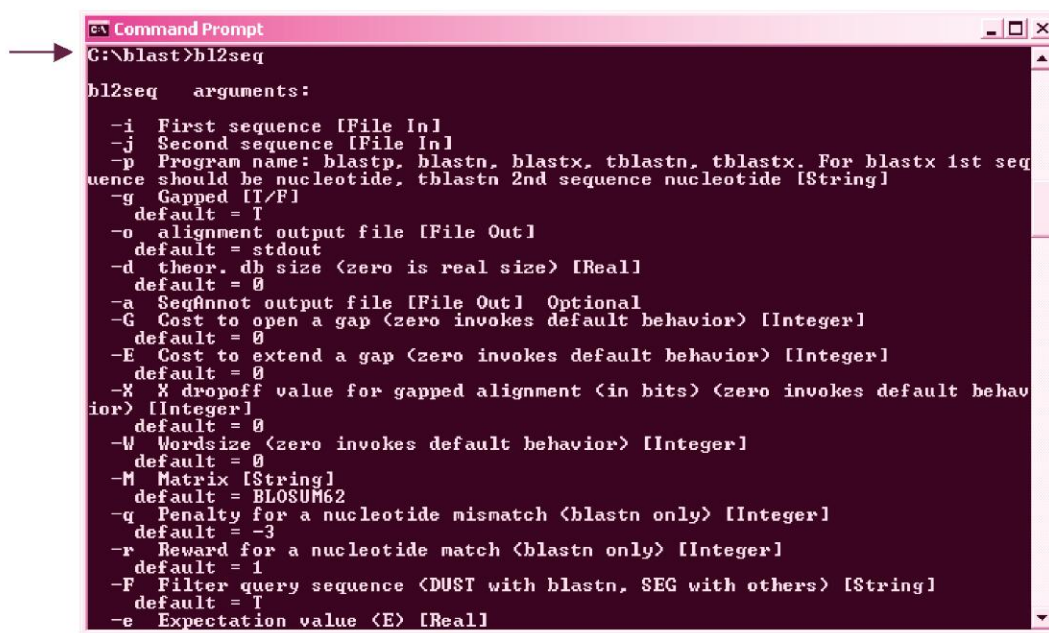
```
c:\blast>fastacmd -d ecoli.nt -i list.txt > list.out
```

The output can be put into another file with the -o option:

```
c:\blast>fastacmd -d ecoli.nt -i list.txt -o list.out
```

3.8 bl2seq

The options used with bl2seq can be listed by typing bl2seq on the command-line (Figure 3.24).



```

C:\blast>bl2seq
bl2seq arguments:
-i First sequence [File In]
-j Second sequence [File In]
-p Program name: blastp, blastn, blastx, tblastn, tblastx. For blastx 1st sequence should be nucleotide, tblastn 2nd sequence nucleotide [String]
-g Gapped [T/F]
  default = T
-o alignment output file [File Out]
  default = stdout
-d theor. db size <zero is real size> [Real]
  default = 0
-a SeqAnnot output file [File Out] Optional
-G Cost to open a gap <zero invokes default behavior> [Integer]
  default = 0
-E Cost to extend a gap <zero invokes default behavior> [Integer]
  default = 0
-X X dropoff value for gapped alignment <in bits> <zero invokes default behavior> [Integer]
  default = 0
-W Wordsize <zero invokes default behavior> [Integer]
  default = 0
-M Matrix [String]
  default = BLOSUM62
-q Penalty for a nucleotide mismatch <blastn only> [Integer]
  default = -3
-r Reward for a nucleotide match <blastn only> [Integer]
  default = 1
-F Filter query sequence <DUST with blastn, SEG with others> [String]
  default = T
-e Expectation value <E> [Real]
  
```

Fig. 3.24 bl2seq options

As with blastall, the general command for performing a BLAST2 alignment is:

```
C:\blast>bl2seq -i seq1 -j seq2 -p program -o outputfile
```

where,

-i first sequence

-j second sequence

- p BLASTP for protein and BLASTN for nucleotide sequences
- o output file



3.9 PERFORMING LOCAL BLAST SEARCHES WITH PERL

The `system` function in Perl can be used to execute other programs from within Perl scripts. When external programs are called, the main program is suspended; control returns to the main program after the external program finishes executing.

The syntax for the `system` command is:

```
system("command");
```

where `command` is the actual command that you want to execute. Written like this, the `command` within the double quotes is sent to the system shell for interpretation. It is the same as executing `command` on the system command-line. For example, on DOS, the following line of code will perform `blastall` analysis as exactly stated:

```
system("blastall -p blastn -d ecoli.nt -i lactamase.txt -o lactamase.out");
```

To perform batch `blastall` analyses with multiple sequences in a multiple Fasta file, the `system` command can be used as before except that each sequence has to be retrieved from the file in turn and supplied to the command in a loop.

```
foreach $seq(@sequences) {
    system("blastall -p $program -d $db -i $file -o $file.out");
}
```

where,

<code>\$program</code>	Specifies BLAST program to use
<code>\$db</code>	Specifies database to be used
<code>\$file</code>	file containing a single sequence
<code>\$file.out</code>	Output file for BLAST results



3.10 SEQUENCE ANNOTATION

Sequence annotation is the process of defining the structure and function of a given sequence. For a raw DNA sequence, this may mean defining what genes, if any, are present on it, what their intron-exon structures are and, ultimately, gathering evidence to determine what their function is. For proteins, in a similar fashion, this means understanding the domains that are present on the protein and the function of the protein. The evidence that is needed to ascribe a function to a gene or protein comes from a variety of sources, the most common one being a BLAST search. In general, the steps towards a programmatic approach to annotation (using BLAST) are:

1. Extract the genes/proteins that you need to annotate. If they are in a file, loop through it to create a temporary file and use it to perform a BLAST.
2. Perform a BLAST search against a set of databases such as nr, nt, EST, etc.
3. Parse the output of BLAST and find out what the top hits are.

Taking the example of the hypothetical rice protein from the BAC OSJNBa0058E19 (GI number 13129470):

```
>gi|13129470|Hypothetical protein [Oryza sativa]
MNLIVVQIRKMKSLFLLHSISSKAAMGLWPSARRCRRQMVTPLGGHRSSASGESEQRMFSGGCACRAIDW
MYPKGCMMHGTHRSSDEV RVGLSDDDAEDVPSALYLLHSNRNRRDLVAAVHCVRSGAPAGEVAFFPNHC
MIEAEIRGDGTGIERRRWNTREKETIAAQ
```

a BLASTP search against the nr database gives the following hits:

Sequences producing significant alignments:	(bits)	Value
gb AAK13128.1 AC083945_3 (AC083945) Hypothetical protein [Oryza ...	349	6e-096
gb AAK13125.1 AC080019_17 (AC080019) Unknown protein [Oryza sati...	48	4e-005
ref NP_566398.1 (NM_112002) expressed protein [Arabidopsis thal...	34	0.77
gb AAF02137.1 AC009918_9 (AC009918) unknown protein [Arabidopsis...	34	0.77
ref XP_146511.1 (XM_146511) similar to Circadian Oscillatory P...	33	1.0
gb AAF63493.1 AF239684_1 (AF239684) polymerase [green turtle her...	33	1.7
dbj BAB21235.1 (AP002953) hypothetical protein [Oryza sativa (j...	33	1.7
ref NP_037202.1 (NM_013070) utrophin (homologous to dystrophin)...	30	8.5
gb AAF87665.1 AF223648_1 (AF223648) esterase [uncultured bacterium]	30	8.5

Looking at the output, it appears that the most significant hits (lowest E values 6e-096, 4e-005) are for hypothetical proteins or unknown proteins, which is not of great use to us in identifying the function of the rice protein. The first protein that has a “known function” appears to be “similar to Circadian Oscillatory Protein” (XP_146511.1, highlighted).

If you look further down, where the sequence alignments are provided, you will see that this is a protein that has been annotated as “similar to Circadian Oscillatory Protein (SCOP)” and is from *Mus musculus* (mouse):

```
>ref|XP_146511.1| (XM_146511) similar to Circadian Oscillatory Protein (SCOP) [Mus musculus]
Length = 1177
Score = 33.5 bits (75), Expect = 1.0
Identities = 24/78 (30%), Positives = 33/78 (41%), Gaps = 6/78 (7%)
Query: 83 SSDEV RVGLSDDDAEDVPSALYLLHSNRNRRDLVAAVHCVRSGAPAGEVAFPPNHCM 142
      SS++ GLSDDD + V + R ++ +HC R P P N
Sbjct: 993 SSNQSDNGLSDDD-QPVEGVI-----TNGSRVEVEVDIHCCRGREPESPPLPKNSSNA 1046
Query: 143 EAEIRGDGTGIERRRWNT 160
      +E R G G RR N+
Sbjct: 1047 CSEERARGAGFGIRRQNS 1064
```

You can also go to <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Protein> (Entrez at NCBI) and look up the protein by its ID (XP_146511.1). Note that the E value is not very significant (E value of 1.0—means that this could be just a chance, not a true hit). Other hits are even less significant based on the high E values of 1.7–8.5. A thumb rule is to look at hits with E values < 0.001.

However, it is possible that one of these low significant hits is a true hit. This is where biology comes in. The next step would be to look at these hits and use your knowledge of the protein(s) involved to make a judgement about the putative function of the protein and design biological experiments to determine the actual function. Based on the evidence gathered from such studies, the role of the Bioinformaticist is to provide clues on what the most plausible function of a given hypothetical protein could be. The Bioinformaticist also looks at other supporting data, for example, if the gene for this hypothetical protein was available, what does the results of BLASTN vs. the EST database indicate for this particular gene? Does it give more information that is not available with BLASTP?

Since ESTs are derived from genes that are expressed, significant hits to ESTs is a strong indication that the protein is expressed in a given organism. This is important because many of these “hypothetical proteins” are the result of gene prediction programs such as GenScan, and GenScan makes mistakes in prediction. It is quite possible that the hypothetical protein is a wrong prediction and that it doesn’t really exist in nature. If all you get is hits to other hypothetical proteins (which themselves could be predicted by a prediction program) and no hits at all to ESTs, that is an indication that this could possibly be a spurious prediction. We will learn more about gene prediction in the next chapter.

Note that the exercise of annotation requires some insight into the protein that you are studying and also involves some subjective analysis on your part. There is also a ‘manual’ part involved, in that, at some level you have to look at the alignments and the evidence to make a judgement. What you can do with Perl is reduce the manual portion to a minimum by parsing the BLAST output further.



For example, write a Perl program to extract sequences to be annotated, BLAST them against a set of databases, parse the BLAST output file to examine the most significant hits and generate a report for each protein analyzed. Large Biotech firms have developed a whole “annotation pipeline” that does this on a regular basis in a high-throughput fashion on all sequences of interest.



Assignments

1. Retrieve the GenBank record for the BACs OSJNBa0058E19 and OSJNBa0094H10 from PubMed. Write a script that does the following:
 - extracts all protein sequences that have been annotated as hypothetical (having no known function). These are marked /product=“Hypothetical protein”
 - performs a local BLASTP and TBLASTN against the databases nr and EST (For ESTs, download the file est_others.tar.gz - this contains all non-human and non-mouse ESTs from ftp://ftp.ncbi.nih.gov/blast/db/FormattedDatabases/. This is a formatted database so you do not have to run formatdb on it.)
 - extracts ALL the sequences producing significant alignments

Use this information to arrive at the best possible annotation for the hypothetical proteins.

2. Write a script that automatically retrieves from NCBI protein sequences corresponding to a given set of GenBank IDs and runs a local BLASTP against nr to arrive at a plausible annotation. Use the enclosed file of GenBank IDs for the assignment.





Web-based Sequence Analysis: Gene Prediction



4.1 INTRODUCTION

Gene prediction and annotation are fundamental aspects of genome sequencing projects. These activities involve determination of complete gene structures from the raw DNA sequence and attributing functions to them, most commonly, by way of computational methods. Specifically, these processes try to understand how the various structural elements such as coding, non-coding and regulatory elements are organized within genes.

To make predictions about gene structure, gene prediction programs are designed to recognize genetic signals that are embedded in DNA sequences. Some of these signals are: promoters, splice sites, exons, introns, transcription start and end points, poly-adenylation sites, CpG islands and translation start and stop sites. Some of the terminology associated with biology is presented as follows:



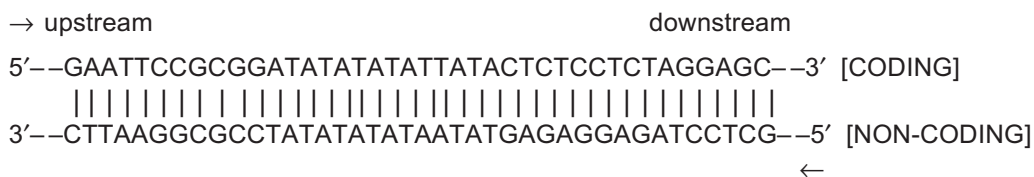


4.2 TERMINOLOGY AND CONCEPTS

DNA structure: DNA is composed of monomeric units called nucleotides. A DNA molecule is, therefore, a 'polynucleotide' polymer composed of a long chain of nucleotides. Each nucleotide is made up of a sugar called deoxyribose, a nitrogen-containing base attached to the sugar, and a phosphate group. There are four types of nucleotides found in DNA, differing only in the composition of the nitrogenous base: Adenine (A), Guanine (G), Cytosine (C) and Thymine (T). A and G are the purine bases while C and T are the pyrimidine bases.

DNA double-helix: DNA is actually composed of two such polynucleotide strands held together by base pairing between the nucleotides. The upper strand is called the coding strand and the lower or complimentary strand is called the non-coding strand.

The pairing rules are that A binds to T and C binds to G so that a double-stranded (ds) DNA molecule can be represented as a linear chain of nucleotides paired according to the rules above:



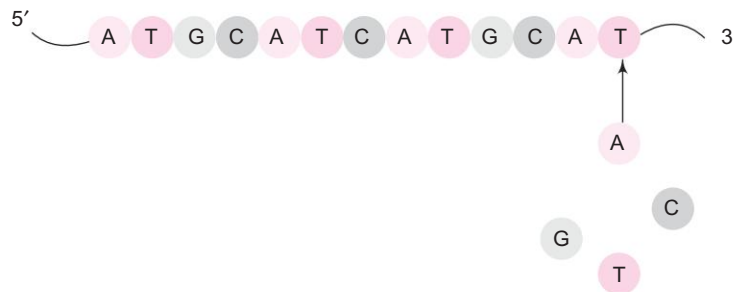
Note, however, that this does not indicate the nature of the bonding between the pairs of nucleotide and is only a schematic representation. In reality, A forms two hydrogen bonds with T on the opposite strand, and G forms three hydrogen bonds with C on the opposite strand, meaning also that greater energy is required to break a G-C bond than an A-T bond.

Also, the two DNA stands are not linear as shown above. Instead, they are entwined with each other, forming a right-handed helical structure, much like a spiral staircase. The two polynucleotide chains run in opposite directions and this is indicated by the 5' and 3' notation on the two strands above.

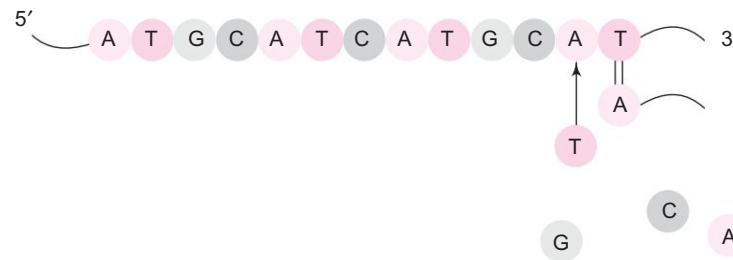
In relation to their location on the strands, elements within DNA are referred to as upstream and downstream. Upstream elements refer to sequences closer to the 5' end of the DNA and downstream elements refer to sequences closer to the 3' end of the DNA. In the above schematic, the EcoRI enzyme restriction site (GAATTC) is said to be upstream of the TATA element.

DNA synthesis: By convention, DNA synthesis always proceeds in the 5' → 3' direction. Figure 4.1 shows the step-wise addition of the four nucleotides in the 5' to 3' direction (Steps 1–3) until synthesis is complete (Step 4).

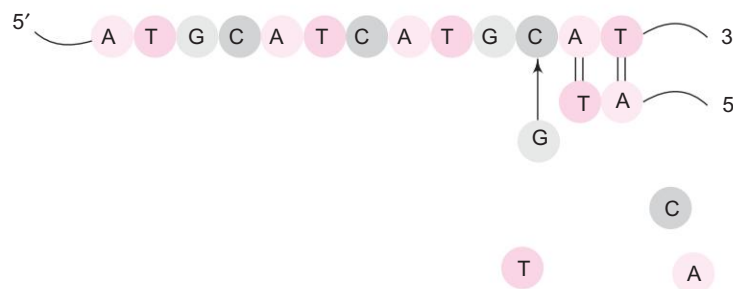
Step 1:



Step 2:



Step 3:



Step 4:

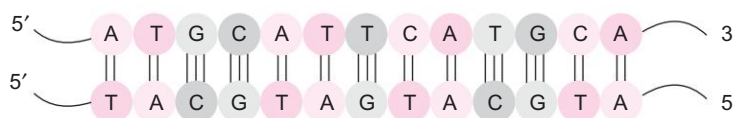


Fig. 4.1 DNA synthesis

Transcription and translation: Transcription is the process by which a DNA molecule is copied into an RNA molecule. Translation is the process by which the RNA sequence is used by the cellular machinery to synthesize a protein.

Transcription may result in one of three types of RNA: Messenger RNA (mRNA), transfer RNAs (tRNA) or ribosomal RNA (rRNA). mRNA molecules serve as 'messengers' that specify the code for the synthesis of amino acids (during translation) and, therefore, the name messenger RNA. tRNAs form covalent attachments to individual amino acids and recognize the encoded sequences of the mRNAs to allow correct insertion of amino acids into the elongating polypeptide chain during translation. rRNAs are assembled together with numerous proteins to form complexes known as ribosomes. Ribosomes engage mRNAs and form a catalytic domain into which the tRNAs enter with their attached amino acids. The proteins of the ribosomes catalyze all of the functions of polypeptide synthesis.

During the process of transcription, the DNA double helix unwinds and one strand serves as the template for the synthesis of the RNA strand. Either strand can serve as the template—which strand becomes the template depends on a combination of transcription initiation and termination signals (such as promoter and enhancer sequences) that are present on the DNA. Transcription is actually a polymerization reaction in which individual nucleotides are linked together by an enzymatic reaction (catalyzed by the enzyme RNA polymerase) into a chain to form another polymer: the RNA.

RNA structure: RNA, like DNA is a polymer composed of four nucleotides. The difference between RNA and DNA is the nature of the sugar moiety: RNA has the ribose sugar, while DNA has the deoxyribose sugar. RNA has the same purine bases as DNA: adenine (A) and guanine (G), and the same pyrimidine cytosine (C), but instead of thymine (T), it uses the pyrimidine uracil (U). The same base pairing rules apply so that the appropriate nucleotide is added based upon the nucleotide on the DNA template.

CpG islands: Regions within DNA that often occur near the beginning of genes, where the frequency of the dinucleotide CG (that is, the nucleotide bases cytosine and guanine) is more than in the rest of the genome.

Introns and Exons: Higher organisms (eukaryotes) have what are called split genes, that is, a large proportion of their genes are not continuous linear entities, but may be interrupted throughout their length by sequences that do not code

for protein. A piece of DNA may, therefore, contain coding sequences with intervening non-coding sequences. The intervening non-coding segments are called the introns and do not code for protein. The coding sequences are exons and do code for protein. For example, the cystic fibrosis transmembrane regulator (CFTR) gene's coding regions (exons) are scattered over 250,000 base pairs of genomic DNA and is made up of 27 exons. During transcription, introns are removed from the CFTR gene and exons are pieced together by a process known as RNA splicing to form a 6100-bp mRNA transcript that is translated into the 1480 amino acid sequence of CFTR protein. In contrast, the 384 nucleotide human pancreatic ribonuclease gene is intronless and codes for a 128 amino acid protein. A highly schematic view of the RNA splicing process is shown in Figure 4.2.

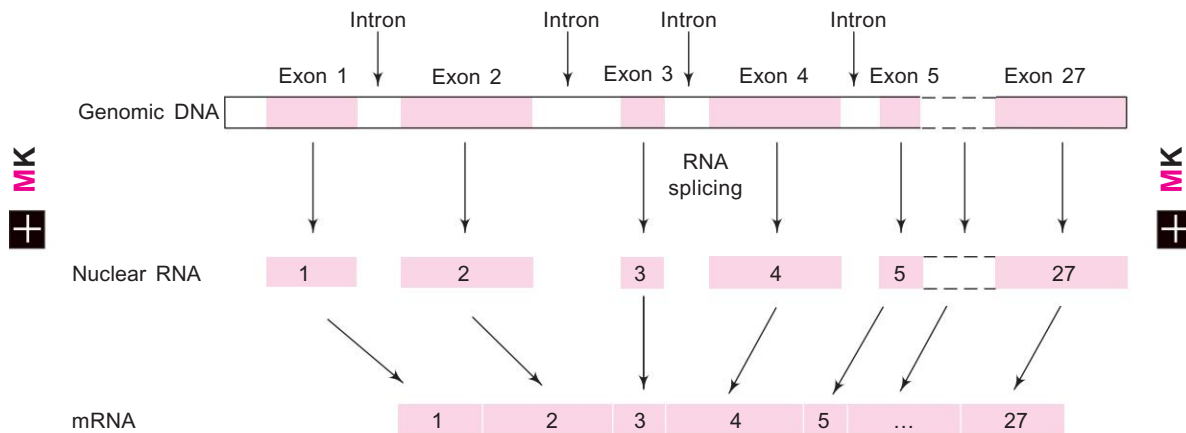


Fig. 4.2 RNA splicing

4.3 GENE PREDICTION PROGRAMS

There are a large number of gene prediction programs available today (Table 4.1, not an exhaustive list). Most of these are able to correctly identify nucleotides that encode proteins 90 per cent of the time or identify exact exon boundaries (70–75 per cent accuracy). However, they are relatively poor at correctly identifying complete gene structures (50 per cent accuracy). We are far from achieving absolute accuracy in computational gene prediction largely because our understanding of complex underlying genetic processes is far from adequate.

Table 4.1 *Gene/exon prediction programs*

Name	URL	Organization
1 Gene Recognition and Assembly Internet Link Version 1.3	http://compbio.ornl.gov/Grail-1.3/	Oak Ridge National Laboratory (ORNL)
2 GeneMark	http://opal.biology.gatech.edu/GeneMark	Georgia Institute of Technology
3 GenScan	" http://genes.mit.edu/GENSCAN.html "	Stanford/MIT
4 Glimmer/ GlimmerM	http://www.tigr.org/software/glimmer/	The Institute of Genomic Research (TIGR)
5 NetGene2	http://genome.cbs.dtu.dk/services/NetGene2/	Technical University of Denmark
6 HMMgene	http://genome.cbs.dtu.dk/services/HMMgene/	Technical University of Denmark
7 MZEF	http://argon.cshl.org/genefinder/	Cold Spring Harbor Laboratory (CSHL)
8 GeneParser	" http://beagle.colorado.edu/~eesnyder/GeneParser.html "	University of Colorado
9 Genie	" http://www.fruitfly.org/seq_tools/genie.html "	Lawrence Berkeley National Laboratory
10 FGeneH	http://genomic.sanger.ac.uk/gf/gf.shtml	Sanger Center

Consequently, the most effective strategy towards gene identification in unknown DNA sequences is an approach where the results of prediction programs are combined with the results of similarity or database homology searches, matches to ESTs, etc. In addition, multiple gene/exon prediction programs are generally used to minimize the possibility of false positive predictions—for example, the validity of a prediction is in question when only one of a set of programs predicts the existence of a certain exon or exons. This can arise purely by error on part of the prediction program.

The approach where results of multiple gene prediction programs are combined with the results of similar searches is not without problems either. Consider the case where a sequencing experiment gives a piece of DNA that has no known homologs. In such cases, gene prediction methods that rely only on information that is encoded in the sequence can be used. These are called *Ab initio* (Latin: from the beginning) gene prediction programs and use signals within DNA such as splice sites, start and stop codons, promoters and terminators of transcription, polyadenylation sites, ribosomal binding sites, CpG islands, and various transcription factor binding sites. *Ab initio* methods such as GenScan rely on probabilistic models known as Hidden Markov Models (HMMs) to discern patterns within DNA. Others such as GRAIL use neural networks for gene prediction. Neural networks form an information-processing paradigm based on the densely interconnected, parallel structure of neurons in the mammalian brain. Neural networks are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. Neural networks are composed of a large number of highly interconnected processing elements that are analogous to neurons and are tied together with weighted connections that are analogous to synapses. In the case of GRAIL, seven separate sensor algorithms, each designed to provide the coding potential of a given piece of DNA, form the core of the system. A neural network then integrates the information from the sensors and predicts the locations of coding regions. An HMM, on the other hand, models the states that a DNA sequence can exist in and the transition probabilities between the states. The different states are promoter, intron, exon, etc. The term 'Hidden' comes from the fact that the sequence itself is visible but the states are hidden.

4.4 GENSCAN

To date the most effective among the many prediction programs are the exon prediction programs. For the purpose of illustration, we will focus on one such program called GenScan.

GenScan was written by Chris Burge and Samuel Karlin at the Department of Mathematics, Stanford University. GenScan utilizes the same basic signals described earlier to build complete gene structures (that is, introns + exons) from human genomic sequences. Specifically, these include transcriptional,

translational and splicing signals (including elements present in most eukaryotic promoters such as the TATA box and cap site), as well as length distributions and compositional features of exons, introns and intergenic regions. Importantly, GenScan also makes use of the many substantial differences in gene density and structure based on GC composition of the human genome. For example, it is known that gene density in GC rich regions is five times higher than in regions with moderate GC content and 10 times higher in AT rich regions. Four categories of DNA were identified based on their GC content:

1. < 43 % GC
2. 43–51 % GC
3. 51–57 % GC
4. > 57 % GC

These are known as isochores. Thus, if the input genomic sequence has a GC content of 45 per cent, it is said to have an isochore value of two.

A functional classification of the various gene prediction methods along with the underlying algorithms they use is given in Table 4.2.

Table 4.2 *A functional classification of gene prediction methods*

Ab initio: HMM methods

FGENEH	http://genomic.sanger.ac.uk/gf/gf.shtml
Genie	http://www.fruitfly.org/seq_tools/genie.html
GeneID	http://www1.imim.es/geneid.html
GeneMark	http://genemark.biology.gatech.edu/GeneMark/eukhmm.cgi
GenScan	http://genes.mit.edu/GENSCAN.html
HMMGene	http://www.cbs.dtu.dk/services/HMMgene/

Ab initio: Neural network methods

GRAIL	http://compbio.ornl.gov/Grail-1.3/
NetGene2	http://www.cbs.dtu.dk/services/NetGene2/

Homology based

Genewise	http://www.sanger.ac.uk/Software/Wise2
Procrustes	http://www-hto.usc.edu/software/procrustes/index.html

Ab initio programs, traditionally, have been poor at predicting genes in regions containing multiple genes, especially when present on both DNA strands.

GenScan addresses these problems by using an explicitly double-stranded genomic sequence model which has the likelihood of genes occurring on both DNA strands. Second, while most programs assume the presence of exactly one complete gene in the input sequence, GenScan treats the more general case in which the sequence may contain a partial gene, a complete gene, multiple complete (or partial) genes on either strand, or no gene at all. Another significant difference in GenScan is the incorporation of splice donor signal information based on the mechanism of donor splice site recognition in pre-mRNA sequences by U1 small nuclear ribonucleoprotein particle (U1 snRNP).

Notes

1. U1 snRNP is an important component of the RNA splicing machinery and is the first splicing factor to contact the pre-mRNA. After pre-mRNA binding, the U1 snRNP interacts with other RNAs and proteins to form a bridge that brings the ends of the intron together for splicing. The removal of the intron brings the two neighboring exons together; these are subsequently pieced together to form one continuous sequence.
2. Most introns start with the sequence GU and end with the sequence AG and are referred to as the splice donor and splice acceptor sites, respectively.

4.5 RUNNING GenScan ANALYSES

Running and interpreting a GenScan analysis is rather straightforward. Point your browser to the GenScan server at MIT: <http://genes.mit.edu/GENSCAN.html> (Figure 4.3). For this exercise, we will use a 175 kilobase human BAC with the accession number AC092818. Download the BAC and save it on your computer as AC092818.txt. GenScan has been 'trained' to work with vertebrate, arabidopsis and maize sequences (Figure 4.4). Since we are analyzing a human BAC, we choose the vertebrate option. We will use the default sub-optimal exon cutoff value of one for our purposes. This value defines the threshold which determines whether exons that do not meet the criteria (sub-optimal exons) will be displayed or not.

You can assign a sequence name if you are analyzing a large number of sequences and want to label each output by a unique identifier. In this case, we will just use the BAC accession number (Figure 4.5). The program gives us

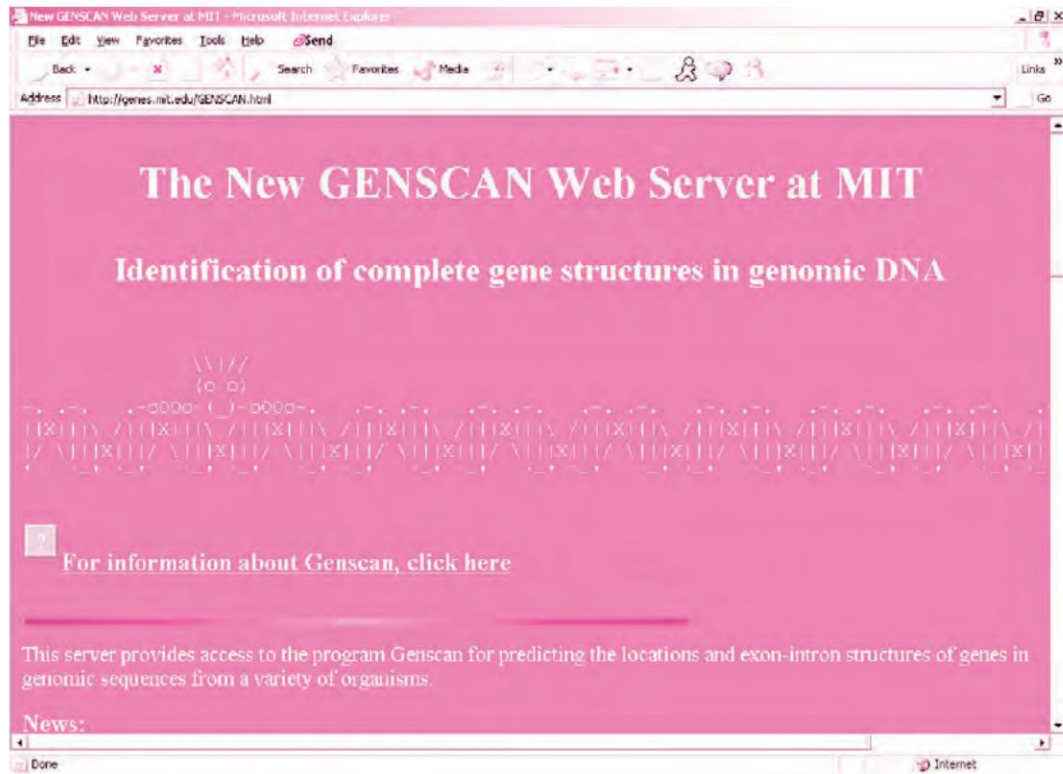


Fig. 4.3 The GenScan web server

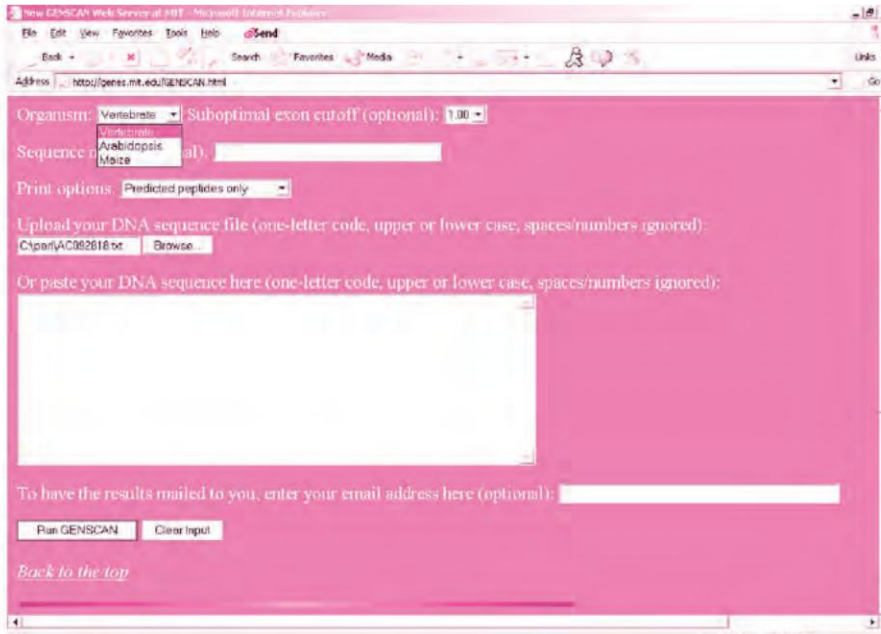
an option of printing the predicted proteins alone or the predicted proteins with their nucleotide sequences. We will choose the latter option (Figure 4.6).

The sequence can be either uploaded or pasted directly in the text box. Uploading a sequence is more convenient if you are handling very large sequences, as is the case here (Figure 4.7). Finally, you can specify an email address if you want to receive the results via email. This is usually the case with large sequences which may take a while to process. In this case, we will hit the “Run GenScan” button and wait to see the results in the browser (Figure 4.8). The results of this analysis are enclosed as a text file (AC092818gsn.txt).



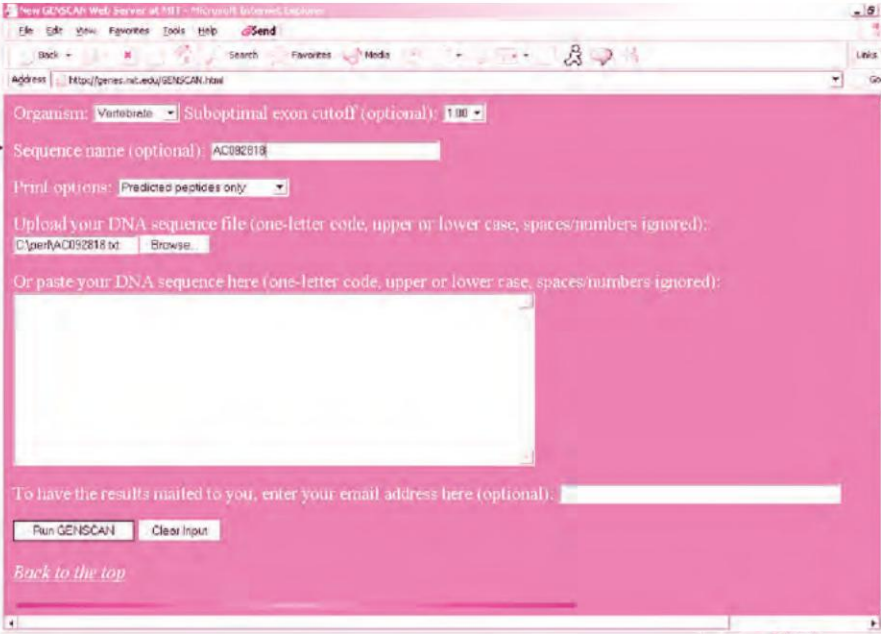
4.6 ANALYZING GENSCAN OUTPUT

The GenScan header gives information on the input sequence and the



A screenshot of a web browser window displaying the GenSCAN web server interface. The browser's address bar shows the URL <http://genes.mit.edu/GENSCAN.html>. The page has a pink background. At the top, there is a navigation bar with links like "File", "Edit", "View", "Favorites", "Tools", and "Help". Below this, the "Organism" dropdown menu is set to "Vertebrate", and the "Suboptimal exon cutoff (optional)" is set to "1.00". The "Sequence name (optional)" field is empty. The "Print options" dropdown is set to "Predicted peptides only". There are two input sections: "Upload your DNA sequence file (one-letter code, upper or lower case, spaces/numbers ignored):" with a "Browse..." button, and "Or paste your DNA sequence here (one-letter code, upper or lower case, spaces/numbers ignored):" with a large text area. At the bottom, there is an email input field with the placeholder text "To have the results mailed to you, enter your email address here (optional):", and two buttons: "Run GENSCAN" and "Clear Input". A link "Back to the top" is also present.

Fig. 4.4 Setting GenScan parameters



A screenshot of the same GenSCAN web server interface as in Figure 4.4, but with the "Sequence name (optional)" field now filled with the text "AC092818". All other elements, including the organism dropdown, suboptimal exon cutoff, print options, input fields, and buttons, remain the same.

Fig. 4.5 Entering an identifier

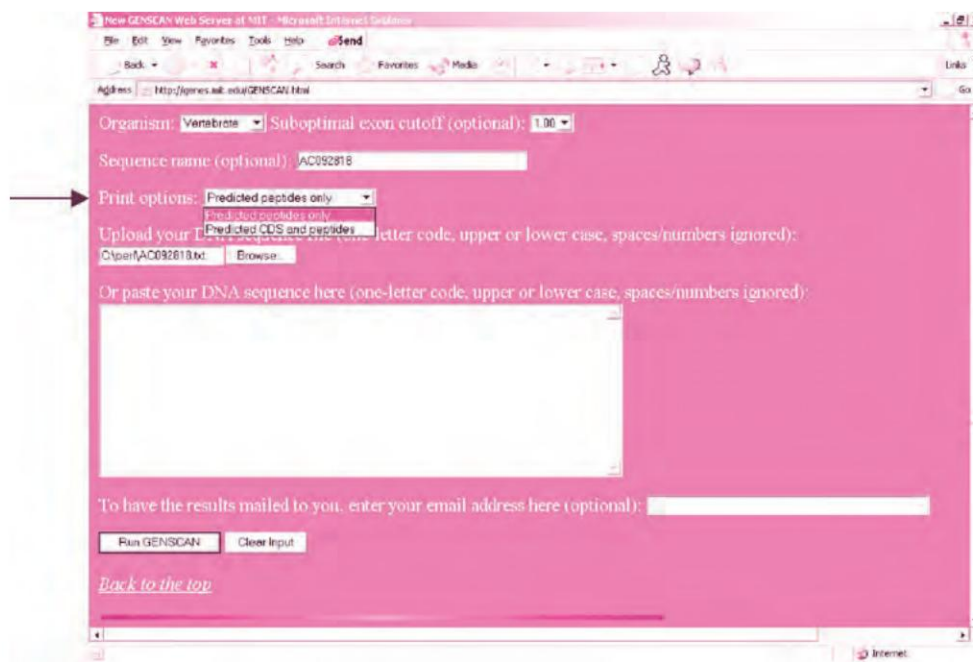


Fig. 4.6 Printing peptides and the corresponding coding sequences (CDS)

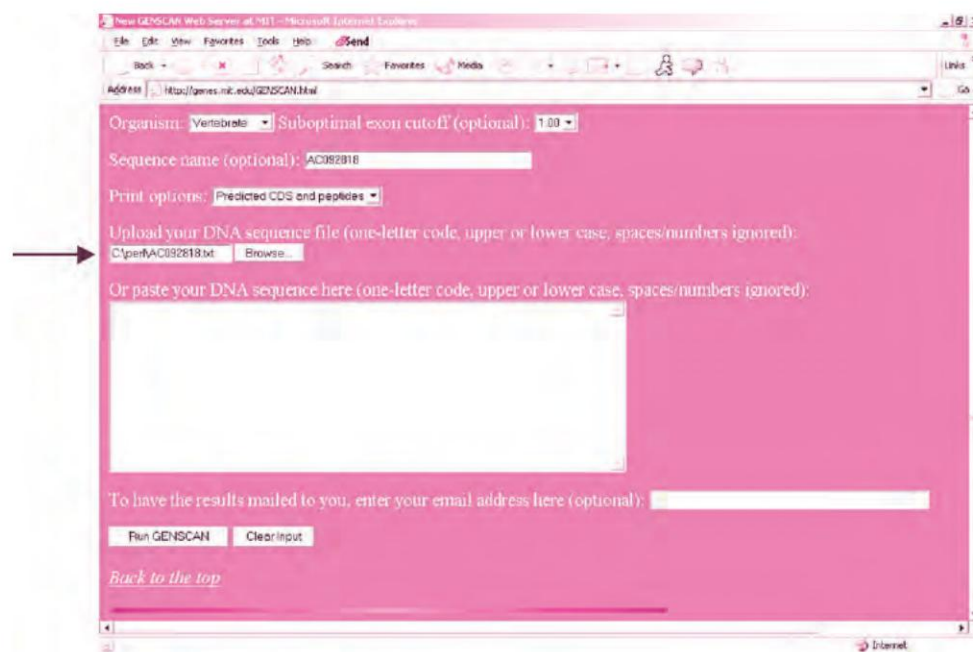


Fig. 4.7 Uploading the BAC sequence

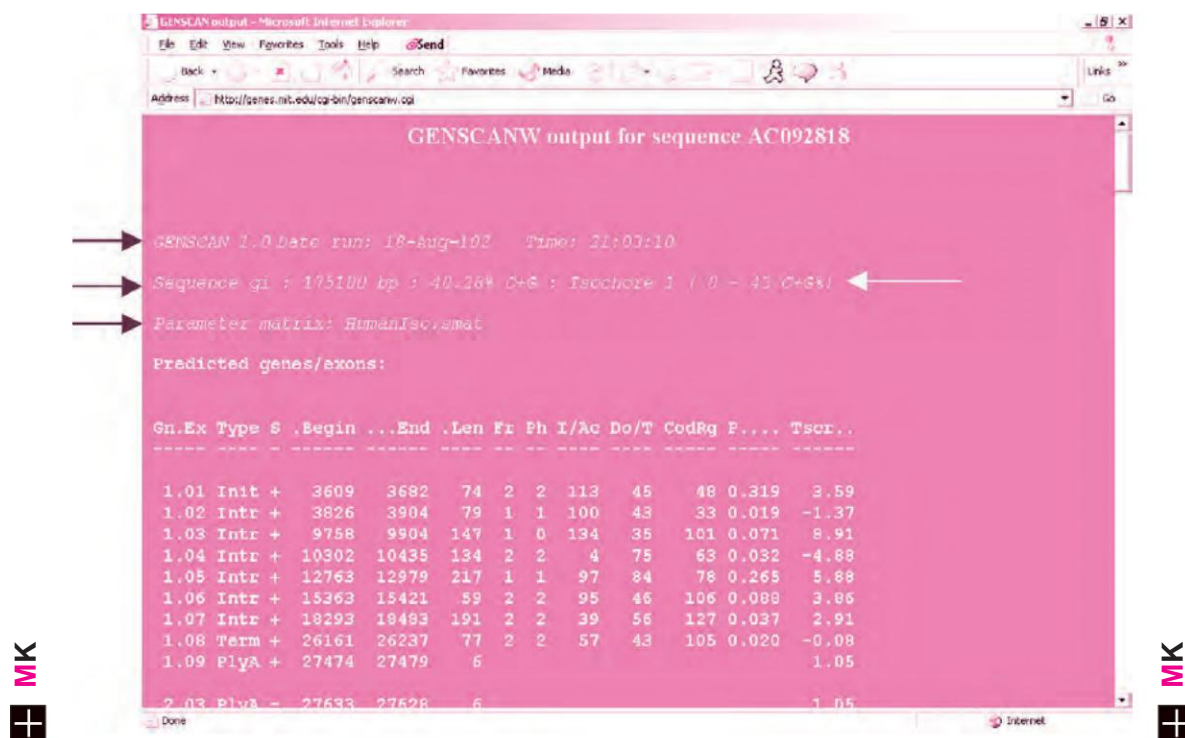


Fig. 4.8 GenScan output: Header information

parameters used, for example, name, size and isochore classification of the sequence, and the matrix used for the analysis (HumanIso.smat).

The body of the analysis consists of the predicted peptide and the corresponding CDS sequences. As is evident from the output, there were eight predicted peptides in this sequence. The complete gene structure of each peptide is listed after the header (Table 4.3).

Table 4.3 Gene structures

Gn.Ex	Type	S	.Begin	...End	.Len	Fr	Ph	I/Ac	Do/T	CodRg	P....	Tscr..
1.01	Init+		3609	3682	74	2	2	113	45	48	0.319	3.59
1.02	Intr+		3826	3904	79	1	1	100	43	33	0.019	-1.37
1.03	Intr+		9758	9904	147	1	0	134	35	101	0.071	8.91

(Contd.)

Table 4.3 (Contd.)

1.04	Intr+	10302	10435	134	2	2	4	75	63	0.032	-4.88
1.05	Intr+	12763	12979	217	1	1	97	84	78	0.265	5.88
1.06	Intr+	15363	15421	59	2	2	95	46	106	0.088	3.86
1.07	Intr+	18293	18483	191	2	2	39	56	127	0.037	2.91
1.08	Term+	26161	26237	77	2	2	57	43	105	0.020	-0.08
1.09	PlyA+	27474	27479	6							1.05

```

>gi|GENSCAN_predicted_peptide_1|1325 aa
MALISFTSTFNFICKKSWQCITEAGFDKVDETIIFVISQSSRNVI VGEFLQDPCQGLPLL
KDLSSKQAANLEFWQMEAVACDILLIMQFEGQPAFLQGMSSRLSGAAEQVGSWSMRSQ
RHSLLWSVPEFVQAGFLFPEALQSAGCFLEPNIGLQVLOQFWTLGLTSVVCQGLSGLWPQ
LEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSLSDSKAAVLENIGLLPVTHL
SECSRGSTQTGISGLKTELGAQKVARVCCQAEYGGESHAEREFWTFTEESLRVYKRLISSA
SGISVDEGSLPGLTTRTFEGYEP

>gi|GENSCAN_predicted_CDS_1|1978 bp
atggcctaatacagttttacatctcgttttaattttattggaagaagagctggcaatgc
atcacagaggdcggcctttgacaaagtggatgaaacaattatcttctgttatcagccaaagc
agttagaaatgtgatagttggggaatttttcaggacccatgcagggcttaacctctgcta
aaggattttgtctctcaaaagcaggcagcaaatctgttcccttggcagagcgtggaagccgtg
gctttgtgacattctctgataatgcagccagggccagccagccagcatttctgcagggg
atgcactccagcctcagtggggcagcagcagcagcagcagcagcagcagcagcagcagcagc
cgtcattctctgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgt
gaagccctccaaagtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgt
ttttggactctttggaattacatcagtggtttgcccaggactctcaggcctttggcctcag
attgaaggtgtgcaatgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgtgt
ttgtctctcagcttgcagacagcctattgtgggacttcaacttggatcattccagcagc
ctttcggtattccaaagcggctgtctgtgaaatatagggtctcttccactaaaccacctc
tctgaatgcagcagaggtggaaccacagacagggatcagtggtttaaagacagagctggga
gccaaggtagccagcagctttgcagcagcagcagcagcagcagcagcagcagcagcagcagc
ttctggaacactaccaggaatctcttccagcagcagcagcagcagcagcagcagcagcagcagc

```

Fig. 4.9 GenScan output II: predicted sequences

The most important aspects of this table are the gene and exon number, the type of exon, the strand information (+/-), the start and end positions, the length of each exon in basepairs, the frame and the scores. The key to the abbreviations is provided at the end of the output (Table 4.4).

Table 4.4 *Abbreviations and explanations*

Gn.Ex	: gene number, exon number (for reference)
Type	: Init = Initial exon (ATG to 5' splice site) Intr = Internal exon (3' splice site to 5' splice site) Term = Terminal exon (3' splice site to stop codon) Sngl = Single-exon gene (ATG to stop) Prom = Promoter (TATA box / initiation site) PlyA = poly-A signal (consensus: AATAAA)
S	: DNA strand (+ = input strand; – = opposite strand)
Begin	: beginning of exon or signal (numbered on input strand)
End	: end point of exon or signal (numbered on input strand)
Len	: length of exon or signal (bp)
Fr	: reading frame (a forward strand codon ending at x has frame $x \bmod 3$)
Ph	: net phase of exon (exon length modulo 3)
I/Ac	: initiation signal or 3' splice site score (tenth bit units)
Do/T	: 5' splice site or termination signal score (tenth bit units)
CodRg	: coding region score (tenth bit units)
P	: probability of exon (sum over all parses containing exon)
Tscr	: exon score (depends on length, I/Ac, Do/T and CodRg scores)

Each pair of peptide and CDSs are in Fasta format and have unique identifiers where the sequences are numbered sequentially.

```
>gi|GENSCAN_predicted_peptide_1|325_aa
MALISFTSPFNFIGKKSQWQCITEAGFDKVDETIIFVISQSSRNIVGEFLQDPCQGLPLL
KDLSSKQAANLFPWQRMEAVACDILLMQPGHGQPAFLQGMSSRLSGAAEQVGSWSMRSQ
RHSLLWSVPEPVQQAGFLFPEALQSAGCFLPSNIGLQVLQFWTLGLTSVVCQGLSGLWPQ
IEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSSLSDSKAAVLENIGLLPLTHL
SECSRGGTQTGISGLKTELGAQVARVCQAEGGESHAEREFWTPTEESLRVYKRGGLISSA
SGISVDHGSLPEGLTKTFIPEGYEP
```

```
>gi|GENSCAN_predicted_CDS_1|978_bp
atggccctaatacagttttacatctccgttaattttattggaaagaagagctggcaatgc
atcacagaggccggccttgacaaagtgatgaaacaattatctcgttatcagccaaagc
agtagaaatgtgatgtggggaattttgcaggacccatgccagggttacctctgcta
```



```

aaggattgtcctcaaagcaggcagcaaatctgttccttggcagaggatggaagccgtg
gcttgtacattctcctgataatgcagccaggccacgggcagccagcatttctgcagggg
atgagctccaggctcagtggggcagcagagcaagtggggagctggtccatgaggagtcat
cgtcattcctgtgtgtgtctgttctgaaccagtccaacaggctggcttctgttcca
gaagccctccaaagtgtggatgcttctgccatcgaaacattggactccaagttcttcag
ttttggactcttggacttacatcagtggtttccaggggactctcaggcctttggcctcag
attgaaggctgcactgtcggtctctacttttgaggttttgggactcggactggcttcc
ttgctcctcagcttgcagacagcctattgtgggacttcaccttgatcattccagcagc
ctttcgattccaaagcggctgtcctggaaaatatagggctccttccactaaccacctc
tctgaatgcagcagaggtggaaccagacagggatcagtgggttaaagacagagctggga
gccaaggtagccagagtttccaggcagagtatggcggagagagccacgcagagagagaa
ttctggacacctacggaggaatctcttcgagtataaaaaggaggactgatcagcagtga
tcaggatatctgttgatcatggttctttacccgaaggactgactaaaacctttattcct
gaagggtatgaaccatag

```

MK



4.7 GENSCAN ANALYSIS WITH LWP::USERAGENT

MK



LWP::UserAgent is a Perl module that is used to send requests to specific applications on the World Wide Web. This module, along with HTTP::Request.pm and other modules, forms the core of the libwww-perl library. We saw an example of another module in the library when we used the LWP::Simple module. Here, we will learn how to use LWP::UserAgent to perform a GenScan analysis over the World-Wide Web.

The LWP::UserAgent module fully exposes the object oriented capabilities of Perl. As with other object oriented programs, the first step is the creation of an object of the type LWP::UserAgent. This is done through the 'constructor' which simply creates a new instance of the LWP::UserAgent object using the **new** keyword. All this means is that to use an object oriented module, we have to create an object of the type LWP::UserAgent (in this case), before we can use its methods:

Step 1: Create an object of type LWP::UserAgent:

```
$ua = new LWP::UserAgent;
```

Step 2: Create an instance of `HTTP::Request` encoding the GenScan request. We use the `new` keyword to create an object of type `HTTP::Request`. One key difference between the `LWP::Simple` and the `LWP::UserAgent` module that we have used above is in the way we have formulated the request.

With `LWP::Simple`, the request is created directly and the various parameters are visible in the URL. For a BLAST2 operation, for example:

```
$url = "http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?program=blastp&matrix=BLOSUM62&one=$gbid1&two=$gbid2&Action=submit";
```

where, `$gbid1` and `$gbid2` are the two gene IDs.

In contrast, with the `LWP::UserAgent` module, the data is sent as part of the HTTP request. The information doesn't appear in the URL and, therefore, is more 'secure'. In addition, it allows a greater number of parameters to be set. The code for the instantiation step is as follows:

```
$request = new HTTP::Request (POST=>'GenScan URL');
```

We can use the GenScan server at MIT for this code:

```
http://genes.mit.edu/cgi-bin/GenScanw.cgi
```

Next, we formulate the request and specify the various parameters we want to use:

```
$request->content("parameters");
```

The parameters are:

```
Content => ['-o' => "$organism",
            '-e' => "$value",
            '-n' => "$name",
            '-p' => "$option",
            '-u' => [$file],      #filename of sequence, OR
            #'-s' => "$seq",      #the sequence itself
            ]
```

where,

```
$organism = Vertebrate/Arabidopsis/Maize (Matrix)
```

`$value` = Cutoff E value
`$name` = Arbitrary sequence name
`$option` = Print options (Predicted peptides OR Predicted peptides and CDS)

We also need to specify another piece of information known as the MIME type or content type. MIME—Multi-purpose Internet Mail Extensions—specify a standard way of classifying file types on the Internet. The purpose of MIME types is to enable Internet programs such as web servers and browsers to transfer files of the same content type in a standardized manner, independent of the underlying operating system. The MIME type enables programs to determine how files of a given type are opened, how they are viewed, etc. A MIME type has two parts: a type and a sub-type. They are separated by a slash (/). For plain text, for example, the MIME type is simple *“text/plain”*. Since we are using the information to plug information into a World Wide Web form, the MIME type we need is:



`'form-data'`

This information is specified as follows:

```
$request->content_type('form-data');
```

The results of the analysis are printed out using the `as_string` method:

```
print $req->as_string;
```

The complete code (with `Getopt::Long` for command-line arguments) is as follows:

```

use LWP::UserAgent;
use Getopt::Long;
use HTTP::Request::Common;

GetOptions("o|organism=s"=>\$organism,    "e|eval=f"=>\$value,
           "n|name=s"=>\$name, "p|option=s"=>\$option, "f|file=s"=>\$file);

my $ua = LWP::UserAgent->new;

$req = $ua->request(POST "http://genes.mit.edu/cgi-bin/GenScanw.cgi",

```



```

Content_Type => 'form-data',
Content => ['-o' => "$organism",
           '-e' => "$evalue",
           '-n' => "$name",
           '-p' => "$option",
           '-u' => [$file],
           #'-s' => "$seq",      #the sequence itself
          ]
);
print $req->as_string;

```

The program can be executed as follows:

```
>GenScan.pl -f AC090419.txt -e 1 -o vertebrate -n testseq -p "predicted
peptides only"
```

The output is shown in Figure 4.10.

Assignments

1. As with BLAST, the process of gene prediction with GenScan can be automated with a Perl script. Download the human BAC AC092818 from NCBI and write a script that sends sequence(s) contained in a local file to the GenScan server, and performs analysis based on the parameters specified by the user on the command line:

```
% GenScan.pl -matrix vertebrate -print peptides -seq AC092818.txt
```

2. The logical next step after gene prediction is determination of the function of each of the predicted peptides and this is most commonly done with BLAST. Extend the previous script to analyze each of the predicted peptides by a BLASTP against the nr database (performed either locally or remotely) using an E value of 0.00001. Arrive at the best annotation for each peptide.

```
% GenScan.pl -matrix vertebrate -print peptides -p Blastp -e 0.00001
-seq AC092818.txt
```

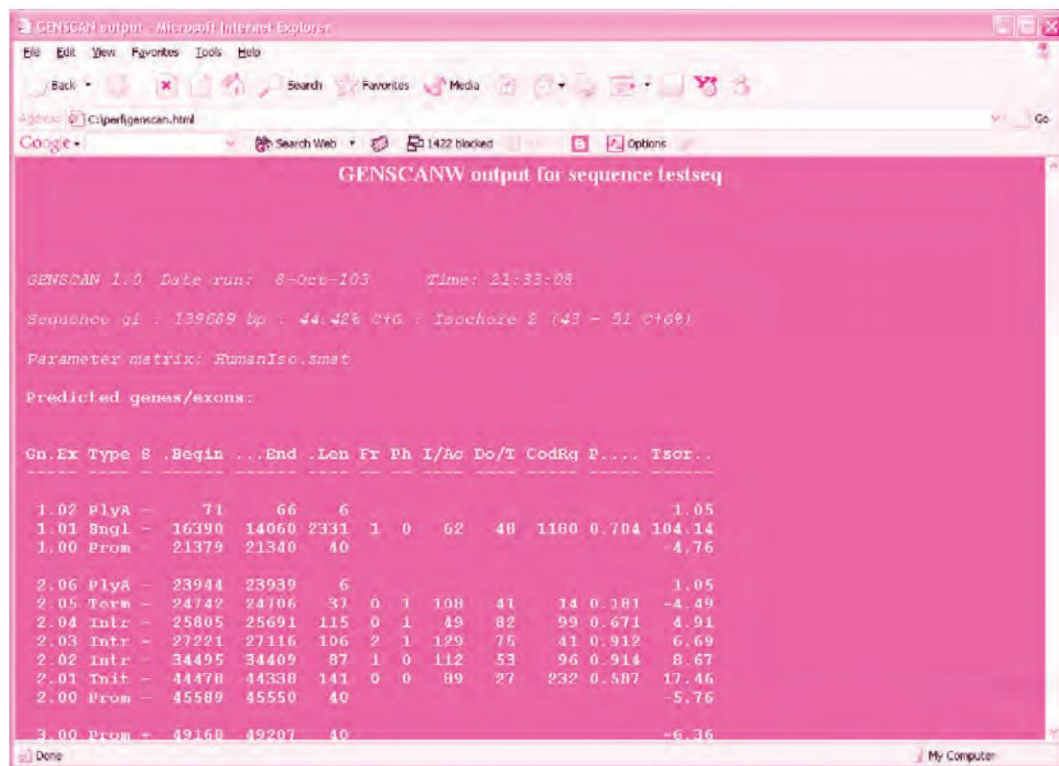


Fig. 4.10 Output of GenScan.pl



Web-based Sequence Analysis: HMMER



5.1 INTRODUCTION

In the previous chapter, we learnt the use of Hidden Markov Models (HMMs) in gene prediction with the GenScan program. HMMs have been applied to other problems in biology as well and have been immensely successful in aiding researchers with Bioinformatics-assisted analyses of biological sequence information. Here, we will understand how HMMs are applied to sequence data to discern relationships between protein families. This is based on a set of programs collectively known as HMMER (pronounced “Hammer”), developed by Sean Eddy and co-workers at the Department of Genetics at the Washington University School of Medicine (St. Louis, MO).



5.2 DOWNLOADING HMMER

The HMMER package can be downloaded (Figure 5.1) from:

<ftp://ftp.genetics.wustl.edu/pub/eddy/hmmer/2.2g/>

A number of distributions of the program are available depending on your platform of choice. Download the version that is appropriate for your operating system. For Mac OS X, for example, download the `hmmmer-2.2g.bin.apple-osx.tar.gz` file. We would be using DOS to run HMMER commands and for this platform, you would need to download `hmmmer-2.2g.bin.dos-cygwin.zip`, expand the compressed file (using WinZip, for example) and save all the files (the executables and the `cygwin1.dll` file) in a directory such as `D:\hmmmer`. The downloaded package and the component programs should appear as shown in Figure 5.2. You may also download, for your reference, the User Guide that comes along with the distribution (Figure 5.1).

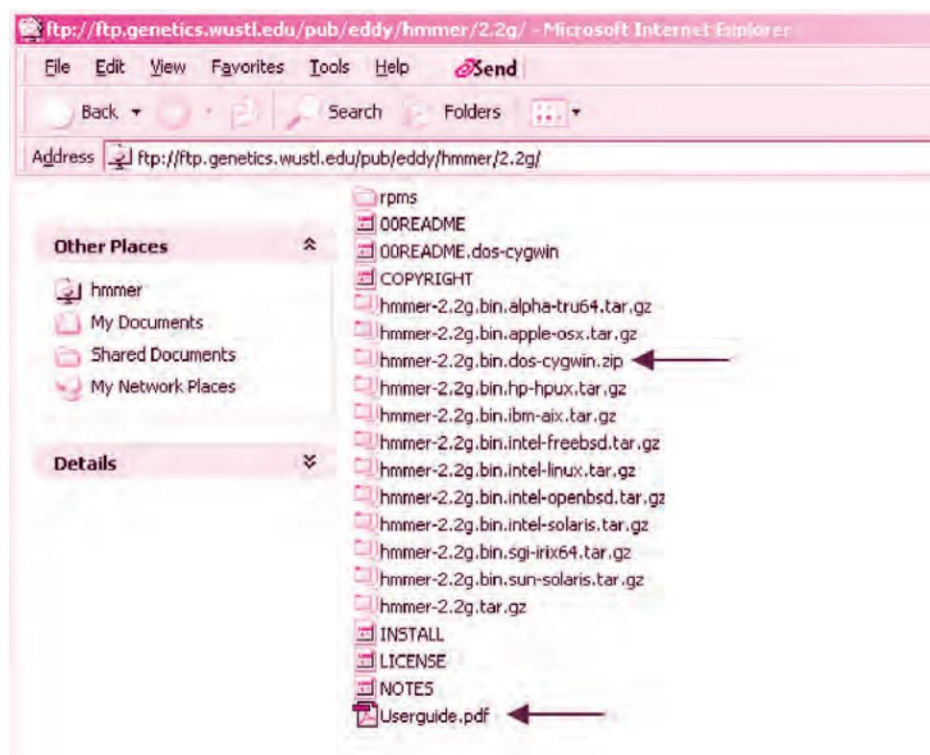


Fig. 5.1 The HMMER ftp site

To use the HMMER programs, you would, in addition, need a multiple sequence alignment program such as ClustalW (Thompson et al., 1994) or ClustalX (Thompson et al., 1997), which provides a windows interface for

```

C:\ Command Prompt
D:\hmmmer>dir
Volume in drive D has no label.
Volume Serial Number is 70AE-6E52

Directory of D:\hmmmer

09/14/2002  10:49 AM    <DIR>          .
09/14/2002  10:49 AM    <DIR>          ..
01/08/2002  12:35 PM             340,900  afetch.exe
01/08/2002  12:35 PM             344,484  alistat.exe
11/12/2001  01:34 PM             731,464  cygwin1.dll
01/08/2002  12:35 PM             561,976  hmmalign.exe
01/08/2002  12:35 PM             612,328  hmmbuild.exe
01/08/2002  12:35 PM             571,133  hmmcalibrate.exe
01/08/2002  12:35 PM             559,379  hmmconvert.exe
01/08/2002  12:35 PM             561,427  hmmemit.exe
01/08/2002  12:35 PM             556,819  hmmfetch.exe
01/08/2002  12:35 PM             558,355  hmmindex.exe
01/08/2002  12:35 PM             567,095  hmmfam.exe
01/08/2002  12:35 PM             568,119  hmmsearch.exe
01/08/2002  12:35 PM             342,470  seqstat.exe
01/08/2002  12:35 PM             348,598  sfetch.exe
01/08/2002  12:35 PM             345,550  shuffle.exe
01/08/2002  12:35 PM             341,924  sindex.exe
01/08/2002  12:35 PM             344,484  sreformat.exe
               17 File(s)              8,256,505 bytes
                 2 Dir(s)  39,733,563,392 bytes free

D:\hmmmer>

```

Fig. 5.2 HMMER installation on DOS

ClustalW. This can be downloaded from <ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalX/> (although it may be available on other sites too). Again, download the version that is appropriate for your operating system. For our purposes, we will download the Windows version: clustalx1.8.msw.zip (Figure 5.3).

Finally, it may also be useful to have a dendrogram viewing program such as TreeView which plots phylogenetic relationships between proteins. TreeView is available for Windows/Macintosh/Unix and Linux and can be downloaded from <http://taxonomy.zoology.gla.ac.uk/rod/treeview.html>. Extract the zipped file in the usual manner and store the files in a directory such as D:\treeview. Install the application by double-clicking the Setup icon.

Uncompress the file as usual and save all the files in a directory such as D:\clustalw. For DOS, the downloaded files appear as shown in Figure 5.4.

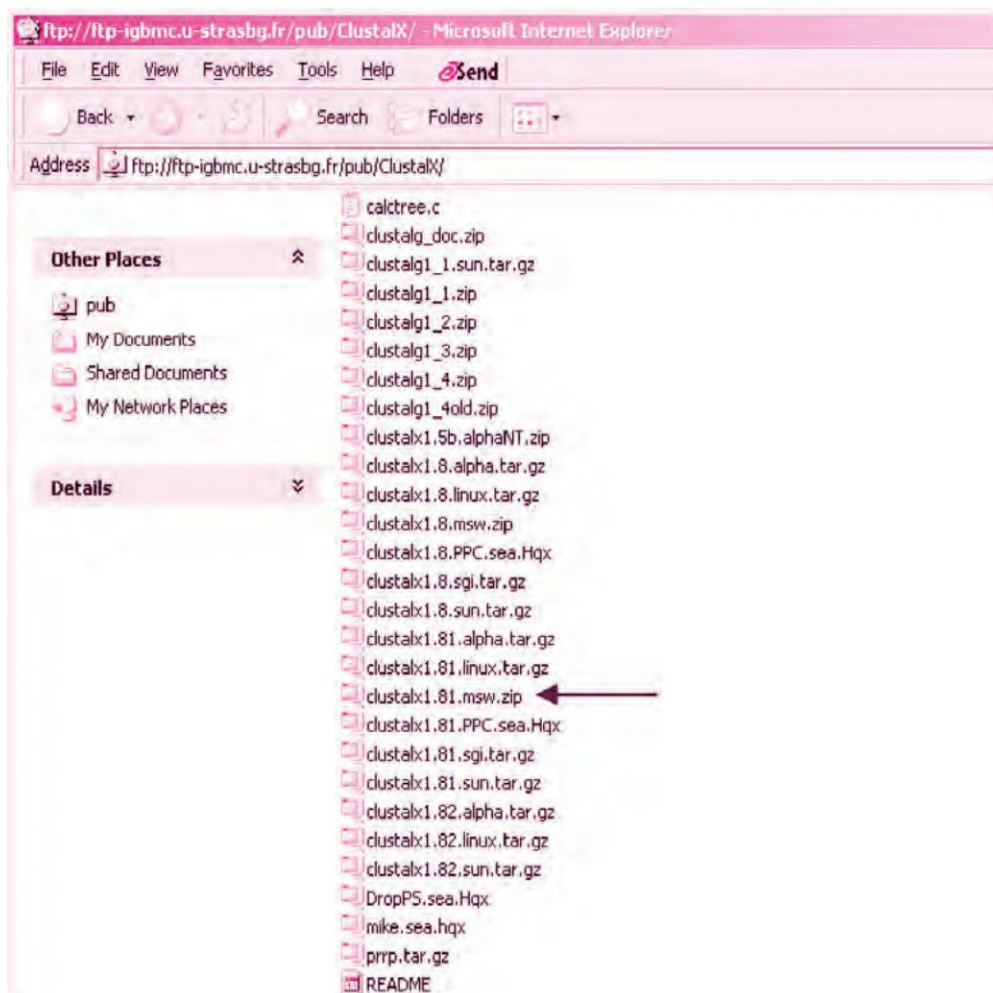


Fig. 5.3 ClustalX ftp site



5.3 WHY USE HMMER?

HMMER is used to perform sensitive database searches to identify distant members of sequence families. For example, you may be looking for hitherto unknown novel members of a certain protein family and you may want to identify all such protein family members that may be present in high-throughput genomic (HTG) sequences. HMMER allows you to use previously characterized (known) sequences of a protein family that you are interested in

```

CA Command Prompt
D:\clustalw>dir
Volume in drive D has no label.
Volume Serial Number is 70AE-6E52

Directory of D:\clustalw

09/14/2002  12:15 PM  <DIR>          .
09/14/2002  12:15 PM  <DIR>          ..
07/01/1999  10:16 AM             2,405  alnscore.c
07/01/1999  10:16 AM             39,610  amenu.c
07/01/1999  10:16 AM            14,611  calcgapcoeff.c
07/01/1999  10:16 AM             3,085  calcprf1.c
07/01/1999  10:16 AM             1,785  calcprf2.c
07/01/1999  10:16 AM            22,698  calctree.c
07/01/1999  10:16 AM            77,555  clustalw.doc
07/01/1999  10:16 AM             2,498  clustalw.c
07/01/1999  10:16 AM            34,148  clustalw.doc
07/01/1999  10:17 AM             7,723  clustalw.h
07/01/1999  10:16 AM            32,093  clustalw.hlp
07/01/1999  10:16 AM             2,917  clustalx.c
07/05/1999  09:34 AM           467,456  clustalx.exe
07/01/1999  10:16 AM            63,686  clustalx.hlp
07/01/1999  10:16 AM             359  coldna.par
06/09/1997  01:26 PM            1,170  color4.par
06/09/1997  01:26 PM            1,429  color8.par
07/01/1999  10:16 AM             366  colprint.par
07/01/1999  10:16 AM            1,429  colprot.par
07/01/1999  10:16 AM            2,699  dayhoff.h
06/09/1997  01:26 PM             323  dna.par
07/01/1999  10:16 AM             320  gcgcheck.c

```

Fig. 5.4 ClustalX installation on DOS

to create a profile or signature of the protein family and to use that profile to search a set of unknown sequences.

Research laboratories use this approach to identify novel members of their favorite protein family. This approach is also commonly used by Genomic companies to identify potential new targets belonging to the highly “druggable” class of proteins such as kinases, phosphatases, proteases, etc. (called “targets”). The term druggable is applied to proteins that take part in important signaling pathways (viz., via protein-protein interaction that affects a disease process) and that can be inhibited using small molecule drugs to achieve a pharmacologic effect. An example of a target protein is the epidermal growth factor receptor-tyrosine kinase (EGFR), a protein of the kinase family that contributes to a number of processes involved in tumor survival and growth including cell proliferation, inhibition of apoptosis, angiogenesis and metastasis. Small molecule drugs that inhibit growth signals within the cell mediated by EGFR could potentially be useful for the treatment of cancer. Indeed, a number of such drugs are currently undergoing pre-clinical and clinical trials in the US and elsewhere.

The identification of novel targets, by using Bioinformatics approaches, is an important aspect of modern Genomics-based drug development and is generally referred to as “New Target Identification”.



5.4 RUNNING HMMER COMMANDS

The programs supported in the HMMER 2 package and their common usage is listed in Table 5.1. The table uses the following files for the sample commands:

proteins.aln	Output of ClustalW (contains the multiple sequence alignment)
proteins.hmm	Output of hmmbuild: hmm_output_file (contains the Hidden Markov Model)
htg.db	Database of unknown protein sequences

MK

Table 5.1

HMMER command explanation and usage

MK



hmmbuild	Builds an HMM model from a multiple sequence alignment. Takes a multiple sequence alignment file (file has extension .aln) generated by a program such as ClustalW as input. The output file has the extension .hmm for ease of identification. Usage: <code>hmmbuild hmm_output_file input_file</code> Example: <code>hmmbuild proteins.hmm proteins.aln</code>
hmmcalibrate	Increases the sensitivity of a database search by calculating more accurate E values. Uncalibrated models may miss remote homologs and, therefore, this is an important step in the process. Usage: <code>hmmcalibrate hmm_output_file</code> Example: <code>hmmcalibrate proteins.hmm</code>

Note: `hmmcalibrate proteins.hmm` overwrites existing (uncalibrated) `hmm` file unless output is directed to a different file using the redirection (`>`) operator. The uncalibrated HMM may not be needed and, therefore, it is usually safe to overwrite it.

hmmsearch	Searches a sequence database for matches to an HMM. Usage: <code>hmmsearch hmm_output_file database</code> Example: <code>hmmsearch proteins.hmm htg.db</code>
------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Note: The database can also be a simple Fasta file of sequences.

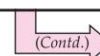


Table 5.1 (Contd.)

hmmalign	Aligns sequences to an existing model.
hmmconvert	Converts a model file into different formats, including a compact HMMER 2 binary format, and “best effort” emulation of GCG profiles.
hmmemit	Emits sequences probabilistically from a profile HMM.
hmmfetch	Gets a single model from an HMM database.
hmmindex	Indexes an HMM database.
hmmpfam	Searches an HMM database for matches to a query sequence.
HMMER also provides a number of utility programs:	
alistat	Shows simple statistics about a sequence alignment file. Alistat can, for example, be run on the output of ClustalW to determine the numbers of sequences that were present in the multiple Fasta file of protein sequences that was used as input to the ClustalW command, the level of identity found between the different proteins in the file, the average length of sequences, etc. Usage: alistat alignment_file Example: alistat proteins.aln
getseq	Retrieves a (sub-)sequence from a sequence file.
seqstat	Shows some simple statistics about a sequence file.
sreformat	Reformats a sequence file into a different format.

Note: Documentation on each command can be invoked by typing the name of the command followed by -h (for help) as shown in Figure 5.5 for hmmcalibrate.

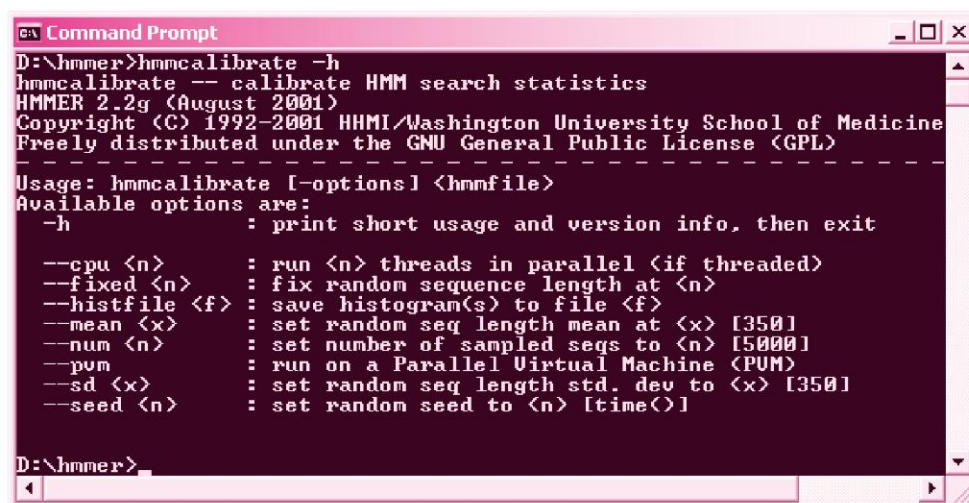
The steps to build and use an HMM are as follows:

1. Prepare a Fasta file of protein sequences
2. Align the sequences using ClustalW
3. Build the HMM (hmmbuild)
4. Calibrate the HMM (hmmcalibrate)
5. Search a database with the HMM (hmmsearch)



5.5 HMMER: A PRACTICAL EXAMPLE

Start the multiple sequence alignment program by typing ClustalX on the command prompt. If you saved the installation in D:\clustalw, this is where



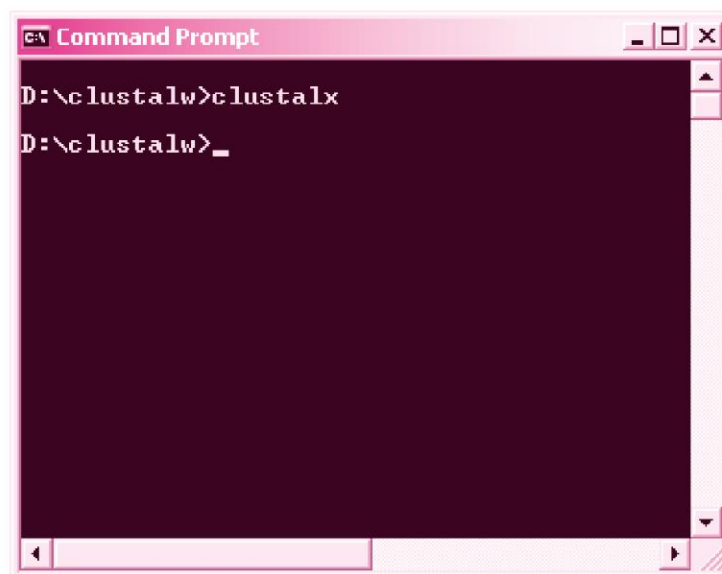
```
C:\ Command Prompt
D:\hmmmer>hmmcalibrate -h
hmmcalibrate -- calibrate HMM search statistics
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
-----
Usage: hmmcalibrate [-options] <hmmfile>
Available options are:
  -h                : print short usage and version info. then exit

  --cpu <n>         : run <n> threads in parallel (if threaded)
  --fixed <n>       : fix random sequence length at <n>
  --histfile <f>    : save histogram(s) to file <f>
  --mean <x>        : set random seq length mean at <x> [350]
  --num <n>         : set number of sampled seqs to <n> [5000]
  --pvm             : run on a Parallel Virtual Machine (PVM)
  --sd <x>          : set random seq length std. dev to <x> [350]
  --seed <n>        : set random seed to <n> [time()]

D:\hmmmer>
```

Fig. 5.5 Invoking help for hmmcalibrate

the command should be issued. This should bring up the graphical ClustalW interface (Figures 5.6 and 5.7). Download a set of cysteine proteases as a Fasta file on your system and build a multiple sequence alignment with ClustalX



```
C:\ Command Prompt
D:\clustalw>clustalx
D:\clustalw>
```

Fig. 5.6 Starting ClustalX

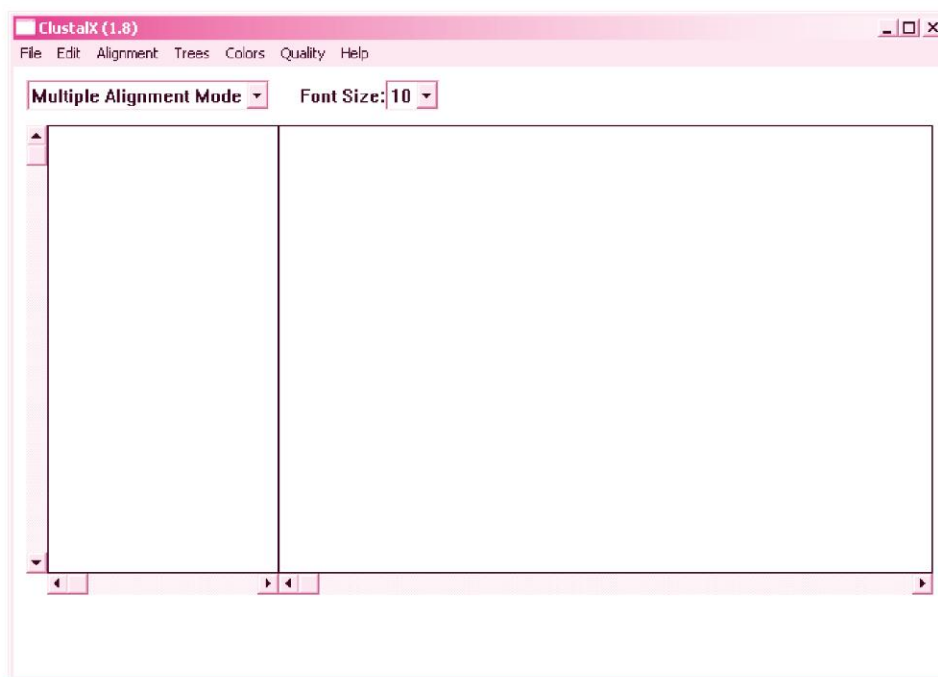


Fig. 5.7 The ClustalX interface

(using File → Load sequences → cproteases.txt). This should bring up the alignment (Figure 5.8). ClustalX color codes to the alignment assigning the same colors to residues that have similar properties. For example, the positively charged residues Lysine (K) and Arginine (R) are colored red, hydrophobic residues Leucine (L), Valine (V), Alanine (A) and Isoleucine (I) are colored blue and so on.

Save the alignment by selecting Alignment → Do complete alignment and specifying an output directory for the dendrogram and the sequence alignment. In this case, both are saved in the D:\sequences directory as cprotease.dnd and cprotease.aln (Figure 5.9).

You can, at this point, see the phylogenetic relationship between the five cysteine proteases by viewing the .dnd file in TreeView. Start the program either on the DOS prompt by typing treev32 or by selecting TreeView from the start icon (Start→Programs→TreeView). Load the .dnd file and select the Rectangular Cladogram option from the menu bar (Figure 5.10). The shape of the

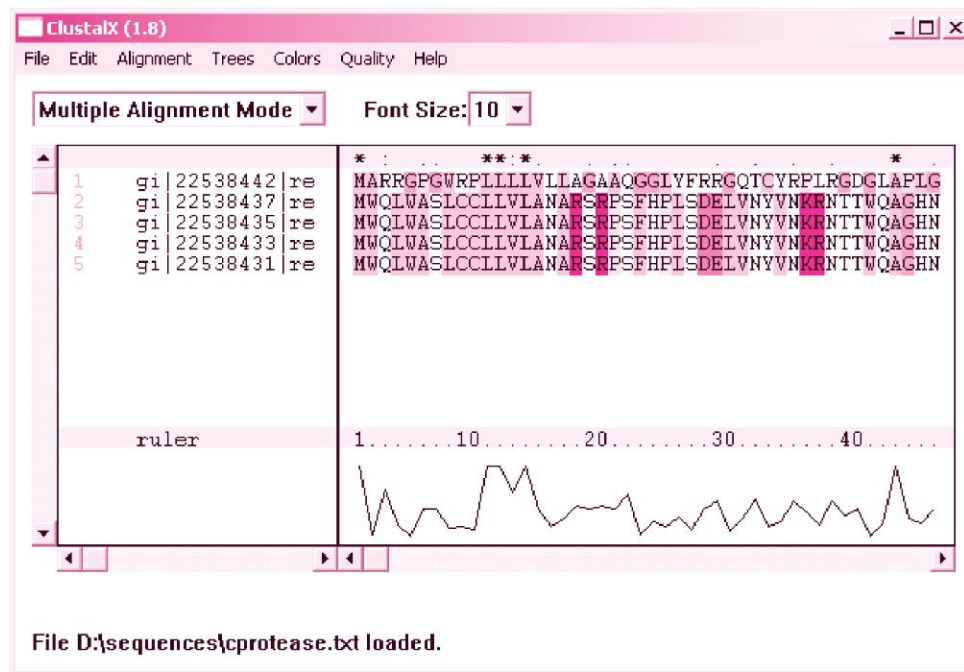


Fig. 5.8 Multiple sequence alignment with ClustalX

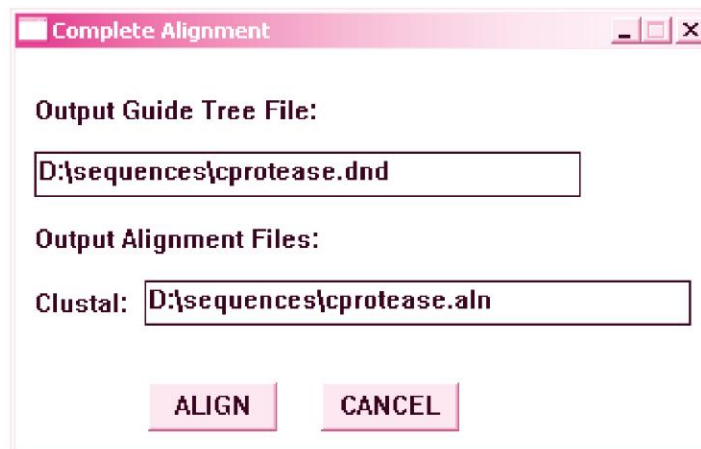


Fig. 5.9 Saving the alignment

tree indicates that the proteins gi:22538442 and gi:22538437 shown as follows are the most closely related as compared to the others:

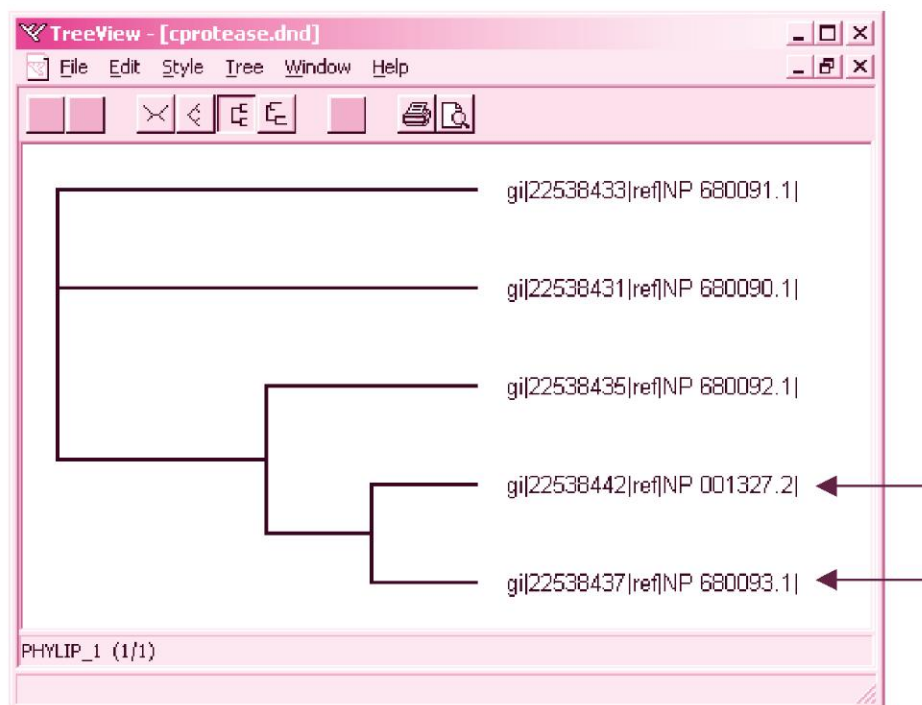


Fig. 5.10 Viewing the tree with TreeView

>gi|22538442|ref|NP_001327.2| cathepsin Z preproprotein; cathepsin X precursor; preprocathepsin P; cathepsin Z precursor [Homo sapiens]

MARRGPGWRPLLLVLLAGAAQGGLYFRRGQTCYRPLRGDGLAPLGRSTYPRPHEYLSPADLPKSWDWRN
VDGVNYASITRNQHQPQYCGSCWAHASTSAMADRINIKRKGAWPSTLLSVQNVIDCGNAGSCEGGNDLSV
WDYAHQHGPDETCNNYQAKDQECDFNQCGTCNEFKECHAIRNYTLWRVGDYGSLSGREKMMAEIYANG
PISCGIMATERLANYTGGIYAEYQDTTYINHVVSVAGWGISDGTETYWIVRNSWGEPWGERGWLRIVTSTY
KDGKGARYNLAIEEHCTFGDPIV

>gi|22538437|ref|NP_680093.1| cathepsin B preproprotein; APP secretase; preprocathepsin B; cathepsin B1; amyloid precursor protein secretase [Homo sapiens]

MWQLWASLCCLLVLANARSRPSFHPLSDELVNYVNKRNTTWQAGHNFYNDMSYLRKLCGTLGGPKPPQ
RVMFTEDLKLPAFDAREQWPQCPTIKEIRDQSGSCSWAFGAVEAISDRICHTNAHVSEVSAEDLLT
CCGSMCGDGCNGGYPAEAWNFWTRKGLVSGGLYESHVGCPRYSIPPCEHHVNGSRPPCTGEGDTPKCSKI
CEPGYSPTYKQDKHYGNSYSVSNSEKDIMAIEYKNGPVEGAFSVYSDFLLYKSGVYQHVTEGMMGGHAI
RILGWGVENGTPYWLANSWNTDWDGNGFFKILRGQDHCGIESEVVAGIPRTDQYWEKI

Note

A cladogram is simply a phylogenetic tree that describes the relatedness of the objects being compared.

The next step after the alignment is to build an HMM from the cprotease.aln multiple sequence alignment file. The command `hmmbuild` and its output is shown in Figure 5.11. The `hmmcalibrate` command next calculates the relevant parameters and adds them to the HMM file (cprotease.hmm, Figure 5.12). This, as explained earlier, is needed to make searches with the HMM more sensitive.

```

C:\ Command Prompt
D:\hmmmer>hmmbuild cprotease.hmm d:\sequences\cprotease.aln
hmmbuild - build a hidden Markov model from an alignment
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)

-----
Alignment file:                d:\sequences\cprotease.aln
File format:                   Clustal
Search algorithm configuration: Multiple domain (hmmls)
Model construction strategy:   MAP (gapmax hint: 0.50)
Null model used:               (default)
Prior used:                    (default)
Sequence weighting method:     G/S/C tree weights
New HMM file:                  cprotease.hmm
-----

Alignment:                      #1
Number of sequences:            5
Number of columns:             344

Determining effective sequence number    ... done. [2]
Weighting sequences heuristically        ... done.
Constructing model architecture          ... done.
Converting counts to probabilities       ... done.
Setting model name, etc.                ... done. [d:\sequences\cprot

Constructed a profile HMM (length 343)
Average score:    956.31 bits
Minimum score:    829.64 bits
Maximum score:    987.98 bits
Std. deviation:   70.81 bits

Finalizing model configuration    ... done.
Saving model to file             ... done.
//

D:\hmmmer>

```

Fig. 5.11 Building an HMM from an alignment

```

C:\ Command Prompt
D:\hmm>hmmcalibrate cprotease.hmm
hmmcalibrate -- calibrate HMM search statistics
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)

-----
HMM file:                cprotease.hmm
Length distribution mean: 325
Length distribution s.d.: 200
Number of samples:       5000
random seed:             1032055457
histogram(s) saved to:   [not saved]
-----

HMM      : d:\sequences\cprotease
mu       : -231.936859
lambda   :  0.140046
max      : -189.341003
//
D:\hmm>

```

Fig. 5.12 Calibrating cprotease.hmm

MK
+
 The next step is to use the calibrated HMM to search a database of sequences. This, as we mentioned before, is to find sequences in a set of unknown sequences that may be related to the sequences that we built the HMMs from. In this case, if we search a set of sequences with cprotease.hmm, we would expect the search to yield proteins which may be cysteine proteases. For the purpose of illustration, let's say we were interested in finding all cysteine proteases in the worm (*C. elegans*) genome. Why would anyone want to do that?

C. elegans is a nematode (of the same class as roundworms and threadworms) that is widely used in developmental biology studies. It was the first multicellular eukaryote whose genome was completely sequenced (1998, *C. elegans* Sequencing Consortium). Despite its small size (< 1 mm in length, total number of somatic cells: 959), *C. elegans* emerged as a model organism for the study of mammalian processes (see magnified image, Figure 5.13). This is because of the dramatic similarity between many of its genes and human genes. This is especially true of genes involved in cellular differentiation and development as they relate to the development and function of the nervous system. Today, due to the efforts of a large body of developmental biologists around the world, the developmental origins of each of the 300-odd neurons of *C. elegans* is known. A significant number of CNS disorders such as Alzheimer's disease, amyotrophic lateral sclerosis, Duchenne muscular dystrophy, neurofibromatosis



Fig. 5.13 *C. elegans*—A model organism in developmental biology

type 1, spinal muscular atrophy, etc. that affect human beings are known to involve genes that have homologs in *C. elegans*. It is hoped that by identifying and analyzing the proteins encoded by these genes, from both worm and man, it will be possible to identify protein targets for the study and treatment of human neurological diseases.

Now that we have made a case for the study of counterparts of human proteins in the worm, let's go ahead and find out how many cysteine proteases our HMM can identify in the *C. elegans* genome.

Download the *C. elegans* amino acid sequences file `wormpep_current.tar.gz` from <http://www.wormbase.org/downloads.html>. Expand the archive and save the `wormpep86` file on your system. The command `hmmsearch` and a partial output is shown in Figure 5.14. The output of the command was redirected to a text file (`worm_protease.txt`), parts of which are reproduced in Figures 5.15–5.17.

Some of the relevant information in the figures is boxed; for example, the header reveals the `hmm` file (`cprotease.hmm`) used for the analysis, the sequence database searched (`wormpep86`) and also indicates that the `hmm` was calibrated (Figure 5.16).

The results of the search (Figure 5.16) are similar to the output of a BLAST search: top hits are listed in the order of their significance, beginning, with the lowest E values. The fields on each line are: name of target sequence, description of sequence, raw score (in bits), E value, and the total number of (cysteine protease) domains detected in the sequence. The next section lists top hits by domain (Figure 5.17). These are also ranked by E value.

```

D:\hmmsearch>hmmsearch cprotease.hmm d:\sequences\wormpep86 > worm_protease.txt

D:\hmmsearch>more worm_protease.txt
hmmsearch - search a sequence database with a profile HMM
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)

-----
HMM file:                cprotease.hmm [d:\sequences\cprotease]
Sequence database:       d:\sequences\wormpep86
per-sequence score cutoff: [none]
per-domain score cutoff:  [none]
per-sequence Eval cutoff:  <= 10
per-domain Eval cutoff:   [none]
-----

Query HMM:  d:\sequences\cprotease
Accession:  [none]
Description: [none]
[HMM has been calibrated; E-values are empirical estimates]

Scores for complete sequences (score includes all domains):
Sequence      Description                                     Score      E-value      N
-----
W07B8.5       CE14682      locus:cpr-5 thiol protease statu      338.6       2.4e-98      1
F32B5.8       CE09855      cysteine proteinase status:Part      334.8       3.5e-97      1
F57F5.1       CE05999      cysteine protease status:Partia      313.1       1.2e-90      1
W07B8.4       CE14680      thiol protease status:Partially      305.7       1.9e-88      1
F44C4.3       CE07251      locus:cpr-4 cathepsin B-like cys     297.1       7.9e-86      1
C52E4.1       CE08943      cathepsin-like cysteine proteas      293.4       9.7e-85      1
C25B8.3b      CE30876      locus:cpr-6 status:Confirmed IM      289.4       1.6e-83      1
C25B8.3a      CE04078      locus:cpr-6 status:Confirmed SW      289.4       1.6e-83      1
T10H4.12      CE27590      locus:cpr-3 cathepsin protease s     260.3       9.2e-75      1
F36D3.9       CE30347      cysteine protease status:Predic      253.0       1.4e-72      1
M04G12.2      CE12424      cysteine protease status:Confir      241.3       4.7e-69      1
W07B8.1       CE14674      thiol protease status:Partially      237.9       5.2e-68      1

```

Fig. 5.14 Running hmmsearch on *C. elegans* peptides

```

worm_protease - Notepad
File Edit Format View Help Send

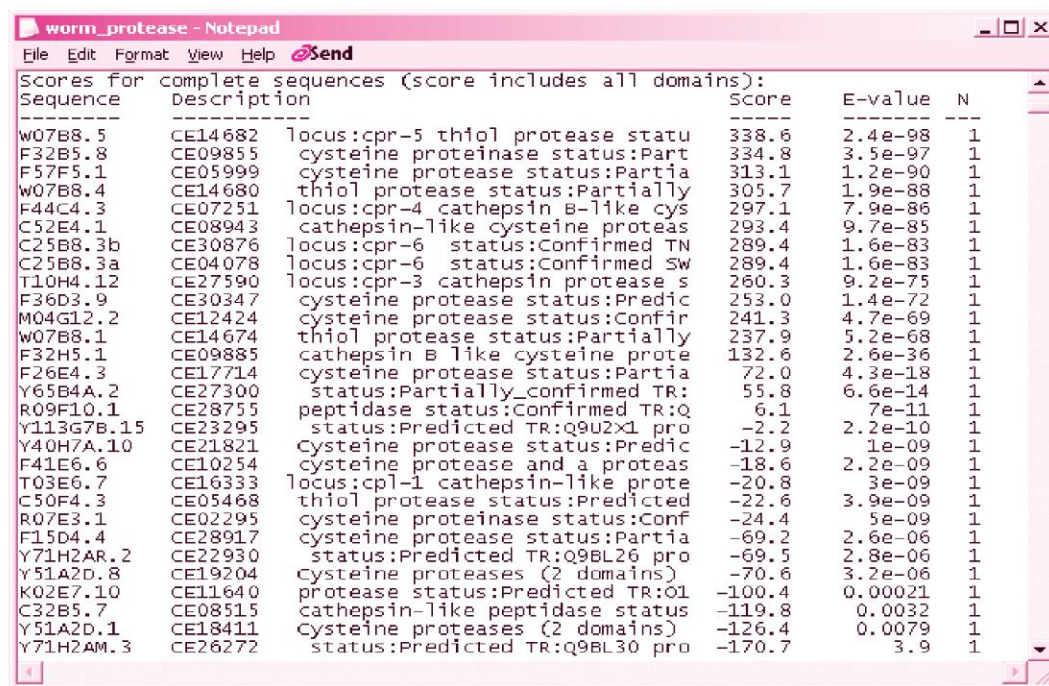
hmmsearch - search a sequence database with a profile HMM
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)

-----
HMM file:                cprotease.hmm [d:\sequences\cprotease]
Sequence database:       d:\sequences\wormpep86
per-sequence score cutoff: [none]
per-domain score cutoff:  [none]
per-sequence Eval cutoff:  <= 10
per-domain Eval cutoff:   [none]
-----

Query HMM:  d:\sequences\cprotease
Accession:  [none]
Description: [none]
[HMM has been calibrated; E-values are empirical estimates]

```

Fig. 5.15 hmmsearch results—Header information



Sequence	Description	Score	E-value	N
w07B8.5	CE14682 locus:cpr-5 thiol protease statu	338.6	2.4e-98	1
F32B5.8	CE09855 cysteine proteinase status:Part	334.8	3.5e-97	1
F57F5.1	CE05999 cysteine protease status:Partia	313.1	1.2e-90	1
w07B8.4	CE14680 thiol protease status:Partially	305.7	1.9e-88	1
F44C4.3	CE07251 locus:cpr-4 cathepsin B-like cys	297.1	7.9e-86	1
C52E4.1	CE08943 cathepsin-like cysteine proteas	293.4	9.7e-85	1
C25B8.3b	CE30876 locus:cpr-6 status:Confirmed TN	289.4	1.6e-83	1
C25B8.3a	CE04078 locus:cpr-6 status:Confirmed SW	289.4	1.6e-83	1
T10H4.12	CE27590 locus:cpr-3 cathepsin protease s	260.3	9.2e-75	1
F36D3.9	CE30347 cysteine protease status:Predic	253.0	1.4e-72	1
M04G12.2	CE12424 cysteine protease status:Confir	241.3	4.7e-69	1
w07B8.1	CE14674 thiol protease status:Partially	237.9	5.2e-68	1
F32H5.1	CE09885 cathepsin B like cysteine prote	132.6	2.6e-36	1
F26E4.3	CE17714 cysteine protease status:Partia	72.0	4.3e-18	1
Y65B4A.2	CE27300 status:Partially_confirmed TR:	55.8	6.6e-14	1
R09F10.1	CE28755 peptidase status:Confirmed TR:Q	6.1	7e-11	1
Y113G7B.15	CE23295 status:Predicted TR:Q9U2X1 pro	-2.2	2.2e-10	1
Y40H7A.10	CE21821 Cysteine protease status:Predic	-12.9	1e-09	1
F41E6.6	CE10254 cysteine protease and a proteas	-18.6	2.2e-09	1
T03E6.7	CE16333 locus:cpl-1 cathepsin-like prote	-20.8	3e-09	1
C50F4.3	CE05468 thiol protease status:Predicted	-22.6	3.9e-09	1
R07E3.1	CE02295 cysteine proteinase status:Conf	-24.4	5e-09	1
F15D4.4	CE28917 cysteine protease status:Partia	-69.2	2.6e-06	1
Y71H2AR.2	CE22930 status:Predicted TR:Q9BL26 pro	-69.5	2.8e-06	1
Y51A2D.8	CE19204 Cysteine proteases (2 domains)	-70.6	3.2e-06	1
K02E7.10	CE11640 protease status:Predicted TR:O1	-100.4	0.00021	1
C32B5.7	CE08515 cathepsin-like peptidase status	-119.8	0.0032	1
Y51A2D.1	CE18411 Cysteine proteases (2 domains)	-126.4	0.0079	1
Y71H2AM.3	CE26272 status:Predicted TR:Q9BL30 pro	-170.7	3.9	1

Fig. 5.16 Top hits yielded by hmmsearch

5.6 HMMER UTILITIES

Finally, let's take a quick look at two utilities—seqstat and alistat available with HMMER to generate simple statistical reports on sequence and alignment files.

Alistat shows simple statistics about a sequence alignment file. Alistat can, for example, be run on the output of ClustalW to determine the number of sequences that were present in the multiple Fasta file of protein sequences used as input to the ClustalW command, the level of identity found between the different proteins in the file, the average length of sequences, etc.

Usage: alistat alignment_file

Seqstat shows some simple statistics about a sequence file.

Usage: seqstat sequence_file


worm_protease - Notepad									
File Edit Format View Help 									
Parsed for domains:									
Sequence	Domain	seq-f	seq-t		hmm-f	hmm-t	score	E-value	
w07B8.5	1/1	1	343	[.	1	343	[]	338.6	2.4e-98
F32B5.8	1/1	124	426	..	1	343	[]	334.8	3.5e-97
F57F5.1	1/1	50	400	.]	1	343	[]	313.1	1.2e-90
w07B8.4	1/1	1	333	[.	1	343	[]	305.7	1.9e-88
F44C4.3	1/1	1	335	[]	1	343	[]	297.1	7.9e-86
C52E4.1	1/1	13	340	.]	1	343	[]	293.4	9.7e-85
C25B8.3b	1/1	14	365	..	1	343	[]	289.4	1.6e-83
C25B8.3a	1/1	15	366	..	1	343	[]	289.4	1.6e-83
T10H4.12	1/1	6	345	..	1	343	[]	260.3	9.2e-75
F36D3.9	1/1	17	344	.]	1	343	[]	253.0	1.4e-72
M04G12.2	1/1	149	461	..	1	343	[]	241.3	4.7e-69
w07B8.1	1/1	1	334	[.	1	343	[]	237.9	5.2e-68
F32H5.1	1/1	1	355	[.	1	343	[]	132.6	2.6e-36
F26E4.3	1/1	155	478	..	1	343	[]	72.0	4.3e-18
Y65B4A.2	1/1	67	421	.]	1	343	[]	55.8	6.6e-14
R09F10.1	1/1	110	382	..	1	343	[]	6.1	7e-11
Y113G7B.15	1/1	16	326	..	1	343	[]	-2.2	2.2e-10
Y40H7A.10	1/1	66	342	..	1	343	[]	-12.9	1e-09
F41E6.6	1/1	205	497	..	1	343	[]	-18.6	2.2e-09
T03E6.7	1/1	51	336	..	1	343	[]	-20.8	3e-09
C50F4.3	1/1	68	372	..	1	343	[]	-22.6	3.9e-09
R07E3.1	1/1	86	398	..	1	343	[]	-24.4	5e-09
F15D4.4	1/1	171	457	..	1	343	[]	-69.2	2.6e-06
Y71H2AR.2	1/1	18	298	..	1	343	[]	-69.5	2.8e-06
Y51A2D.8	1/1	89	384	..	1	343	[]	-70.6	3.2e-06
K02E7.10	1/1	15	298	..	1	343	[]	-100.4	0.00021
C32B5.7	1/1	8	250	.]	1	343	[]	-119.8	0.0032
Y51A2D.1	1/1	65	381	..	1	343	[]	-126.4	0.0079
Y71H2AM.3	1/1	18	281	..	1	343	[]	-170.7	3.9

Fig. 5.17 Domain top hits

The output of these files is shown in Figures 5.18 and 5.19.

Taking the first line of the output above, the meaning of each of the fields is explained below:

parsed for domains:

sequence	Domain	seq-f	seq-t		hmm-f	hmm-t	score	E value	
w07B5.5	1/1	1	343	[.	1	343	[]	338.6	2.4e-99

Domain: 1/1 means the first domain of the number of domains detected (1)

seq-f: sequence from (start)

seq-t: sequence to (end)

Alignment key:

- [or] → alignment goes all the way to the end of the sequence
- . → alignment does not go all the way to the end
- [. → alignment starts at the beginning of the sequence, but stops before it ends
- .] → alignment starts internally and goes all the way to the end
- [] → alignment spans the entire sequence
- .. → alignment is local within the sequence

hmm-f: start point of the consensus coordinates of the model

hmm-t: end point of the consensus coordinates of the model

Alignment key (with respect to the model). Here, [] means that complete matches to the entire model are found.

```

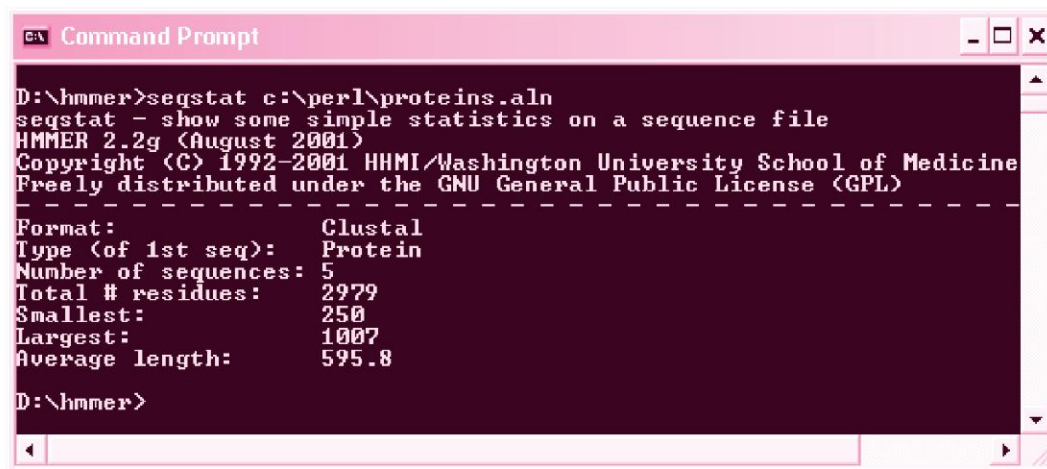
D:\hmmer>alistat c:\perl\proteins.aln
alistat - show some simple statistics on an alignment file
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
-----
Format:           Clustal
Number of sequences: 5
Total # residues:  2979
Smallest:         250
Largest:          1007
Average length:   595.8
Alignment length: 1032
Average identity:  12%
Most related pair: 19%
Most unrelated pair: 6%
Most distant seq: 12%
//
D:\hmmer>

```

Fig. 5.18 alistat utility

Assignments

1. Chromosome 21 is the smallest human chromosome. The fact that trisomy 21 (Down's syndrome), the most frequent genetic disorder associated with



```
Command Prompt
D:\hmmmer>seqstat c:\perl\proteins.aln
seqstat - show some simple statistics on a sequence file
HMMER 2.2g (August 2001)
Copyright (C) 1992-2001 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
-----
Format:          Clustal
Type (of 1st seq): Protein
Number of sequences: 5
Total # residues: 2979
Smallest:        250
Largest:         1007
Average length:  595.8
D:\hmmmer>
```

Fig. 5.19 seqstat utility

MK
+

significant mental retardation, is associated with aberrations of this chromosome has given chromosome 21 a prominent position in biomedical research. Download the peptide sequences: protein.fa.gz from ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/protein/. . Generate separate HMMs for all the four protease families following the steps outlined in Section 5.4 and search chromosome 21 peptide sequences for the presence of members of the protease families. For each of the top 10 hits of the `hmmsearch`, extract the identifier from the first column and extract the annotation for the protein from GenBank.

MK
+

To create a Fasta file of the proteases, search NCBI Entrez and select ~20 representative sequences for all four families. Automate the process of creating and calibrating an HMM, searching a sequence database and annotating the top hits using Perl. The command for the program should be something like:

```
% searchdb.pl -hmmfile aspartyl.hmm -database chr21.txt
```

Note

If you get a memory error running HMMER commands, divide the Chromosome 21 sequence into smaller files of ~10 Mb each. If you still have a problem, download a smaller genome, e.g., HIV-1 and perform the same analysis.

References

- Thompson JD, Higgins DG and Gibson TJ. (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(22):4673–80.
- Thompson JD, Gibson TJ, Plewniak F, Jeanmougin F and Higgins DG (1997) The ClustalX windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Research*, 24:4876–82.





PSI-BLAST



MK



6.1 INTRODUCTION

Position-Specific Iterated BLAST or PSI-BLAST is a variant of the BLAST program developed by Altschul et al. in 1997. A position specific scoring matrix, PSSM, is constructed (automatically) by calculating position-specific scores for each position in the alignment of a multiple alignment in the highest scoring hits in an initial BLAST search. The PSS is calculated by assigning high scores to highly conserved positions and near zero scores to weakly conserved positions. The profile is then used to perform a second BLAST search and the results of each “iteration” is used to refine the profile. This iterative searching strategy results in increased sensitivity. Thus, PSI-BLAST is a highly sensitive homology search program generally used with a query of amino acid sequence against an amino acid sequence database.



6.2 PSI-BLAST AND PROTEIN ANALYSIS

Many functionally and evolutionarily important protein similarities are recognizable only through comparison of three-dimensional structures. When

MK



such structures are not available, patterns of conservation identified from the alignment of related sequences can aid the recognition of distant similarities. In essence, for each position in the derived pattern, every amino acid is assigned a score. If a residue is highly conserved at a particular position, that residue is assigned a high positive score, and others are assigned high negative scores. At weakly conserved positions, all residues receive near zero scores. Position-specific scores can also be assigned to potential insertions and deletions.

The power of profile methods can be further enhanced through iteration of the search procedure. After a profile is run against a database, new similar sequences can be detected. A new multiple alignment, which includes these sequences, can be constructed, a new profile abstracted, and a new database search performed. The procedure can be iterated as often as desired or until convergence, until no new statistically significant sequences are detected. PSI-BLAST is an example of such a tool.

MK



6.3 WHEN IS PSI-BLAST BETTER THAN BLASTP?



PSI-BLAST can beat BLASTP if BLASTP finds some reliable alignments to database sequences. (Moderately distant matches are particularly useful.) Then, PSI-BLAST (which starts by running BLASTP) can determine the positions, in the query sequence that are conserved during evolution and devise an appropriate position-specific scoring matrix which can be used to identify relatives at a further evolutionary distance. If the original BLASTP run cannot find any reliable alignment, PSI-BLAST is powerless.

MK



6.4 THE DESIGN OF PSI-BLAST

Iterated profile search methods have led to biologically important observations, but, for many years, were quite slow and generally did not provide precise means to evaluate the significance of their results. This limited their utility in the systematic mining of protein databases. The principal design goals in developing the PSI-BLAST program were speed, simplicity and automatic operation.

The procedure PSI-BLAST uses can be summarized in five steps:

1. PSI-BLAST takes as 'input' a single protein sequence and compares it to a protein database, using the gapped BLAST program.
2. The program constructs a multiple alignment, and then a profile, from any significant local alignments found. The original query sequence serves as a template for the multiple alignment and profile, whose lengths are identical to that of the query. Different numbers of sequences can be aligned in different template positions.
3. The profile is compared to the protein database, again seeking local alignments. After a few minor modifications, the BLAST algorithm can be used for this directly.
4. PSI-BLAST estimates the statistical significance of the local alignments found. Because profile substitution scores are constructed to a fixed scale and gap scores remain independent of position, the statistical theory and parameters for gapped BLAST alignments remain applicable to profile alignments.
5. Finally, PSI-BLAST iterates, by returning to step 2, until convergence (when further iterations do not produce better results).

Profile alignment statistics allow PSI-BLAST to proceed as a natural extension of BLAST; the results produced in iterative search steps are comparable to those produced from the first pass. Unlike most profile-based search methods, PSI-BLAST runs as one program, starting with a single protein sequence, and the intermediate steps of multiple alignment and profile construction are invisible to the user.



6.5 ADVANTAGES OF PSI-BLAST

PSI-BLAST offers exciting opportunities to discover new types of relationships in protein databases and use them to infer evolutionary origins of proteins. PSI-BLAST will search a protein sequence database with a query sequence motif, a matrix with rows representing sequence positions and columns representing variations in that position. The motif represents the observed variations in the alignment of a set of related proteins. PSI-BLAST has been engineered to find database matches almost as rapidly as BLASTP finds matches

to a query sequence. However, there are some differences between the motifs found by PSI-BLAST. First, the motif covers the entire sequence length, whereas motifs usually cover only a short stretch of the sequences. Second, the same gap penalties are used throughout the procedure and there is no position-specific penalty as in other programs. Third, each subsequent motif is based on using the query sequence as a master template to produce a multiple sequence alignment of the same length as the query sequence. Columns in the alignment involve varying numbers of sequences depending on the extent of the local alignment of each sequence with the query, and columns with gaps in the query sequence are ignored. Sequences >98 per cent similar to the query are not included in order to avoid biasing the motif. Thus, the alignment is a compilation of the pair-wise alignments of each matching database sequence with the query sequence.



6.6 LIMITATIONS OF PSI-BLAST



The main difficulty with searching for subtle sequence relationships based on similarity is determining the significance of the motifs that are found. Such similarities may be evidence of structural or evolutionary relationships but they could also be due to matching of random variations that have no common origin or function. Protein structures are, in general, comprised of a tightly packed core and outside loops. Amino acid substitutions within the core are common but only certain substitutions will work at a given amino acid position in a given structure. Thus, sequence similarity is not usually a good indicator of structural similarity and the motifs found need to be carefully evaluated before any firm conclusions can be drawn. Another difficulty with the PSI-BLAST approach is that the procedure follows a type of algorithm called a 'greedy' algorithm. Put simply, once additional sequences that match the query are found, they influence the finding of more sequences like themselves, and so on. If a different set of query sequences were initially used, a different group with the possible overlaps with the first set may be found. Thus, there is no guarantee that the group finally discovered authentically represents a functional group.





6.7 EXAMPLE OF A PSI-BLAST SEARCH

6.7.1 Example #1: ATP-dependent Lon Protease (*Wigglesworthia brevipalpis*)

We shall now illustrate how to run PSI-BLAST using a protease protein from *Wigglesworthia brevipalpis*, a bacteria, as an example. The protein sequence of the ATP-dependent Lon protease is as follows:

```
>gi|24324229|ref|NP_715593.1| ATP-dependent Lon protease, bacterial type
[Wigglesworthia brevipalpis]
```

```
MNPEHSQQIDIPVLPLRDVVVPHMVVPLFVGREKSIRCLEISMDKDKKIMLIAQKEASKDEPNIDDLFL
VGTISSILQMLKLPDGTVKVLVEGISRARIISLKNNGDYFTAEANYFNNTTSVNEQEQLIRATINQFEN
YIKLNKKIPTEVLSLSSINDAARLADTIASHMPLKLSGKQAVLEMISVAERLEYLMAMMESEMDLLQIE
KRIRNRVKKQMEKSQREYYLNEQIKAIQKELGEMEDNPDEHESLKRKIELSKMPKEVKKKADSELQKLKM
MSPMSAEATVVRGYIDWMISVPWHNRSKIKKNLSIAQKILDKDHYGLKKVKDRILEYLAVQSRVLKIKGP
ILCLMGPPGVGKTSLGQSIKATGRKYIRMA LGGM RDEAEIRGHRRTYIGSMPGKIIQKMSKVGKVNPLF
LLDEIDKMSTDMRGDPASALLEVDPEQNI AFNDHYLEV D YDLS DVMFVATSN SMRIPAPLLDRMEVIRL
SSYTEDEKLNIA RKHLFPKQVNRNALKENEIYVEDNALMGII RYYTREAGVRNLEREISKL CRKSVKIIL
MNKNINRIKINKKNLKD FLGVKKFDY GKAEIENKIGQVIGLAWTEVGGDLLTIETACVP GKGKLIYTGSL
GEVMQESIQAALT VRSRANKLGIKSDFY EKNDIHHVPEGSTPKDGPSAGIAMCTALVSCLTKNPVNSS
LAMTGEITLRGQILPIGGLKEKLLAAHRGGIKTVLIPYENKRNLENMPENVIKELNIHPVKIIDEVFNIS
LQDSIF
```

We employ the World Wide Web version of PSI-BLAST. The URL is as follows:

<http://www.ncbi.nlm.nih.gov/blast/>

The steps to perform a PSI-BLAST with the above protein as a query sequence are as follows:

1. Connect to the above-mentioned URL and open the BLAST page of NCBI
2. Choose the PSI and PHI-BLAST (arrow) option [refer Figure 6.1]
3. Paste the above sequence in the “search” section [Figure 6.2]. Alternatively you can simply write the GI number of the protein.

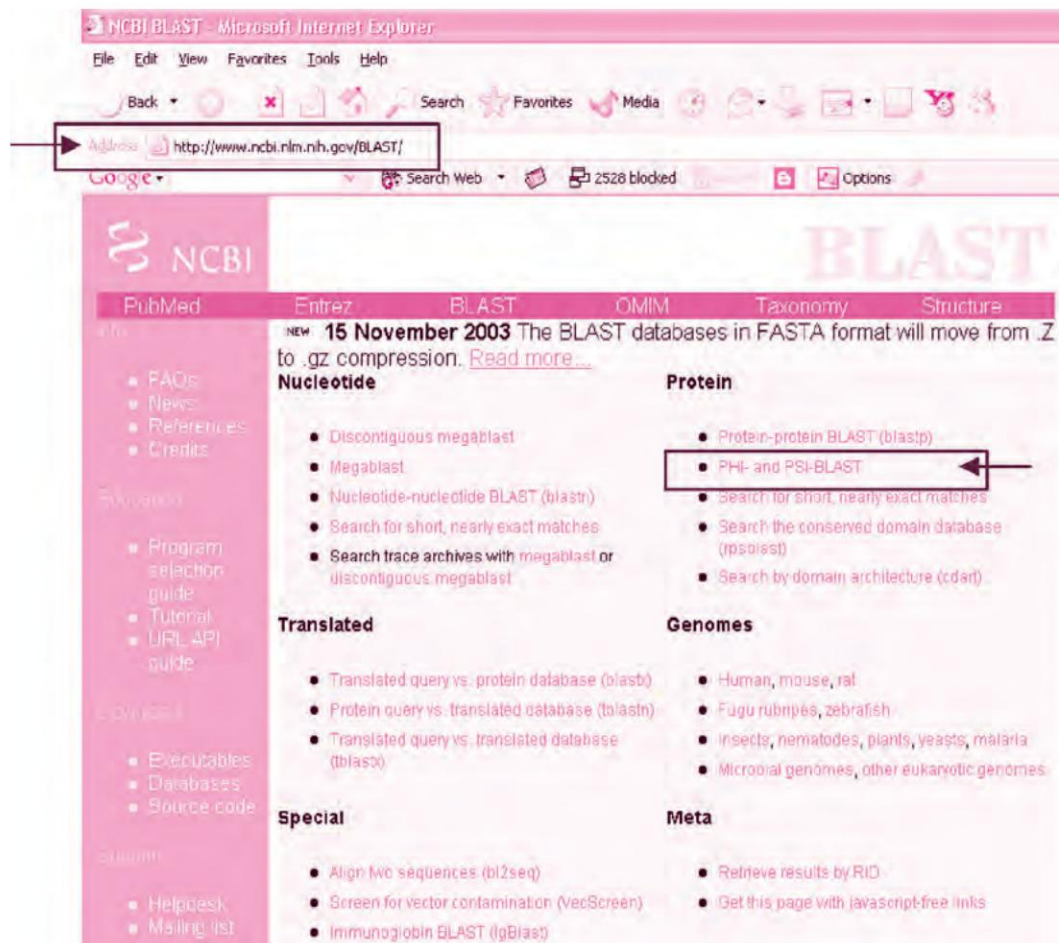


Fig. 6.1 The PSI-BLAST service at NCBI

4. Scroll down the page to the format section to set formatting parameters for the PSI-BLAST. In this case, we want to find very distant relatives of a common protein-protease. Therefore, I have selected the number of descriptions to be 250 [Figure 6.3, Box A]. You can choose to see a lower number of descriptions if the expected number of initial BLAST hits is low. This will help you save time as you will not have to go through a large number of hits to select the one you want to include in the second round of iteration.
5. The threshold values to include protein hits is determined by how divergent the proteins you are interested in finding are. In this case,

Fig. 6.2 Preparing a sequence for PSI-BLAST

we want to find a mammalian ortholog of a bacterial protein. Since the distance between the two is expected to be large, the threshold value is set at 10 [Figure 6.3, Box B]. If the divergence sought is small, say between two different species of bacteria, or if the number of hits expected is below 10, the default value setting for the threshold (0.005) is used. In general, the value is set between 0.01 and 10. The exact value comes from experience and an understanding of the biology of the query.

6. The rest of the parameters are generally used at the set default settings.
7. Click on the “BLAST” button [Figure 6.3, Box C] to initiate the first round of PSI-BLAST search.
8. On clicking the “BLAST” option, your browser window connects to the BLAST server and begins running the algorithm. You are brought to the new page, which gives you a “request ID”. This simply means that your request is in queue and is being processed.

Fig. 6.3 Configuring PSI-BLAST parameters

9. Click on the format button [Figure 6.4, arrow] to get the results of the first PSI-BLAST in formatted readout.
10. Figure 6.5 shows the result of the first iteration. As seen in the figure, the first few hits are exact matches of the protein query with an E value of 0. De-select those that do not fit your search criteria (in this case, hits of non-mammalian proteins), leaving those that do (proteins of mammalian origin) as selected [Figure 6.6, arrow]. Note that you may have to open the selected protein to confirm its origin (in this case, mammalian). In some cases, the first round of iteration results in convergence with respect to proteins hits having an E value higher than the threshold, but does not include proteins of biological interest to you. You would then have to select proteins that have a lower alignment score than the set threshold value (See for example #2).



Fig. 6.4 First PSI-BLAST iteration

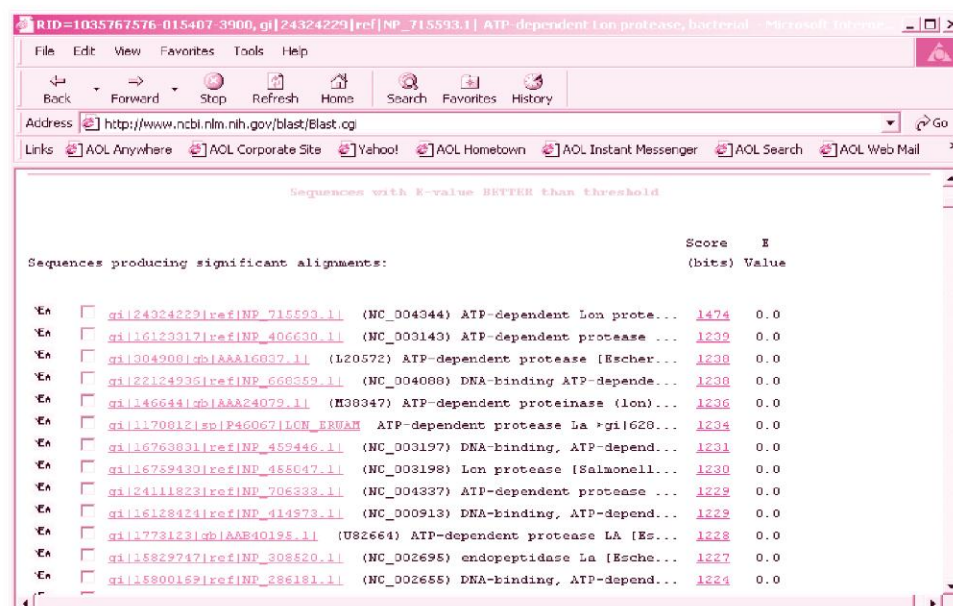


Fig. 6.5 Hits from the first PSI-BLAST iteration

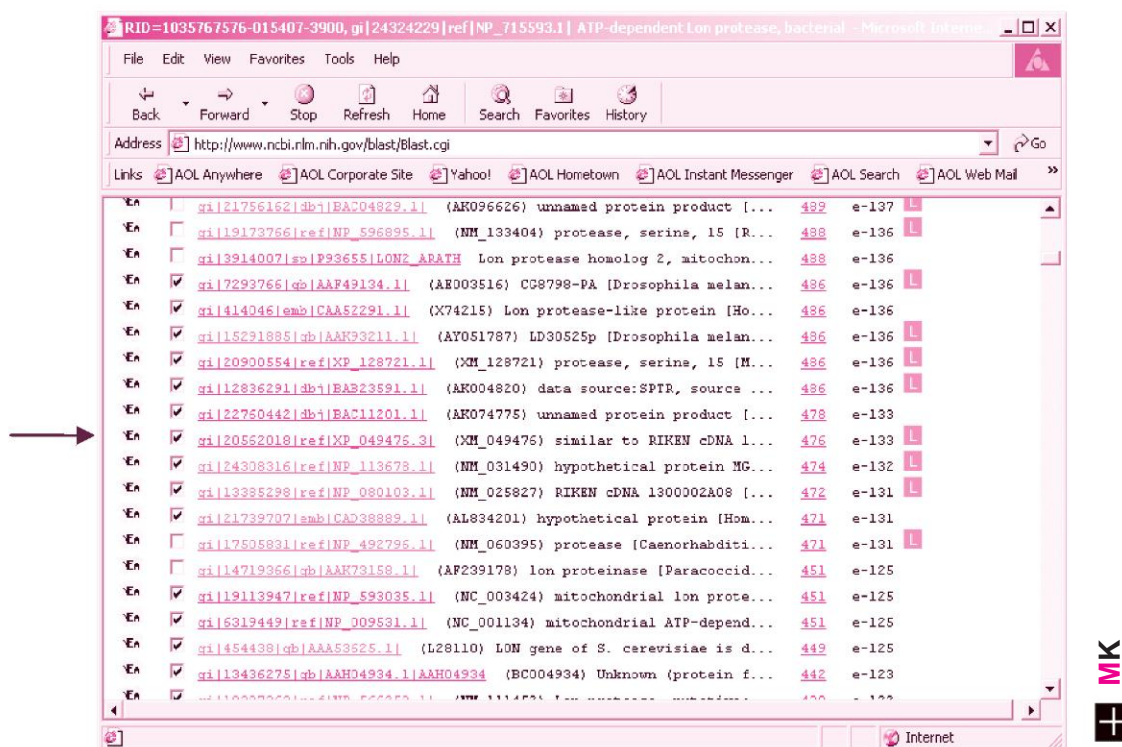


Fig. 6.6 Selecting hits from the first iteration of PSI-BLAST

11. Go to the bottom of the page and click on the “Run PSI-BLAST iteration 2” button to run the selected proteins through the second round of iteration [Figure 6.7, arrow].
12. Your browser window now shows the result of the second round of iteration through PSI-BLAST. It is important to understand the legend on top of the hit list [Figure 6.8, arrow].

The legend will help you determine the significance of the hits from the second round of iteration. As one would expect, the hits contain three classes of hits:

- (a) Some hits from the previous iteration. These are represented by a green dot next to them “●”.
- (b) Some new hits that have an E value higher than the threshold set (i.e. these are new proteins that have high similarity to the new set of collective query from the previous iteration).

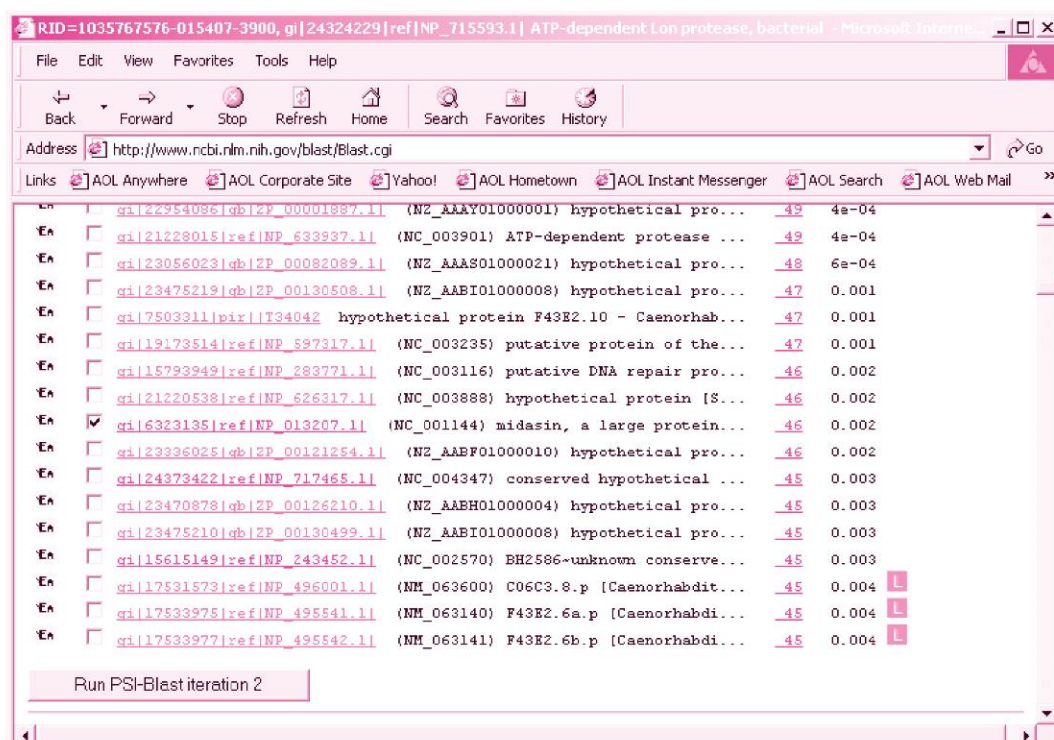


Fig. 6.7 Results of the second iteration of PSI-BLAST

(c) some new hits that have an E value lower than the set threshold limit (i.e. these are proteins that although are similar, are distant in relation to alignment scores. These are represented by the letters “Ea” in a yellow background “Ea”).

The criteria to select the proteins that will form the query set for the third round of iteration will depend on your ultimate goal of the PSI-BLAST search. If you are searching for orthologs or analogs of closely related species, or organisms that are not very divergent, say, between two bacteria or between a bacteria and a virus, then it may be safe to select only those protein hits that have a high E value. On the other hand, if the search is for very distant or divergent relationships (in this case, bacterial to mammalian), it would be advisable to include those that have a lower E value but may be significant with respect to the proteins sought in the search. Ultimately, the proteins that will constitute the query set for your next round of iteration will depend

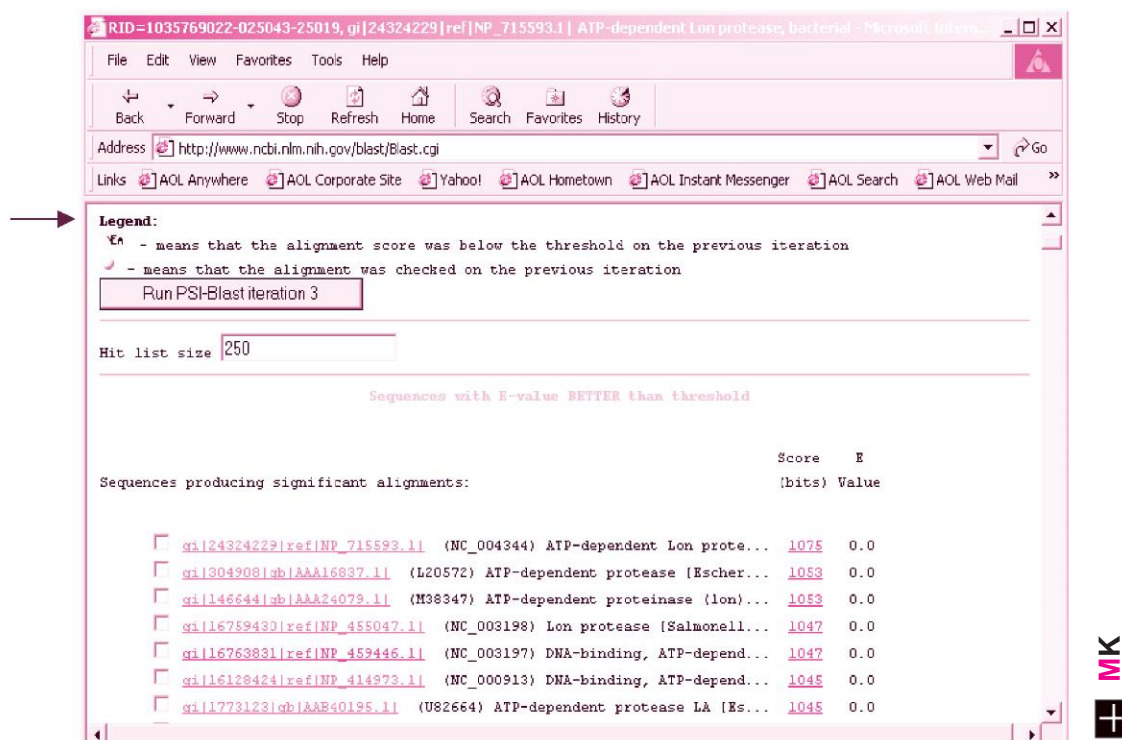


Fig. 6.8 Results of the 2nd iteration of PSI-BLAST

on the biological question you are looking to answer. Since, in our case, we are looking to find orthologs of a bacterial protein within a distant mammalian species, we shall be including proteins with an E value lower than the threshold value, as long as they are of mammalian origin. As seen in Figure 6.9, we have selected some hits from the previous iterations and included those that have an E value lower and above the set threshold value.

13. Go to the bottom of the screen and click on the “run PSI-BLAST iteration 3” button to run the selected proteins through the third round of iteration [Figure 6.10, arrow].
14. You would now repeat steps 12 and 13 till a convergence of proteins is achieved or no further convergence is possible. In our example, the highest convergence was with a serine protease (gi:21396489) of human origin, localized on chromosome 19.

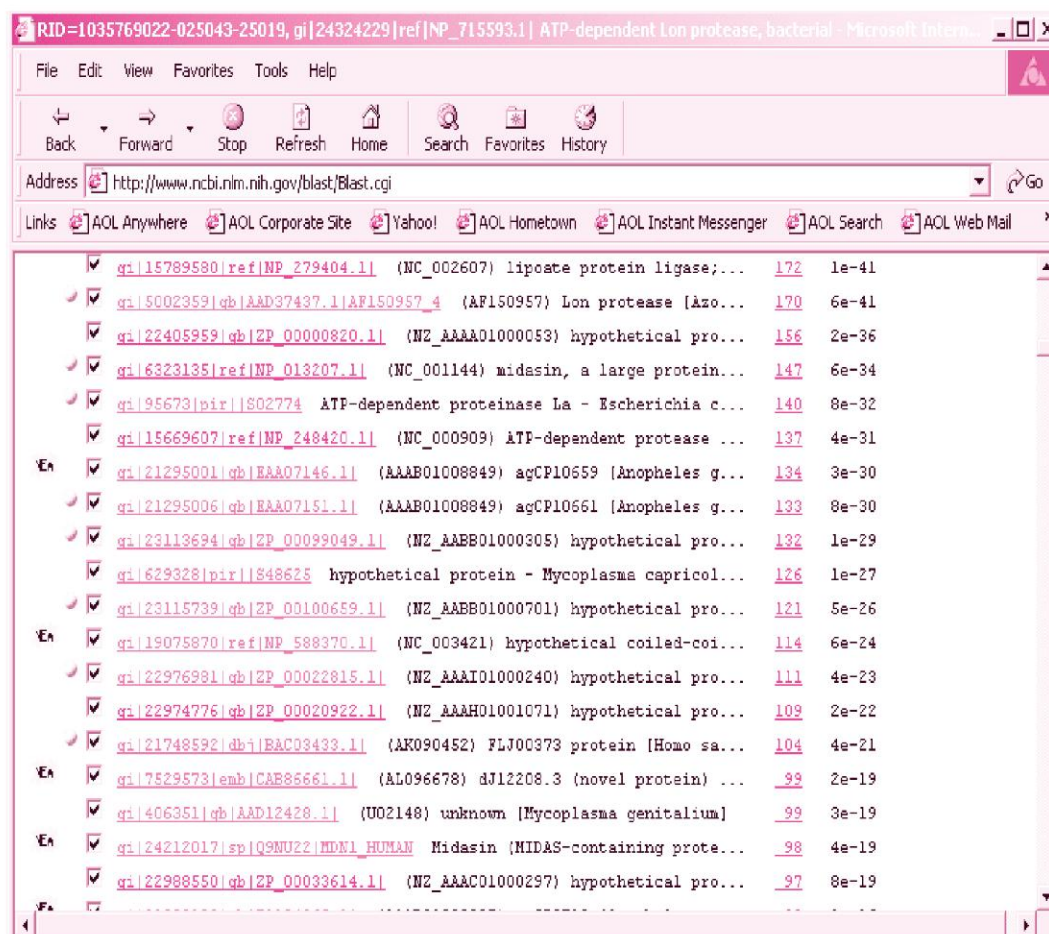


Fig. 6.9 Third round of PSI-BLAST iteration

6.7.2 Example #2: Repair Endonuclease of *Arabidopsis thaliana*

The protein used in this case is a repair endonuclease of *Arabidopsis thaliana*. The sequence of the protein is as follows:

```
>gi|6013183|gb|AAF01274.1|AF160500_1 repair endonuclease [Arabidopsis thaliana]
MALKYHQIISDILLEDNSGGLLILSSGLSLAKLIASLLILHSPSQGTLLLLLSPAAQSLKSRIIHYISSL
DSPTPTTEITADLPANQRYSLYTSGSPFFITPRILIVDLLTQRIPVSSLAGIFILNAHSISETSTEAFIIR
IVKSLNSSAYIRAFSDRPQAMVSGFAKTERTMRALFLRKIHLWPRFQLDVSQELEREPPEVVDIRVSMNS
```

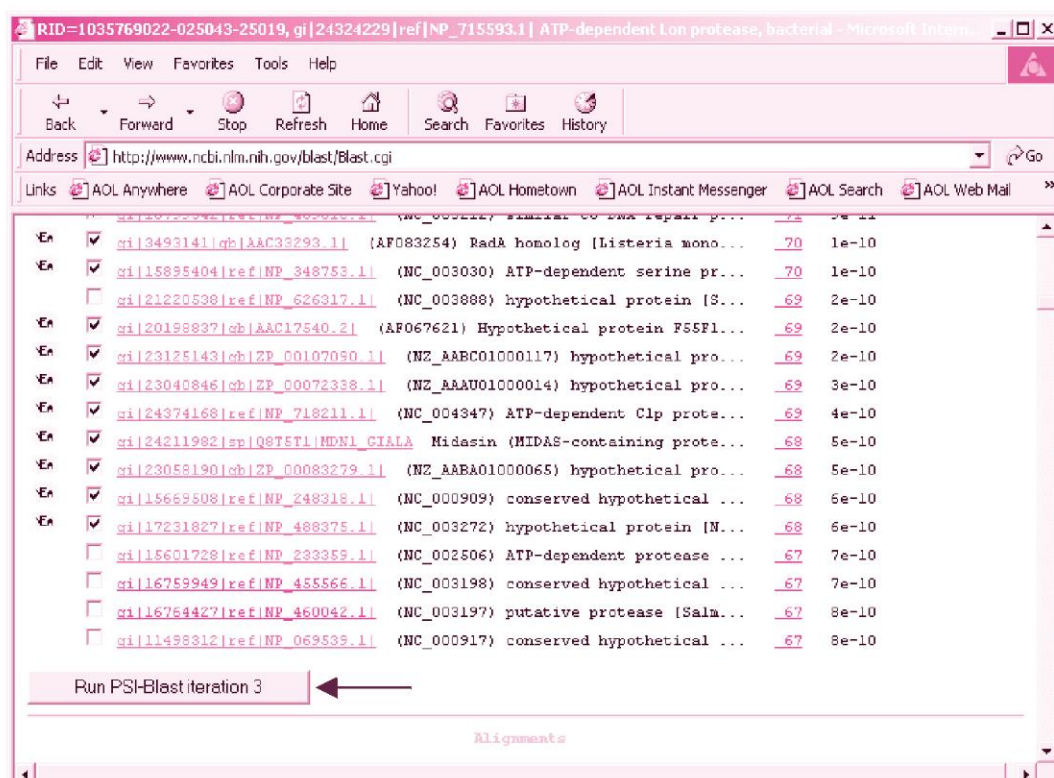


Fig. 6.10 Third iteration of PSI-BLAST

YMVGIQKAIIEVMDACLKEMKKTNKVDVDDLTVESGLFKSFDEIVRRQLDPIWHTLGKRTKQLVSDLKTL
 RKLLDYLVRYDAVSFLKFLDLTVSESYRSVWLFAESSYKIFDFAKKRVYRLVKASDVKSKEHVKNKSGK
 KRNSKGETDSVEAVGGETATNVATGVVVEEVLEEAPKWVKVLRILEETQEERLKQAFSEEDNSDNGIVL
 VACKDERSCMQLEDCTNNPQKVMREEWEMYLLSKIELRSMQTPQKKKQKTPKGFGLDGVVPVTTIQNS
 EGSSVGRQEHEALMAAASSIRKLGKTTDMASGNNNPEPHVDKASCTKGKAKKDPTSLRRSLRSCNKKTTN
 SKPEILPGPENEEKANEASTSAPQEANAVRPSGAKKLPPVHFYALESDQPILDILKPSVIVYHPDMGFV
 RELEVYKAENPLRLKLVYFIFYDESTEVQKFEASIRRENEAFESLIRQKSSMIIPVDQDGLCMGNSNSTE
 FPASSTQNSLTRKAGGRKELEKETQVIVDMREFMSSLPNVLHQKGMKIIPVTLEVGDYILSPSICVERKS
 IQDLFQSFTSGRLFHQVEMMSRYRIPVLLIEFSQDKSFSFQSSSDISDDVTPYNIISKLSLLVLHFPRLL
 RLLWSRSLHATAEIFTTLKSNQDEPDETRAIRVGVPSEEGHIENDIRAENYNTSAVEFLRRLPGVSDANY
 RSIMEKCKSLAELASLPVETLAELMGGHKVAKSLREFLDAKYPTLL

The sequence of the Arabidopsis XPF DNA repair gene was used to query the Swissprot database, with an E value setting of 0.01, requesting 10

descriptions and alignments with otherwise the recommended default program settings. The initial iteration found three matching sequences, and these were used to enter iteration 1. Iteration 1 did not produce any additional matches at the chosen level of significance, and the program indicated that the search had converged with no more sequences at the chosen level of significance. Therefore, for iteration 2, the sequences scoring worse than the threshold were used. Since only those lower scoring sequences that have an alignment with the query could influence the result, this option could potentially find additional sequences. A yeast transport protein was then reported. With another iteration using the four sequences above threshold, another set of sequences were now pulled into the high scoring group. This search, therefore, revealed that the Swissprot database has three other sequences strongly related to the query sequence but that other sequences of lower scoring similarity were also present.

Step 1: PSI-BLAST initial iteration

sp Q92889 XPF_HUMAN DNA-REPAIR PROTEIN COMPLEMENTING XP-F CELL ...	504	e-142
sp P06777 RAD1_YEAST DNA REPAIR PROTEIN RAD1	300	6e-81
sp P36617 RA16_SCHPO DNA REPAIR PROTEIN RAD16	231	3e-60

Step 2: PSI-BLAST iteration 1 (with sequences scoring better than E threshold)

sp Q92889 XPF_HUMAN DNA-REPAIR PROTEIN COMPLEMENTING XP-F CELL ...	1020	0.0
sp P06777 RAD1_YEAST DNA REPAIR PROTEIN RAD1	953	0.0
sp P36617 RA16_SCHPO DNA REPAIR PROTEIN RAD16	897	0.0

Step 3: PSI-BLAST iteration 2 (with sequences scoring worse than E threshold)

sp Q92889 XPF_HUMAN DNA-REPAIR PROTEIN COMPLEMENTING XP-F CELL ...	1020	0.0
sp P06777 RAD1_YEAST DNA REPAIR PROTEIN RAD1	967	0.0
sp P36617 RA16_SCHPO DNA REPAIR PROTEIN RAD16	939	0.0
sp P25386 USO1_YEAST INTRACELLULAR PROTEIN TRANSPORT PROTEIN USO1	53	3e-06

Step 4: PSI-BLAST iteration 3 (with sequences scoring better than E threshold)

sp Q92889 XPF_HUMAN DNA-REPAIR PROTEIN COMPLEMENTING XP-F CELL ...	1007	0.0
sp P06777 RAD1_YEAST DNA REPAIR PROTEIN RAD1	950	0.0
sp P36617 RA16_SCHPO DNA REPAIR PROTEIN RAD16	884	0.0
sp P25386 USO1_YEAST INTRACELLULAR PROTEIN TRANSPORT PROTEIN USO1	294	5e-79
sp Q08696 MST2_DROHY AXONEME-ASSOCIATED PROTEIN MST101(2)	52	4e-06
sp Q62209 SCP1_MOUSE SYNAPTONEMAL COMPLEX PROTEIN 1 (SCP-1 PROT...	49	5e-05
sp Q03410 SCP1_RAT SYNAPTONEMAL COMPLEX PROTEIN 1 (SCP-1 PROTEIN)	49	5e-05
sp Q02224 CENE_HUMAN CENTROMERIC PROTEIN E (CENP-E PROTEIN)	45	5e-04

You can continue the steps until you reach convergence. The results of the above iterations with the selected proteins allow the user to successively converge on the sequences of interest. The subset of proteins on convergence is representative of the distant relatives of the plant protein.

Assignments

- The following is a sequence from the *Rattus norvegicus* (Norway rat) genome that was predicted automated computational analysis using GenomeScan. The protein has the GenBank ID (gi) number 27721631 and is annotated as “similar to hypothetical protein KIAA0527—human (fragment)”. The complete GenBank record for the protein is available at Entrez (<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Nucleotide>).

Perform a PSI-BLAST analysis and provide your analysis of the putative function of the protein. Support your answer with an analysis of the different domains present in the protein.

```

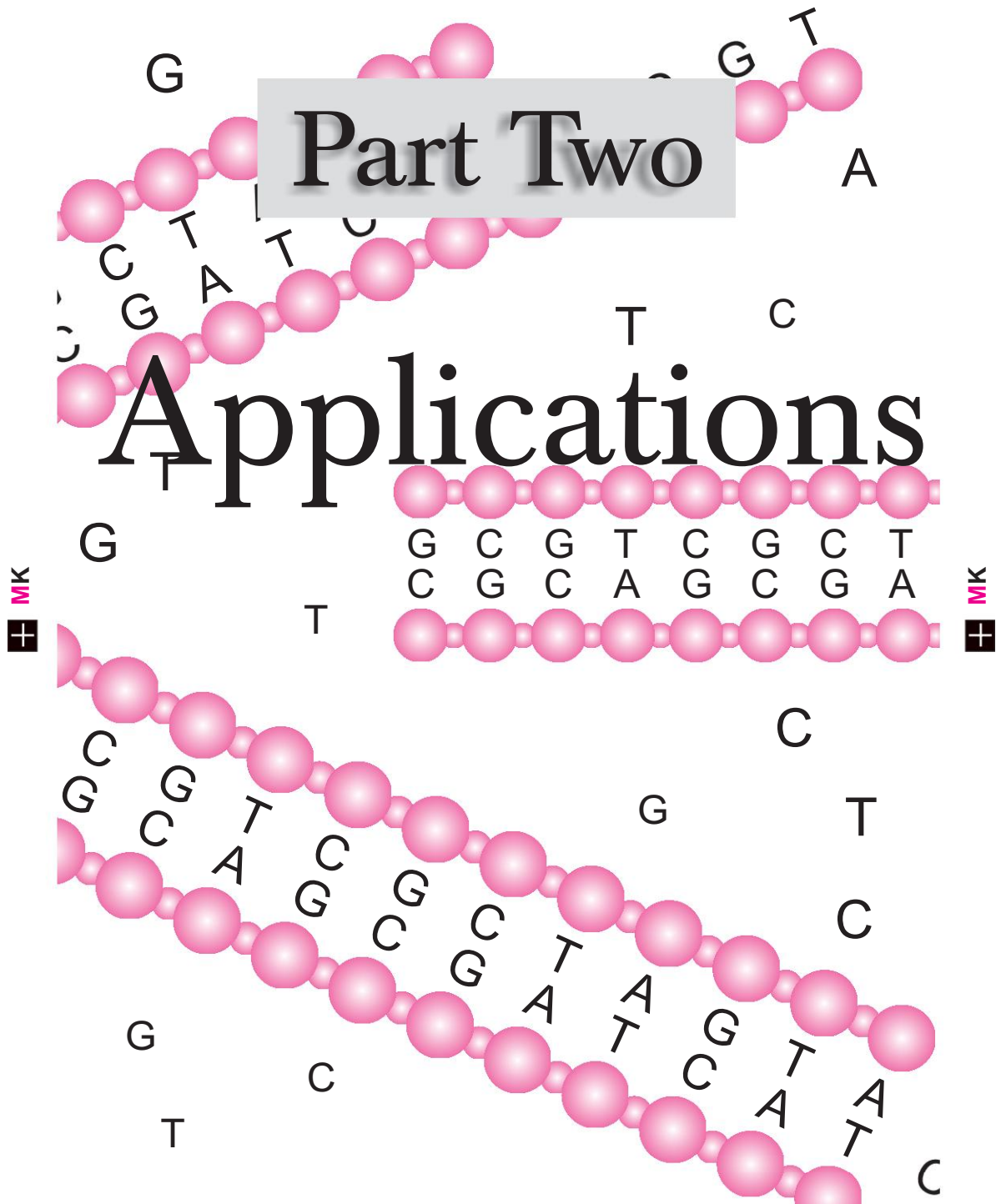
1  msagpqwapap  rslpalcaal  llalqpppv  raegklfvld  sqngsqgldl  etarqscksr
61  gahlvsagel  krvvqdcasa  vcttgwladg  tlgttvcskg  sgeqpvlrai  dvtidshpvp
121  gakynalcik  deerpcgdpp  sfphltlqgr  tglemgdell  yvcvpgsvtg  hretatllc
181  nscgewyglv  qacgkdeaea  hidyeenfpd  drsvsfrelm  edsraegeke  kaqedasdet
241  pkqdrlvfts  vskeniaqek  afvpttglpg  agssfhtdwp  rsrlhrkysl  wfpaetfhks
301  elekdvd det  keplpardth  sdekpapees  etrlvyatty  spsepfadrn  dskaedigvs
361  ssddswldgy  pvtgdawrkv  eagqeddedk  gdgsvgpdds  vlmspdqpik  nvtvissesv
421  iyssispsqm  ldvealvpqp  invseterph  tgdadltyny  stiprrvtq  qspmatpse
481  lttsttqetv  ltlqpthkh  spssnveatq  ppaevtapev  qdnfpyllse  dflgqegpgp
541  gaseerllpt  lapcvgdecp  sfrkgpviat  ivtlcllfl  lavsgavwgy  rrcqhkssvy
601  klnvgqrqar  hyhqqiemek  v

```

- PSI-BLAST was used to analyze members of the BRCT superfamily in the paper by Altschul et al. (Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. (1997) *Nucleic Acids Research*, 25(17): 3389–3402). Describe the rationale and the findings of the study.

Part Two

Applications





Accessing Sequence Information Using BioPerl



7.1 BIOPERL INSTALLATION

We will begin the chapter by learning how to install BioPerl on Windows. The assumption here is that you already have Perl (version 5.005 or 5.6) running on your system. If not, please download Perl (for example, from the ActiveState website: <http://aspn.activestate.com/ASPN/Perl/Downloads/>), before proceeding.

Most Perl installations come with an in-built mechanism to download Perl modules. For the ActiveState Perl installation, this is called the Programmer's Package Manager (PPM). PPM provides a command-line interface for searching, installing, updating or removing modules from your system. We will use PPM to install the BioPerl package.

PPM makes the job of installing BioPerl very straightforward. All you need to do is point it to the source of the distribution, search for the particular package you want to install and issue the install command. PPM does the rest



for you. These steps are illustrated in Figures 7.1 to 7.7. To invoke the Programmer's Package Manager, start the DOS prompt, change directory to where Perl is installed (for example, C:\Perl) and just type ppm (lower or upper case). This will bring you to the PPM interactive shell with the ppm> prompt (Figure 7.1).

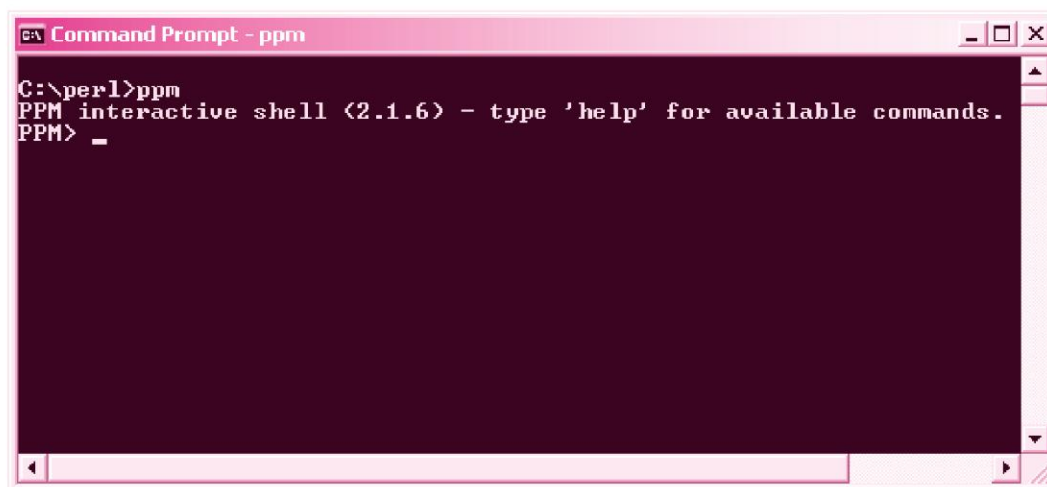


Fig. 7.1 Invoking PPM

Next, specify the location (URL for the repository) of the package that you want to download. The command for this is:

```
ppm> set repository name location
```

In our case, the location has the URL: <http://bioperl.org/DIST/> and the command, therefore, becomes (Figure 7.2):

```
ppm> set repository bioperl http://bioperl.org/DIST/
```

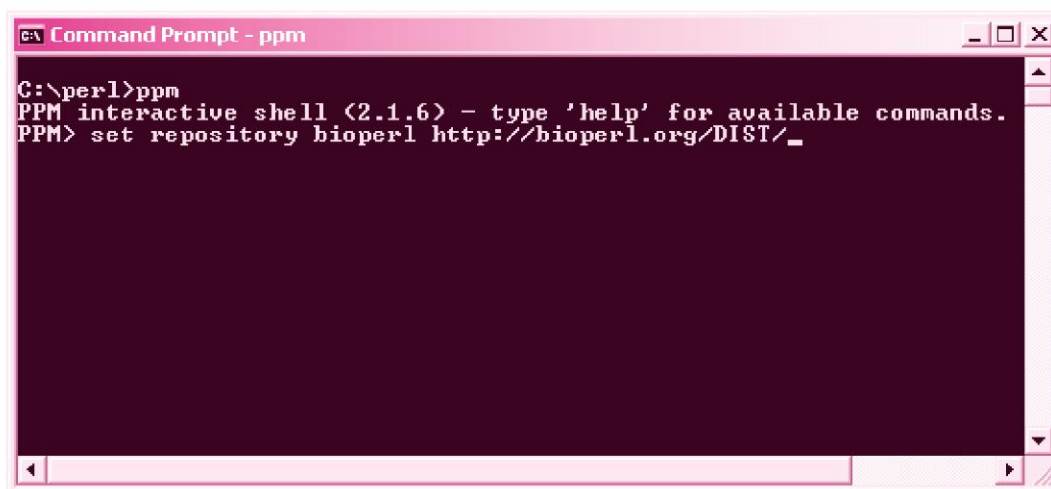
Next, we search for bioperl with the search command:

```
ppm> search bioperl
```

which gives us a list of available packages (Figure 7.3).

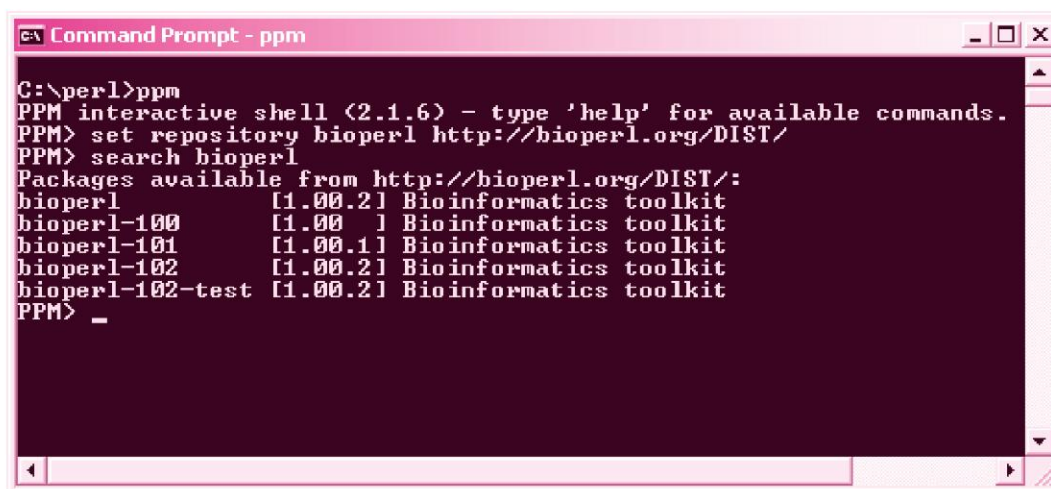
At this point, typing:

```
ppm> install bioperl
```



```
C:\perl>ppm
PPM interactive shell (2.1.6) - type 'help' for available commands.
PPM> set repository bioperl http://bioperl.org/DIST/
PPM>
```

Fig. 7.2 Specifying a repository to use



```
C:\perl>ppm
PPM interactive shell (2.1.6) - type 'help' for available commands.
PPM> set repository bioperl http://bioperl.org/DIST/
PPM> search bioperl
Packages available from http://bioperl.org/DIST/:
bioperl      [1.00.2] Bioinformatics toolkit
bioperl-100  [1.00.1] Bioinformatics toolkit
bioperl-101  [1.00.1] Bioinformatics toolkit
bioperl-102  [1.00.2] Bioinformatics toolkit
bioperl-102-test [1.00.2] Bioinformatics toolkit
PPM>
```

Fig. 7.3 Searching for available packages

will install the package on to your computer (Figures 7.4–7.6).

You can return to the DOS prompt with the quit command (Figure 7.7).

The messages that appear during the installation indicate where the package has been installed—in this case it is the D:\Perl\site\lib\ directory; it may be in a different drive on your computer, for example, C:\Perl\site\lib\ etc. The

```

C:\perl>ppm
PPM interactive shell (2.1.6) - type 'help' for available commands.
PPM> set repository bioperl http://bioperl.org/DIST/
PPM> search bioperl
Packages available from http://bioperl.org/DIST/:
bioperl          [1.00.2] Bioinformatics toolkit
bioperl-100      [1.00.1] Bioinformatics toolkit
bioperl-101      [1.00.1] Bioinformatics toolkit
bioperl-102      [1.00.2] Bioinformatics toolkit
bioperl-102-test [1.00.2] Bioinformatics toolkit
PPM> install bioperl
Install package 'bioperl?' <y/N>:

```

Fig. 7.4 Installing BioPerl

```

C:\perl>ppm
PPM interactive shell (2.1.6) - type 'help' for available commands.
PPM> set repository bioperl http://bioperl.org/DIST/
PPM> search bioperl
Packages available from http://bioperl.org/DIST/:
bioperl          [1.00.2] Bioinformatics toolkit
bioperl-100      [1.00.1] Bioinformatics toolkit
bioperl-101      [1.00.1] Bioinformatics toolkit
bioperl-102      [1.00.2] Bioinformatics toolkit
bioperl-102-test [1.00.2] Bioinformatics toolkit
PPM> install bioperl
Install package 'bioperl?' <y/N>: y
Installing package 'bioperl'...
Bytes transferred: 1165497

```

Fig. 7.5 Installing BioPerl

BioPerl modules however, will always be stored in a sub-directory called Bio, which means that to use them, you have to include the following statement in your scripts:

```
use Bio::module;
```



```

C:\ Command Prompt - ppm
Installing D:\Perl\site\lib\Bio\DB\GFF\RelSegment.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\transcript.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\alignment.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\clone.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\none.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Util\Rearrange.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Util\Binning.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\memory_iterator.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\biofetch.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\memory.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\mysqlopt.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\caching_handle.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\mysql.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\iterator.pm
Writing D:\Perl\site\lib\auto\bioperl\packlist
PPM>

```

Fig. 7.6 Installing BioPerl

```

C:\ Command Prompt
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\alignment.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\clone.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Aggregator\none.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Util\Rearrange.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Util\Binning.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\memory_iterator.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\biofetch.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\memory.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\mysqlopt.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\caching_handle.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\mysql.pm
Installing D:\Perl\site\lib\Bio\DB\GFF\Adaptor\dbi\iterator.pm
Writing D:\Perl\site\lib\auto\bioperl\packlist
PPM> quit
Quit?
C:\perl>

```

Fig. 7.7 Quitting PPM

Modules in Perl end with the extension `pm` (for Perl module). For the module called `SeqIO.pm`, for example, the use statement becomes:

```
use Bio::SeqIO;
```

Note that by convention, the extension `pm` is omitted from the use statement. The use statement simply provides Perl an indication of where the modules are located. In reality,


```
use Bio::SeqIO
```

translates to:

```
use Bio/SeqIO.pm
```

where `Bio/SeqIO.pm` is simply the path to the `SeqIO.pm` file in the directory `Bio`. You don't need to use the full path because that is already stored in a special Perl array variable called `@INC` that contains the list of directories that Perl should search (for modules) while executing scripts. To find what is stored in the variable you can run a simple one-line script that prints the variable out:

```
print "@INC\n";
```

On my system, this prints out:

```
D:/Perl/lib D:/Perl/site/lib
```

as expected.

Note

If, for some reason, you have downloaded the (BioPerl) modules in a directory other than the default location, Perl will not be able to find them and you will get an error that may look like this:

```
Can't locate Bio/SeqIO.pm in @INC (@INC contains: D:/Perl/lib D:/Perl/site/lib) at C:\perl\getids.pl line 6.
```

```
BEGIN failed—compilation aborted at C:\perl\getids.pl line 6.
```

To avoid the error, you should explicitly tell Perl where to look for the modules by using the “`use lib`” statement. If you have stored the modules in `D:\myModules\`, then you should include the following statements at the top of the script. For Windows:

```
use lib 'D:\myModules\';
```

```
use Bio::SeqIO;
```

If you are on Unix, the statement should be placed after the `#!/usr/bin/perl` (or its equivalent on your system) line:

```
#!/usr/bin/perl
```

```
use lib "/home/myModules/";
```

```
use Bio::SeqIO;
```



7.2 BIOPERL MODULES

If the BioPerl installation has been completed, without any errors, you will see the following 25-odd directories on your computer (in the D:\perl\lib\site directory, for example):

Align	AlignIO	Annotation
Biblio	DB	Event
Factory	Graphics	Index
LiveSeq	Location	Map
MapIO	Root	Search
SearchIO	Seq	SeqFeature
SeqIO	Structure	Symbol
Tools	Tree	TreeIO
Variation		



Each of these represents a top level folder containing several BioPerl modules. For example, the Seq directory contains the following modules:

LargePrimarySeq.pm
 LargeSeq.pm
 PrimaryQual.pm
 Quall.pm
 RichSeq.pm
 RichSeqI.pm
 SeqWithQuality.pm

Each of these modules, in turn, carry out a specific function. A list of the commonly used modules and their functions are outlined in Table 7.1. We will add to this list of modules in later chapters.



Table 7.1 *BioPerl modules*

Top-level folder	Modules	Function
Align	AlignI	Provides an interface to describe sequence alignments
AlignIO	bl2seq	Provides methods for sequence alignments
	clustalw.pm	Provides methods to read and write ClustalW flat file databases
	emboss.pm	Provides parsing and writing pair-wise sequence alignments from the EMBOSS suite
	fasta.pm	Provides fasta sequence input/output stream methods
	meme.pm	Provides methods to manipulate meme output
	pfam.pm	Provides methods to transform Bio::SimpleAlign objects to and from pfam flat file databases
	phylip.pm	Provides methods to transform Bio::SimpleAlign objects to and from interleaved phylip format
	psi.pm	Provides methods to read and write PSI-BLAST profile alignment files
DB	prodom.pm	Provides methods to read and write Prodom flat file databases
	GenBank.pm	Allows the dynamic retrieval of sequence objects (Bio::Seq) from the GenBank database at NCBI, via an Entrez query
	EMBL.pm	Allows the dynamic retrieval of sequence objects from the EMBL database using the dbfetch script at EBI: http://www.ebi.ac.uk/cgi-bin/dbfetch
	Fasta.pm	Provides indexed access to one or more Fasta files allowing the retrieval of very large sequences
Seq	LargePrimarySeq.pm	Stores a very large sequence (100s of MB long) as a series of files in a temporary directory
	LargeSeq.pm	
	PrimaryQual.pm	Associates sequences with their corresponding quality values
	SeqWithQuality.pm	

(Contd.)

(Contd.)

	RichSeq.pm	Implements a sequence and an interface for sequences created from a rich sequence database such as EMBL, GenBank and SwissProt
SeqFeature	RichSeqI.pm	
	FeaturePair.pm	Holds information about a sequence feature on two coordinates: the genomic sequence and the corresponding protein sequence.
	Generic.pm	Provides all information for a feature on a sequence
	Similarity.pm	Provides information on sequence features based on similarity, for example bit score, % identity, etc.
	SimilarityPair.pm	Provides information on the similarity between two sequences
SeqFeature::	Exon.pm/ExonI.pm	Implements a feature representing an exon, an intron, a gene structure, a transcript, a poly adenylation site, an untranslated region respectively
	Intron.pm	
	GeneStructure.pm/	
	GeneStructureI.pm	
	Transcript.pm/	
	TranscriptI.pm	
	Poly_A_site.pm	
SeqIO	UTR.pm	Handles interconversions for Bio::Seq objects to and from the ace, BSML*, EMBL, Fasta, GCG, GenBank, SwissProt and raw file format respectively
	ace.pm	
	bsml.pm	
	embl.pm	
	fasta.pm	
	gcg.pm	
	genbank.pm	
	swiss.pm	
	raw.pm	
	MultiFile.pm	Joins a large number of files of a particular format (e.g., Fasta) into a single stream

*Bioinformatic Sequence Markup Language or BSML is an extensible language specification based on XML (Extensible Markup Language) and SGML (Standard Generalized Markup Language) for the storage, display and dissemination of bioinformatic data. It enables the integration of data of a diverse kind: sequence, annotation, images, etc. into one standard format.



7.3 OBJECT ORIENTED PROGRAMMING

The use of BioPerl modules requires a basic understanding of object oriented programming (OOP). Some essential concepts are outlined below.

The object—which is at the heart of OOP—is simply a term used to visualize the entity being modeled or programmed. The idea is that if you can visualize the object, it is easier to build a program that mimics the properties and the behaviors of the object. Virtually anything you see around you can be an object—a calculator, your telephone, computer, etc. As is perhaps obvious, each such object has some general and some unique characteristics associated with it—every calculator, for example, has buttons for numbers and mathematical operations and a display that allows you to see the results of a computation. However, calculators can be of different types—you may have a simple calculator that just does additions, subtractions, multiplications and divisions. On the other hand, you may have a specialized calculator with advanced mathematical software that allows you to solve algebraic expressions or a calculator that print results on paper tape.

MK



A class is a container that contains the object. It is the OOP way of representing an object. In the above example, the calculator is a class that represents the object of type calculator. The general methods that are common to all objects of a certain type are called the class methods. The ability to perform a basic operation such as adding two numbers is a fundamental operation and an example of a class method. Every object of the class calculator will have this method. On the other hand, the ability to solve complex algebraic expressions is a specialized operation that may be present on some but not all calculators. This is an example of a method that is associated with a particular type of calculator. Such methods are called instance methods because they are specific to a particular 'instance' of an object of the class calculator.

MK



Similarly, properties of an object—for example, the ability to print calculations on a paper tape that makes a calculator unique (as compared to an instrument that outputs results on an electronic display)—are called instance variables, i.e., the variables associated with an object. When the entire class, its methods and variables are stored in one file with the same name as the class, it becomes a Perl module. For example, for the class calculator, the file (the module) will be called calculator.pm.

Once a class that models the behavior of a certain object is created, how is the object used in code? This is done by a process known as instantiation because it creates an instance of the class. The OOP keyword to do this is the term 'new'. When a new object of a class is instantiated, the function is called *class constructor*. These terms will become clearer as we work with some examples.

We will begin with the `Bio::SeqIO` module which handles IO functions and is commonly used to interconvert sequences from one format into another, for example, fasta into EMBL. The module also provides methods for a large number of operations that can be performed on sequences. We will understand how to perform percentage GC calculation on a DNA sequence in a Fasta file using the `Bio::SeqIO` module.

It would be helpful to see how this operation is performed using standard Perl for a single sequence file (Listing 7.1).

Listing 7.1 Using standard Perl for per cent GC calculation

```
#!/usr/bin/perl
$/ = undef;
use Getopt::Long;
(GetOptions("f|filename=s" => \$file));
open (IN, $file) or die "Cannot read $file: $!\n";
$line = <IN>;
$a = ($line =~ tr/A//);
$t = ($line =~ tr/T//);
$g = ($line =~ tr/G//);
$c = ($line =~ tr/C//);
$total = ($a + $t + $g + $c);
$gc = (($g+$c)/$total)*100;
print "A          : $a\n";
print "T          : $t\n";
print "G          : $g\n";
print "C          : $c\n";
print "Total       : $total\n\n";
printf "GC content : %.1f%\n", $gc;
```




7.4 USING BIOPERL

The steps to write a program using the BioPerl module are fairly straightforward.

1. Include the BioPerl module in the program: As with standard Perl modules, BioPerl modules are included in programs with the `use` statement:

```
use Bio::SeqIO;
```

2. Instantiate an object of the class. The `new()` class method instantiates a new `Bio::SeqIO` object:

```
$filestream = Bio::SeqIO->new(-file => $filename, -format => 'fasta');
```

This line of code requests a stream object for a particular format (in this case Fasta). The newly created object can then be used to manipulate sequence information. In this case, `new()` accepts the following parameters:

`file`: A file path to be opened for reading or writing.

The following conventions apply:

```
'file'      :    open file for reading
'>file'     :    open file for writing
'>>file'    :    open file for appending
'+<file'    :    open file read/write
```

You can specify the `$filename` parameter through the `Getopt::Long` module via the `Getoptions()` function or on the command-line using the `$ARGV[0]` variable.

`format`: The file format can be EMBL, Fasta, SwissProt, GenBank, etc.

3. Call the methods provided by the object. Each stream object has the following functions:

```
$stream->next_seq();
```

This function reads the next sequence object in the stream.

In addition, once you create a sequence object `$seq` from the input stream, you can use the `moltype`, `desc`, `id`, and `seq` methods to extract information about the current sequence being read:

```

$seq    = $filestream->next_seq();
$type   = $seq->moltype();
$id     = $seq->id();
$desc   = $seq->desc();
$dnaseq = $seq->seq();

```

The right arrow notation invokes the object method and substitutes the parameter to the left of the arrow as the first parameter passed to the object method.

For a DNA sequence in Fasta format:

```

>gij153423|gb|M88615.1|STMRI8ON S. aureofaciens ribonuclease gene, complete cds
CGGCGGAACGGACAACGGCGTCCGTCCCGCCCGGCCGTACGTCCTTGTTGGTGTGCGCGGGCGGTGCCCCGT
CGCGTGGTGCTCGTGTGTGGTGTCCGTATGTGTGGGACACGCCGCGCGCGCGCGCGCCTGCCAAGC
TGGACCGCATGACCAGAAGCAAGAGCCCGCTCGTGTGCGGGGCCGTCTGATCCTGGCCGTGCTCGCCGG
GGTTGGGTACCTGCTCGCCGGCAGGGGCGGCGACACCCACCCCAAGGCCGCGCGAGCTCCGCCGCCGCG
GGCACCTCGGCCCCCAATCCTCGGCTCCCAAGCCGTCCCCGCCCGCGCGCGCGCCTGGACGCCCCGCC
ACCCGGCGCTGGCCGACGTCTGCCGACCAAGCTGCCAGCCAGGCCAGGACACCCCTGCGCCTGATCGC
CAAGAACGGCCCCCTACCCGTACAACCGGGACGGCGTCTTTCGAGAACC GCGAGAGCCGCCTGCCGAAG
AAGGGCAACGGCTACTACCACGAGTTACCGTGGTCAACCCCGGCTCCAACGACCGAGGCACCCGACGGG
TCGTACCCGGCGGCTACGGCGAGCAGTACTGGTCCCCGGACCACTACGCGACCTTCCAGGAGATCGACCC
GCGCTGCTGAGCGCTGGCCTCACAGAAAATACTTTCCAACCTGGAAAGGTCGTGGCACAGTGGAGCCACC
GTCACCTTCCGAGGGGGAACATGAGTGACTCCGCTCCACCCCGCGCTCCGCCGCCGAGGCACGGCCTG
CCGACCGGGCGCGCGGCTGCCCGGCGCCCGGGCGAGCTGCCCCGCTGGTACGGGCGCGCCTTCGCCTT
CGCCTTCGCGCTCTACGGCCTGGGGATCGGCCACGTATCGAGACCGGGCAGATCGCCGTGATCGGCGGG
CTCGCGCTTCGCCGCCCTGACCGGTGGCCTCGCGGCCTTCGCCATGCGCAGCGACGGCATCGTCCGC

```

the Bio::SeqIO module provides the following methods:

```

moltype*      : Type of molecule (DNA or protein)
desc          : Description of the molecule (from the Fasta header)
display_id**  : Sequence identifier (from the Fasta header)
seq           : The actual sequence itself

```

*`motype` has been replaced by a new method: `alphabet()`. If you are using an older version of BioPerl, you may get an error saying:

“`motype: prev1.0 method. Calling alphabet() instead`”.

In this case, use `alphabet()`.

**The `display_id` field is the common name or the identifier of the sequence. This is the LOCUS field of the GenBank/EMBL databanks and the ID field of the SwissProt/sptrembl database.

The values of the individual fields (for the *Streptomyces aureofaciens* ribonuclease gene) are:

```
motype      : DNA
desc        : S. aureofaciens ribonuclease gene, complete cds
display_id  : gi|153423|gb|M88615.1|STMRIBON
```

and the sequence,

seq :

```
CGGCGGAACGGAACAACGGCGTCCGTCCCGCCCGGCCGTACGTCTTGTGGTGTGCGCGGGCGGGTGCCCGTC
GCGTGGTGCTCGTGTGGTGTCCGTATGTGTTGGACACGGCCGGCCGGCGCGGGCGCCTGCCAAGCTG
GACCGCATGACCAGAAGCAAGAGCCCGCTCGTCGTGCGGGCCGTCTGATCCTGGCCGTGCTCGCCGGGGT
TGGGTACCTGCTCGCCGGCAGGGGCGGCAGCACCCACCCCAAGGCCGCCGCGAGCTCCGCCGCCGCGGGCA
CCTCGGCCCCCAATCCTCGGCTCCCAAGCCGTCCCGCCCGCGCGGCGCCTGGACGCCCGCCGACCCG
GCGCTGGCCGACGTCTGCCGCACCAAGCTGCCAGCCAGGCCAGGACACCCTCGCCCTGATCGCCAAGAA
CGGCCCTACCCGTACAACCGGGACGGCGTCTTTCGAGAACC GCGAGAGCCGCTGCCGAAGAAGGGCA
ACGGCTACTACCACGAGTTCACCGTGGTCACCCCGGCTCCAACGACCGAGGCACCCGACGGGTGCTACC
GGCGGTACGGCGAGCAGTACTGGTCCCGGACCACTACGCGACCTTCAGGAGATCGACCCGCGCTGCTG
AGCGCTGGCCTCACAGAAAATACTTTCCAACCTGGAAAGGTCGTGGCACAGTGGAGCCACCGTCACTTTCC
GAGGGGGAAACATGAGTGACTCCGCTCCACCCCGCGTCCGCCCGCGAGGCACGGCCTGCCGACCGGGCC
GGCGCGGTGCCCGCGCCCGGGCGAGCTGCCCCGCTGGTACGGGCCGGCCTTCGCCTTCGCCTTCGCGCT
CTACGGCCTGGGGATCGGCCACGTCATCGAGACCGGGCAGATCGCCGTGATCGGCGGGCTCGCGCTTCGCC
GCCCTGACCGGTGGCCTCGCGGCCTTCGCCATGCGCAGCGACGGCATCGTCCGC
```

The code to read a complete DNA sequence file and to calculate its GC content using the `Bio::SeqIO` module is as follows (Listing 7.2).

Listing 7.2 Calculating per cent GC using BioPerl

```

use Bio::SeqIO;
use Getopt::Long;

GetOptions("f|filename=s"=>\$filename);

$filestream = Bio::SeqIO->new(-file => \$filename, -format => 'fasta');
my $seq = $filestream->next_seq();
$id      = $seq->id();
$desc    = $seq->desc();
$type    = $seq->moltype();

print "ID: $id\nMolecule type: $type\nName: $desc\n\n";

$dnaseq = $seq->seq();
$a = ($dnaseq =~ tr/[Aa]//);
$t = ($dnaseq =~ tr/[Tt]//);
$g = ($dnaseq =~ tr/[Gc]//);
$c = ($dnaseq =~ tr/[Cc]//);

$total = ($a + $t + $g + $c);
$gc = (($g+$c)/$total)*100;

print "A          : $a\n";
print "T          : $t\n";
print "G          : $g\n";
print "C          : $c\n";

print "Total       : $total\n\n";
printf "GC content : %.1f%\n", $gc;

```

The execution and output of the script is shown in Figure 7.8.

```
C:\perl>atgc.pl -f sa_rnase.txt
```

where, *sa_rnase.txt* is a file that contains the ribonuclease gene from *S. aureofaciens* (gi 153423) in Fasta format.

```

C:\perl>atgc.pl -f sa_rnase.txt
ID: gi11534231gb1M88615.11STMRI B0N
Molecule type: dna
Name: Streptomyces aureofaciens ribonuclease gene, complete cds

A      : 137
T      : 130
G      : 324
C      : 386
Total  : 977

GC content : 72.7%

C:\perl>

```

Fig. 7.8 Running atgc.pl with such sa_rnase.txt as input file

7.5 THE write_seq() FUNCTION



The write_seq() function:

```
$stream->write_seq($seq);
```

writes a sequence object to a user specified location. An application of the above function is to interconvert sequences from one format to another. For example, the script in Listing 7.3 will change a Fasta format file into an EMBL format file.

Listing 7.3 Converting a Fasta file into EMBL format

```

use Bio::SeqIO;
use Getopt::Long;

GetOptions("i|infilename=s"=>\$infilename,
"o|outfilename=s"=>\$outfilename);

$instream = Bio::SeqIO->new(-file => \$infilename, -format => 'fasta');
$outstream = Bio::SeqIO->new(-file => ">\$outfilename", -format => 'EMBL');

```

(contd.)

(contd.)

```
while(my $seq = $instream->next_seq()) {  
    $outstream->write_seq($seq);  
}
```

Let us see the output of this program using a Fasta file of two Zebrafish (*Danio rerio*) kinase sequences:

>gi|28856247|gb|BC048050.1| *Danio rerio*, Similar to creatine kinase, mitochondrial 1 (ubiquitous), clone MGC:55538 IMAGE:2642172, mRNA, complete cds

GTACAGCACCACTTGAGAAGACCAACTTCTGCTGGATTGAGAAGCATCTGACCACATTCATCTGGAGT
GCTCTGTTCTGCGGTTGAGGTGTTAAATGGCAAGCAGCTTCGCACGGATTTTGTGAGGTAACAGGAAGG
TTGGCATCTTGTGCTGGTGGTGGGATCTCTGACCGTCGGGTTCTTCTGAACAGGGAGCAGCATGT
CAGCGCAGGATCGAGCGTCCGGAGAATCTATCCCCGAGTGCTGAATATCCAGATTTGCGTAAGCACAAAT
AACTGTATGGCCAGTCACCTGACTCCTGCCGTATACGCAAAGCTGTGTGATAAATCCACTCCGAACGGTT
ACACTTTGGACGAAGCCATTGAGCTGGCGTGGACAATCCAGGTCATCCTTTATAAAGACAGTAGGAAT
GGTGGCAGGAGATGAAGAGTCATATGAGGTTTTTGTGACATCTTCAACCCAGTCATCAAAGAAAGGCAC
AATGGTTATGACCCCTGCAACATGAAACACCCCACTGACCTGGATTCCAGTAAGATACGAGGAGGCATGT
TTGATGAGAAGTACGTGCTGTCTTCTCGAGTCAGGACGGGAGGAGTATCCGAGGCTGAGTCTCCCCC
GGCCTGCACCCGTGCTGAGCGCAGAGAGGTGGAGAGGGTTGTGGTTGATGCTTTGGCAGGCCTAAAAGGG
GATTTGACTGGAATACTACAGCCTGACTGTAATGACTGAACAGGAGCAGCAGCAGCTTATTGATGATC
ACTTCTGTTTGATAAACCTGTATCGCCATTGCTGACATGTGCGGGTATGGCTCGAGATTGGCTGACGC
TAGAGGCATCTGGCACAACAATGAGAAAACCTTCTGGTGTGGATCAACGAGGAAGATCACACCCGTGTG
ATCTCCATGGAGAAGGGAGGCAACATGAGAAGGGTCTTTGAGCGTTTCTGCAAGGGTCTCCAAGAGTTG
AGAGACTAATTCAGGAGAAGGGTTGGGAATTCATGTGGAATGAGCGTCTGGGTTACATTCTCACCTGTCC
GTCAAACCTGGGCACTGGGCTGCGAGCTGGAGTCCATGTTAATCTTCTCGCCTCAGCAAGGACCCTCGC
TTTTCTAAATCCTGGATAACCTGCGGCTCCAGAAAAGAGGGACTGGAGGAGTGGACACGGCTGCTGTTG
GAAGCACTTTTGATATTCCAATCTGGACAGGCTGGGCAATCAGAGGTCCAGCTGGTGCAGACTGTGAT
AGACGGAGTGAATATCTCATTGAATGTGAGAGGAACTGGAGAAAGGCCAAGACATCAAATCCCCGCC
CCCATCAGCCAGTTTAAATAGACTGGGTGGTCTTTCGTAGCTCCTCCCTCTCTCCTGGCTCCACCTCTC
GTGTGCTCCTCCACCTTAAATATTCCGTAAAACCGATCCATAAAGCATGCCTCTGTGTTACATCCGA
CATATTCGACACTCCAGTATAACGCAGGAGTGAATGTATCGTACATCATGATTGTGTTCAATTTGTGGCGT
TAATGATTTGTACAATGTATCTAAGTATTGTGTTGCAATAACGTTATTAGAGCCATTGCTGAAAAAT

GCCTCAGGTGTACAAGAATTAACAGGTTTTGTGGAAGATGCTGATGATATAACTTGCTGACGGATCCT
GAGAGTGTATTATTGTGTGCTTCACTGATGTTATACTTAAAGGAAGTGCTTTAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

>gi|28856170|gb|BC048055.1| Danio rerio, Similar to NIMA (never in mito-
sis gene a)-related kinase 2, clone MGC:55602 IMAGE:2643611, mRNA,
complete cds

GTTTTCTCAACTCACAAAATACGTACACGCGTTTTAACTAAGGGTAAACAATAAACACACTGGCAGTAA
AACCACAGCGTTCAAGTAGTGAGTGAGTTACGTGTGTTGGATCTGCTGAGTTGCTGTTGAACGCGATGC
CATCCAAAACCTGAAGACTACGAGGTGCTGCTCACCATAGGATGTGGATCTTATGGAAAATGCCAGAAAAT
CAAGAGGAAATCCGATGGAAGATTCTGGTCTGGAAGTTCTGGACTATGGCACTATGGCCGAGGGAGAG
AAACAGATGCTGGTGTGAGAAGTCAACTTGCTCCGTGAGCTGAAGCACCCAAATATCGTCCGATACCATG
ACCGAATTATTGACAGAACGAACACGACATTATATAGTGATGGAATACTGTGAGGGTGGAGATCTCGC
CAGCCTCATCAACAGAAGCATCAAAGACAAGCGATACCTGGAGGAAGAATTCATCCTCCGTGTGATGGCA
CAGTTGTCTCTGGCGTTAAAGAATGCCATGGTAGGAGTAACGGCAGCAGTACAGTTCTGCACCGAGACC
TGAAACCAGCAAACATCTTTCTGGATGCCAAACAGAATGTAAAGCTTGGCGATTCGGTTTAGCTCGCAT
ACTAAACCACGATACAAGCTTTGCTAAACGTTTGTGGAACGCCATATTACATGTCGCCAGAACAATG
AATCGCATGTCCTATAATGAGAAATCGGATATATGGTCTTTAGGGTGTCTACTCTATGAACTATGTGCTT
TATCGCCCCATTTACAGCATACAACCAGACAGAGCTGGCTCGAAAAATCAGAGAAGGCAGATTTCAAG
AATCCCATACCGATACTCGGATGAGCTAAACACACTGCTTTCAAAAATGCTCAACTTAAAGGATTATCTG
AGGCCCTCTGTGGAGTCCATCCTGCAGAATGGTTTGATCTCCGTTATGTGGCCCTCGAGCAGAAGAGGC
TCCAGGAGAAACAGCGGCGCAGATCAGATGAGGCAGAGCAGCCCAAACATCCAGAGTCAACCACTTCTGGC
AGAGCTGCGGCTTAAAGAGCAGATTCTCGAGAGCGAGAGCAGGCCCTCAAAGAGCGAGAGCAGCGGCTA
GAGCAAAGGGAACAAGAACTGTGTGTCCGAGAACAGCAAATAATGAAAAGCTGGTCAGAGCCGAGAGCA
TGTGAAGGCGTTTAATCTGATTGACAGCAGAGGGCGCTATCTCTGCTCAGCGCCAGCGACACAGAGAA
TGAAGAGAACATCTCTCCAGGGAAAAAGAGGGTTCACTTTGCAGGAGACGGGAAGGAGAACGGCAGACTG
ATCATGAAACCTCAGGAGCACATCCTTGAGAAGAGACACCAGCTGATGAACAAGCGCATAACAGACTCG
GAGAGGAGGAGAAGATGATCCACTCGCCAAAACACAGAGAAATGCAGGGAATCCGCTAGCCTTCAAAGAC
ACTGCCAGTGTGGTCAATCACGTGAGAGGATTCGTCTAGTTGTGTATTAAGAGATAGATTCCTGCT
TTAATTTATTTGCTGCAGCATTTGTACAGTACACTGGAGCATTTCAATTAAGGGACAGCTAACCCAAA
TTCAAACTCTCTTATCATTTATTCACCTCCAATTTTCCAAACCTGTTGTTTCTTTTTTTATTTTCT
GATCACACAGAAATGAAGATATTTAGAGAAATGTTGGGAATCAGTAGCCATTAACCTTAATAAAATTTGT
TATATCCCACTTTAGATGTCTGACTATCAATTACAAACATTCTTCCAAATATCTCCTTTTGTGAAGAGAA
AGGGAACATCTTGAGGGTGAATAAATGGTAAGTAATGTTTTATTTGGGTGAAGTGTCTTTTAAATGTGC
ATTACACTGGACTATTATATAATAGCATTTTATGGATGTTTTATTGCTAGAAGCATTGTTTTATTTGGA
ATAAATAAATGAATAATGGTTGCAAAAAAAAAAAAAAA

To run the program, save the sequences in a file called `kinase.fasta`, save the script as `fasta2embl.pl` and specify the input and output files as in Figure 7.9. Figure 7.9 also shows the output in EMBL format using the above script (only the first sequence is shown).

```

C:\perl>fasta2embl.pl -i kinase.fasta -o kinase.embl

C:\perl>more kinase.embl
ID      gi|28856247|gb|BC048050.1|standard; DNA; UNK; 1812 BP.
XX
DE      Danio rerio, Similar to creatine kinase, mitochondrial 1 (ubiquitous),
DE      clone MGC:55538 IMAGE:2642172, mRNA, complete cds
XX
FH      Key
FH      Location/Qualifiers
XX
SQ
Sequence 1812 BP; 525 A; 389 C; 457 G; 441 I; 0 other;
gtacagcacc acacttgaga agaccaactt ctgctggatt cagaagcacc tgaccacatt      60
catctggagt gctctgttct gcggttgagg tgttaaaatg gcaagcagct tcgcacggat      120
tttctcaggt aacaggaagg ttggcatctt gtcgctggtc ggtgcgggat ctctgaccgt      180
cgggttcttc ttgaacaggg agcagcatgt cagcgcagga tcgagcgtcc ggagaattcta      240
tcccccgagt gctgaatatc cagatttgcg taagcacaat aactgtatgg ccagtcacct      300
gactcctgcc gtatacgcaa agctgtgtga taaatccact ccgaacgggt acactttgga      360
cgaagccatt cagactggcg tggacaatcc aggtcatcet ttcataaaga cagtaggaat      420
ggtggcagga gatgaagagt catatgaggt ttttctgac atcttcaacc cagtcatcaa      480
agaaagggcac aatggttatg acccctgcaa catgaaacac cccactgacc tggattccag      540
taagatacga ggaggcatgt ttgatgagaa gtactgtctg tcttctcgag tcaggacggg      600
caggagtatc cgaggcctga gtctccccc gccctgcacc cgtgctgagc gcagagaggt      660
ggagaggggt gtggttgatg ctttggcagg cctaaagggt gatttgactg gaaaatacta      720
cagcctgact gtaatgactg aacaggagca gcagcagctt attgatgac acttctgtt      780
tgataaacct gtatcgccat tgcgtacatg tgcgggtatg gctcgagatt ggctgacgc      840
tagaggcate tggcacaaca atgagaaaac ctctctggtg tggatcaacg aggaagatca      900
caccctgtgt atctccatgg agaagggagg caacatgaga aggttcttgy acggtttctg      960
caagggctctc caagaggttg agagactaat tcaggagaag ggttgggaat tcagtgtgaa      1020
tgagcgtctg ggttacattc tcacctgtcc gtcaaacctg ggcactgggc tgcgagctgg      1080
agtccatggt aatcttcttc gcctcagcaa ggaccctcgc ttttctaaaa tcctggataa      1140
cctgcggctc cagaaaagag ggactggagg agtggacacg gctgctgttg gaagcacttt      1200
tgataatttc aatctggaca ggctgggcca atcagaggtc cagctggtgc agactgtgat      1260
agacggagtg aactatctca ttgaatgtga gaggaacctg gagaaaggcc aagacatcaa      1320
aatccccgcc cccatcagcc agtttaaata gactgggtgg tcttctgtag ctctccctc      1380
tctcttggct ccacctctc gtgtgtctct cccaccttaa aatatctcgc taaaaccgat      1440
ccataaagca tgcctctgtg ttacatccga catattcgac actccagtat aacgcaggag      1500
tgaatgtate gtacatcatg attgtgttca tttgtggcgt taatgatttg tcacaatgta      1560
tctaacttga ttgtgttga ataacyttat tagagccatt gctgaaaatt gcctcaggtg      1620
tacaagaatt aaacaggttt ttgtggaaga tgctgatgat ataacttgct gacggatcct      1680
gagagtgttt attgtgtgct tcaactgatgt tatacttaaa ggaagtgcct taaaaaaaaa      1740
aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa      1800
aaaaaaaaaa aa
//
  
```

Fig. 7.9 kinase.embl (EMBL format)

Assignments

1. Download a set of kinases from GenBank and store them as a multiple Fasta file. Write a program to calculate the GC content of each of the sequences.

2. Write two separate scripts, one using the `Bio::SeqIO` module and one using standard Perl, to break up a multiple Fasta file into individual files, each containing a single sequence in Fasta format. The resulting files should be named by the gi or accession number of the DNA sequence it contains.



Appendix I: Installing External Modules

BioPerl has external package dependencies. This means it needs additional packages to provide functionality that it does not have on its own. Some examples are the `IO::Scalar` and `IO::String` modules. These are available as the `Bundle-BioPerl` package and should be downloaded along with your BioPerl installation. Search for 'BioPerl' at the ppm prompt and use the install command as illustrated in Figure 7A below:

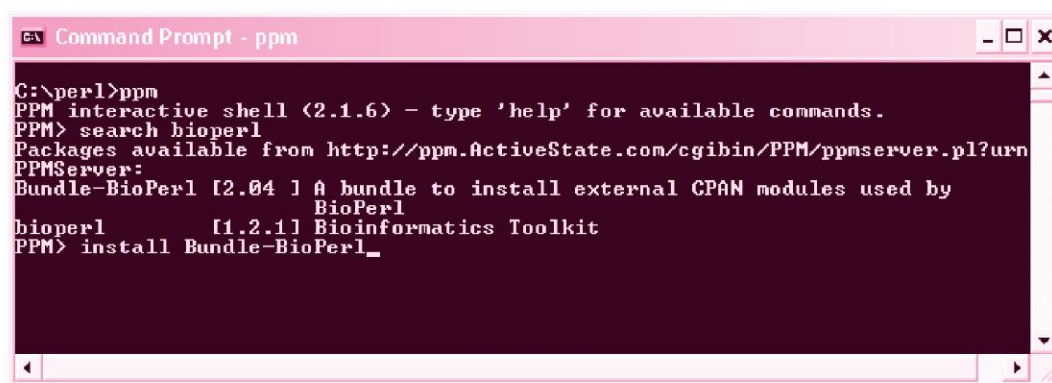


Fig. 7A Installing BioPerl modules

Appendix II: Upgrading BioPerl

BioPerl may have released a new version since you last downloaded it on your system. The steps below illustrate the method to check for new updates and install a recent version.

Step 1: Check for availability of a new version:

```
ppm>verify bioperl
```

If an upgrade is available, ppm will respond with:

“An upgrade to package bioperl is available”

Step 2: To install it, do:

```
ppm> verify --upgrade bioperl
```

After it has been installed, ppm will end with a message saying:

“Package bioperl upgraded to version x.x.x.x”

Appendix III: Testing for Availability of Individual Modules

To test for the presence of individual modules on your system, issue the following command on the command-line

```
> perl -e "use Module"
```

The > sign represents the command prompt (which could be > or %, etc. on Unix or simply, C:\Perl> on Windows). To check for Bio::SeqIO.pm, for example, the command would be:

```
> perl -e "use Bio::SeqIO"
```

If this statement exits without errors, the module has been loaded properly. If not, you will get an error message saying:

```
Can't locate Bio/SeqIO.pm in @INC (@INC contains: D:/Perl/lib
D:/Perl/site/lib) at -e line 1.
```

```
BEGIN failed—compilation aborted at -e line 1.
```



The actual directory paths would be different though. The useful piece of information here is the “(@INC contains: D:/Perl/lib D:/Perl/site/lib)” part which identifies the location where you should place all Perl modules. The above error messages indicate, for example, that there are two locations: D:/Perl/lib and D:/Perl/site/lib where modules can be placed.



If you get the “Can’t locate...” error, you should try to reload the module or manually place it in one of the directories listed in @INC.



Bio::DB::GenBank



MK



8.1 INTRODUCTION

MK



In this chapter we will learn how to automate sequence downloads from GenBank with the Bio::DB::GenBank module. Sequences in GenBank can be accessed via a number of different identifiers such as accession numbers, GenInfo Identifier (GI) numbers and version numbers. There are important differences in these various identifiers.

Accession numbers are unique identifiers for a sequence record and do not change even if the information in the record changes. Accession numbers are combinations of letter(s) and numbers. The accession number for some rice BACs from chromosome 10, for example, are AC080019, AC078839 and AC083945. On the other hand, both the GI number and the version numbers change whenever the sequences change. Sequences are continually submitted to GenBank and as they are updated with more accurate versions, they are assigned a new GenBank number and a new version number. However, they retain the same accession number.

8.2 STRUCTURE OF A GENBANK RECORD

Before we get into the Bio::DB::GenBank module, we will see how a GenBank record looks and understand what different elements a typical GenBank record consists of. The steps to download the GenBank record for the accession number AC080019 are described below.

1. Open the Entrez server at <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi> (Figure 8.1).



Fig. 8.1 Entrez web-server at NCBI

Note

Entrez is a common gateway that provides free access to molecular biology resources such as sequence and structure data and scientific publications. It is a service developed and maintained by the National Center for Biotechnology Information (NCBI). NCBI was founded in 1988, as a division of the National Library of Medicine, under the aegis of the National Institutes of Health (NIH)—the apex body that regulates scientific research in the United States. GenBank, likewise, is a searchable central repository of annotated nucleotide sequences derived from data submitted by researchers from all over the world. Make a permanent link to this site on your computer since you will be using it frequently.

- From the drop-down box called Search on the left, select 'Nucleotide' since you will be downloading DNA sequences (Figure 8.1) for this example.
- Enter the accession number of the BAC (AC080019) in the "Search for" box and press enter. The sequence record should appear as in Figure 8.2. View the sequence data by clicking the accession number that is hyperlinked to the sequence record. The page you get represents what a typical GenBank record looks like (Figure 8.3).



Fig. 8.2 Displaying the AC080019 entry in GenBank

Note some of the useful information present in the header (shown boxed) such as:

- Definition Genomic sequence for *Oryza sativa*, Nipponbare strain clone OSJNBa0094H10, from chromosome X, complete sequence.
- Accession number AC080019
- Total size of BAC 149654 basepairs (bp)
- Chromosomal location Chromosome X

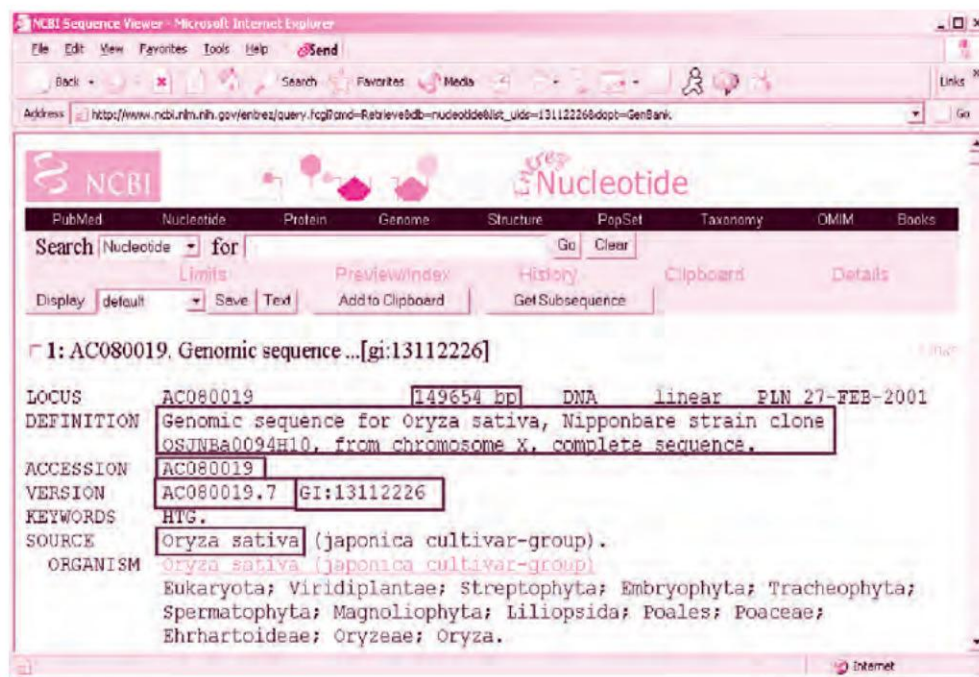


Fig. 8.3 Displaying the AC080019 record

- Clone name OSJNBa0094H10
- Version number AC080019.7
- GI number 13112226
- Source organism Oryza sativa

Scroll down the record to see the actual BAC nucleotide sequence (Figure 8.4) and its base composition (Base Count). Alternatively, select Fasta from the dropdown box next to 'Display' on the left (Figure 8.5) and press the Display button to see the sequence in Fasta format (Figure 8.6).

A record such as this with a header beginning with only a '>' sign on the first line and followed by the complete sequence starting from the second line is called a Fasta record or file.

Note the sequence of the BAC (Figure 8.6) with a single header line that reads:

```
>gi|13112226|gb|AC080019.7|AC080019 Genomic sequence for Oryza
sativa, Nipponbare strain clone OSJNBa0094H10, from chromosome X,
complete sequence
```

```

BASE COUNT    39952 a   35699 c   35539 g   38464 t
ORIGIN
1  aagcttgact  tcaacagggg  ggcactccac  tccacaggca  atccttgacg  gcagcgacaa
61  aaccacctc  tctgagacgg  ctccaactga  gaagaacttc  aactctgggt  tgggggaaaa
121 gagagcaaga  ctgagtactg  cccactgtac  tcagcaagtc  ataccgaaag  aggaggtatg
181 atgcaggata  taaccaaagg  aagctagggg  ttcttttgca  taaagcaggc  atttcaaagc
241 agtagttgaa  agcagtaaaa  cagctgtagt  aattaatcaa  tattaaccaa  tcaactgtca
301 acgtatcacc  aogttgcaac  aggccaacc  aaccaactga  actacaccag  ttcatgaagc
361 taaactaggg  gtgagactaa  tcacggtgaa  tctggttgat  cgcccataac  cgccggcgag
421 gctattogaa  tagttttact  ctggccagag  gtgtacaact  gtacccacaa  gacacgatcc
481 ccacatgtc  gccatgcccc  gaagtatcac  cctgatactg  caaaggggga  aatcgtgaca
541 agacccctca  cataaccctc  cctaaccat  ccacaccacg  ctaaggtttc  acccccaccc
601 ctcaaaaggc  agtggcggtg  cccctcttgc  gccgcgtgga  atccggcgag  tggacaacgc
661 gacaccccg  cgcacccaac  tccatcagc  ccacccctgc  caecggtgcc  taggaagggg
721 tgcagctata  ctccagacca  agcagttacc  cactcccgct  tgtggttaag  acggtaagtc
781 tcccagggtt  tctcgtgaac  cggctcttaa  ctgctatggg  tgcgatcagc  aaaaccatgc
841 acccacagcc  caccattcag  tgtattttaa  ttaactaaca  ccattcggtg  ggcaccaatc
901 taaagctatg  ccaatagaca  aagtcctatg  aataatgtga  tccccatttg  tgtactagtt
961 gaactaagca  tggctaagca  ttctctaagc  caacatctag  tcattttgat  acccaagtta
1021 tcaatggcat  aaggtaacca  atatgtggct  gaggaatagg  acccatccca  cattacatbg
1081 taaaagaatg  caacatttaa  tagaaatgcg  ggatattggt  aaattgggta  caatatgac
1141 aaacgtattg  catgacttgc  cttgctctcg  aactgatgag  acctcagcaa  cgtcttcgag
1201 aaaccgggga  tcgacgaaac  ggcggaacc  tacgcgacaa  acaagcaca  caagcaaaac
1261 atgctataag  actactgaaa  caggaaacaa  aaccattttt  aatggattct  ttgcattttt
1321 cttgatttac  tgagacttga  atggacttaa  acggagctcg  gatgaattac  ttatgtattt
1381 tagaagataa  actgtgtttt  tactaataaa  gaaaaagtc  ttaattaatt  attcgtgat
1441 aatacccagg  gctgacgtca  tctaaggggg  gcggcgccga  caggcggggc  ccacgggtca
1501 gcaqctcaag  gtqcccqtc  taccqtgac  cgggaccac  cgggtggtc  accqccqtc

```

Fig. 8.4 AC080019 nucleotide sequence

GenBank uses a number of different identifiers to refer to the entry above. The identifiers are listed below:

GI	13112226
Accession number	AC080019.7 (Version number 7)
Clone name	OSJNBa0094H10

A little later in the chapter, we will use each of these to access this entry using the Bio::DB::GenBank module.

For short sequences, you can copy and paste the sequence in a text editor in the usual manner. For a large sequence such as this, use the 'Send to Text' button to download the sequence on your computer (Figure 8.7).

When you view this document in a text editor, the first few lines look like Figure 8.8 (Notepad on Windows). Here, even though the header overflows on two lines, it is really just one line.

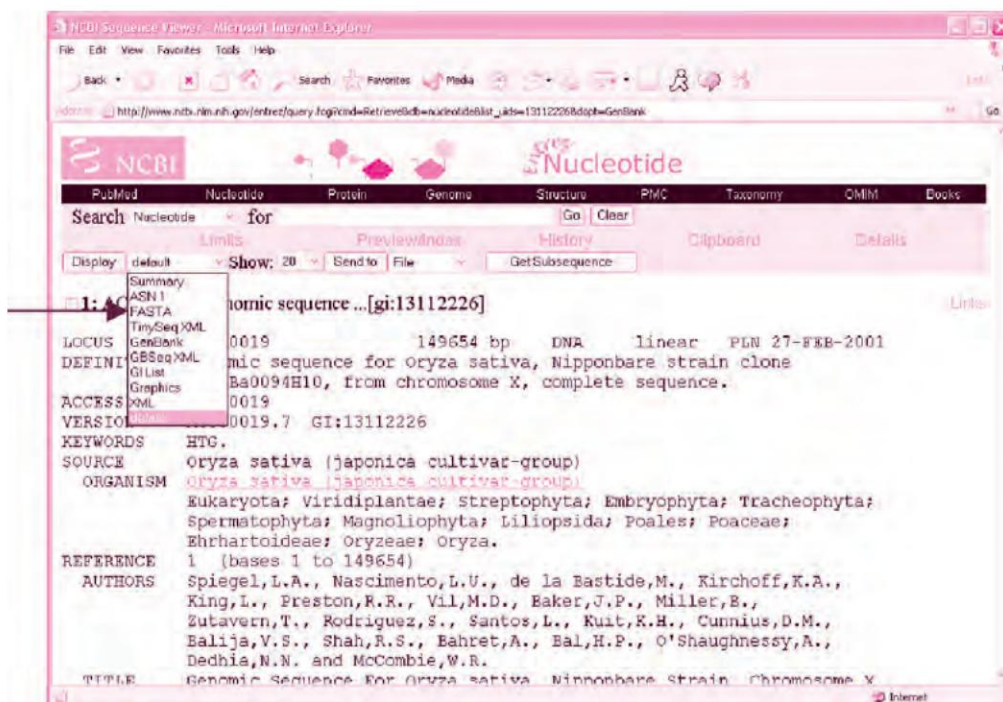


Fig. 8.5 Selecting Fasta format

8.3 THE Bio::DB::GENBANK MODULE

In its simplest form, Bio::DB::GenBank provides methods to retrieve from GenBank sequences identified by GI accession numbers, etc. The general usage is as follows:

```
use Bio::DB::GenBank;
```

```
$gb = new Bio::DB::GenBank;
```

```
$seqobj = $gb->get_Seq_by_id('identifier');
```

where, identifier = a unique ID

or,

```
$seqobj = $gb->get_Seq_by_acc('accession');
```

where, accession = accession number

NCBI Sequence Viewer - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print

Address http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?cmd=&db=nucleotide&extrafeat=-1&view=fasta&dispmax=20

NCBI Nucleotide

PubMed Nucleotide Protein Genome Structure PMC Taxonomy

Search Nucleotide for Go Clear

Limits Preview/Index History Clipboard

Display FASTA Show: 20 Send to: File Get Subsequence

1: AC080019. Genomic sequence ...[gi:13112226]

>gi|13112226|gb|AC080019.7|AC080019 Genomic sequence for Oryza sativa,
AAGCTTGACTTCAACAGGGGGCGACTCCACTCCACAGGCAATCCTTGACGGCAGCGACAAAACCAACCTC
TCTGAGACGGCTCCAACGTGAGAAGAACTTCAACTCTGGTGTGGGGGAAAAGAGAGCAAGACTGAGTACTG
CCCACTGTACTCAGCAAGTCATACCGAAAGAGGAGGTATGATGCAGGATATAACCAAAGGAAGCTAGGGG
TTCTTTTGATAAAGCAGGCATTTCAAAGCAGTAGTTGAAAGCAGTAAACAGCTGTAGTAATTAATCAA
TATTAACCAATCACTGTCCAACGCTACACCACGTTGCAACAGGCCCAACCAACCACCTGAACCTACACCAG
TTCATTAGCTAAACTAGGGGTGAGACTAATCACGGTGAATCTGGTTGATCGCCCATAAACCGCGGGCAGG
GCTATTCGAATAGTTTTACTCTGGCCAGAGGTGTACAACCTGTACCCACAAGACACGATTCCACACATGTC
GCCATGCCCCGAAGTATCACCATGATACTGCAAAGGGGGAAATCGTGACAAGACCTCCACATAACCCCTC
CCCTAACCATCCACACCACGCTAAGGTTTCAACCCCAACCCCTCAAAGGCAGTGGGCGGTCCCTCTTGC
GCCGCGGTGAATCCGGCAGCTGGACAACCGGACACCCCGCGGACCCCAACTCCATCACGCCCAACCCCTCGC
CACCGGTGCCCTAGGAAAGGGTCGAGCTATACTTCAGACCAAGCAGTTACCCACTCCCGCTTGTGGTAAGC
ACGGTAAGTCTCCAGGTTTCTCGTGAACCGGTCCTTAACCTGCTATGGGTGCGATCAGCAAAACCATGC
ACCCACAGCCCAACCATTCAGTGTATTTTAATTAACCTAACACCATTCGCGGTGGCACCACATCTAAAGCTATG
CCAATAGACAAAGTCTATGTAATATGTGATCCCATTTGTGTACTAGTTGAACCTAAGCATGGCTAAGCA
TTTCCTAAGCCAACATCTAGTCATTTTGATACCCAAGTTATCAATGGCATAAGGTAACCAATATGTGGCT
GAGGAATAGGACCCATCCACATTACATTGTAAGAAGTGAACATTTAATAGAAATGCGGGATATTGGT

Fig. 8.6 AC080019 sequence in Fasta format

Once you get the \$seqobj object, you can use the same methods we used earlier in Chapter 1 to retrieve information about the GenBank entry:

```
$type      = $seqobj->motype();    #DNA, RNA or protein?
$id        = $seqobj->id();        #id
$description = $seqobj->desc();    #description
$dnaseq    = $seqobj->seq();       #DNA or protein sequence
$length    = $seqobj->length();    #length of sequence
```

The `get_Seq_by_id` method retrieves a `Bio::Seq` object by a unique ID such as its GI number and returns a `Bio::Seq` object. If the ID is not found, the



Fig. 8.7 Saving the AC080019 Fasta sequence to file

method displays an “ID does not exist” exception. This method works for IDs that uniquely identify the sequence entry. For example, all the three IDs below will give the same information because they uniquely identify the sequence record:

By Accession number:

```
$seqobj = $gb->get_Seq_by_id('AC080019');
```

By Accession + version number:

```
$seqobj = $gb->get_Seq_by_id('AC080019.7');
```

By gi number:

```
$seqobj = $gb->get_Seq_by_id('13112226');
```

```

>gi|13112226|gb|AC080019.7|AC080019 Genomic sequence for oryza sativa,
Nipponbare strain clone OSJNb0094H10, from chromosome X, complete sequence
AAGCTTGACTTCAACAGGGGGCGACTCCACTCCACAGGCAATCCTTGACGGCAGCGACAAAACCAACCTC
TCTGAGACGGCTCCAACAGAGAACTTCAACTCTGGTGTGGGGGAAAAGAGAGCAAGACTGAGTACTG
CCCCTGTACTCAGCAAGTCATACCGAAAGAGGAGGTATGATGCAGGATATAACCAAAGGAAGCTAGGGG
TTCTTTTGCATAAAGCAGGCATTTCAAAGCAGTAGTTGAAAGCAGTAAAACAGCTGTAGTAATTAAATCAA
TATTAACCAATCACTGTCCAACGCTACACCACGTTGCAACAGGGCCCAACCAACCACTGAACCTACACCAAG
TTCATTAAGCTAACTAGGGGTGAGACTAATCACGGTGAATCTGGTTGATCGCCCATAACCGCGGGCAGC
GCTATTCGAATAGTTTTACTCTGGCCAGAGGTGTACAACCTGTACCCACAAGACAGATTCCACACATGTC
GCCATGCCCGGAAGTATCACCATGATACTGCAAGGGGGGAAATCGTGACAAGACCCTCCACATAACCCCTC
CCCTAACCATCCACACCAGCTAAGGTTTCAACCCACCCCTCAAAAGGCAGTGGGCGGTCCCTCTTGC
GCGCGGTGAATCCGGCAGCTGGACAACCGGACACCCCGCGCCGACCCAACTCCATCACGCCCACCCCTCGC
CACCGGTGCCTAGGAAAGGGTCTGAGCTATACCTCAGACCAAGCAGTTACCCACTCCCGCTTGTGGTAAGC
ACGGTAAGTCTCCAGGGTTTCTCGTGAACCGGTCCTTAACTGCTATGGGTGCGATCAGCAAAACCATGC
ACCCACAGCCCAACATTCACTGTATTTAATTAACTAACACCATTTGCGGTGGCACCAATCTAAAGCTATG
CCAATAGACAAAGTCTATGTAATAATGTGATCCCCATTTGTGTACTAGTTGAACCTAAGCATGGCTAAGCA
TTTCCTAAGCCAACATCTAGTCATTTTATACCCAAGTTATCAATGGCATAAGGTAACCAATATGTGGCT
GAGGAATAGGACCCATCCACATTACATTGTAAAAGAATGCAACATTTAATAGAAATGCGGGATATTGGT
AAATTGGGTACAATATGATCAAACGTATTGATGACTTGCCTTCTCGAACTGATGAGACCTCAGCAA
CGTCTTCGAGAAACCGCGATCGACGAAACGGCCGAAACCTACGCGACAAACAAAGCACACAAGCAAAAC
ATGCTATAAGACTACTGAAACAGGAACAAACCAATTTTAAATGGATTCTTTGATTTTCTTGATTTAC
TGAGACTTGAATGGACTTAAACGGAGCTCGGATGAATTAATTTATGATTTTGAAGATAACTGTGTTTT
TACTAATAAAGAAAAAGTCCTTAATTAATTATTGCGTGATAATACCCAGGGCTGACGTCATCTAAGGGGG
GCGGGCGCCGACAGGGGGGGCCACGGGTCAGCAGCTCAAGGTGGCCGGTCTACCGTGGACCGGACACG
CGGGTGGTCCACCGCGGTCCACGGGACCGACGGTCCGGATCGGCCGGGAGGCCGATCGGACGGCACGGC
GGCGGCTAGGCACGGCTCGACTCGGCTTCAAAACCGGCGGTGCGCCATGGCAACTGGCGGCGACGCGCA
CGGTCAACCGCGACGGCAATCGGCGGCGCGGGAAGCGATGGCGGACGGGTAAAGGACGGCGGCGTGCAC
CGAAGGCGCGGCGGCGGTGGAGAGAGCGCGCAACGAGAGAGAGGGGAAGAGGGAGGAGGGGGGG
ATCCTCACCGGCGAGTATGGCGGCGGAGCGGAGGATGAAGGTGGCGGCGACGACCTGGCGATGAGGAGG
GGCGGACGGGCGGCGGACGACCTACGGAGGGAGTTGAGAAGGATTAGGAGTAGAGGGAGTGACCACGGC

```

Fig. 8.8 AC080019 Fasta file in Notepad

By clone name:

```
$seqobj = $gb->get_Seq_by_id('OSJNb0094H10');
```

To test this, let's write a small script that takes the ID from the command-line and see what output we get for the different identifiers.

```
use Bio::DB::GenBank();
```

```
$gb = new Bio::DB::GenBank();
```

```
$seqobj = $gb->get_Seq_by_id("$ARGV[0]");
```

```
$id = $seqobj->id();
```

```
$description = $seqobj->desc();
```

```
$moltype = $seqobj->moltype();
```

```
$length = $seqobj->length();
```



```

$dnaseq      = $seqobj->seq();
print "ID: $id\n";
print "Description: $description\n";
print "Molecule type: $moltype\n";
print "Sequence: $dnaseq\n";
print "Length: $length\n";

```

Note that `$ARGV[0]` is a sequence identifier, not a multiple Fasta file saved locally. As explained earlier, `$ARGV[0]` could be the accession number, GI number, etc. Try the above code with the different identifiers. The output should be the same in each case.

Similarly, the `get_Seq_by_acc` method retrieves a `Bio::Seq` object by its accession number and returns a `Bio::Seq` object. If the accession number is not found, the method displays an "ID does not exist" exception.

 If you want to obtain just a part of the sequence, you can use the `subseq` method and supply the start and end nucleotide positions of the sequences that you want to extract. 



```

use Bio::DB::GenBank();

$gb = new Bio::DB::GenBank();
$seqobj = $gb->get_Seq_by_id("$ARGV[0]");

#get description
$description = $seqobj->desc();
print "Description: $description\n";

#get a subsequence
$subseq = $seqobj->subseq(1, 100);
print "Sequence from 1 to 100:\n$subseq\n";

```



This will give only the sequence from nucleotide positions 1 to 100. The command for the script (called `getSequence.pl`) run with the accession number and its output is shown in Figure 8.9.



```
C:\perl>getSequence.pl AC080019
Description: Genomic sequence for Oryza sativa, Nipponbare strain
94H10, from chromosome X, complete sequence.

Sequence from 1 to 100:
AAGCTTGACTTCAACAGGGGGCGACTCCACTCCACAGGCCAATCCTTGACGGCAGCGACAAAACCA
ATCCTCAACTGAGAAGAACTTC

C:\perl>
```

Fig. 8.9 Extracting subsequences



Assignments



1. Download and uncompress the nr.Z database from NCBI (<ftp://ftp.ncbi.nih.gov/blast/db/>). Write a script to download all rat sequences (*Rattus norvegicus*) from nr in Fasta format. How many rat sequences are present in the current release of nr?
2. For each of the rat sequences, write a script to parse the GenBank record and create an HTML table containing the following information:

GI number

Accession number

Protein product name

Complete protein sequence

Domain information (see below)

For example, the GenBank record for rat protein “delayed rectifier potassium channel Kv4” (gi 111574) contains the following information as identified by the Features list:

Parse each of the “Region” entries and add it to the Domain information field in the following manner:

NCBI Sequence Viewer - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media

Address: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=protein&list_uids=111574&dopt=GenPept

FEATURES

	Location/Qualifiers
source	1..585 /organism="Rattus norvegicus" /db_xref="taxon:10116"
<u>Protein</u>	1..585 /product="delayed rectifier potassium channel Kv4, neuronal" /note="potassium channel protein Raw2"
→ <u>Region</u>	191..209 /region_name="domain" /note="transmembrane"
→ <u>Region</u>	245..266 /region_name="domain" /note="transmembrane"
→ <u>Region</u>	278..298 /region_name="domain" /note="transmembrane"
→ <u>Region</u>	310..328 /region_name="domain" /note="transmembrane"
→ <u>Region</u>	345..364 /region_name="domain" /note="transmembrane"
→ <u>Region</u>	415..436 /region_name="domain" /note="transmembrane"

Region name: Transmembrane

Coordinates: 191–209

Subsequence: YVAFASLFFILVSITTFCL



Accessing GenBank Data



MK



9.1 INTRODUCTION

In the last chapter we saw how a GenBank record is structured and how we can use the `Bio::DB::GenBank` module to extract basic information on sequence data using unique identifiers such as the accession number, GI number or clone name. In this chapter, we will learn how to use the methods provided by the `Bio::Seq`, `Bio::SeqIO` and `Bio::SeqFeatureI` packages to access further information from a GenBank record.



9.2 GENBANK TAGS

Before we do so, we need to understand how BioPerl views a GenBank record. If you remember, the GenBank record consists of a header portion at the top which provides such information on the sequence such as the locus, definition, accession number, the source organism, the authors, etc. (Figure 9.1).

The record then provides information on the sequence itself in extensive detail (especially if it is a fully annotated record). These are listed under the

MK



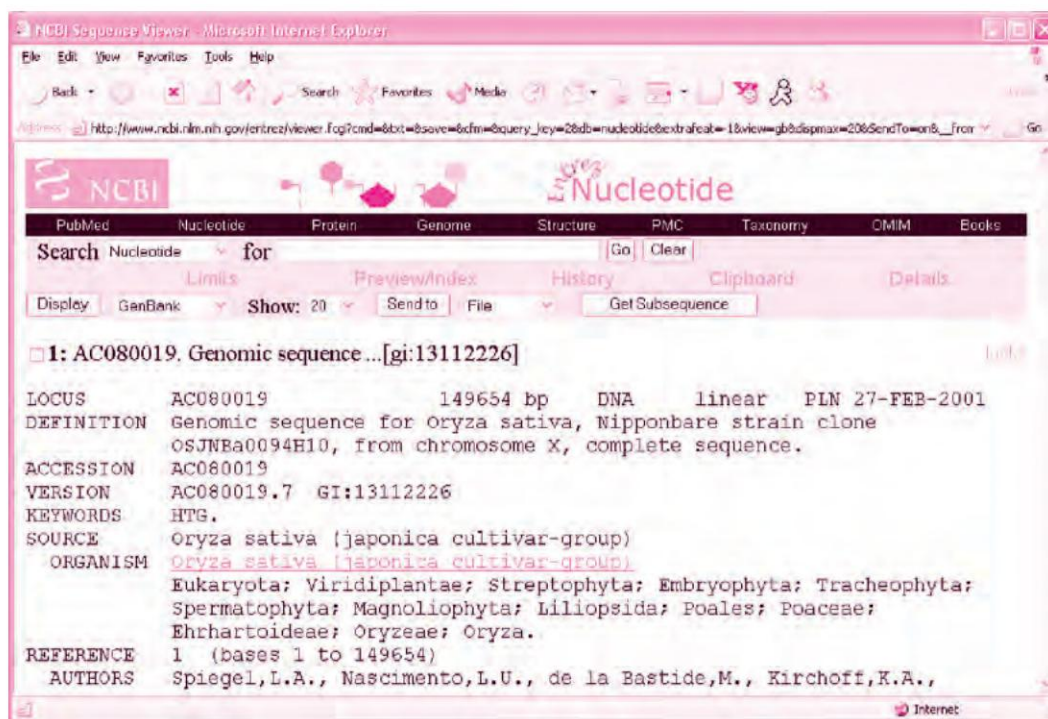


Fig. 9.1 GenBank header for rice BAC AC0080019

title “Features” and include such information as the source of the sequence (viz., the name of the organism it is derived from, its chromosomal location, etc.) and the genes, repeat regions and proteins found on the sequence (Figure 9.2), with specific details on each of the features.

BioPerl calls these features “tags” (Figure 9.3). All the main headings under Features such as source, repeat_region, gene, CDS, etc. are called Primary tags. Each Primary tag has a value and, in addition, carries information below it in the form of sub-tags. For example, the first Primary tag “gene” in Figure 9.3 has a value of 55..1885, which is just the start and stop coordinates of the gene and has sub-tags called “gene” and “note” appended with a/sign. Each of these sub-tags, in turn, have values, which, in this case, describe the identifier and the definition of the gene: “OSJNBa0094H10.1” and “Hypothetical protein” respectively. Some examples of Primary and sub-tags and their corresponding values as represented in the AC0080019 GenBank record for Rice BAC OSJNBa0094H10 are presented in Table 9.1. In each case, the primary features are highlighted.

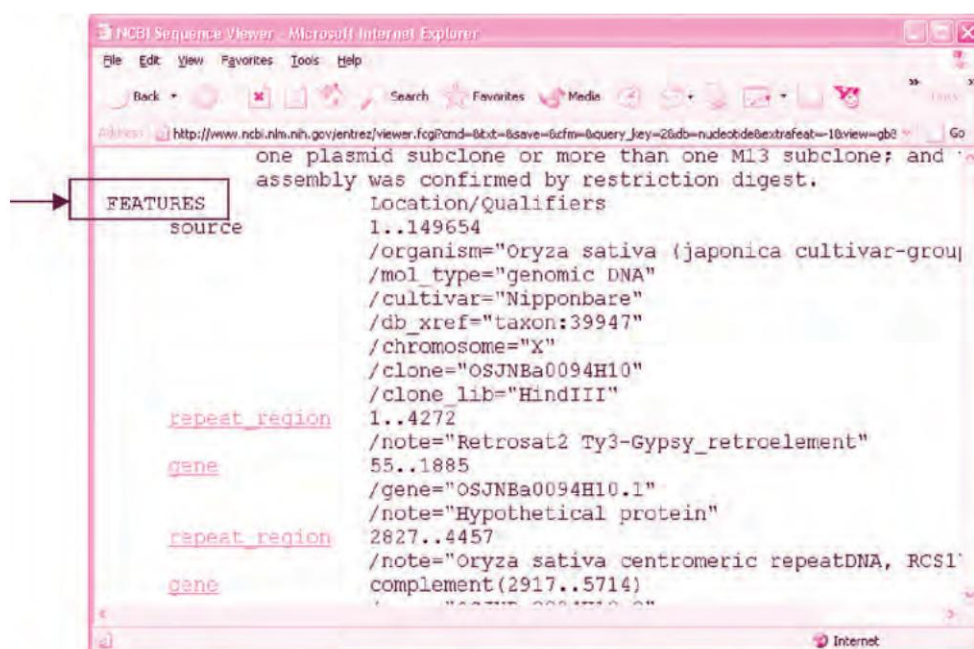


Fig. 9.2 Sequence features on a GenBank record

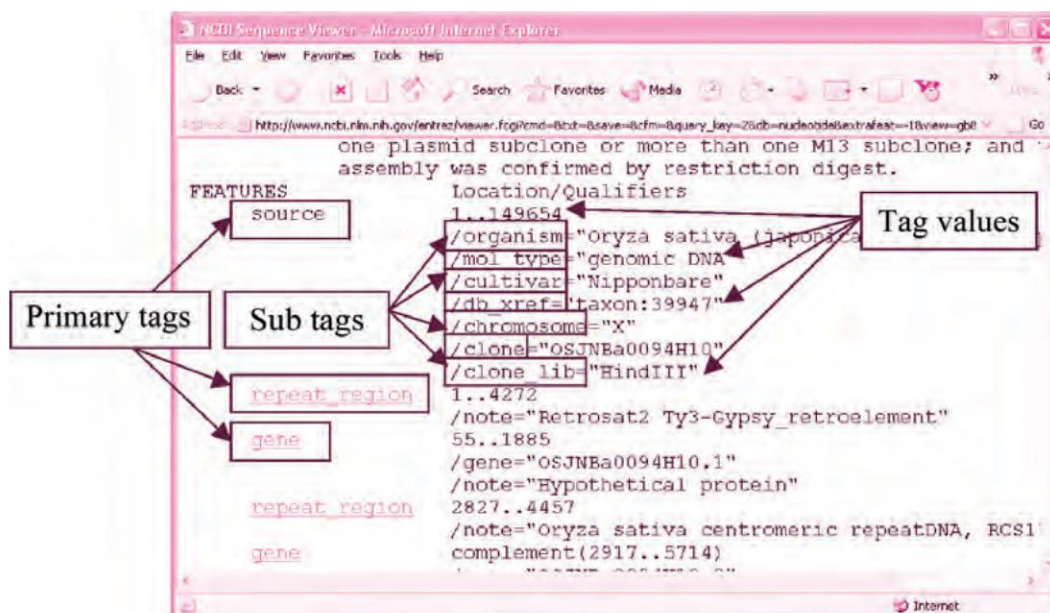


Fig. 9.3 Tag-value pairs in a GenBank record

Table 9.1 Tag-value pairs in the AC080019 GenBank record

	Tag type	Tag name	Tag value
1	Primary	Source	1..149654
	→Sub-tag	Organism	Oryza sativa (japonica cultivar-group)
	→Sub-tag	mol_type	genomic DNA
	→Sub-tag	cultivar	Nipponbare
	→Sub-tag	db_xref	taxon:39947
	→Sub-tag	chromosome	X
	→Sub-tag	clone	OSJNBa0094H10
	→Sub-tag	clone_lib	HindIII
2	Primary	repeat_region	1..4272
	→Sub-tag	note	Retrosat2 Ty3-Gypsy_retroelement
3	Primary	gene	55..1885
	→Sub-tag	gene	OSJNBa0094H10.1
	→Sub-tag	note	Hypothetical protein
4	Primary	CDS	complement(join(35752..36640,36704..36883, 37831..37865))
	→Sub-tag	gene	OSJNBa0094H10.6
	→Sub-tag	codon_start	1
	→Sub-tag	product	Hypothetical protein
	→Sub-tag	protein_id	AAK13109.1
	→Sub-tag	db_xref	GI:13129451
	→Sub-tag	translation	MKVEKGRDAPKVPLPSLPVVP ... LSFPRLVAAWW



9.3 EXTRACTING TAGS AND THEIR VALUES

BioPerl provides mechanisms to access both the Primary tag and each of the sub-tags below it. These methods are derived from the `Bio::SeqFeatureI` module and are described in Table 9.2.

Table 9.2 *Bio::SeqFeatureI methods*

Method name	Function	Usage
start	Start coordinate of feature	<code>\$feature→start</code>
end	End coordinate of feature	<code>\$feature→end</code>
strand	Orientation of feature (1 for the forward strand, -1 for the reverse strand, 0 if not relevant)	<code>\$feature→strand</code>
length	Length of feature	<code>\$feature→length</code>
all_tags	Extracts all tags for a feature	<code>\$feature→all_tags</code>
each_tag_value	Extracts all values for a tag	<code>\$feature→each_tag_value</code>
has_tag	Checks if a tag is present	<code>\$feature→has_tag</code>
source_tag	Extracts the source tag for a feature (i.e., where the feature comes from, e.g., BLAST, GenScan, etc.)	<code>\$feature→source_tag</code>

The use of these methods in actual code is illustrated below.

As always we begin with a `use` statement that includes the `Bio::SeqIO` module in our program.

```
use Bio::SeqIO;
```

We then create a new instance of the `Bio::SeqIO` object by invoking the `new()` class method, and assign it to `$infile`.

```
my $instream= Bio::SeqIO->new(-file => $ARGV[0], -format => "Genbank");
```

Note that this statement can also be written as:

```
my $instream= new Bio::SeqIO(-file => $ARGV[0], -format => "Genbank");
```

As we saw in Chapter 7, `new()` accepts two parameters:

- `file` (path to file to be opened for reading or writing)
- `format` (the file format: EMBL, Fasta, SwissProt, GenBank, etc.)

In this case, we have simply chosen to provide the file name on the command-line (`$ARGV[0]`); the format of the input file is GenBank.

We can now create a `$seq` object and extract the DNA sequence from the BAC along with other header information such as the accession number, the primary ID, the display ID and description.

```

my $seq      = $instream->next_seq()
$accession   = $seq->accession_number();
$seq         = $seq->seq();
$desc        = $seq->desc(); # Description
$dnaseq      = $seq->seq(); # DNA sequence
$did         = $seq->display_id(); # Display ID
$pid         = $seq->primary_id(); # Primary ID

```

The `subseq()` method can be used to obtain a subsequence:

```
$substr = $seq->subseq(start,end); #Obtain a subsequence
```

where `start` and `stop` represent the required start and end coordinates.

In addition, once we have the `$seq` object, we can extract information on the species to which the sequence belongs using the methods provided by the `Bio::Species` module. These methods are described in Table 3.3.

Table 9.3 *Bio::Species methods*

Method name	Function	Usage
<code>common_name</code>	The common name of the organism	<code>\$species->common_name</code>
<code>genus</code>	Extracts the genus of the organism	<code>\$feature->genus</code>
<code>binomial</code>	The full scientific name of the organism	<code>\$feature->binomial</code>
<code>classification</code>	Returns the classification list in the object as an array in the order species, genus, ..., kingdom.	<code>\$feature->classification</code>

The `classification` method yields an array of terms used to describe the taxonomy of the organism. In the case of rice, the classification is:

```

'sativa',
'Oryza',
'Oryzeae',
'Ehrhartoideae',

```


'Poaceae',
'Poales',
'Liliopsida',
'Magnoliophyta',
'Spermatophyta',
'Tracheophyta',
'Embryophyta',
'Streptophyta',
'Viridiplantae',
'Eukaryota'

which simply means that rice (*Oryza sativa*) belongs to the Kingdom Viridiplantae (which represents green plants and green algae), the Phylum Magnoliophyta (representing flowering plants), the Class Monocotyledoneae (also called Liliopsida for the grasses), the Family Poaceae, the Tribe Oryzeae, the Genus *Oryza* and the Species *sativa*.

If the BAC represented a human sequence, the corresponding classification would be:

'sapiens',
'Homo',
'Hominidae',
'Catarrhini',
'Primates',
'Eutheria',
'Mammalia',
'Euteleostomi',
'Vertebrata',
'Craniata',
'Chordata',


```

'Metazoa',
'Eukaryota'

```

which has a similar connotation.

The methods are used as follows:

```

my $organism = $species->common_name; # Organism
my $genus    = $species->genus; # Genus
my $name     = $species->binomial(); # Full scientific name
my @class    = $species->classification(); # Full taxonomy

```

Importantly, we can now extract the tags from the BAC along with their values. To get all the features, viz., gene, mRNA, repeat_region, CDS, etc., use the `all_seqFeatures()` method from the `Bio::Seq` package (which provides an abstract interface of annotated sequence):

```
@features = $seq->all_SeqFeatures();
```

which returns an array of all the features associated with the sequence. Now, we can iterate over the array and retrieve all the associated values:

```

foreach my $feat(@features) {
    print "Feature   = ", $feat->primary_tag,
          "\nStrand  = ", $feat->strand,
          "\nFrom    = ", $feat->start,
          " to      = ", $feat->end,
          "\nSource  = ", $feat->source_tag(), "\n";
}

```

This will give only the information associated with the Primary tag. To get all the sub-tags and their values, we need to iterate over each of the tags:

```

@tags = $feat->all_tags();
foreach $tag(@tags) {
    @values = $feat->each_tag_value($tag);
    print $tag, " = ", @values, "\n";
}

```

To get all tags of the type “translation”, we can use the `has_tag()` method to check for the presence of that string.

```
#get all translations
if ($feat->has_tag('translation')) {
    @proteinids = $feat->each_tag_value('translation');
    print "@proteinids\n";
}
```

Similarly, to get all the gene names in the BAC (eg., OSJNBb0076H04.1, OSJNBb0076H04.2, etc), check for the tag “gene”.

```
#get all genes
if ($feat->has_tag('gene')) {
    @geneids = $feat->each_tag_value('gene');
    print "@geneids\n";
}
```



9.4 SAMPLE SCRIPTS

A few sample scripts and their outputs are shown here to illustrate the use of the methods described above.

Listing 9.1

Sample script 1

```
use Bio::SeqIO;
my $instream = new Bio::SeqIO(-file => $ARGV[0], -format => "Genbank");
my $seq      = $instream->next_seq();
my $accession = $seq->accession_number();
my $dnaseq   = $seq->seq();
my $did      = $seq->display_id();
my $pid      = $seq->primary_id();
```

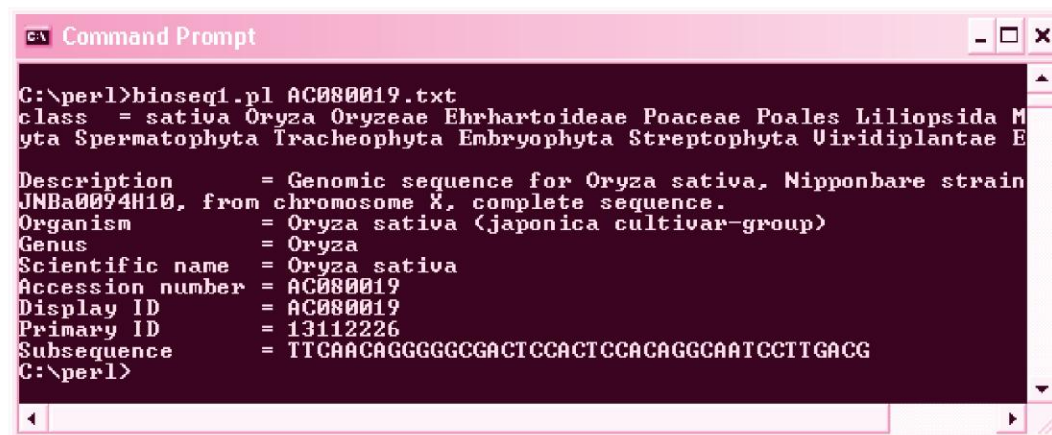
(Contd.)

(Contd.)

```

my $desc      = $seq->desc();
my $substr    = $seq->subseq(10,50);
my $species   = $seq->species();
my $organism  = $species->common_name;
my $genus     = $species->genus;
my $name      = $species->binomial();
my @class     = $species->classification();
print "class  = @class\n";
print "
Description  = $desc
Organism     = $organism
Genus        = $genus
Scientific name = $name
Accession number = $accession
Display ID   = $did
Primary ID   = $pid
Subsequence  = $substr";

```



```

C:\perl>bioseq1.pl AC080019.txt
class = sativa Oryza Oryzeae Ehrhartoideae Poaceae Poales Liliopsida M
yta Spermatophyta Tracheophyta Embryophyta Streptophyta Viridiplantae E
Description      = Genomic sequence for Oryza sativa, Nipponbare strain
JNBa0094H10, from chromosome X, complete sequence.
Organism         = Oryza sativa (japonica cultivar-group)
Genus            = Oryza
Scientific name  = Oryza sativa
Accession number = AC080019
Display ID       = AC080019
Primary ID       = 13112226
Subsequence      = TTCAACAGGGGGCGACTCCACTCCACAGGCAATCCTTGACG
C:\perl>

```

Fig. 9.4 Output of Listing 3.1

Listing 9.2 **Sample script 2 (Extract Primary tags only)**

```

use Bio::SeqIO;

my $instream = new Bio::SeqIO(-file => $ARGV[0], -format => "Genbank");
my $seq = $instream->next_seq();
@feats = $seq->all_SeqFeatures();
foreach my $feat (@feats) {
    $count++;
    print "$count] Feature = ", $feat->primary_tag,
        "\nStrand   = ", $feat->strand,
        "\nFrom     = ", $feat->start,
        " to       = ", $feat->end,
        "\nLength   = ", $feat->length,
        "\nSource  = ", $feat->source_tag(),
        "\n";
}

```

**Listing 9.3** **Sample script 3 (Extract all tags)**

```

use Bio::SeqIO;

my $instream = new Bio::SeqIO(-file => $ARGV[0], -format => "Genbank");
my $seq = $instream->next_seq();
@feats = $seq->all_SeqFeatures();
foreach my $feat (@feats) {
    $count++;
    print "$count]
        Feature   = ", $feat->primary_tag,
        "\nStrand  = ", $feat->strand,
        "\nFrom    = ", $feat->start,
        " to      = ", $feat->end,
        "\nLength  = ", $feat->length,

```

(Contd.)

```

C:\perl>bioseq2.pl AC080019.txt > bioseq2out.txt
C:\perl>more bioseq2out.txt
1) Feature    = source
Strand = 1
From   = 1 to   = 149654
Length = 149654
Source = EMBL/GenBank/SwissProt
2) Feature    = repeat_region
Strand = 1
From   = 1 to   = 4272
Length = 4272
Source = EMBL/GenBank/SwissProt
3) Feature    = gene
Strand = 1
From   = 55 to   = 1885
Length = 1831
Source = EMBL/GenBank/SwissProt
4) Feature    = repeat_region
Strand = 1
From   = 2827 to   = 4457
Length = 1631
Source = EMBL/GenBank/SwissProt
5) Feature    = gene
Strand = -1
From   = 2917 to   = 5714
Length = 2798
Source = EMBL/GenBank/SwissProt

```

Fig. 9.5 Output of sample script 2

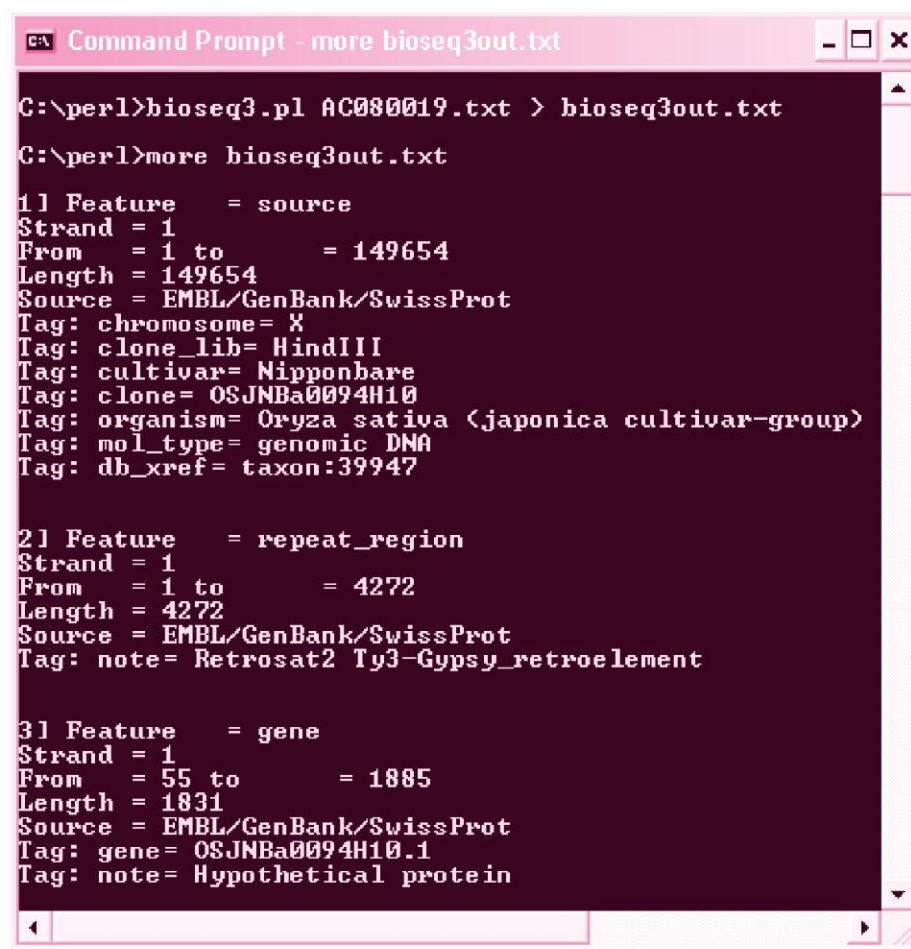
(Contd.)

```

        "\nSource = ", $feat->source_tag(),
        "\n";

#get all tags for all primary features viz., gene, mRNA,
@tags = $feat->all_tags();
foreach $tag (@tags) {
    @values = $feat->each_tag_value($tag);
    print "Tag: ", $tag, "= ", @values, "\n";
}
print "\n";
}

```



```
Command Prompt - more bioseq3out.txt

C:\perl>bioseq3.pl AC080019.txt > bioseq3out.txt
C:\perl>more bioseq3out.txt

11 Feature      = source
Strand = 1
From   = 1 to      = 149654
Length = 149654
Source = EMBL/GenBank/SwissProt
Tag: chromosome= X
Tag: clone_lib= HindIII
Tag: cultivar= Nipponbare
Tag: clone= OSJNBa0094H10
Tag: organism= Oryza sativa (japonica cultivar-group)
Tag: mol_type= genomic DNA
Tag: db_xref= taxon:39947

21 Feature      = repeat_region
Strand = 1
From   = 1 to      = 4272
Length = 4272
Source = EMBL/GenBank/SwissProt
Tag: note= Retrosat2 Ty3-Gypsy_retroelement

31 Feature      = gene
Strand = 1
From   = 55 to      = 1885
Length = 1831
Source = EMBL/GenBank/SwissProt
Tag: gene= OSJNBa0094H10.1
Tag: note= Hypothetical protein
```

Fig. 9.6 Output of sample script 3

Assignments

Download a fully annotated BAC sequence for a sequence from *Rattus norvegicus*. Write a script to parse the GenBank record and create an HTML table containing the following information (wherever available):

- GI number
- Accession number
- Protein product name

Complete protein sequence

Domain information (see below)

For example, the GenBank record for rat protein “delayed rectifier potassium channel Kv4” (GI 111574) contains the following information as identified by the Features list:

FEATURES	Location/Qualifiers
source	1..585 /organism="Rattus norvegicus" /db_xref="taxon:10116"
Protein	1..585 /product="delayed rectifier potassium channel Kv4, neuronal" /note="potassium channel protein Raw2"
Region	191..209 /region_name="domain" /note="transmembrane"
Region	245..266 /region_name="domain" /note="transmembrane"
Region	278..298 /region_name="domain" /note="transmembrane"
Region	310..328 /region_name="domain" /note="transmembrane"
Region	345..364 /region_name="domain" /note="transmembrane"
Region	415..436 /region_name="domain" /note="transmembrane"

Parse each of the “Region” entries and add it to the Domain information field in the following manner:

Region name: Transmembrane

Coordinates: 191–209

Subsequence: YVAFASLFFILVSITTFCL



BioPerl BLAST Modules



MK



10.1 INTRODUCTION

In this chapter, we will introduce the Basic Local Alignment Search Tool (BLAST) that is commonly used in sequence analysis and demonstrate how the searches can be automated using both conventional Perl modules and BioPerl modules.

In Chapter 1, we had used BLAST to find high scoring local alignments between an input sequence and sequences in NCBI database through the web-based tool that is available through their website at <http://www.ncbi.nlm.nih.gov/BLAST/>. We had also demonstrated how to perform automated BLAST analyses using standard Perl code. In this chapter, we will learn how to run searches using standard Perl as well as the direct use of specialized BioPerl modules.



10.2 BLAST PROGRAMS

There are a number of variants of the BLAST algorithm and the choice of a

MK



particular algorithm primarily depends on the type of sequence to be analyzed (that is, nucleotide or protein).

The search programs and their applications are described in Table 10.1.

Table 10.1 *BLAST programs*

Program	Query sequence of type	Database of type	Comparison	Application
BLAST2	DNA or protein	DNA or protein	DNA ↔ DNA or Protein ↔ protein Compares a nucleotide or a protein query sequence against another nucleotide or protein sequence.	Find level of sequence similarity or identity between the input nucleotide or protein sequences.
BLASTN	DNA	DNA	DNA ↔ DNA Compares a nucleotide query sequence against a nucleotide sequence database.	Find DNA sequences that match the query.
BLASTP	Protein	Protein	Protein ↔ protein Compares an amino acid query sequence against a protein sequence database.	Find identical (homologous) proteins.
BLASTX	DNA	Protein	Protein ↔ protein Compares a nucleotide query sequence translated in all reading frames against a protein sequence database.	Find what protein the query sequence codes for.
TBLASTN	Protein	DNA	Protein ↔ protein Compares a protein query sequence against a nucleotide sequence database dynamically translated in all reading frames.	Find genes in unknown DNA sequences.

(Contd.)

Table 10.1 (Contd.)

TBLASTX	DNA	DNA	Protein ↔ protein Compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.	Discover gene structure. (Find degree of homology between the coding region of the query sequence and known genes in the database.)
---------	-----	-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------



10.3 BLAST2

We will begin the exercise with the BLAST2 algorithm which is used to perform a search between two input sequences. This is also known as a “pair-wise” search and its purpose is to examine the level of identity between the input sequences. This analysis is performed at the nucleotide or amino acid level and both sequences need to be either DNA or protein sequences.

This program is available on the NCBI site and can be accessed at:

<http://www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html>

To perform a BLAST2 analysis, you need to specify:

1. Two input sequences (nucleotide or protein) or their GI numbers.
2. The BLAST program to use (BLASTN or BLASTP for a nucleotide or protein sequence respectively).
3. The matrix (we will use the default BLOSUM62 for BLASTP).
4. Parameters such as gap and extension gap penalties, etc. (we will use the default settings).

The parameters and their values (for a protein-protein BLAST2) are as follows:

Name of program	value = BLASTP
Name of matrix	value = BLOSUM62
First sequence	name = one
Second sequence	name = two
Action (Command)	name = submit

The screenshot shows the NCBI BLAST 2 SEQUENCES web interface. Annotations include:

- Program and Matrix:** Points to the 'Program' dropdown menu (set to 'blastn') and the 'Matrix' dropdown menu (set to 'BLOSUM62').
- Search parameters:** Points to the 'Open gap', 'extension gap', 'penalties', 'gap x_dropoff', 'expect', 'word size', 'Filter', and 'Align' options.
- Paste sequence 1:** Points to the input field for 'Sequence 1'.
- Paste sequence 2:** Points to the input field for 'Sequence 2'.
- Or enter GI number Or specify file:** Points to the 'Browse' buttons for both sequence inputs.

Fig. 10.1 BLAST2 at NCBI

10.4 PERL MODULES FOR BLAST2

You can use the standard Perl modules LWP::Simple and LWP::UserAgent to perform BLAST2.

The basic code using the modules for a pair-wise BLAST between two sequences with GenBank IDs \$gbid1 and \$gbid2 is shown below:

1. Using LWP::Simple:

```
#!/usr/bin/perl
$!=undef;
use LWP::Simple;
$url =
"http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?program=blastp&matrix=BLOSUM62&one=$gbid1&two=$gbid2&Action=submit";
$page = get($url);
print "$page";
```

Of course, the matrix can be changed to any of the available options (PAM40, PAM120, PAM250, BLOSUM50, BLOSUM62 and BLOSUM90) depending on the specific purpose of the analysis. No alternate scoring matrices are available for a nucleotide-nucleotide BLAST2.

Now, let's see the equivalent code using the `LWP::UserAgent` module. Unlike `LWP::Simple`, the `LWP::UserAgent` module provides an object oriented interface to the World Wide Web (WWW). It provides the user with methods to create an agent (hence the name 'UserAgent') which in turn is used to issue requests (for example, perform a pair-wise BLAST) to specific services (e.g., the NCBI BLAST2 server) on the WWW and obtain a response (the result of the BLAST2 analysis). Both of these latter functions (formulating a request and obtaining a response) are handled by a different class. These are called `HTTP::Request` and `HTTP::Response` respectively.

As with other object oriented programs, the first step is the creation of an object of the type `LWP::UserAgent`. This is done through the 'constructor' which simply creates a new instance of the `LWP::UserAgent` object using the new keyword:

Step 1: Create an object of type `LWP::UserAgent`:

```
$ua = new LWP::UserAgent;
```

Step 2: Create an instance of `HTTP::Request` encoding the BLAST2 request. Again, we use the `new` keyword to create an object of type `HTTP::Request`. One key difference between the `LWP::Simple` and the `LWP::UserAgent` modules that we have used above is in the way we have formulated the request.

With `LWP::Simple`, the request is created directly and the various parameters are visible in the URL:

```
$url =
"http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi?program=blastp&matrix=BLOSUM62&one=$gbid1&two=$gbid2&Action=submit";
```

In contrast, with the `LWP::UserAgent` module, the data is sent as part of the HTTP request. The information doesn't appear in the URL and, therefore, is more 'secure'. In addition, it also allows a greater number of parameters to be set. The code for the instantiation step is as follows:

```
$request = new HTTP::Request
(POST=>'http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi');
```


Next, we formulate the request and specify the various parameters we want to use:

```
$request->
content("program=blastp&matrix=BLOSUM62&one=1350818&two=133198&
Action=submit");
```

Here, we have used the GenBank IDs 1350818 and 133198.

We also need to specify another piece of information known as the MIME type or content type. MIME—Multi-purpose Internet Mail Extensions—specify a standard way of classifying file types on the Internet. The purpose of MIME types is to enable Internet programs such as Web servers and browsers to transfer files of the same content type in a standardized manner, independent of the underlying operating system. The MIME type enables programs to determine how to open files of a given type, how to view them, etc. A MIME type has two parts: a type and a sub-type. They are separated by a slash (/). For plain text, for example, the MIME type is simply *text/plain*.

Since we are using the information to plug information into a WWW form, the MIME type we need is:

```
application/x-www-form-urlencoded
```

This information is specified as follows:

```
$request->content_type('application/x-www-form-urlencoded');
```

This request is then passed through the UserAgent request() method, which dispatches it using the relevant protocol, and returns an HTTP::Response object:

```
$response = $ua->request($request);
```

Finally, if the request is properly processed, we can obtain the response from the server:

```
if ($response->is_success()) {
    print $response->content();
}
else { warn "Unsuccessful attempt at Blast2!\n"; }
```

The complete code is as follows:

2] Using LWP::UserAgent:

```
#!/usr/bin/perl
$/ = undef;
```

```

use LWP::UserAgent;
#1
$ua = new LWP::UserAgent;
#2
$request = new HTTP::Request
(POST=>"http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi");
#3
$request->
content("program=blastp&matrix=BLOSUM62&one=1350818&
two=133198&Action= submit");
#4
$request->content_type('application/x-www-form-urlencoded');
#5
$response = $ua->request($request);
if ($response->is_success()) {
    print $response->content();
}
else { warn "Unsuccessful attempt at Blast2!\n"; }

```

Save the program as blast2.pl. Run the program and capture the information as a HTML file:

```
C:\perl> Blast2.pl > blast2.html
```

Open the output file using a web browser and the result should appear as shown in Figure 10.2.

The HTML file can be parsed using regular expressions to extract relevant information.



10.5 USING BIOPERL FOR BLAST2

We will now see how the same operation can be performed using a BioPerl module. For this, we need to use the Bio::Tools::Run::StandAloneBlast module.

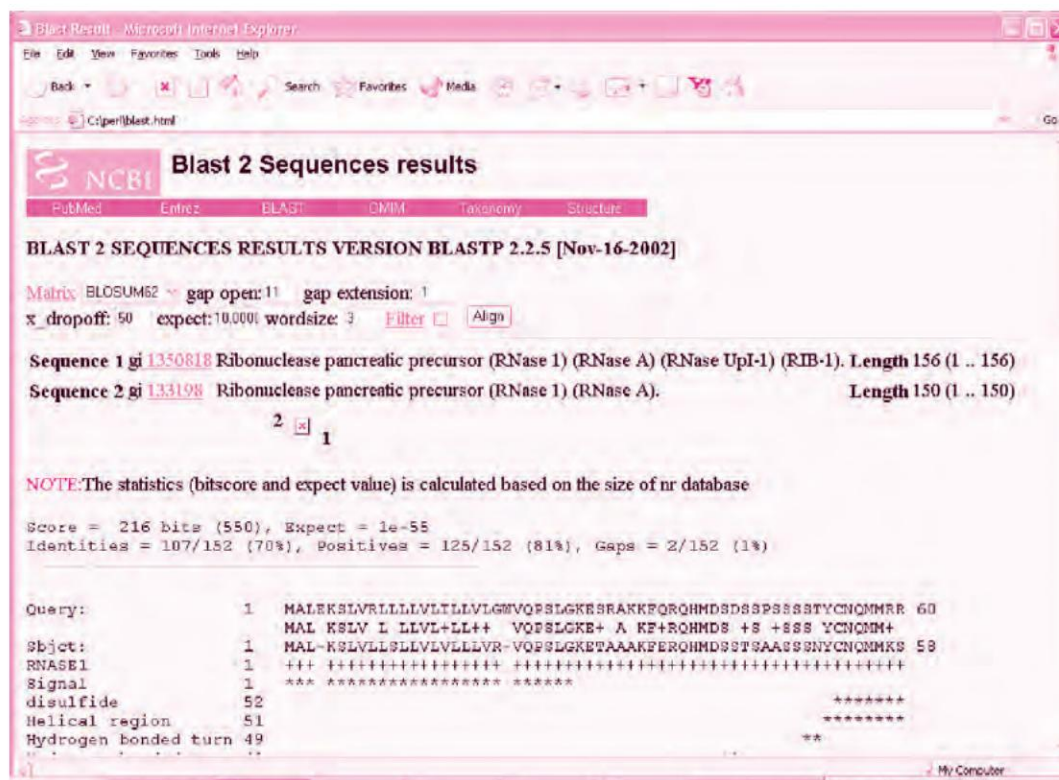


Fig. 10.2 Blast2.pl output

Check if this module is present on your system with the following command:

```
perl -e "use Bio::Tools::Run::StandAloneBlast"
```

If the command exits without issuing any error messages, the module has been installed. If you get an error message saying,

"Can't locate Bio/Tools/Run/StandAloneBlast.pm in @INC (@INC contains: D:/Perl/lib D:/Perl/site/lib .)"

then you need to download it to the appropriate directory on your system as explained in Chapter 1.

You also need to install on your computer the blastall executable from the NCBI ftp site. The next few sections explain how this is done.

10.6 STANDALONE BLAST

The executables for Standalone versions of BLAST are available from the NCBI ftp site (<ftp://ftp.ncbi.nih.gov/blast/executables/snapshot/2004-07-25/>) and can be downloaded by anonymous ftp. Figure 10.3 shows the BLAST versions available for the various platforms. The executable for the Windows version of BLAST, called `blast-20040725-ia32-win32.exe`, is available as a self-extracting archive from the ftp site (indicated in the figure).

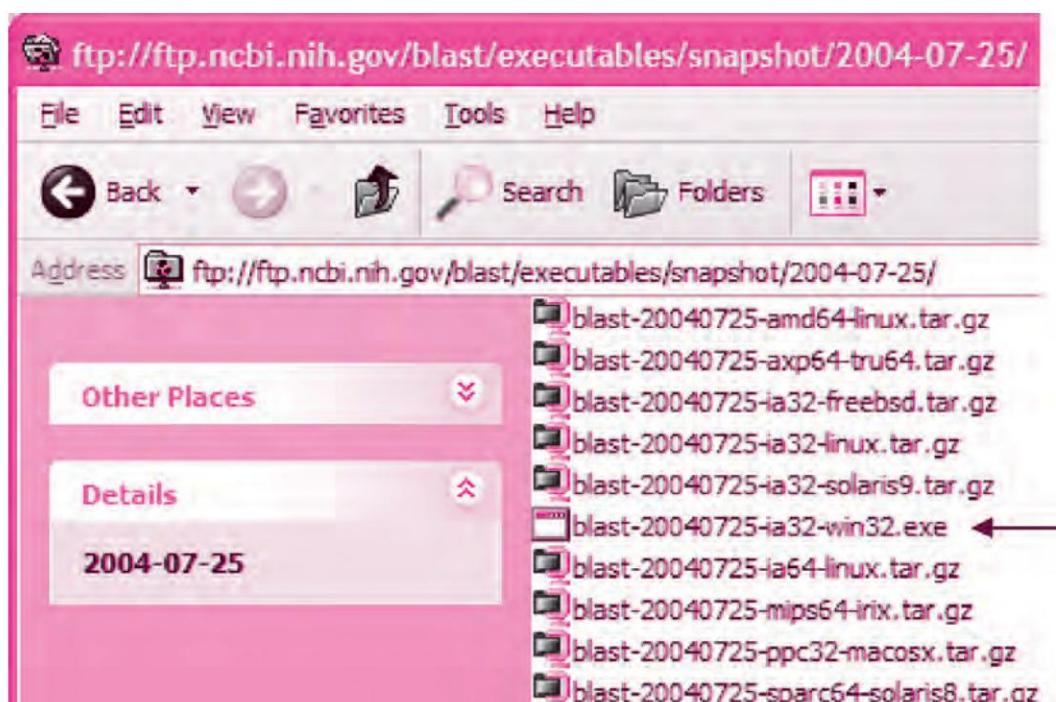


Fig. 10.3 Standalone versions of BLAST at NCBI

To install `blastall` on Windows, download the executable and extract its contents into an appropriate location such as `C:\blast`. Figure 10.4 shows the various programs installed as part of the BLAST suite. Some of these are explained in Table 10.2.

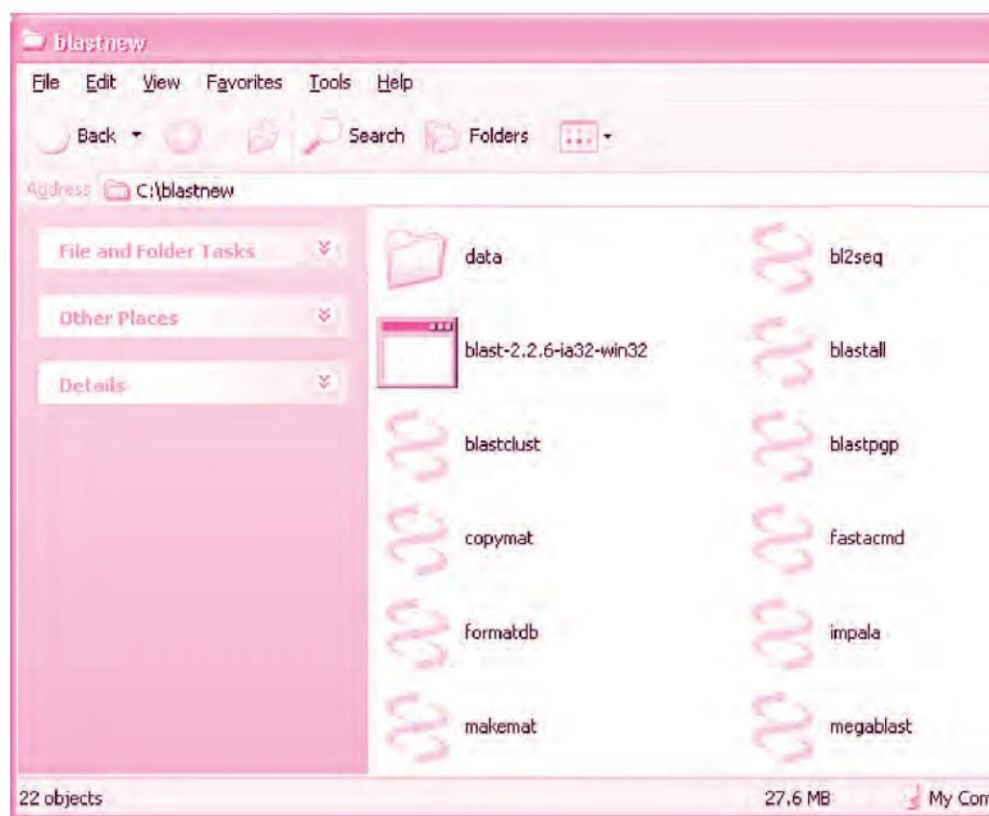


Fig. 10.4 Blast suite of programs

Table 10.2 List of programs installed with BLAST

Program	Function
Blastall	Performs local BLAST searches using any of the five algorithms: BLASTN, BLASTP, BLASTX, TBLASTN or TBLASTX.
Blastpgp	Performs gapped BLASTP searches and can be used to perform iterative searches using PSI-BLAST (Position-Specific Iterative BLAST) and PHI-Blast (Pattern-Hit Iterative BLAST).
Megablast	Alignment program for nucleotide sequence where the sequences differ slightly as a result of sequencing or other similar errors. It is upto 10 times faster than more common sequence similarity programs and, therefore, can be used to swiftly compare two large sets of sequences against each other.

(Contd.)

Table 10.2 (Contd.)

bl2seq	Performs a local alignment between two sequences using either BLASTN or BLASTP. Both sequences must be either nucleotide or protein sequences.
Blastclust	Clustering program for protein or DNA sequences based on pairwise matches found using the BLAST algorithm in case of proteins or Mega BLAST algorithm for DNA.
Rpsblast	Reversed Position-Specific Blast. RPSBLAST performs a BLAST search of a protein sequence vs. a database of conserved protein family domains. Used to derive putative protein family information for an unknown protein sequence.
Seedtop	Performs a search between a sequence and a database of patterns and identifies which patterns occur in the sequence.
Fastacmd	Program to retrieve FASTA formatted sequences from a BLAST database.
Formatdb	Program to format BLASTable databases downloaded from NCBI.

MK



10.7 CONFIGURING blastall

+

After the executable has been installed, create a file called “ncbi.ini” in the Windows or WINNT directory on your machine (C:\Windows or C:\WINNT, etc. depending on the version of Windows you are running). The path to the file will be C:\Windows\ncbi.ini or C:\WINNT\ncbi.ini for the above two examples. Add the following lines to the ncbi.ini file:

```
[NCBI]
Data="C:\path\data\"
```

where,

```
C:\path\data\
```

is the path to the location of the Standalone BLAST “data” subdirectory which should be present in the directory where the downloaded file was extracted. To check if everything has been installed properly, test the bl2seq command as follows:

```
C:\blast>bl2seq.exe -i c:\perl\hpraa.txt -j c:\perl\bpraa.txt -p blastp
```

Where hpraa.txt and bpraa.txt are the protein sequences of the Human and Bovine pancreatic ribonuclease respectively:

MK

+


```
>GI:1350818
malekslvrlillvlvlgwvqpslgkesrakkfqrqhmdsdsspsststycnqmmrr
rnmtqgrckpvnrtfvheplvdvqnvfcqekvtckngqgncysnssmhitdrltngsry
pncayrtspkerhiivacegspyvpvhfdasvedst
>GI:133198
malkslvllslvlvllvrvqpslgketaaakferqhmdsstsaasssnycnqmmksrn
ltkdrckpvnrtfvhesladvqavcsqknvackngqtnncyqsytsmsitdcretgsskypn
caykttqankhiivacegnpyvpvhfdasv
```

If the command succeeds, you should get the following output (Figure 10.5).

```

C:\blastnew>bl2seq.exe -i c:\perl\hpraa.txt -j c:\perl\hpraa.txt -p blastp
Query= GI:1350818
      (156 letters)
>GI:133198
      Length = 150
  Score = 192 bits (489), Expect = 2e-054
  Identities = 87/130 (66%), Positives = 100/130 (76%)
Query: 23  UQPSLGKESRAKKFQRQHMXXXXXXXXXXXTYCNQMMRRRNMTQGRCKPUNTFUHEPLUDU 82
           UQPSLGKE+ A KF+RQHM                YCNQMM+ RN+T+ RCKPUNTFUHE L DU
Sbjct: 21  UQPSLGKETAAAKFERQHMDSSTSAASSSNYCNQMMKSRNLTKDRCKPUNTFUHESLADU 80
Query: 83  QNUCFQEKUTCKNGQGNQCYKSNSSMHITDCRLTNGSRYPNCAYRTSPKERHIIIVACEGSP 142
           Q UC Q+ U CKNGQ NCY+S S+M ITDCR T S+YPNCAY+T+ +HIIIVACEG+P
Sbjct: 81  QAUCSQKNUACKNGQTNQCYQSYSTMSITDCRETGSSSKYPNCAYKTTQANKHIIIVACEGNP 140
Query: 143 YUPVHFDASU 152
           YUPVHFDASU
Sbjct: 141 YUPVHFDASU 150

Lambda      K      H
0.319      0.131  0.414

Gapped
Lambda      K      H
0.267      0.0410  0.140

Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 119
Number of Sequences: 0
Number of extensions: 2
Number of successful extensions: 1
Number of sequences better than 10.0: 1
Number of HSP's better than 10.0 without gapping: 1

```

Fig. 10.5 Pair-wise BLAST output



10.8 Bio::Tools::Run::StandAloneBlast

We are now ready to use the Bio::Tools::Run::StandAloneBlast module to run the pair-wise BLAST we performed on the command-line. The basic code for the operation is as follows:

```
use Bio::Tools::Run::StandAloneBlast;

$seqobj = Bio::SeqIO->new(-file=>'c:\perl\kinesins.txt' ,
                        '-format' => 'Fasta' );

$seq1 = $seqobj->next_seq();
$seq2 = $seqobj->next_seq();

@params = ('program'=> blastp, 'outfile' => 'c:\perl\bl2seq.txt');
$factory = Bio::Tools::Run::StandAloneBlast->new(@params);
$bl2seq_report = $factory->bl2seq($seq1, $seq2);
```



As before, the output is directed to the file defined in the parameter list (bl2seq.txt). An alternate way is to provide the sequences on the command-line:



```
use Bio::Tools::Run::StandAloneBlast;

@params = ('program'=> blastp, 'outfile' => 'c:\perl\bl2seq.txt');
$factory = Bio::Tools::Run::StandAloneBlast->new(@params);
$seqfile1 = Bio::SeqIO->newFh ( -file => $ARGV[0],
                              -format => 'fasta' );

$seq1 = <$seqfile1>;
$seqfile2 = Bio::SeqIO->newFh ( -file => $ARGV[1],
                              -format => 'fasta' );

$seq2 = <$seqfile2>;

$report = $factory->bl2seq($seq1, $seq2);
```

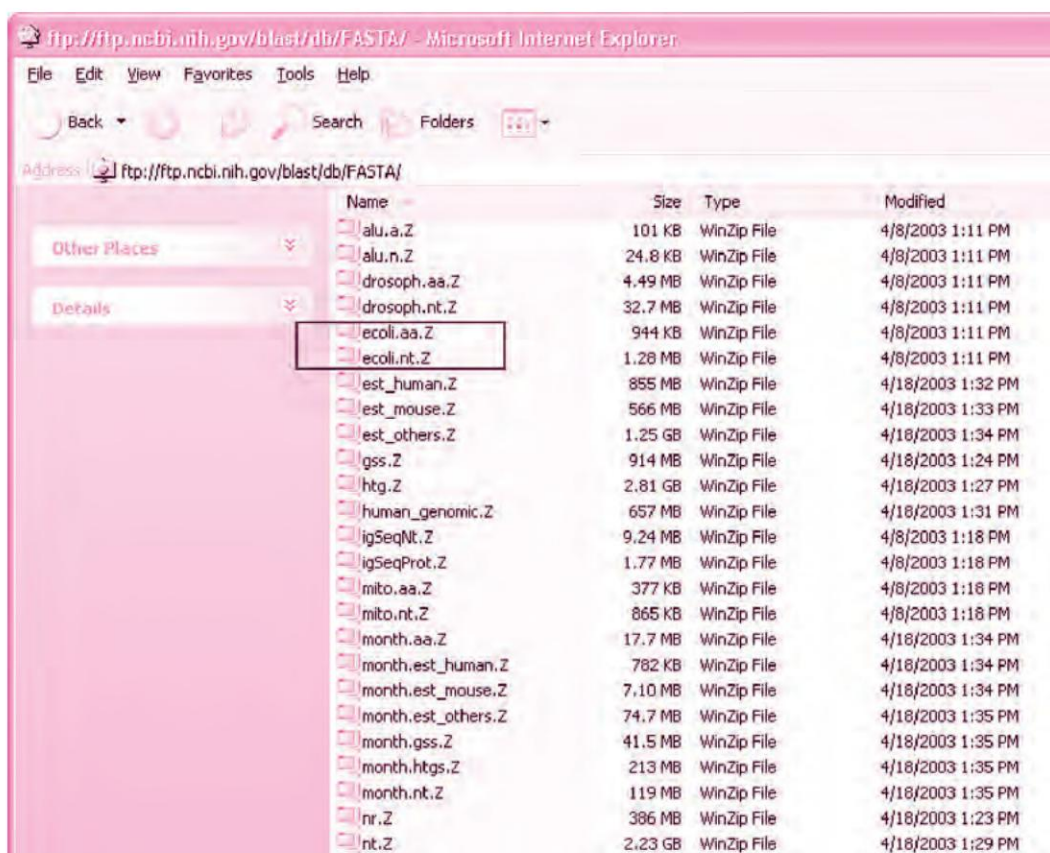
Save this script as bl2seq.pl and run it as follows:

```
C:\perl> bl2seq.pl hpraa.txt bpraa.txt
```

10.9 PERFORMING BLAST SEARCHES

BLASTing a sequence against a database can be done in a similar manner. The only difference is that you must have a database locally installed on your computer to run the local BLAST. The next few sections will explain how to download and format databases.

Figure 10.6 shows a list of databases available for download at NCBI. The ftp site is <ftp://ftp.ncbi.nih.gov/blast/db/FASTA/>. We advise installing a small database such as *ecoli.nt* or *ecoli.aa* (the nucleotide and amino acid databases of the bacterium *E. coli* respectively) to begin with. To do so, click on any of the *ecoli.nt.Z* or *ecoli.aa.Z* files and save it on your computer.



Name	Size	Type	Modified
alu.a.Z	101 KB	WinZip File	4/8/2003 1:11 PM
alu.n.Z	24.8 KB	WinZip File	4/8/2003 1:11 PM
drosoph.aa.Z	4.49 MB	WinZip File	4/8/2003 1:11 PM
drosoph.nt.Z	32.7 MB	WinZip File	4/8/2003 1:11 PM
ecoli.aa.Z	944 KB	WinZip File	4/8/2003 1:11 PM
ecoli.nt.Z	1.28 MB	WinZip File	4/8/2003 1:11 PM
est_human.Z	855 MB	WinZip File	4/18/2003 1:32 PM
est_mouse.Z	566 MB	WinZip File	4/18/2003 1:33 PM
est_others.Z	1.25 GB	WinZip File	4/18/2003 1:34 PM
gss.Z	914 MB	WinZip File	4/18/2003 1:24 PM
htg.Z	2.81 GB	WinZip File	4/18/2003 1:27 PM
human_genomic.Z	657 MB	WinZip File	4/18/2003 1:31 PM
igSeqNt.Z	9.24 MB	WinZip File	4/8/2003 1:18 PM
igSeqProt.Z	1.77 MB	WinZip File	4/8/2003 1:18 PM
mito.aa.Z	377 KB	WinZip File	4/8/2003 1:18 PM
mito.nt.Z	865 KB	WinZip File	4/8/2003 1:18 PM
month.aa.Z	17.7 MB	WinZip File	4/18/2003 1:34 PM
month.est_human.Z	782 KB	WinZip File	4/18/2003 1:34 PM
month.est_mouse.Z	7.10 MB	WinZip File	4/18/2003 1:34 PM
month.est_others.Z	74.7 MB	WinZip File	4/18/2003 1:35 PM
month.gss.Z	41.5 MB	WinZip File	4/18/2003 1:35 PM
month.htgs.Z	213 MB	WinZip File	4/18/2003 1:35 PM
month.nt.Z	119 MB	WinZip File	4/18/2003 1:35 PM
nr.Z	386 MB	WinZip File	4/18/2003 1:23 PM
nt.Z	2.23 GB	WinZip File	4/18/2003 1:29 PM

Fig. 10.6 NCBI databases

10.10 FORMATTING NCBI'S DATABASES

You need to format databases before you can run searches on them. NCBI provides a tool called `formatdb` that is part of the BLAST suite of programs to create your own BLAST-searchable database. To format a nucleotide database such as `ecoli.nt` database, run the following command from the DOS prompt:

```
C:\blast>formatdb -i ecoli.nt -p F -o T
```

The corresponding command to format a protein sequence database such as `ecoli.aa` is:

```
C:\blast>formatdb -i ecoli.aa -p T -o T
```

The options `-i`, `-p` and `-o` used with `formatdb` are some of the most commonly used arguments. The individual options are explained in Table 10.3.

Table 10.3 *formatdb arguments*

Option	Function
-i	Input file for formatting
-p	Type of file T — protein sequences (default) F — nucleotide sequences
-o	Parse options T — True: Parse SeqId and create indexes. F — False: Do not parse SeqId. Do not create indexes.
-t	Title for database file
-n	Base name for BLAST files. Produces a database with a different name than that of the original FASTA file. To create a database called <code>myecoliDB</code> from <code>ecoli.nt</code> , for example, type: <code>formatdb -i ecoli.nt -p F -o T -n myecoliDB</code>
-s	Create indexes limited only to accessions—sparse [T/F]. Default = F. This option limits the indices for the string identifiers used by <code>formatdb</code> to accessions (i.e., no locus names) and is especially useful for sequences sets like the ESTs where the accession and locus names are identical. <code>formatdb</code> runs faster and produces smaller temporary files if this option is used. It is strongly recommended for EST, STS, GSS and HTG sequences.

Some of these arguments such as title of database, base name of database, etc. are optional. When a BLAST-searchable database is created, a number of files are produced. Using `formatdb`, these files will have extensions `.phr`, `.pin`, `.psq` for protein databases and `.nhr`, `.nin`, `.nsq` for nucleotide databases. The `ecoli.nt` file can be removed once `formatdb` has been run.



10.11 RUNNING blastall

To run `blastall` against the `ecoli.nt` database, download a test *E. coli* sequence from NCBI (<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=nucleotide>) such as the *E. coli* beta-lactamase nucleotide sequence, save it on your computer and run the following command:

```
C:\blast>blastall -p blastn -d ecoli.nt -i lactamase.txt -o lactamase.out
```

MK Note that you may get the “[NULL_Caption] WARNING: test: Could not find index files for database” error message when `blastall` cannot find the database you have specified. If any of these databases or files is on a different directory than where BLAST is installed, you may need to specify the full path to the database. For example, **+**

```
c:\blast\blastall -p blastp -d d:\blastdb\nr\nr -i kinase.txt
```

An explanation of common command-line flags used with the `blastall` command is provided in Table 10.4.

Table 10.4 *blastall options*

Option	Function	Values
-p	Program name	blastn, blastp, blastx, tblastn or tblastx
-d	Database name	nr, swissprot, est, etc.
-I	Input (query) sequence file	cftr.txt, etc.
-o	BLAST results (output file)	cftrout.txt, etc.
-e	E value	0.1, 0.01, etc. Default = 10.
-F	Filter query sequence	T or F (for true or false)
-q	Penalty for a nucleotide mismatch	integer



Table 10.4 (Contd.)

-r	Reward for a nucleotide match	integer
-v	Number of one line descriptions	integer
-b	Number of alignments to show	integer
-g	Perform gapped alignment	T or F (for True or False)
-M	Matrix	matrix name
-W	Word size	integer
-T	Produce HTML output	T or F (for True or False)

To look at the contents of the BLAST results, open the `blast.out` file using the `more` command on the DOS command-line or with a text editor such as Notepad.



10.12 RUNNING BLAST WITH Bio::Tools::Run::STANDALONEBLAST



The code to run BLAST using `Bio::Tools::Run::StandAloneBlast` is as follows:

```
use Bio::Tools::Run::StandAloneBlast;

@params = ('database' => 'ecoli\ecoli.aa', 'program'=> blastp,
'outfile' => 'c:\perl\blastout.txt', '_READMETHOD' => 'Blast');

$factory = Bio::Tools::Run::StandAloneBlast->new(@params);

#Blast a sequence against a database:
$seqfile = Bio::SeqIO->new(-file=>'c:\perl\kinesins.txt' ,
                           '-format' => 'Fasta' );

$seq = $seqfile->next_seq();

$blast_report = $factory->blastall($seq);
```



Note

The location of the database should be `C:\blast\data`. This is assuming you have downloaded the `blastall` executable in `C:\blast`. The database location specified as `'ecoli\ecoli.aa'` actually means:

'C:\blast\data\ecoli\ecoli.aa'

The BLAST output is directed to the file specified in the parameters list. Note that this is the raw BLAST output that can be viewed with a text editor such as Notepad. In the next chapter, we will see how to parse BLAST reports generated by these scripts.

Assignments

1. Write a script that generates a pair-wise alignment between a set of sequences in a multiple Fasta file and parses the output for the E values, Identities and Positives. Vary the matrix used and find out the difference in the output obtained.

The script should be run as follows:

```
blast2.pl -p blastp -m Blosum62 -f filename
```

where

MK +	<pre>[-p Program name (any of the five BLAST programs) [-m substitution matrix, example., Blosum62, PAM30 [-f Fasta file, use zfkinase.txt</pre>	<pre>]]]</pre>	MK +
---------	-----------------------------------------------------------------------------------------------------------------------------------------------------------	------------------	---------

and the output should be two tables:

Table1: Alignment scores

Program used: Blastp

Matrix used: Blosum62/other

E value used: 10/other

ID1	ID2	Score	(bits)	Expect	Identities	Positives	Gaps
...							

...

Table 2: Protein sequence data

ID	Name	Length
...		

Note that if no similarity is found, this should be stated as zero identities, zero positives, etc.

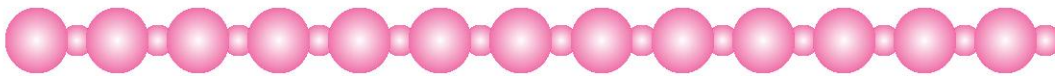
2. Given the partial DNA sequence for an unknown gene:

```
aacccgaaaa tccttccttg caggaaacca gtctcagtg ccaactctt aaccttgaa
ctgtgagaac tctgaggaca aagcagcgga tacaacctca aaagacgtct gtctacattg
aattgggatac tgattcttct gaagataccg ttaataaggc aacttattgc agtgtgggag
atcaagaatt gttacaaatc acccctcaag gaaccaggga tgaaatcagt ttggattctg
caaaaaaggc tgcttgtaaa tttctgaga cggatgtaac aaatactgaa catcatcaac
ccagtaataa tgattgaac accactgaga agcgtgcagc tgagaggcat ccagaaaagt
atcagggtag ttctgtttca aacttgcattg ttgagccatg tggcacaaat actcatgcca
gctcattaca gcatgagaac agcagtttat tactactaa agacagaatg aatgtagaaa
aggctgaatt ctgtaataaa agcaaacagc ctggcttagc aaggagccaa cataacagat
```

List three possible BLAST programs that you can use to analyze this sequence. Perform each of these analyses separately and compare the first 10 hits from each of the outputs. Use your knowledge of E values, matrices, gap penalties, etc. to set parameters that may be optimal for the search. What is the effect of varying word length and gap penalties on the output? Identify the gene and describe its structure. What is the significance of this gene and its protein product?



Parsing BLAST Output



MK



In the last chapter, we learnt how to perform a local BLAST using the `Bio::Tools::Run::StandAloneBlast.pm` module. All the programs yielded the raw BLAST output which, you must have realized, can be quite verbose, complex and difficult to interpret. This is because it generally holds a lot of data on the different aspects of the hits that the search reveals. It would be easier if there was a way to parse the output so that only the most relevant pieces of information could be extracted and presented in a more readable form. In this chapter, we will learn how to apply BioPerl methods to parse these raw files and utilize the information more effectively. In particular, we will use the `Bio::Tools::Blast` module. `Bio::Tools::Blast` supports NCBI Blast1.x, Blast2.x, and WashU-Blast2.x, including both gapped and ungapped alignments.

MK



11.1 GENERATING A RAW BLAST REPORT

Let's first set up a simple BLAST analysis to generate a raw BLAST output file. Figure 11.1 shows the result of a protein-protein BLAST done with the human pancreatic ribonuclease (HPR):

```

C:\ Command Prompt - more hprout.txt
D:\blast>blastall.exe -d d:\blastdb\nr\nr -i c:\perl\hpraa.txt -o hprout.txt -p
blastp
D:\blast>more hprout.txt
BLASTP 2.2.6 [Apr-09-2003]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= GI:1350818
      (156 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF
          1,047,264 sequences; 330,262,426 total letters

Sequences producing significant alignments:

                                Score   E
                                (bits) Value
ref|NP_002924.1|<NM_002933> ribonuclease, RNase A family, 1 (pa... 277 2e-
74
pir|INRHU1 pancreatic ribonuclease (EC 3.1.27.5) precursor - human 276 7e-
74
gb|AAL87050.1|AF449629_1 (AF449629) pancreatic ribonuclease [Gor... 275 9e-
74
gb|AAL87052.1|AF449631_1 (AF449631) pancreatic ribonuclease [Hyl... 274 2e-

```

Fig. 11.1 BLASTP search with HPR sequence

```

>gi|1350818|sp|P07998|RNP_HUMAN Ribonuclease pancreatic precursor (RNase 1) (RNase
Upl-1) (RIB-1)

MALEKSLVRLLLLVILLVLGWVQPSLGKESRAKKFQRQHMDSDSSPSSSSTYCNQMMRRRNMTQGRCKP
VNTFVHEPLVDVQNVCFQEKVTCKNGQGNCYKSNSSMHITDCRLTNGSRYPNCAYRTSPKERHIIVACEG
SPYVPVHFDASVEDST

```

The command was run with the following parameters:

Database nr

Input file hpraa.txt (the HPR amino acid sequence)

Program BlastP

and the output was redirected to a file called hprout.txt:

```
blastall -d d:\blastdb\nr\nr -i c:\perl\hpraa.txt -o hprout.txt -p blastp
```

The raw BLAST results performed on the command-line and through the NCBI website are shown in Figures 11.2 and 11.3.

```

hprout - Notepad
File Edit Format View Help
BLASTP 2.2.6 [Apr-09-2003]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= GI:1350818
      (156 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF
          1,047,264 sequences; 330,262,426 total letters

Sequences producing significant alignments:

                                Score   E
                                (bits) value
ref|NP_002924.1| (NM_002933) ribonuclease, RNase A family, 1 (pa... 277 2e-074
pir|NRH01 pancreatic ribonuclease (EC 3.1.27.5) precursor - human 276 7e-074
gb|AAL87050.1|AF449629.1 (AF449629) pancreatic ribonuclease [Gor... 275 9e-074
gb|AAL87052.1|AF449631.1 (AF449631) pancreatic ribonuclease [Hyl... 274 2e-073
gb|AAL87049.1|AF449628.1 (AF449628) pancreatic ribonuclease [Pan... 272 8e-073
gb|AAL87051.1|AF449630.1 (AF449630) pancreatic ribonuclease [Pon... 271 2e-072
emb|CAA55817.1| (X79235) pancreatic ribonuclease [Homo sapiens] 267 3e-071
gb|AAL87063.1|AF449642.1 (AF449642) pancreatic ribonuclease [Pyg... 265 2e-070
pir|I53530 pancreatic ribonuclease (EC 3.1.27.5) precursor - hu... 264 3e-070
gb|AAL87058.1|AF449637.1 (AF449637) pancreatic ribonuclease [Sai... 262 8e-070
gb|AAL87060.1|AF449639.1 (AF449639) pancreatic ribonuclease [Ate... 260 3e-069
gb|AAL87059.1|AF449638.1 (AF449638) pancreatic ribonuclease [Sag... 258 2e-068
gb|AAL87061.1|AF449640.1 (AF449640) pancreatic ribonuclease [Lag... 255 1e-067
emb|CAA44718.1| (X62946) pancreatic ribonuclease [Homo sapiens] 253 4e-067

```

Fig. 11.2 Raw BLAST output (hprout.txt)

11.2 THE BIO::TOOLS::BLAST MODULE

The minimal code to parse a BLAST report using the use Bio::Tools::Blast module is as follows:

```

use Bio::Tools::Blast;

%parameters = ( specify parameters );

$blastObj = Bio::Tools::Blast->new(%parameters);

foreach $hit($blastObj->hits) { extract data }

```

where,

```

%parameters [parameters for parsing Blast reports]

```




The screenshot shows a web browser window with the address bar displaying `http://www.ncbi.nlm.nih.gov/blast/Blast.cgi#19386943`. The page title is "RID=1059874874-08456-25300, gj|1350818|ep|P07998|RNase_HUMAN Ribonuclease pancr...". The browser's menu bar includes File, Edit, View, Favorites, Tools, and Help. The address bar has a "Go" button. The main content area displays the results of a BLAST search, titled "Sequences producing significant alignments:". The results are presented in a table with three columns: "gi|...", "Score (bits)", and "E Value". The table lists various sequences, including pancreatic ribonucleases and ribonuclease families, with their corresponding scores and E-values.

gi ...	Score (bits)	E Value
gi 19386943 gb AAL87051.1 AF449630.1	268	3e-71
gi 19386957 gb AAL87058.1 AF449637.1	262	2e-69
gi 19386961 gb AAL87060.1 AF449639.1	260	5e-69
gi 4506547 ref NP_002924.1	259	8e-69
gi 1360656 pir INRHU1	259	1e-68
gi 19386941 gb AAL87050.1 AF449629.1	258	2e-68
gi 19386945 gb AAL87052.1 AF449631.1	257	4e-68
gi 19386959 gb AAL87059.1 AF449638.1	256	6e-68
gi 19386939 gb AAL87049.1 AF449628.1	255	2e-67
gi 19386963 gb AAL87061.1 AF449640.1	252	1e-66
gi 19386967 gb AAL87063.1 AF449642.1	251	2e-66
gi 488413 emb CAA55817.1	251	3e-66
gi 19386969 gb AAL87064.1 AF449643.1	239	1e-62
gi 2135882 pir IS3530	237	3e-62
gi 19386965 gb AAL87062.1 AF449641.1	237	4e-62
gi 35281 emb CAA44718.1	234	5e-61
gi 14278127 pdb 1E21 A	232	1e-60
gi 20150003 pdb 1H6X A	229	1e-59
gi 19399882 pdb 1D2A A	229	1e-59
gi 19386953 gb AAL87056.1 AF449635.1	228	2e-59
gi 19386947 gb AAL87053.1 AF449632.1	227	4e-59
gi 19386951 gb AAL87055.1 AF449634.1	227	6e-59
gi 19386949 gb AAL87054.1 AF449633.1	223	1e-57

Fig. 11.3 Output of BLASTP performed at NCBI

`$blastobj` [an instance of `Bio::Tools::Blast`]

`$hit` [each sequence element returned by BLAST]

As always, a BLAST object can be instantiated using the new keyword in this manner as well:

```
$blastobj = new Bio::Tools::Blast (%parameters);
```

The parameters for parsing a BLAST report file are presented in the form of a hash (key and value pairs) and are used to specify information shown in Table 11.1.

Table 11.1 *Parameters for parsing BLAST*

Parameter (key)	Function (value)
-file	name of file containing BLAST report
-signif	cutoff E value. Hits with E value greater than this will be ignored
-filt_func	Subroutine added to filter output by special criteria, e.g., gaps < 10
-check_all_hits	0 or 1. If set to 1 (true), all hits will be parsed to check if they meet the significance criteria specified by the parameters -signif and -filt_func. The default = 0 (or false), which means parsing of hits will stop when the significance criteria fail. This speeds the parsing process.
-stats	0 or 1. If set to 1, the program will collect information on the matrix, filters, etc. used in the BLAST report. Default = false.
-best	0 or 1. If set to 1, the program will only process the best hit for each report. Default = false.
-strict	0 or 1. If set to 1, uses strict mode for all BLAST objects created to enhance error trapping.

The signif can be a float (e.g., 0.001) or a number in scientific notation (e.g., 1e-10). In addition, a parameter called “parse”, with a value of 1, must be specified for the parse to work:

```
parse    [boolean, (=1) to parse the BLAST report ]
```

Note that all the parameters are specified in the form of a hash which is nothing but a Perl data type that holds variables and their corresponding values written as pairs separated by a delimiter (commonly =>):

```
%parameters = ( -file      => 'path_to_BLAST_report',
                 -parse     => 1,
                 -filt_func => \&filter_function
               );
```

The Perl shorthand to represent a hash is a % sign, thus the name %parameters for the hash above.

Note

We have used hashes (also called associative arrays) in previous chapters (although, perhaps, without explicitly stating so). The function `GetOptions()` from the module `Getopt::Long`, for example, uses a hash to obtain command-line arguments that you want to plug into the code at run-time. To search a file with a given search term, for example, the hash key-value pairs can be set up as follows:

```
GetOptions("f|filename=s"=>\$filename, "s|searchterm =s"=>\$searchterm);
```

Here the keys are `filename` and `searchterm` and their values are `\$filename` and `\$searchterm` respectively (Table 11.2).

Table 11.2 *Key-value pairs in a hash*

Key	Delimiter	Value
F filename=s	=>	\\$filename
S searchterm=s	=>	\\$searchterm

The “=s” after each variable, as we have seen earlier, means that a string is expected (rather than, say, an integer or a float). Adding a back-slash (\) before a variable creates a reference to that variable and here it means that the value of each of the keys is a reference to the variables `\$filename` and `\$searchterm` respectively. The actual string values of the variables (the file name and the search term itself) are obtained by the `GetOptions()` function (by a process called dereferencing) when the code executes.

The methods provided by `Bio::Tools::Blast` to extract information about the individual hits (matches found to the query sequence in the database) and the corresponding code are shown in Table 11.3.

Table 11.3 *Bio::Tools::Blast methods*

Methods to extract top-level information on every hit	
Method	Code
Sequence identifier of a hit	<code>\\$hit->name;</code>

(Contd.)

Table 11.3 (Contd.)

E value of a hit	<code>\$hit->expect;</code>
Number of high scoring pairs for each hit	<code>\$hit->num_hsps;</code>
Number of identities between query and subject sequence	<code>\$hit->frac_identical;</code>
Number of gaps in the alignment between query and subject sequence	<code>\$hit->gaps;</code>
Most significant hit	<code>\$hit = \$blastObj->hit;</code>
E value of most significant hit (for NCBI BLAST2 reports only)	<code>\$eval = \$blastObj->lowest_expect;</code> or, <code>\$eval = \$blastObj->hit->expect;</code>
P value of most significant hit (for BLAST1/WashU-BLAST2 reports only. P values are not reported in NCBI BLAST2 reports)	<code>\$pval = \$blastObj->lowest_p;</code> or, <code>\$pval = \$blastObj->hit->p;</code>
Start coordinates of hit (subject) sequence	<code>\$sobj->start('query');</code>
End coordinates of hit (subject) sequence	<code>\$sobj->end('sobj');</code>
Get both query and subject in array context	<code>(\$query_start, \$subject_start) = \$sobj->start();</code> <code>(\$query_end, \$subject_end) = \$sobj->end();</code>

Methods to obtain information on high scoring pairs (HSPs) for each hit

E value of HSP	<code>\$hsp->expect</code>
Raw score of HSP	<code>\$hsp->score</code>
Score in bits	<code>\$hsp->bits</code>
The fraction of identical positions within the given HSP. Returns a float with a two-decimal precision.	<code>\$hsp->frac_identical</code>
Get the fraction of conserved positions ('positives') within the given HSP. Returns a float with a two-decimal precision.	<code>\$hsp->frac_conserved</code>
The number of gaps in the query.	<code>\$hsp->gaps('query')</code>
The number of gaps in the subject (hit)	<code>\$hsp->gaps('sobj')</code>
The full query sequence as a string	<code>\$hsp->seq_str('query');</code>
The full subject sequence as a string	<code>\$hsp->seq_str('sobj');</code>

Note

To recap what we have learnt in earlier chapters, the significance of hits returned by BLAST is gauged with the help of two numbers: the bit score and the E value. The bit score is defined as

$$S' \text{ (bits)} = [l * S - \ln K] / \ln 2$$

where,

S' [bit score]

S [raw score]

and,

λ (lambda) and K are Karlin-Altschul parameters

The conversion of the raw score into a normalized bit score makes it independent of the matrix used (e.g., BLOSUM62). The larger the bit score, the greater the significance of the hit.

The E value, on the other hand, is an estimate of the statistical significance of the match, specifying the number of matches, with a given score, that are expected to occur in a search of a database of a particular size purely by chance alone. An E value of 0.001, for example, means that there is a chance of 1 in 1000 that the match has occurred by chance. One would also expect that as the size of a database increases, there is more likelihood of getting hits with a certain score that occur by chance alone. For this reason, the E value depends on the size of the database searched. It is easy to see that as the E value for a particular match decreases, the significance of the match increases—that is, we are more confident that the match is real. Thus, the smaller the E value, the greater the significance of the hit.

Methods for obtaining data on high-scoring segment Pairs or HSPs are provided by the `Bio::Tools::Blast::HSP` module. However, `Bio::Tools::Blast::HSP` methods are accessed not directly but through the `Bio::Tools::Blast` module as follows:

```
use Bio::Tools::Blast;

%parameters = ( ... );

$blastObj = Bio::Tools::Blast->new(%parameters);

$hit = $blastObj->hit;           #obtain data on top hit

$hsp = $blastObj->hit->hsp;      #obtain data on HSPs for top hit
```

Each of these objects are hashes representing a `Bio::Tools::Blast::Sbjct` and a `Bio::Tools::Blast::HSP` object.

To get all hits, use a `foreach` loop to iterate through the BLAST output:

```
foreach $hit($blastObj->hits) {
    printf "%s\t",      $hit->name;
    printf "%.1e\t",    $hit->expect;
    printf "%d\t",      $hit->num_hsps;
    printf "%.2f\t",    $hit->frac_identical;
    printf "%d\n",      $hit->gaps;
}
```

The `printf` function is used in place of the standard `print` function to format the individual variables as needed. For example, `%s` is used to format the hit name since it is a string. `%e` is used to format the E values in scientific notation while `frac_identical` is formatted as a float with two decimal points with `%.2f`.



The separate `printf` statements can also be written as one combined statement:

```
foreach $hit($blastObj->hits) {
    printf "%s\t %.1e\t %d\t %.2f\t %d\n",
        $hit->name, $hit->expect, $hit->num_hsps,
        $hit->frac_identical, $hit->gaps;
}
```



Similarly, a `foreach` loop is used to obtain all the HSPs for each hit:

```
foreach $hsp ($hit->hsps) {
    printf "%.1e\t %d\t %.1f\t %.2f\t %.2f\t %d\t %d\n",
        $hsp->expect, $hsp->score, $hsp->bits,
        $hsp->frac_identical, $hsp->frac_conserved,
        $hsp->gaps('query'), $hsp->gaps('sbjct');
}
```

Since each hit may have one or more HSPs, the above code needs to be placed inside the `foreach` loop for the individual hits:

```

foreach $hit ($blastObj->hits) {
    printf "%s %d", $hit->name, $hit->num_hsps;
    print "\n";
    foreach $hsp ($hit->hsps) {
        printf "%.1e\t %d\t %.1f\t %.2f\t %.2f\t %d\t %d\n",
            $hsp->expect, $hsp->score, $hsp->bits,
            $hsp->frac_identical, $hsp->frac_conserved,
            $hsp->gaps('query'), $hsp->gaps('sbjct');
    }
}

```

To output the parameters and filters used to parse the BLAST output, use the `display()` function:

```
$blastObj->display(-SHOW=>'stats');
```



11.3 PARSING THE HPR BLAST REPORT



Let's parse the HPR BLAST output with a program that incorporates what we have learnt so far (Listing 11.1).

Listing 11.1 **parseblast1.pl**

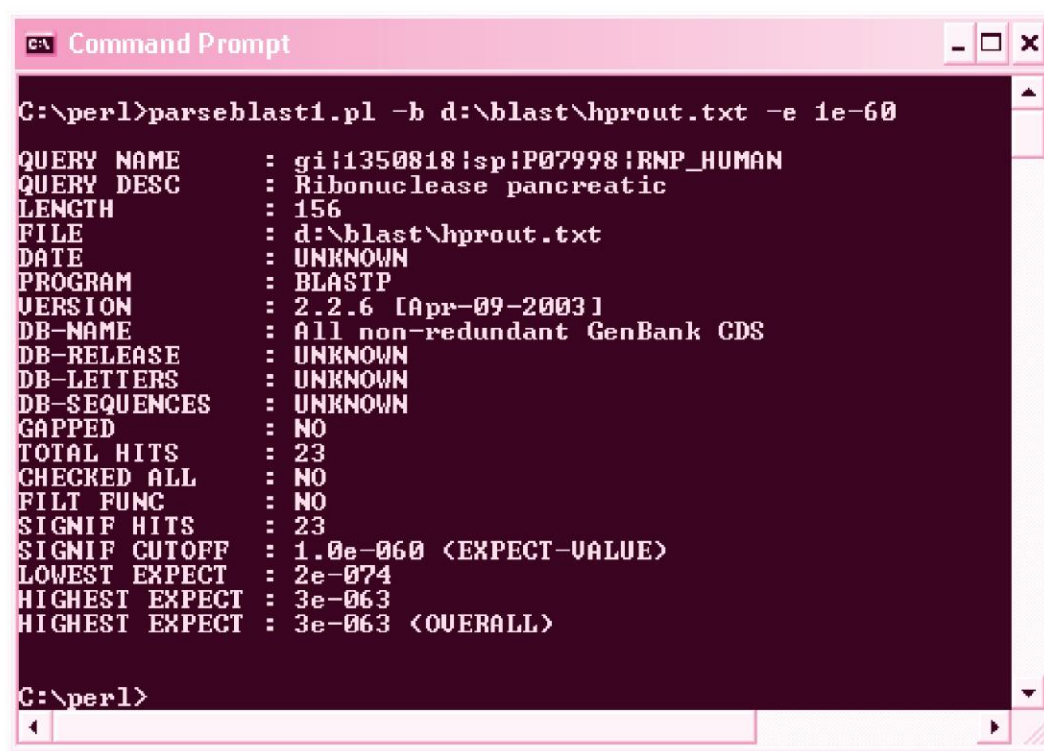
```

use Getopt::Long;
use Bio::Tools::Blast;
GetOptions("b|blastfile=s"=>\$blastfile, "e|evalue=f"=>\$evalue);
%parameters = ( -file      => "$blastfile",
                 -parse    => 1,
                 -signif   => "$evalue",
                 );
$blastObj = Bio::Tools::Blast->new(%parameters);
$blastObj->display(-SHOW=>'stats');
```


Save this program on your computer and run it with an E value of 1e-60. The command used is:

```
C:\perl>parseblast1.pl -b d:\blast\hprout.txt -e 1e-60
```

The output is shown in Figure 11.4.



```
C:\perl>parseblast1.pl -b d:\blast\hprout.txt -e 1e-60

QUERY NAME      : gi|1350818|sp|P07998|RNP_HUMAN
QUERY DESC      : Ribonuclease pancreatic
LENGTH         : 156
FILE            : d:\blast\hprout.txt
DATE            : UNKNOWN
PROGRAM         : BLASTP
VERSION         : 2.2.6 [Apr-09-2003]
DB-NAME         : All non-redundant GenBank CDS
DB-RELEASE      : UNKNOWN
DB-LETTERS      : UNKNOWN
DB-SEQUENCES    : UNKNOWN
GAPPED         : NO
TOTAL HITS      : 23
CHECKED ALL     : NO
FILT FUNC       : NO
SIGNIF HITS     : 23
SIGNIF CUTOFF   : 1.0e-060 <EXPECT-VALUE>
LOWEST EXPECT   : 2e-074
HIGHEST EXPECT  : 3e-063
HIGHEST EXPECT  : 3e-063 <OVERALL>

C:\perl>
```

Fig. 11.4 parseblast1.pl output

The output of the `display()` function lists the relevant information about the BLAST search as well as the parameters used for parsing, and extracts the values specified (sequence name, description and input file, BLAST program and database used, cutoff E value and number of significant hits obtained at the cutoff E value used, etc.). This information serves as a handy record about the BLAST search, especially if you are performing a large number of searches.

Try to match the output in Figure 11.4 versus the parameters list in Table 11.1. You will notice that since we did not specify a filter function or set the

check_all_hits parameter to 1 (parse all hits), the values of both parameters in Figure 11.4 are “No” (see arrows, Figure 11.4). Note also that the value of the Total Hits is equal to the value of Signif Hits (23). This indicates that, indeed, the parsing was limited only to the significant hits. To turn the check_all_hits parameter on, simply change the parameters list to the following:

```
%parameters = ( -file          => "$blastfile",
                 -parse        => 1,
                 -check_all_hits => 1,
                 -signif       => "$evalue",
                 );
```

Run the script again. This time the program will check all the hits and the check_all_hits parameter will be set to “Yes”. Note now that the value of Total Hits changes (359) while the Signif Hits remains at 23.

Let’s now enhance the functionality of the program to extract some information on individual hits and their HSPs. Run the code shown in Listing 11.2 to see this in effect.

Listing 11.2 parseblast2.pl

```
use Getopt::Long;
use Bio::Tools::Blast;

GetOptions("b|blastfile=s"=>\$blastfile, "e|evalue=f"=>\$evalue);
%parameters = ( -file      => "$blastfile",
                 -parse    => 1,
                 -signif   => "$evalue"
                 );

$blastObj = Bio::Tools::Blast->new(%parameters);
$blastObj->display(-SHOW=>'stats');
print "Hit #\tSeq Id\t# of hsp\te-value\tRaw score\tBit
score\tFractionIdentical\tPositives\tGaps(query)\tGaps(subject)\n";

foreach $hit ($blastObj->hits) {
    $count++;
```

(Contd.)

(Contd.)

```

print "$count] ";
printf "%s %d", $hit->name, $hit->num_hsp;
foreach $hsp ($hit->hsps) {
    printf " %.1e\t %d\t %.1ft %.2ft %.2ft %d\t %d\n",
        $hsp->expect, $hsp->score, $hsp->bits,
        $hsp->frac_identical, $hsp->frac_conserved,
        $hsp->gaps('query'), $hsp->gaps('sbjct');
}
}

```

The output of the script is shown in Figure 11.5.

```

C:\perl>parseblast2.pl -b d:\blast\hprout.txt -e 1e-60

```

Hit #	Seq Id	# of hsps	e-value	Raw score	Bit score	Fraction
1	ref:INP_002924.1	1	2.0e-074	709	277.0	0.85
2	pir: NRHU1	1	7.0e-074	705	276.0	0.84
3	gb: AAL87050.1	1	9.0e-074	704	275.0	0.84
4	gb: AAL87052.1	1	2.0e-073	701	274.0	0.84
5	gb: AAL87049.1	1	8.0e-073	696	272.0	0.83
6	gb: AAL87051.1	1	2.0e-072	693	271.0	0.83
7	emb: CAA55817.1	1	3.0e-071	682	267.0	0.83
8	gb: AAL87063.1	1	2.0e-070	676	265.0	0.79

Fig. 11.5 parseblast2.pl output

We will now see how we can specify additional filters to process the BLAST results.



11.4 SPECIFYING A FILTER FUNCTION

An example of how a filter function (`-filt_func`) can be used to parse a BLAST report is as follows:

1. Create the subroutine (called `filterBLAST` here) containing the filtering criteria (here we will use the condition `gaps = 0` to filter the BLAST report):

```
sub filterBlast {
    $hit=shift;
    return ($hit->gaps == 0);
}
```

(the `shift` function is used to make the subroutine iterate through each hit)

2. Plug the subroutine into the parameters hash:

```
%parameters = ( -file      => 'path_to_BLAST_report',
                 -parse     => 1,
                 -filt_func => \&filterBlast
               );
```

3. Specify the parameters to the BLAST object using the new keyword:

```
$blastObj = Bio::Tools::Blast->new(%parameters);
```

When the code runs, the BLAST object checks for matches to the specified criteria by calling the `&filterBlast($hit)` subroutine for each hit. The subroutine returns false and stops when a hit fails the criteria. Run the code shown in Listing 11.3 to see this in effect.

Listing 11.3 parseblast3.pl

```

use Getopt::Long;
use Bio::Tools::Blast;
GetOptions("b|blastfile=s"=>\$blastfile, "e|evalue=f"=>\$evalue);
sub filterBlast {
    $hit=shift;
    return ($hit->bits > 250);
}
%parameters = ( -file      => "$blastfile",
                -parse     => 1,
                -filt_func => \&filterBlast,
                -signif    => "$evalue",
                );
$blastObj = Bio::Tools::Blast->new(%parameters);
$blastObj->display(-SHOW=>'stats');
print "Hit #\tSeq Id\t# of hsp\te-value\tRaw score\tBit
score\tFractionIdentical\tPositives\tGaps(query)\tGaps(subject)\n";
foreach $hit ($blastObj->hits) {
    $count++;
    print "$count] ";
    printf "%s %d", $hit->name, $hit->num_hsp;
    foreach $hsp ($hit->hsps) {
        printf " %.1e\t %d\t %.1f\t %.2f\t %.2f\t %d\t %d\n",
            $hsp->expect, $hsp->score, $hsp->bits,
            $hsp->frac_identical, $hsp->frac_conserved,
            $hsp->gaps('query'), $hsp->gaps('sbjct');
    }
}

```

The output is shown in Figure 11.6. Note that the filter function value is now set to "Yes" and that there are now only 15 significant hits in place of 23 (see arrows) due to the additional `$hit->bits > 250` criteria applied to process the report.

```

C:\perl>parseblast3.pl -b d:\blast\hprout.txt -e 1e-60

QUERY NAME      : gi|1350818|sp:P07998|RNP_HUMAN
QUERY DESC     : Ribonuclease pancreatic
LENGTH        : 156
FILE           : d:\blast\hprout.txt
DATE           : UNKNOWN
PROGRAM        : BLASTP
VERSION        : 2.2.6 [Apr-09-2003]
DB-NAME        : All non-redundant GenBank CDS
DB-RELEASE     : UNKNOWN
DB-LETTERS     : UNKNOWN
DB-SEQUENCES   : UNKNOWN
GAPPED         : NO
TOTAL HITS     : 23
CHECKED ALL    : NO
FILT FUNC      : YES
SIGNIF HITS    : 15
SIGNIF CUTOFF  : 1.0e-060 (EXPECT-VALUE)
LOWEST EXPECT  : 2e-074
HIGHEST EXPECT : 1e-066
HIGHEST EXPECT : 3e-063 (OVERALL)

Hit #  Seq Id  # of hsp  e-value  Raw score  Gaps(query)  Gaps(subject)  Bit score  Fraction
-----
1  ref|NP_002924.1| 1 2.0e-074  709  277.0  0.85  0.85  0  0
2  pir|I|NRHU1 1 7.0e-074  705  276.0  0.84  0.84  0  0
3  gb|AAL87050.1|AF449629_1 1 9.0e-074  704  275.0  0.84  0.85  0  0
4  gb|AAL87052.1|AF449631_1 1 2.0e-073  701  274.0  0.84  0.84  0  0
5  gb|AAL87049.1|AF449628_1 1 8.0e-073  696  272.0  0.83  0.84  0  0
6  gb|AAL87051.1|AF449630_1 1 2.0e-072  693  271.0  0.83  0.84  0  0
7  emb|CAA55817.1| 1 3.0e-071  682  267.0  0.83  0.84  0  0
8  gb|AAL87063.1|AF449642_1 1 2.0e-070  676  265.0  0.79  0.83  0  0
9  pir|I|I53530 1 3.0e-070  674  264.0  0.92  0.92  0  0
10 gb|AAL87058.1|AF449637_1 1 8.0e-070  670  262.0  0.80  0.83  0  0
11 gb|AAL87060.1|AF449639_1 1 3.0e-069  665  260.0  0.79  0.82  0  0
12 gb|AAL87059.1|AF449638_1 1 2.0e-068  659  258.0  0.79  0.82  0  0
13 gb|AAL87061.1|AF449640_1 1 1.0e-067  652  255.0  0.78  0.81  0  0
14 emb|CAA44718.1| 1 4.0e-067  647  253.0  0.92  0.92  0  0
15 pdb|1E21:A 1 1.0e-066  643  252.0  0.91  0.91  0  0

C:\perl>

```

Fig. 11.6 Output of parseblast3.pl

11.5 FORMATTING PARSING RESULTS INTO A TABLE OR HTML

The parsed output can still be a little verbose and messy, especially if the input sequence has a large number of hits. Fortunately, for easy navigation of the search results, using a web browser, the Bio::Tools::Blast module also provides methods to create formatted output of filtered data in the form of a table or HTML. The code to achieve this is quite simple. To create a table, add the following line at the end of the program:

```
print $blastObj->table;
```


The output can then be redirected to a file:

```
C:\perl>parseblast4.pl -b d:\blast\hprout.txt -e 1e-60 > hpr_table.txt
```

The output is shown in Figure 11.7.

gi	accession	sp	db	length	e-value	ref	accession	length	e-value
gi 1350818 sp P07998 RNP_HUMAN	156	ref NP_002924.1	156	2.0e-074					
gi 1350818 sp P07998 RNP_HUMAN	156	pir NRHU1	156	7.0e-074	705				
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87050.1	AF449629_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87052.1	AF449631_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87049.1	AF449628_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87051.1	AF449630_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	emb CAA55817.1	152	3.0e-071					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87063.1	AF449642_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	pir I53530	153	3.0e-070	674				
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87058.1	AF449637_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87060.1	AF449639_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87059.1	AF449638_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	gb AAL87061.1	AF449640_1	156					
gi 1350818 sp P07998 RNP_HUMAN	156	emb CAA44718.1	127	4.0e-067					
gi 1350818 sp P07998 RNP_HUMAN	156	pdb 1E21 A	128	1.0e-066	643				

Fig. 11.7 Parsed results in table format

To create an HTML output, add this line at the end of the program:

```
$blastObj->to_html();
```

as shown in Listing 11.4.

Listing 11.4 Creating an HTML output of parse results

```
use Getopt::Long;
use Bio::Tools::Blast;
```

(Contd.)

(Contd.)

```
GetOptions("b|blastfile=s"=>\$blastfile, "e|evalue=f"=>\$evalue);  
sub filterBlast {  
    $hit=shift;  
    return ($hit->bits > 250);  
}  
%parameters = ( -file      => "$blastfile",  
                -parse     => 1,  
                -filt_func => \&filterBlast,  
                -signif    => "$evalue",  
                );  
$blastObj = Bio::Tools::Blast->new(%parameters);  
$blastObj->to_html();
```

Now, run the program and redirect the output to a file with a .html extension using the '>' operator. For example,

```
C:\perl>parseblast4.pl -b d:\blast\hprout.txt -e 1e-60 > hpr_blast.html
```

The output of parseblast4.pl is shown in Figure 11.8. Note the hyperlinks on the sequence identifiers and the E values that allow easy access to additional information about the hits.

Assignments

1. A recent microarray-based experiment on the analysis of DNA copy numbers in human breast cancers indicated that alterations in DNA copy numbers has a direct effect on deregulation of gene expression and may contribute to the development of cancer. The chip used for the study contained several thousand genes, many of which had unknown functions. A partial list of genes used in the study is provided in a file called hypothetical.txt. Create a pipeline that performs the following functions:
 - (a) Extracts only the hypothetical proteins from the list. How many hypothetical proteins are there in the file?

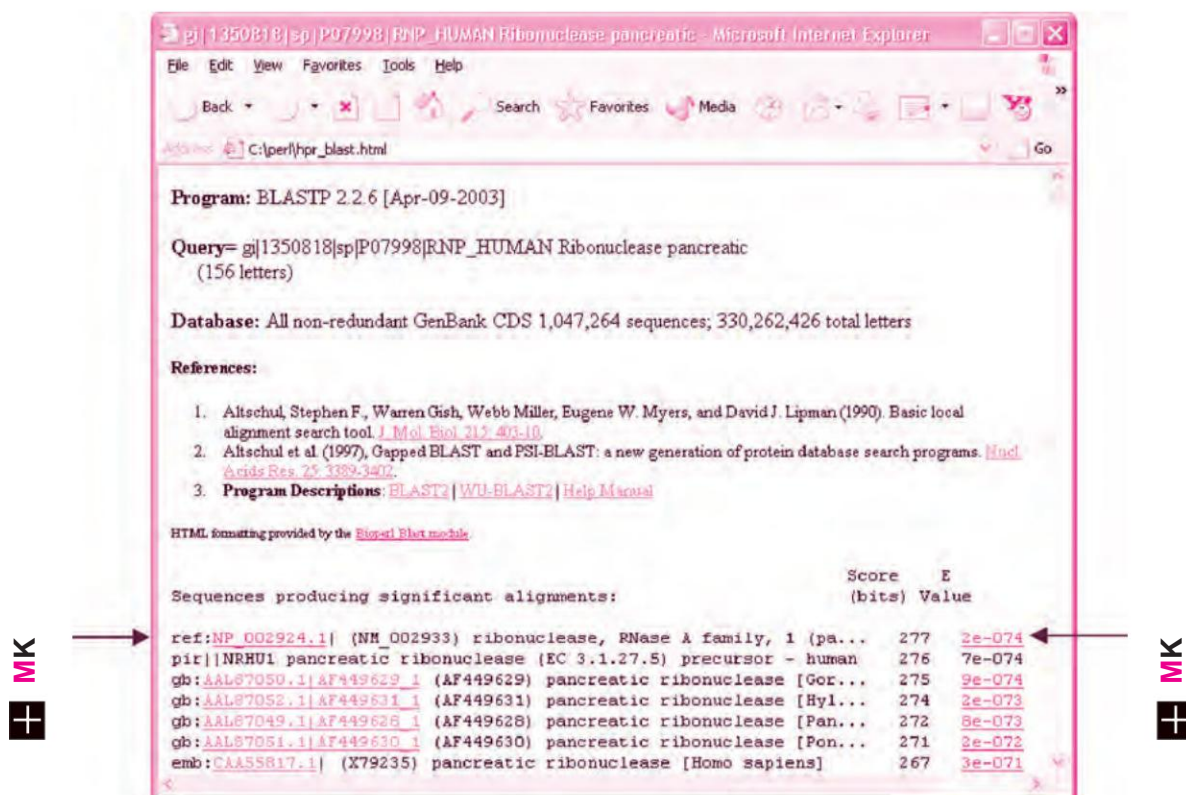


Fig. 11.8 HTML formatted parse results

- Performs a BLASTP for each sequence in an automated fashion against the nr database with an E value of 0.001.
 - Parses the top 10 hits and their associated HSPs. How many top hits are of human origin? How many are non-human origin?
- Use the information from Assignment 1 to arrive at an annotation of a plausible function (wherever possible) for each of the unknown proteins. Can you think of any obvious relationships to cancer for these genes?

Index



Ab initio, 75
Affine gap costs, 11
Alignment score, 11
Alistat, 104. *See* HMMER utilities

 Basic Local Alignment Search Tool (BLAST), 3, 175
 Bio::DB::GenBank, 149, 154
 Bio::module, 130
 Bio::Seq, 134, 155
Bio::Seq FeatureI. *See* Table 9.2
 Bio::SeqFeatureI, 161
 Bio::SeqI, 161
 Bio::SeqIO, 131, 137, 139, 161
 Bio::SimpleAlign, 134
 Bio::Species, 166. *See* Table 9.3
 Bio::Tools::Blast, 194, 196, 209
 Bio::Tools::Blast::HSP, 201
 Bio::Tools::Blast::Sbjct, 202
 Bio::Tools::Run::StandAloneBlast, 181, 187, 191
 Bio::Tools::Run::StandAloneBlast.pm, 194
 BioPerl, 127
 BLAST module, 175

installation, 127
 installing external modules, 147
 module
 Bio::SeqIO, 137
 modules, List. *See* Table 7.1
 testing availability, 148
 upgrading, 147
 using, 138
 Write_seq() function, 142
Bit score, 11
 BLAST, 3, 4, 9, 12, 23, 26, 29, 31, 38, 41, 44, 49, 55, 61, 62, 64, 65, 67, 114, 175, 183, 191, 194, 201, 204
 algorithm, 26
 analysis, 9
 Affine gap costs, 11
 Alignment score, 11
 Bit score, 11
 Expectation value, **E value**, 11
 Gap, 12
 Gap score, 12
 Gapped alignment, 12
 Global alignment, 12
 Heuristics, 12





High scoring pair, HSP, 12
 Local alignment, 12
 Maximal-scoring Segment Pair, 13
 Substitution Matrices, 12
 Automated alignments, 17
 Automating analysis, 41
 Bio::Tools::Blast, 196
 bl2seq, 63, 185
 BLAST programs. *See* Table 10.1
 BLAST2, 177
 blastall, 184
 BLASTALL, 49
 configuring, 49
 Blastclust, 185
 BLASTN, 28
 advanced, 35
 parameters, 30
 Pasting sequence, 30
 Submitting search, 31
 Blastpgp, 184
 comparison, PSI-BLAST v/s BLASTP, 110
 Database searches, 4
 databases, 10
 dbsts, 10
 downloading, pre-formatted, 57
 drosophila, 11
 ecoli, 11
 est, 10
 gss, 10
 mito, 10
 nr, 10
 pat, 10
 pdb. *See*
 yeast, 10
 E-value
 applying, 61
 fastacmd, 62
 Fastacmd, 185
 Formatdb, 185
 formatting database, 54
 formatdb, 54
 -i, 54
 -n, 54
 -o, 54
 -p, 54
 -s, 54
 -t, 54

Formatting results, 32
 generating raw report, 194
 Homologs, 8
 Identity, 5
 matrices, 23
 BLOSUM, 25
 Per cent Accepted Mutation, PAM, 24
 relationship-PAM and BLOSUM, 26
 Scoring matrices, 23
 substitution, 12
 Megablast, 184
 Orthologs, 8
 output
 List of significant alignments, 36
 Biological analysis, 40
 filter, 37
 filter, DUST, 37
 filter, SEG, 37
 graphical view of hits, 31
 Header, 31
 Mouse-over for information, 34
 Sequence alignments, 34
 output, explanation, 31
 output, pair-wise, 186
 Paralogs, 8
 parsing output, 194, 203
 formatting into HTML, 210
 formatting into table, 210
 Specifying filter function, 207
 parsing parameters. *See* Table 11.1
 performing local search, 188
 PERL
 Automated alignments, 17
 Automating analysis, 41
 local BLAST searches, 64
 programs
 BLAST2, 176
 BLASTN, 176
 BLASTP, 176
 BLASTX, 176
 TBLASTN, 176
 TBLASTX, 176
 PSI, 109
 query sequence, 4
 Rpsblast, 185
 Seedtop, 185
 sequence annotation, 65





Similarity, 5
 standalone BLAST, 44
Standalone BLAST
 instillation, 46
 programs installed, 48
 version, 45
 target sequences, 4
 word length, 38
 BLAST2, 176, 177
 Perl module, 178
 using BioPerl, 181
 blastall
 Checking results, 58
 command-line options, 57
 options. *See* Table 10.4
 running, 55
 BLASTALL, 49
 configuring, 185
 options. *See* Table 3.3
 -b, 57
 -d, 56
 -e, 56
 -F, 56
 -g, 57
 -I, 56
 -M, 57
 -o, 56
 -p, 56
 -q, 56
 -r, 57
 -T, 57
 -v, 57
 -V, 57
 running, 190
 BLASTN, 176
 BLASTP, 176
 BLASTX, 176
 BLOSUM62,, 12

 cladogram, 100
 ClustalW, 91, 96
ClustalX, 93
 hmm calibrate, 96
 interface, 97
 Multiple sequence alignment, 98
 Starting, 96

database
 formatting, 189
 databases, 10
 dbsts, 10
 drosophila, 11
 ecoli, 11
 est, 10
 gss, 10
 htgs, 10
 mito, 10
 month, 10
 nr, 10
 pat, 10
 pdb. *See*
 yeast, 10
 DNA, 70
 CpG islands, 72
 double-helix, 70
 Exons, 72
 Introns, 72
 RNA, 72
 structure, 70
 synthesis, 71
 Transcription, 72
 translation, 72

E value, 11

formatdb
 arguments. *See* Table 10.3

Gap, 12
Gap score, 12
Gapped alignment, 12
 GenBank, 149, 162
 accessing data, 161
 sequence feature of record, 163
 structure of records, 150
 tags, 161, 162
 extrating tag values, 164
 extrating tags, 164
 features, 162
 primary tag, 162
 sub-tags, 162
 Tag-value pairs, 163
 Gene prediction programs. *See* Table 4.1



- GenScan, 75
 corresponding coding sequences, CDS, 80
 entering identifier, 79
 output
 Abbreviations and explanations. *See* Table 4.4
 Header information, 81
 predicted sequences, 82
 output analysis, 78
 printing peptides, 80
 running analysis, 77
 setting parameters, 79
 uploading BAC sequences, 80
 web server, 78
 Getopt::Long, 86, 199
Global alignment, 12
- Heuristics**, 12
 Hidden Markov Models
 building alignment, 101
 running search, 103
 search results, 103
High scoring pair, 12
 HMMER, 89
 ClustalX installation, 93
 Downloading, 89
 ftp site, 90
 installation on DOS, 91
 running commands. *See* Table 5.1
 usage, 92
 utilities, 104
 Alistat, 104
 Seqstat, 104
 HMMs (Hidden Markov Models), 89
 HTTP::Request, 85, 179
 HTTP::Response, 179
- IO::Scalar, 147
 IO::String, 147
- Local alignment**, 12
 LWP::Simple, 17
 LWP::UserAgent, 84, 179
- Maximal-scoring Segment Pair**, 13
 method
 Ab initio, 75
 HMM method, 76
 homology based method, 76
 neural network method, 76
 all_seqFeatures(), 168
 alphabet(), 140
 as_string, 86
 Bio::DB::GenBank, 154
 Bio::Seq FeatureI. *See* Table 9.2
 Bio::SeqFeatureI, 161, 164
 Bio::SeqI, 161
 Bio::SeqIO, 161
 Bio::Tools::Blast, 199. *See* Table 11.3
 BioPerl, 194. *See* BioPerl Module, Table 7.1
 class methods, 136
 classification, 166
 data on HSPs
 Bio::Tools::Blast::HSP module, 201
 desc, 138
 extract top-level information on every hit, 199
 gene prediction, 75
 gene prediction, functional classification. *See* Table 4.2
 GenScan, 75
 get_Seq_by_acc, 158
 get_Seq_by_id, 155
 GRAIL, 75
 has_tag(), 169
 id, 138
 information on species
 Bio::Species module, 166
 instance method, 136
 moltype, 138
 new() class, 165
 obtain information on high scoring pairs (HSPs) for each hit, 200
 profile-based search, 110
 score based, 24
 seq, 138
 subseq, 158
 subseq(), 166
 UserAgent request(), 180
 module, 17
 Bio::DB::GenBank, 149
 Bio::module, 130
 Bio::SeqFeatureI, 164
 Bio::SeqIO, 137, 165

- Bio::Species, 166
- Bio::Tools::Blast, 194, 196
- Bio::Tools::Blast::HSP, 201
- Bio::Tools::Run::StandAloneBlast, 181, 187
- Bio::Tools::Run::StandAloneBlast.pm, 194
- BioPerl, 127, 133
- BioPerl BLAST, 175
- BioPerl modules. *See* Table 7.1
- Getopt::Long, 138, 199
- installing external, 147
- IO::Scalar, 147
- IO::String, 147
- LWP::Simple, 17
- LWP::UserAgent, 84, 179
- Perl for BLAST2, 178
- SeqIO.pm, 131
- OOP
 - class, 136
 - class constructor, 137
 - instantiation, 137
 - object, 136
 - properties of an object, 136
- OOP (object oriented programming), 136
- position specific scoring matrix (PSSM), 109
- Position-Specific Iterated (PSI), 109
- position-specific scores (PSS), 109
- PPM
 - invoking, 128
 - quitting, 131
- PPM (Programmer's Package Manager), 128
- PSI-BLAST, 109
 - advantages, 111
 - comparison, PSI-BLAST v/s BLASTP, 110
 - design, 110
 - iteration, 109, 110, 114, 116, 117, 118, 119, 120, 121, 122, 123
 - first iteration, 118
 - second iteration, 119
 - selecting hits, 118
 - third iteration, 121
 - limitations, 112
 - preparing sequence, 115
 - procedure, 111
 - Protein Analysis, 109
- query sequence, 4
- RNA, 72
 - Messenger RNA, mRNA, 72
 - ribosomal RNA, rRNA, 72
 - splicing, 73
 - structure, 72
 - transfer RNA, tRNA, 72
- Seqstat. *See* HMMER utilities
- Sequence annotation, 65
- Specifying filter function, 207
- Standalone BLAST
 - programs installed
 - bl2seq, 48
 - blastclust, 49
 - blastpgp, 48
 - fastacmd, 49
 - Megablast, 48
 - rpsblast, 49
 - seedtop, 49
- Standalone BLAST, 44, 183
 - BIO::TOOLS::RUN::STANDALONEBLAST, 187
 - instillation**, 46
 - programs installed
 - blastall, 48
 - version**, 45
- Substitution Matrices**, 12
- table**, 82, 94, 159, 173, 209, 210
- tags. *See* GenBank Tags
- target sequences, 4
- TBLASTN, 176
- TBLASTX, 176