Modeling and Optimization in Science and Technologies

Gautam B. Singh

Fundamentals of Bioinformatics and Computational Biology

Methods and Exercises in MATLAB

Springer

Modeling and Optimization in Science and Technologies

Volume 6

About this Series

The book series *Modeling and Optimization in Science and Technologies (MOST)* publishes basic principles as well as novel theories and methods in the fast-evolving field of modeling and optimization. Topics of interest include, but are not limited to: methods for analysis, design and control of complex systems, networks and machines; methods for analysis, visualization and management of large data sets; use of supercomputers for modeling complex systems; digital signal processing; molecular modeling; and tools and software solutions for different scientific and technological purposes. Special emphasis is given to publications discussing novel theories and practical solutions that, by overcoming the limitations of traditional methods, may successfully address modern scientific challenges, thus promoting scientific and technological progress. The series publishes monographs, contributed volumes and conference proceedings, as well as advanced textbooks. The main targets of the series are graduate students, researchers and professionals working at the forefront of their fields.

More information about this series at http://www.springer.com/series/10577

Gautam B. Singh

Fundamentals of Bioinformatics and Computational Biology

Methods and Exercises in MATLAB

Springer

Gautam B. Singh
Department of Computer Science
and Engineering
Oakland University
Rochester, Michigan
USA

ISSN 2196-7326 ISSN 2196-7334 (electronic)
ISBN 978-3-319-11402-6 ISBN 978-3-319-11403-3 (eBook)
DOI 10.1007/978-3-319-11403-3

Library of Congress Control Number: 2014949497

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To my family

Preface

The integration of computers in life sciences has been growing for the last two decades. While the first release of GenBank contained a mere half a million DNA sequence bases in 1982, the current release of GenBank has exceeded 100 giga bases of data. With data comes computational challenges for analysis, interpretation, visualization and integration of information. That in a nutshell is the reason to familiarize undergraduate students in computer science and engineering with the nature and use of biological data and thus become prepared to meet the demands of high tech careers in the twenty-first century.

The intended audience of this textbook are students in computer science, engineering and information technology at the undergraduate or lower graduate level. The material is primarily presented in a simplified manner and extensive details are left out. However, pointers to appropriate references should guide those who are interested in exploring specific topics in greater detail.

Topics in this textbook are organized into three parts. **Part I** of this book provides some background to the field of bioinformatics and an introduction to molecular biology and genetics. A survey of biological databases is also included. The material in this part is considered to be fairly fundamental and should be covered in all courses, graduate and undergraduate.

Part II of the book covers methodologies for retrieving information from biological databases and covers simple boolean searches, sequence alignment algorithms, protein alignment, scoring matrices, alignment tools and biolinguistic methods. Undergraduate students should cover basic retrieval techniques and advanced topics such as PAM and BLOSUM may be included based on the amount of time available and level of preparation of the students.

Part III of the book covers the topics related to sequence analysis and covers algorithms for finding patterns and detecting genes.

Part IV focuses on topics in phylogenetics and systems biology and covers the algorithms for distance, character and probabilistic methods for inferring

phylogeny. Also described are some key algorithms for analyzing micro-chip data.

The book is an offshoot of our project aimed at creating bioinformatics educational resources for undergraduates in computer science and engineering. This project is sponsored by the National Science Foundation, USA. Additional details for the project and bioinformatics educational resources are available from `http://bioflow.secs.oakland.edu`.

The author would like to acknowledge the efforts by students who participated in creating resources for the NSF sponsored bioinformatics project: Kenneth DeMonn, Nirmala Venkatraman, David Poe, Guy Lima, Kellie McGowan and Ashwin Kottam. Their work influenced the content and the presentation in this text.

For the *BioFlow* project we are also analyzing student learning styles in collaboration with Professor Christine Hansen, Department of Psychology, Oakland University. The results from obtained from the student assessment studies were very valuable in providing insight into methods that made this text more comprehensible for computer science and engineering undergraduates.

Rochester, MI
USA

Gautam B. Singh
June 2014

Contents

Part I
Background

Chapter 1
Introduction to Bioinformatics

Sequencing of biomolecules began in 1951 when Sanger and Tuppy deduced the thirty residue protein from the insulin B-chain. It was only after 25 years that *real* DNA sequencing methodologies were developed by Maxim & Gilbert and by Sanger *et al.* Today, we are sequencing tens of millions of bases of DNA sequences a year and undertaking the sequencing of genomes from whole organisms. During these times, the sequence databases have continued their exponential growth rate. The computational research in bioinformatics aims at enhancing the retrieval, analysis and interpretation of information that is embedded within the biological databases containing the DNA and protein sequences.

In a manner similar to the transformation of physics and chemistry, the study of biology has been undergoing a transformation since the 1990s. This transformation is aimed at the integration of computational sciences and information technology into the study of life sciences. This transformation has been driven by the computational requirements of genomic research. The experiment-rich field of biology has been generating data at an exponential rate, while there is a dearth of tools for information analysis and visualization. Furthermore, the challenges faced in bioinformatics stem largely from the fact that the languages and techniques utilized in the field of molecular biology are descriptive and experimental in nature, while the methodology in computer science and mathematics is generally based on analytical and precise formulations.

1.1 What Is Bioinformatics?

Bioinformatics is an emerging discipline that draws upon the strengths of computer sciences, mathematics, and information technology to determine and analyze genetic information. Bioinformatics leverage synergies between computational and biological sciences. Although the field of bioinformatics originally aimed at extracting information embedded within the 3 billion

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_1

bases of human DNA, the field has evolved to realize its capabilities for studying *information content* and *information flow* in biological systems and processes in general.

Earlier bioinformatics research aimed at mapping individual genomes and calculating differences to estimate population diversity. The study of bioinformatics now encompasses genomes from other species besides humans. For example, other genomes of interest are those of microbes, plants and fungi. An analysis of plant genome databases leads to advances in the agricultural arena by helping produce plants that are resistant to diseases and have higher yields. Understanding microbial genomics facilitates the development of new therapies for combating infectious diseases. Furthermore, computational analysis of fungal and microbial genomes serve as valuable smaller scale model organisms that lead to understanding functional genomics and improving technological development for applications at a larger "human" scale.

Originally, in the mid-1980s, the field of bioinformatics was defined as the *subject of genetic data collection, analysis and dissemination.* Bioinformatics has come a long way since then and now aims at complex mathematical modeling and simulation as well. For example, bioinformaticians now aim at developing computational models for differential regulation of gene expression that occurs *in-vivo* in a tissue-specific manner. Thus, the field of biology now turns towards mathematics and computation sciences to help make further advancements for understanding its core theoretical principles.

Such a dependence by biological science on the algorithms and formulations provided by analytical sciences was borne out of the necessity to analyze and make interpretations from the large volume of data generated by the Human Genome Project. Biologists look towards computational sciences to help bridge the gap between experimental observations and gaining an understanding of how living systems and processes perform their functions. Such a creation of novel biological *knowledge* in the case of diseases, for example, could lead to our predictive inference on the prognosis and preventive therapeutic treatment based on our understanding of diagnostics data through integrative bioinformatics models.

From an information technology perspective, therefore, bioinformatics may be defined as a scientific discipline encompassing acquisition, storage, processing, analysis, interpretation and visualization of biological information. It encompasses frameworks, theories, algorithms, techniques and tools from mathematics, computer science and biology with the aim of understanding the significance of a variety of biological data.

1.2 The Human Genome Project

Leveraging the technological advancements in molecular biology and genetics in the mid-1980s, the Human Genome Project was initiated with the goal of enabling progress and benefits in biomedicine. The two main goals of the

Human Genome Project were to identify all the approximately 25–30,000 human genes, and to determine the sequences of the 3 billion chemical base pairs that make up human DNA. Completed in 2003, the Human Genome Project (HGP) was a 13-year project coordinated by the U.S. Department of Energy and the National Institutes of Health. Coincidentally, the completion of the human DNA sequence in the spring of 2003 also marked the $50^{th.}$ anniversary of Watson and Crick's discovery of fundamental structure of DNA.

The HGP was truly an international effort, although significant advancements of its goals were accomplished in the United States. Other contributors included the Wellcome Trust (U.K.) who was major partner during the early years of its inception in the 1990s. Contributions to its advancements are also attributed to the commitments by countries such as Japan, France, Germany, China, and others. Being an international effort, the HGP was supported by the technological advancements in database and information retrieval with networking technologies that supported international collaborations.

The completion of the Human Genome Project was celebrated in April 2003 and also marks the completion of the sequencing of the human genome. However, this event also marked the beginning of the *post-genome informatics* era where scientists are collaboratively engaged in understanding the biological function in an integrative manner. As an illustration, consider the initial analysis of the draft human genome sequence that was published in 2001 by the International Human Genome Sequencing Consortium which estimated that the human genome contains only about 20,000 to 30,000 protein-coding genes, an estimate that was significantly lower than previous estimates of around 100,000.

This lower estimate came as a shock to many scientists because counting genes was viewed as a way of quantifying genetic complexity. With around 25,000, the human gene count would be only about 25% greater than that of the simple roundworm *C. elegans* at about 20,000 genes. Thus, science today is embarking upon unraveling the complexity in coordination and regulation of genetic networks in contrast to the number of genes a being the determining factor in an organism's complexity.

Thus, an understanding of biological function is not possible using the sequence information alone. Complete understanding of life's complex processes often necessitates a convergence of computational modeling and experimental sciences. This, incidentally is also the recipe for effective bioinformatics research.

1.3 Genome Data Statistics

The National Center for Biotechnology Information, or the NCBI, is part of the National Library of Medicine. The National Library of Medicine is a component of the National Institutes of Health and the U.S. Department of Health and Human Services. The NCBI was established in 1988 as a national

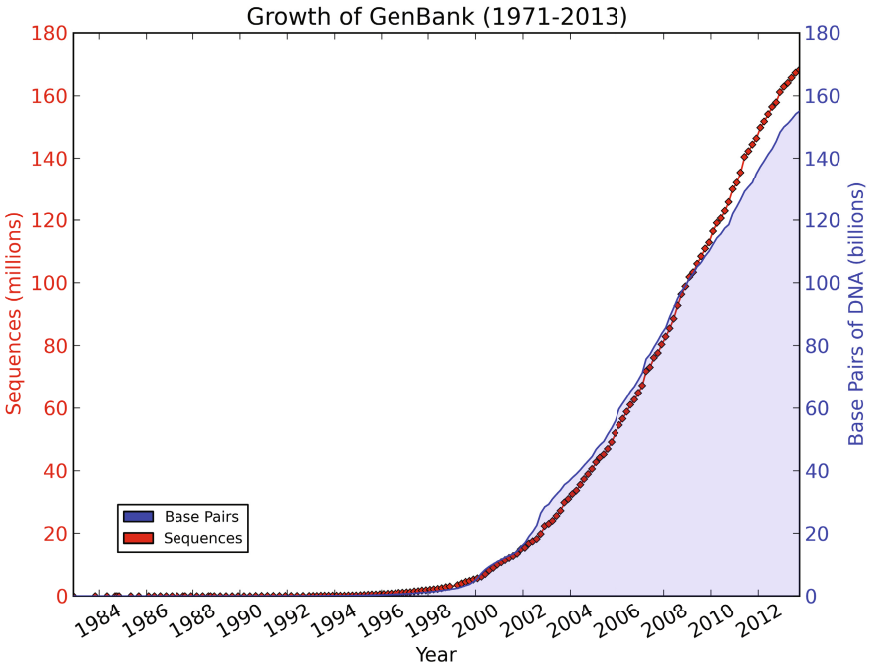


Fig. 1.1 The growth of the nucleotide database GENBANK since its inception till March 2006. The size of the database as well as the number of sequences deposited has been undergoing an exponential growth ever since GENBANK's inception. *Source of data:* `http://www.ncbi.nlm.nih.gov/GenBank/genbankstats.html`

resource for molecular biology information. NCBI creates public databases and develops software tools for analyzing genome data.

GenBank is the NIH genetic sequence database, an annotated collection of all publicly available DNA sequences. NCBI is host to the GenBank nucleotide sequence database. Submitters to GenBank contribute over 3-4 million new DNA sequences per month to the database. As of 2013, there are over 150 billion bases in approximately 170 million sequence records in the various GenBank dataset divisions. The data represents both individual genes and partial and complete genomes of over 165,000 organisms. GenBank provides sequences from single genes from organisms as diverse as humans, elephants, earthworms, fruit-flies, apple trees, and bacteria as well as the sequences for organisms' complete genomes.

As illustrated in Fig. 1.1 the total database size and the number of sequences in the nucleotide database GenBank has been growing exponentially. This growth rate has continued despite the completion of the genome project and is fueled by scientists' interest in continuing to sequence other organisms and plants and mankind's interest in biological diversity.

1.4 Applications of Bioinformatics

There are many applications of the burgeoning field of bioinformatics. Bioinformatics is not limited in its applications and in general can be applied to *any* computational inquiry for solving biological problems. Some of the common applications of bioinformatics are listed below. However, it must be emphasized that the list below is by no means exhaustive. At its core of bioinformatics is an application of computational and mathematical problem solving algorithms to further the understanding of a biological system. Thus, the applications of bioinformatics is really an open ended list.

- **Pattern Discovery:** The need to discover patterns stems from the observation that whenever nature finds a mechanism (the evolution of the eye for example) which bestows a differential fitness to an organism, the mechanism is often reused. Such a reuse of successful recipes also occurs at the molecular level implying that biological sequences belonging to distant species share common patterns in their genetic makeup. Through computational analysis and modeling, bioinformatics algorithms help in discovery and functional interpretation of these biological patterns.
- **Protein Folding:** After a gene has been transcribed and translated, the linear polypeptide chain folds into a three dimensional protein within a matter of seconds or minutes. The protein can only function after it has acquired a three dimensional structure as its interactions with other molecules is largely governed by the protein's shape. Even after a concerted effort over the past few decades, computational algorithms for predicting the protein structure from a protein sequence continues to be an unsolved problem. A solution to this problem will enable scientists to accurately model biochemical pathways and lead to effective drug design *in-silico* without the need for extensive experimentation.
- **Alignment and Homology:** Sequence alignment is a methodology for arranging biological sequences to identify regions of similarity between them. The extent of similarity, or homology, between the DNA from a variety of organism may be used to determine evolutionary relationships and degrees of divergence between them. The type of ancestry would let us establish if a certain structures observed evolved from some structure in a common ancestor (such as arms in humans and wings on bats) and also infer function of an anonymous sequence based on the known function of a homologous sequence. There are a number of computational challenges in developing newer and integrative homology algorithms that can infer homology using data from sequence and structure.
- **Orthologs and Paralogs:** Homologous features share an evolutionary history. Orthologs and paralogs are both homologs, but have more specific meanings: Orthologs have diverged because of a speciation event such as a gene like *Antp* in fruit fly *D. melanogaster* diverged from the gene *mab-5* in roundworm *C. elegans* – that is these two genes evolved from the same gene in their common ancestor. In contrast, paralogs have diverged

because of a gene duplication event in an ancient ancestor as is probably the case with *mab-5* and *lin-39* in *C. elegans*. Also consider the example of the gamma-globin genes in the Anthropoidea which duplicated before the new world monkeys diverged from the lineage to humans and apes. That would make gamma-1-globin and gamma-2-globin paralogous. However, the gamma-1 in humans is orthologous to gamma-1 in chimpanzees. Thus, it is sometimes difficult to distinguish orthologs from paralogs. Sophisticated computational models are needed to effectively develop taxonomies and gene trees and discover novel instances of convergent evolution.

- **Information Retrieval and Data Mining from Biological Databases:** The explosive growth of sequence and biological information has created new challenges for data representation, access, and analysis. With the size and the need for access to this data continually increasing, maintaining database performance and availability is a challenging task. In-silico biology, is a term that life science companies use to refer to the computational tools that translate raw experimental data into workable models or simulations to identify targets for drug development. Bioinformatics tools for processing the overwhelming amount of data gathered through genome research, such as the Human Genome Project, are essential to fuel the *in-silico* life science research. Genome database searching entails developing computational tools for identification of protein-encoding regions of a genome and assign functions to these genes on the basis of sequence similarity with other genes of known function.
- **Data Integration from Multiple Modalities:** Researchers in molecular biology and medicine rely heavily on progress in data management and quality assurance as the necessary underpinnings for effectuating progress. Advancements in life sciences, and particularly in areas like drug development, systems biology, or personalized medicine, are dependent on integration of data from myriad experiments, longitudinal studies, and levels of detail. Numerous unsolved problems, for example the integration of data from proteomics mass spectroscopy and gene sequences, requires fulfilling a vast information gap-filling through data rationalization. These issues are moving towards the forefront as advancements in instrumentation is rapidly generating larger quantities of data in a high throughput manner. The next big challenge is expected to be the integration of genomics, proteomics and individualized medical data routinely collected by hospitals.
- **Analysis of Biological Sequences and Pattern Discovery:** A good understanding of the probabilistic nature of biological sequences is the key to statistical modeling of biological sequences. Genome sequence data analysis leads to the design of novel tools for effective prediction of biological function. Sequence analysis research focuses on finding patterns in biological sequences and associating these patterns with functions necessary for pathways that support life forms. Some of the pattern detection encompasses looking for short range patterns such as binding and initiation sites, while other computational techniques seek to model long range

patterns such as genes, locus control regions, matrix attachment regions, and helitrons where a consensus sequence or even a consensus structure of the model is not yet known. Inductive learning approaches are needed to analyze known data and induce mathematical models that learn as new discoveries are made.

- **Micro-arrays and Differential Gene Expression:** The thousands of genes and their products expressed in a given living organism function in a coordinated and complex manner. The discovery of genes has been followed by various techniques for detecting genes based on their expression levels as gene expression is correlated with the tissue type. For example, the genes that are expressed in the liver are not the same set of genes expressed in kidneys, with the exception of certain housekeeping genes required for processes common to all cells that are expressed in all cells. Methods of identifying differential expression in genes have been developed to establish the levels of gene expression in various tissue types. This can be particularly useful in cancer studies, where mutations that can amplify or turn off gene expression occur in malignant samples. These days the gene chip or microarray technology is greatly accelerated the speed of performing a differential gene expression analysis. With gene chips one can monitor the whole genome using a single chip allowing researchers to better understand the interactions among thousands of genes simultaneously. Among the many applications of micro-arrays or gene chips are gene discovery, disease diagnosis, drug discovery, and toxicology research. Gene expression studies using gene chips draw upon the advances in computational analysis and statistics, image processing and data management.
- **Gene Regulatory Networks:** Gene regulatory networks (GRNs) are the on-off switches that act like rheostats and control the level of expression for each gene by controlling whether and to what level the gene will be transcribed into RNA. These networks are a collection of DNA segments that interact with each other and other substances in the cell and govern the overall rate at which the network is transcribed into mRNA. In a graph theoretic model, the nodes of a GRN are proteins and edges represent individual molecular reactions or protein interactions. Edges may have arrowheads and thus be inductive where the increase in the concentration of one leads to the interaction of the other, or inhibitory (with a circle) where the increase in one leads to the decrease in the other. GRNs thus capture the chemical dynamics of the cell. Construction and simulation of GRNs offer many challenges in mathematical modeling, simulation and visualization.
- **Metabolic Pathway Models:** Metabolic network reconstruction and simulation enables us to gain insight into molecular mechanisms that relate the genome to molecular physiology. The metabolic pathway breaks down a metabolic cycle such as glycolysis, Krebs cycle, or pentose phosphate pathway into their respective reactions and enzymes and analyzes their interactions. Enzymes and genes are correlated by searching genomic

databases. Continual validation of metabolic pathways is needed to keep the pathway database consistent.

Further Readings

1. Searls, D.B.: An online bioinformatics curriculum. PLoS Comput. Biol. 8(9), e1002632 (2012)
2. Maojo, V., Kulikowski, C.A.: Victor Maojo and Casimir A Kulikowski. Bioinformatics and medical informatics: collaborations on the road to genomic medicine? J. Am. Med. Inform. Assoc. 10(6), 515–522 (2003)
3. Li, J., Doyle, M.A., Saeed, I., Wong, S.Q., Mar, V., Goode, D.L., Caramia, F., Doig, K., Ryland, G.L., Thompson, E.R., Hunter, S.M., Halgamuge, S.K., Ellul, J., Dobrovic, A., Campbell, I.G., Papenfuss, A.T., McArthur, G.A., Tothill, R.W.: Bioinformatics pipelines for targeted resequencing and whole-exome sequencing of human and mouse genomes: a virtual appliance approach for instant deployment. PLoS One 9(4), e95217 (2014)
4. Hartwell, L.H., Hopfield, J.J., Leibler, S., Murray, A.W.: From molecular to modular cell biology. Nature 402(6761 suppl.), C47–C52 (1999)
5. Csete, M.E., Doyle, J.C.: Reverse engineering of biological complexity. Science 295(5560), 1664–1669 (2002)
6. Ouzounis, C.A.: Rise and demise of bioinformatics? promise and progress. PLoS Comput. Biol. 8(4), e1002487 (2012)
7. Dymond, J.S., Scheifele, L.Z., Richardson, S., Lee, P., Chandrasegaran, S., Bader, J.S., Boeke, J.D.: Teaching synthetic biology, bioinformatics and engineering to undergraduates: the interdisciplinary build-a-genome course. Genetics 181(1), 13–21 (2009)

Chapter 2
Introduction to Molecular Biology

Molecular biology overlaps the fields of biology and chemistry and mainly aims at developing an understanding of the interactions between the various systems of a cell, including the interrelationship of DNA, RNA and protein synthesis as well as with uncovering the manner in which these interactions are regulated.

Researchers in molecular biology use specific techniques native to molecular biology. However, there is considerable diffusion of ideas from other disciplines such as genetics and biochemistry; there does not seem to be a clear boundary that delineates these fields anymore as the researchers borrow techniques and methodologies from all these related fields.

Biochemistry is defined to be the study of the chemical substances and vital processes occurring in living organisms. Genetics concerns itself with the effect of genetic differences on organisms which often is associated with the absence of a genes as in the study of "mutants." Mutants are organisms which lack one or more functional genes with respect to the so-called "wild type."

Molecular biology synthesizes the above viewpoints by studying the molecular underpinnings of the processes related to genetics. Specifically, those related to the replication, transcription and translation of the genetic material – the so called central dogma of molecular biology discussed later in this chapter.

2.1 Cell Structure

Every cell typically contains hundreds of different kinds of macromolecules that function together to generate the behavior of the cell. Computer graphic of a typical animal cell and its contents or the organelles is shown in Fig. 2.1.

The big and round structure is the nucleus, which carries the cell's genes in the form of DNA and controls cellular activities via genes. The nucleolus is located within the nucleus and is the site for ribosome synthesis. The oval

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_2

bodies are mitochondria which are responsible for the production of ATP through the oxidation of carbohydrates and provide the cell with energy.

The folded, dotted structures are rough endoplasmic reticulum where the folds of membrane carry ribosomes that synthesize proteins. The smooth endoplasmic reticulum is involved in the lipid synthesis. The round bodies containing small particles are vesicles are the lysosomes or peroxisome. Lysosome contains hydrolytic enzymes for intracellular ingestion, while peroxisome is responsible for hydrogen peroxide synthesis and degradation and expelling some of the matter through the cell's outer plasma membrane. The vesicles are made by the golgi apparatus which is the folded structure is the packaging center and the site for manufacture of carbohydrates.

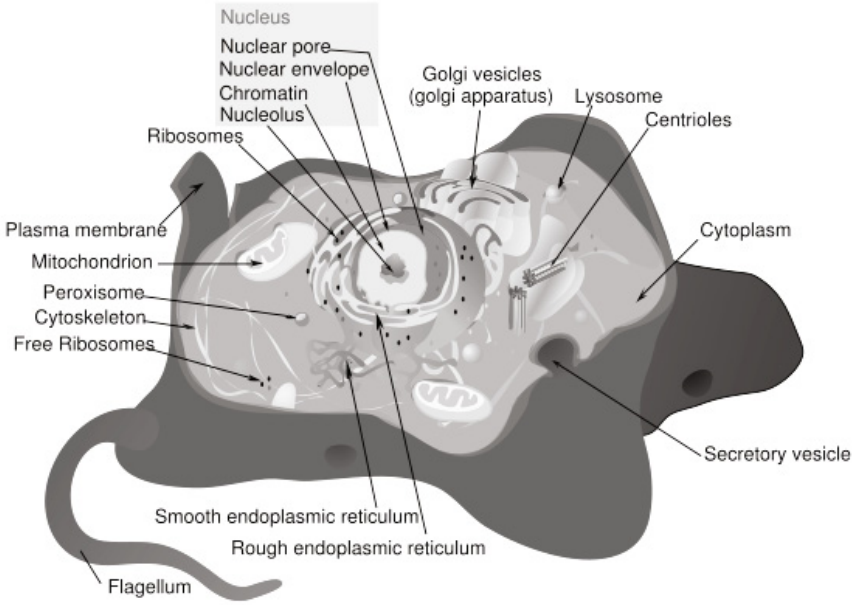


Fig. 2.1 The structure of an animal cell

Most proteins are synthesized by ribosomes in the cytoplasm. This process is also known as protein biosynthesis or simply protein translation. Some proteins, such as those to be incorporated in membranes (membrane proteins), are transported into the ER during synthesis and are also processed further in the golgi apparatus.

2.1.1 Genome

The term genome was coined by Hans Winkler, a Professor of Botany at the University of Hamburg, in 1920 to collectively refer to the complete genetic material of an organism. The genome contains an organism hereditary information encoded in the DNA and packaged into the chromosome. The genome encompasses both the genes and the non-coding sequences of the DNA. More precisely, the genome of an organism is a complete DNA sequence of one set of chromosomes; for example, one of the two sets that a diploid individual carries in every somatic cell.

The term is sometimes qualified in a manner to refer to the complete genetic material is in *nuclear genome* might be used to refer to the complete set of nuclear DNA, can be applied to organelles such as *mitochondria genome* or *chloroplast genome* to refer to the complete contents of the DNA found in mitochondria or chloroplast respectively.

Every somatic cell contains two copies of each chromosome where one is inherited from each parent. Each pair of chromosome is often referred to as a homologous pair where both the chromosomes contain a gene for the same trait at exactly the same loci. A given location in the homologous chromosome pair thus contains a gene pair representing the two *alleles* for that gene with each allele originating from the two parents. For example, one allele in a fruit fly may code for the flies to have long wings, while the other allele may code for flies to be wingless. Alternatively, both the alleles may code for long wings, or may code for no wings at all. Where the two alleles are different the genotype or the genetic makeup is referred to as *heterozygous* while the genotype is referred to as *homozygous* when the two alleles are the same.

The phenotype is the actual expression of the genotype. Purple flowers, brown eyes, or wingless fruit flies are all examples of phenotypes. In a heterozygous individual, the phenotype of the organism is determined by which of the allele is dominant. In the example, whether a heterozygous fruit fly has long wings or no wings depends upon which of the two alleles, the one for long wings or the one for no wings, is dominant. The trait of the dominant allele is expressed, while that of the recessive allele is suppressed. For a homozygote, the expression or the phenotype is the same as that coded by the two identical alleles.

The biological information contained in a genome is encoded in its deoxyribonucleic acid (DNA). The DNA is a macromolecule that contains discrete units of protein coding segments called the genes. Current estimates place the number of genes in the humans to be around 25,000 encoded in on 23 chromosomes. The total length of the human nuclear genome is 3×10^9 base pairs. Note that the length of DNA only considers the of one of the homologous chromosome pairs.

The chromatin, the constituent DNA in a chromosome, becomes observable under a microscope when it is packed in the shape shown in Fig. 2.2 during the metaphase of a cell division or mitosis. While the diameter of a strand

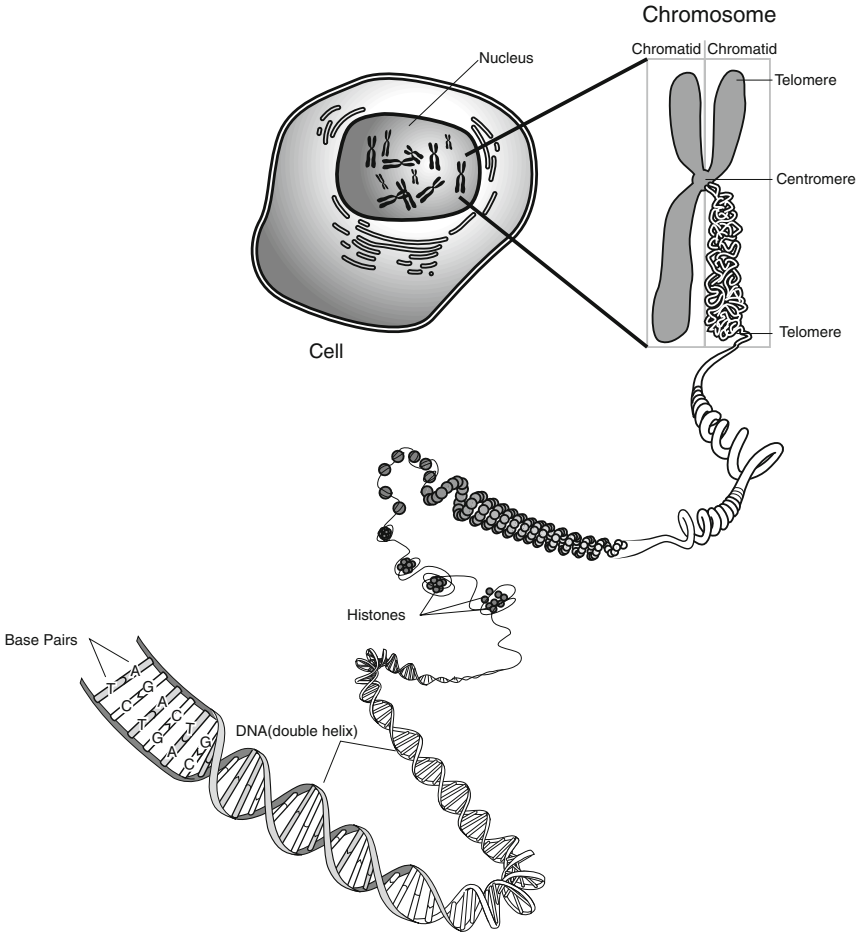


Fig. 2.2 Each chromosome contains a single molecule of DNA organized into several orders of packaging. The packaging of a chromosomes in metaphase is shown where the compact packaging causing length of the chromosome to be about 0.0001 times the length of the DNA molecule. *Reprinted from National Human Genome Research Institute's Educational Resources.*

of DNA is only 2 nm, several levels of packaging facilitated through the packaging proteins called the histones, the chromosomes themselves have a diameter of 1400 nm or 1.4 microns. In order for a gene to be expressed, the loci of the genome containing the gene must be in an open conformation and not tightly packaged as in a metaphase chromosome.

2.1.2 DNA: Deoxyribonucleic Acid

The DNA molecules consist of two complementary chains that are twisted together to form a double helix. The DNA molecule is comprised of four nucleotide bases belonging to two classes. These four bases are adenine (A), guanine (G), cytosine (C) and Thymine (T). The two classes to which these bases belong are purine and pyrimidine. The bases A and G belong to the class purine while C and T belong to the class pyrimidine. The DNA double helical structure is formed by the hydrogen bonding between purines and pyrimidines. The DNA is often thought of as a spiral staircase where the sides of this ladder consist of deoxyribose residues linked together with phosphate bonds and the "rungs" of the ladder are made up of a hydrogen bonded purine and pyrimidine pair.

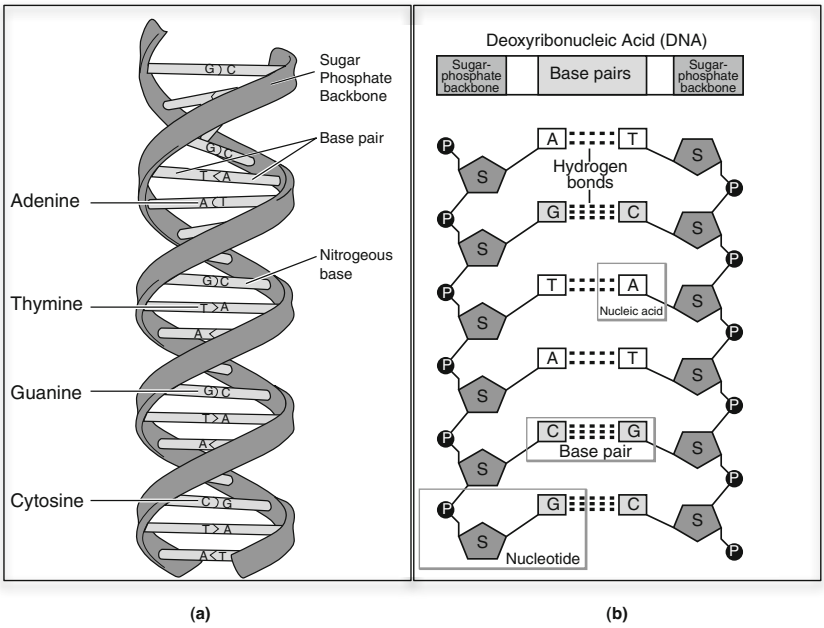


Fig. 2.3 DNA molecule and nucleotide bases. (a) Cytosine (G) always pairs with Guanine (G) while Adenine (A) always pairs with Thymine (T). (b) Triple hydrogen bonds are formed between every C:G pairings while two hydrogen bond exists between each A:T pairs. *Reprinted from National Human Genome Research Institute's Educational Resources.*

Fig. 2.3 shows the structure of a DNA molecule. The DNA is a double stranded molecule with base C pairing with the base G and base A pairing with T. The pairing is not chemical in nature but rather mediated with

hydrogen bonding. This enables the DNA to adopt a single stranded conformation with relative ease as is needed during the cellular processes leading to replication and transcription. The bonding between bases C and G is stronger as characterized by a triple hydrogen bond between this pair. The bases A and T on the other hand are paired using a double hydrogen bond. Generally, due to the C:G bond being stronger, there is ample evidence to suggest that genes are located in the CG rich areas of the genome.

The DNA is read from the direction of 5' (*five-prime*) to 3' (*three-prime*). These labels are indicative of the free carbon atom on the sugar phosphate backbone of the DNA molecule. The 5' carbon on the reverse, or complementary strad is located directly opposite from the nucleotide base attached to the 3' end of the forward strand and vice versa. Further, as the DNA occurs in a double stranded conformation, the length along the molecule is measured in units of *base pairs* or **bp**.

Example 2.1

Consider a 10-bp DNA sequence `TAAGCCTGTA`. WIthout more information we will assume that the sequence provided is for a 5' to 3' read for forward strand. This corresponds to a left to right read. The forward and reverse strands would be shown as follows:

```
5'     forward strand      3'
T  A  A  G  C  C  T  G  T  A
|  |  |  |  |  |  |  |  |  |
A  T  T  C  G  G  A  C  A  T
3'     reverse strand      5'
```

The reverse strand, also read from the 5' to 3' direction, will thus be read from right to left. This would correspond to the sequence `TACAGGCTTA`.

End of Example

2.1.3 Genes

Genes are discrete functional units located on the genome. A gene codes for a protein; that is, a gene is comprised of the blueprint of how a particular protein is to be synthesized. This blueprint is written in an alphabet comprised of the four characters {A,C,T,G}. Only about 1–2% of the genome codes for the approximately 20,000–25,000 genes in the human genome. The percentage of coding regions in other eukaryotes[1] is comparable.

The function of the remainder of the DNA is relatively unknown but is generally believed to be associated with differential regulation of gene expression

[1] Organisms where the DNA is packaged inside the nucleus are called eukaryotes, while DNA floats freely in the cytoplasm in a prokaryote. Higher level life forms, such as mammals, plants, reptiles and fungi are eukaryotes. Lower level life forms such as bacteria are prokaryotes.

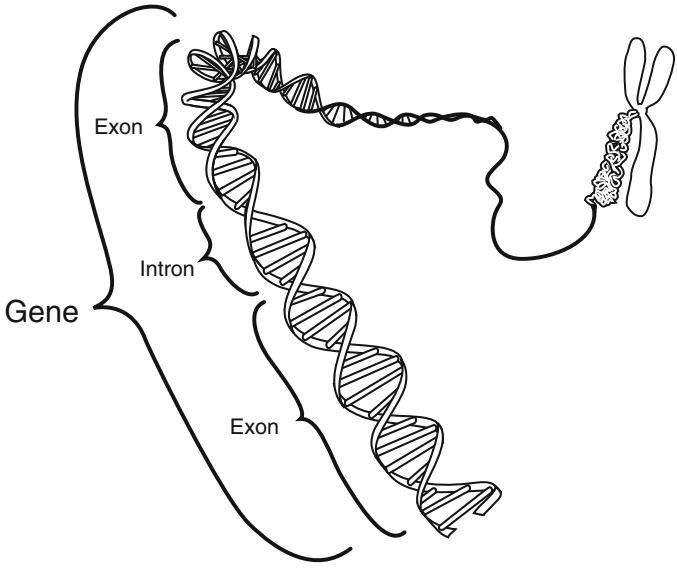


Fig. 2.4 A eukaryotic gene is comprised of protein coding *exons* interspersed by non-coding intervening sequences called *introns. Reprinted from National Human Genome Research Institute's Educational Resources.*

and programmed cell death. As shown in Fig. 2.4, the genes in eukaryotes are organized into intervening regions or introns that do not code for protein. As is further explained in Section 2.2.2, the protein sequence is derived by splicing out the intronic regions before the mature RNA transcript is synthesized in the cytoplasm. The number and size of introns vary in different genes. Albeit the introns do not code for proteins, their number and location is often indicative of evolutionary relationships between specific genes. For example, all apolipoprotein genes, affecting lipoprotein metabolism, have a similar number and size of introns and share a common ancestry.

The organization of a eukaryote gene is shown in Fig. 2.5. The boundaries between the introns and exons are generally well defined. All introns have a dinucleotide `GT` at their 5' end and the dinucleotide `AT` at their 3' end. The *open reading frame* (ORF) is a contiguous sequence of bases on the chromosome that could code for a protein.

The open reading frames or ORFs begin with the trinucleotide `ATG` which is considered as the universal *start codon* specifying the initiation of protein synthesis and is located on the 5' end of a gene. The 3' end of an open reading frame is a *stop codon* which could be a `TAA`, `TAG` or `TGA` and marks the termination of protein synthesis. Thus, a gene is demarcated by a start and stop codon.

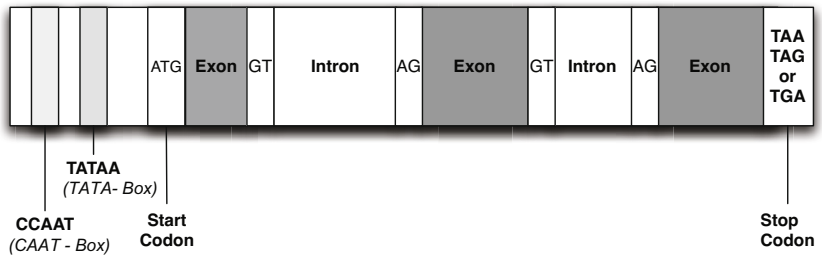


Fig. 2.5 The blueprint of an eukaryotic gene. The gene is demarcated by a start and stop codon. Upstream from the start codon are the CAAT and TATA blocks which facilitate in the process of gene expression. In eukaryotes, the genes are characterized by the presence of intervening sequences that are spliced out of the primary transcript which is capped with a poly-A tail to synthesize the mature RNA which is ultimately used in the synthesis of proteins in the a cell's cytoplasm.

Upstream from the start codon are the CAAT and TATA boxes which facilitate in the process of gene expression. The TATA-boxes are found in most genes and are located about 20–30 base pairs upstream of the transcription start codon `ATG`. The TATA-boxes are thought to direct transcriptional enzymes to the appropriate site for initiating transcription. The CAAT-boxes occur about 70–90 bases upstream of the start codon and are also thought to regulate and facilitate transcription.

2.2 Central Dogma

The central dogma of molecular biology was first proffered by Francis Crick in 1958. Given the three types of information carrying biopolymer sequences, namely the DNA, the RNA and the proteins, the central dogma limits the information transfer capability of a given type of biopolymer to another. It essentially states that once the information transfer has occurred to its final state, the information transfer may not be reverted back.

The final state of information flow is the *protein* as shown in Fig. 2.6. Thus, after a protein has been formed using the information stored on the DNA, the DNA information is not recoverable from protein. Neither may the information from protein be transferred to RNA or to other proteins.[2] In this manner, the central dogma of molecular biology concerns itself with the detailed residue-by-residue transfer of sequential information and represents a framework for understanding the transfer of sequence information between biopolymers.

As illustrated in Fig. 2.6, the flow of information occurs from a DNA to DNA molecule during *replication* where a template strand DNA is used to

[2] Except in the very rare cases where the information from a prion may be transferred to other proteins.

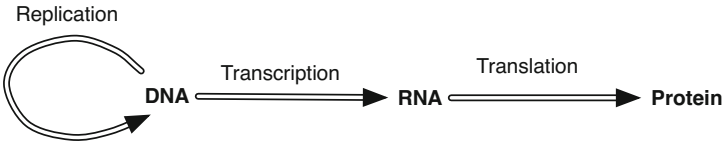


Fig. 2.6 Central Dogma. DNA replicates during cell division where the information on a strand of DNA is used to synthesize a complementary strand. Information on a strand of DNA is transferred to RNA during transcription. The information encoded in a RNA molecule is translated by the ribosome to synthesize a poly-peptide chain which folds into a protein.

synthesize the complementary strand with the help of the enzyme DNA polymerase. The flow of information from DNA to RNA occurs during *transcription* when the nuclear enzyme RNA polymerase synthesizes messenger RNA from the coding strand of the double helix (DNA). The single stranded RNA molecule provides the template utilized by the ribosomes for the *translation* where the chain of amino acids representing the coded protein is synthesized.

2.2.1 Replication

The replication of DNA precedes cell division. An exact copy of DNA must be made to pass on genetic information to the progenitor daughter cells. The DNA must be replicated faithfully and any errors in the replication process represent genetic mutations. Replication is initiated by the enzyme *helicase* which unwinds the superhelix and splits the two strands apart at the AT-rich origin of replication regions. With the two strands exposed, the sequence of bases on each of the separated strands serves as a template for synthesizing complementary set of bases by enzymes such as DNA polymerase.

In human DNA, the copy rate of DNA can reach around 80 bases per second, while some prokayotes are able to replicate DNA at the rate of 1000 bases per second. The missing molecules in of the ribose-phosphate backbone are added by the enzyme ligase. The replication process consumes the original strands. As shown in Fig. 2.7 (a), this results in the creation of two identical strands of DNA where the two new double-stranded molecules created are each comprised of one original strand paired with the other newly-synthesized strand. Thus, DNA replication is defined as a *semi-conservative* process.

Since the average human chromosome contains about 150 million nucleotides, the process of copying a single chromosome would take about a month. However, the process actually takes close to an hour since there are many places on the chromosome where replication simultaneously begins. As illustrated in Fig. 2.7 (b), the synthesized DNA around the replication origins creates “replication bubbles” of that finally fuse and form two new molecules.

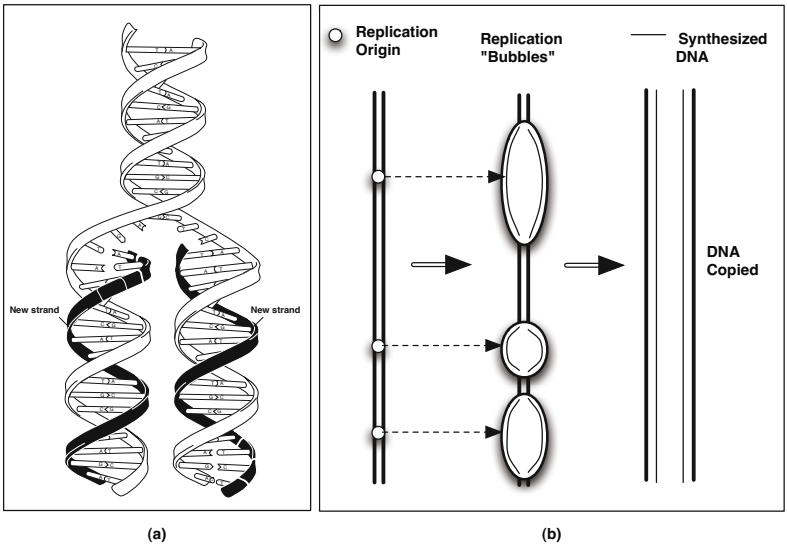


Fig. 2.7 Replication of the DNA. (a) Dark strand shown is the copy resulting from the replication process. *Reprinted from National Human Genome Research Institute's Educational Resources.* (b) Simultaneous initiation of replication at AT-rich replication origins grow into bubbles that eventually merge to create a copy of DNA.

The completion of DNA replication is the pre-requisite for the two types of cell divisions, namely mitosis and meiosis. Each type of cell division is preceded by a DNA synthesis phase where a sister chromatid is formed. The sister chromatid for each chromosome in the homologous pair is attached at the centromere. Thus, if sister chromatids are also counted, the total number of chromosomes in a cell right before division is $4 \cdot N$ after each of the $2 \cdot N$ chromosome is replicated during the synthesis phase preceding mitotic or meiotic cell division. The mitotic cell division produces two daughter cells each with $2 \cdot N$ chromosomes while a meiotic cell division produces four daughter cells each with N chromosomes.

Cell Division - Mitosis

In mitotic cell division, each cell with $2 \cdot N$ chromosomes produces two daughter cells with $2 \cdot N$ chromosomes each. Prior to the division, the parent cell undergoes DNA synthesis, where the cell's genetic material is duplicated in anticipation of mitotic division, often referred to as *binary fission*. The entire cell cycle is often divided into four stages: G_1, S, G_2 and *mitosis*. The G_1 phase, or the first growth phase, immediately follows mitosis with the cell's DNA present in an unreplicated state. G_1 is followed by S or the synthesis phase where the genetic material is replicated in the semi-conservative

manner discussed above and where each chromosome is doubled to form its sister chromatid with the centromere serving as the point of attachment. This is followed by another growth phase called G_2 which lasts till the mitotic division ensues. The stages of mitosis, commenced after G_2 phase, are shown in Fig. 2.8.

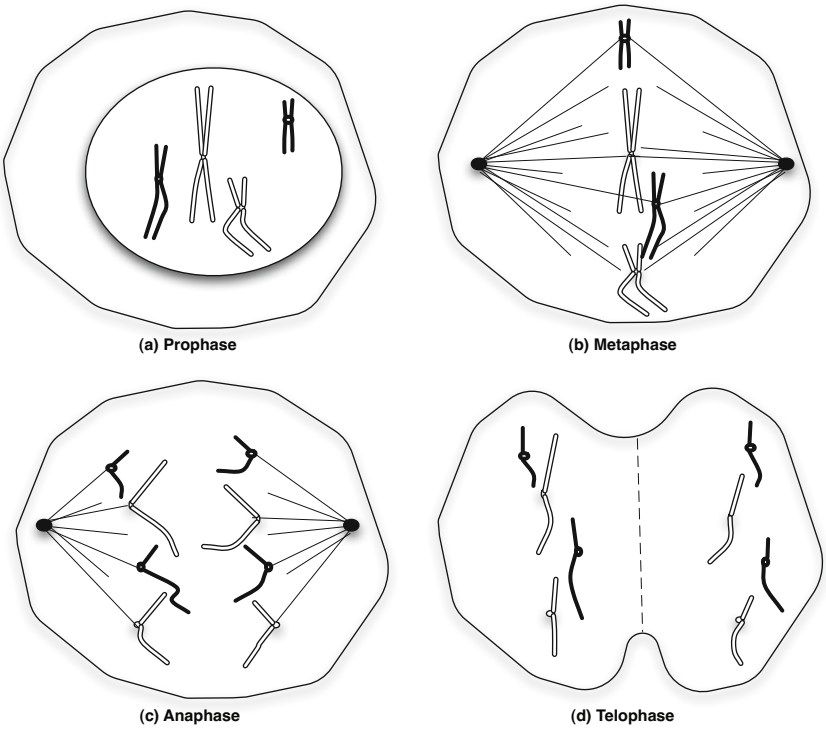


Fig. 2.8 Mitosis. The cell division produces diploid cells. The DNA in the cell must be replicated before the cell division can occur. The DNA replication results in the synthesis of the sister chromatid shown for each of the chromosome. All chromatid pairs are split into two diploid daughter cells.

During mitosis prophase, the chromosomes condense and become visible under a light microscope with each chromosome consisting of parallel strands called the sister chromatids held together by the centromere. During prophase, the nuclear membrane disappears. During metaphase, the chromosomes contract and move towards the center of the cell and spindle fibers form and extend from the centromere to the centrioles located at the opposite poles of the cell. During the anaphase, each of the chromosomes held together at the centromere divide and move to the opposite poles of the cell. Finally, in telophase, the cytoplasm divides and nuclear membrane forms around the

chromosome which begin to de-condense again. The end of mitosis marks the beginning of the interphase and growth phase for the two daughter cells.

In a mitosis cell division diploid daughter cells are produced. The diploid daughter cells each have *2N* chromosomes corresponding to the homologous pair derived from each parent. The mitotic division is followed, in case the daughter cells need to further divide, by the growth and synthesis phase where the DNA is again replicated in the daughter cells.

Cell Division - Meiosis

Under meiosis cell division haploid or germ cells are produced. The gametes are haploid cells and have N chromosomes where a non-sex diploid cells have $2 \cdot N$ chromosomes comprising of the N homologous pairs. The fusion of a male gamete (e.g. the sperm cell) and a female gamete (the egg cell) results in the embryonic cell having $2 \cdot N$ chromosomes.

The meiosis is preceded by one round of DNA synthesis followed by two special cell divisions. The first round of cell division is called the reduction division because the number of chromosomes is reduced from $2 \cdot N$ to N as the two homologous chromosomes are segregated into different daughter cells. During the first metaphase, as shown in Fig. 2.9 (b), the nuclear membrane disappears and the bivalent chromosomes line up along the center of the cell and the spindle connects the centromeres and the centrioles in the opposite poles of the cell. During the anaphase of the first division shown in Fig. 2.9 (c), the homologous chromosomes comprising each bivalent separate from each other and move to the opposite poles. The sister chromatids remain attached at the centromere which, unlike mitosis, do not split during reduction division.

The second cell division is *not* preceded by DNA synthesis. The second cell division is similar to mitosis as each of two sister chromatids attached at the centromere line up along the central plane of the cell and split into the the two daughter cells during the anaphase shown in Fig. 2.9 (e). Four gametes, with N chromosomes each, form during the telophase. Thus, in the case of humans, the daughter cells after meiosis have 23 chromosomes, while the daughter cells after mitosis have 46 chromosomes.

It is also worth noting that the reduction division in meiosis is often accompanied by a cross-over event as shown in Fig. 2.9 (b). As the homologs line up along the central plane of the cell, parts of their genetic material may be exchanged before they are split up during anaphase I. Such an exchange is called cross-over and is basis for genetic diversity. As is evident in Fig. 2.9 (f), two of the gametes contain genetic material that is a combination of the materials exchanged between the two homologs. Exactly one of the four daughter cells will be become fertilized and as such the probability is $\frac{1}{4}$ each that one of the recombined gametes or one of the "pure" homologs will be passed to the offspring.

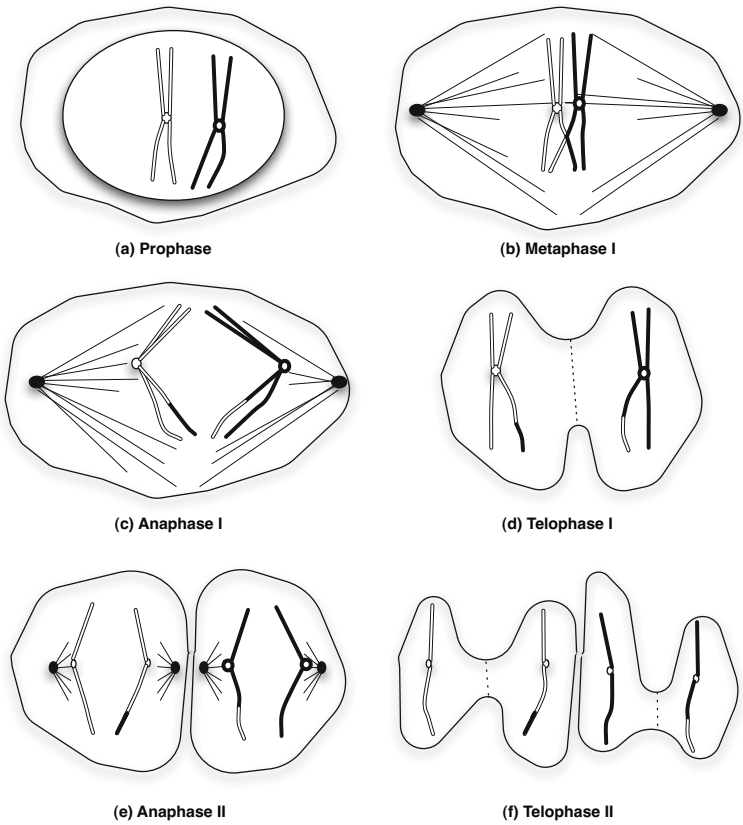


Fig. 2.9 Meiosis. The meiotic division of a diploid cell produces haploid daughter cells known as the gametes

2.2.2 Transcription

A transcription unit is the stretch of DNA that is transcribed into an RNA molecule. The process of transcription is divided into three stages, namely, initiation, elongation and termination. Transcription is initiated with the binding of RNA polymerase to the promoter in DNA. The DNA unwinds and produces a small open complex and RNA synthesis begins and as the complex moves along the template DNA strand the RNA product is elongated. The transcription elongation also involves a proofreading mechanism whereby incorrectly added mRNA is replaced. The transcription terminates when a stop codon is encountered in the template DNA strand. This is accomplished by the formation of a secondary hairpin loop that RNA transcription

complex come off the DNA template. A protein designated "Rho" can pull the mRNA away from polymerase.

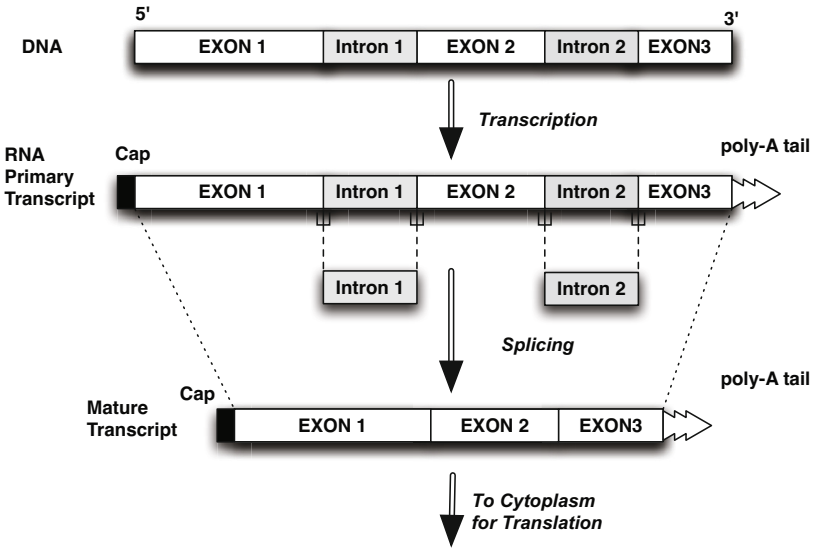


Fig. 2.10 The transcription of a gene results in the synthesis of a primary transcript which is subsequently spliced to remove the introns before the mature transcript is released into the cytoplasm where the translation machinery assembles the protein coded by the gene.

The main stages in transcription and the processing leading to the mature RNA which is ultimately used for synthesis of proteins is shown in Fig. 2.10. The primary transcript is a large strand of RNA which extends from the original 5' to 3' ends and is formed from the entire gene sequence comprising of the exons and introns. The RNA is formed by Watson-Crick base pairing rules. That is, primary transcript contains a A, C, G, or U wherever the DNA sequence contains a T, G, C or A respectively. Note that the base Uracil (U) is used in an RNA sequence instead of thymine (T).

The primary transcript comprising of a large RNA strand is very unstable and is quickly modified in order to stabilize it by adding a cap structure to the 5' end and a string of adenylic acid (poly A) tail at the 3' end. The introns are then removed and the exons are spliced together to form the mature RNA which is transported to the cytoplasm for synthesis of protein. Although the exact mechanism for RNA splicing is not well understood, it is evident that the consensus dinucleotides `GT` and `AG` at the 5' and 3' ends of exon/intron boundaries are crucial for this cleavage to effectively occur.

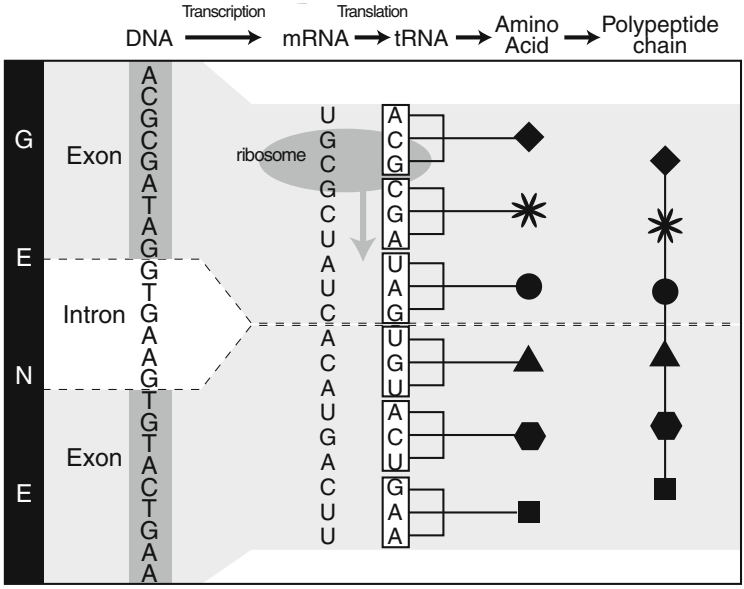


Fig. 2.11 The mature RNA is synthesized by the ribosome which recruits the tRNA with the anticodon that matches the codon in the mRNA sequence. The individual amino acids are linked to form the peptide chain of the coded protein. *Reprinted from National Human Genome Research Institute's Educational Resources.*

2.2.3 Translation

The translation process converts the mature mRNA sequence derived from the coding regions on the DNA into the corresponding polypeptide chain which ultimately folds into the protein that the DNA sequence codes for. In eukaryotic cells, while the site of transcription is the cell nucleus, the site for translation is the cytoplasm. The mRNA carries the protein synthesis instructions from the cell's nucleus to the cytoplasm and must be transported out of the nucleus into the cytoplasm for the synthesis of proteins. This process occurs on the ribosomes located in the cytoplasm.

The mRNA is bound by the ribosome at a nucleotide triplet, or codons. The initial binding site is usually initiator methionine codon, designated by AUG, downstream of the ribosome binding site. The mRNA moves over the surface of the ribosome and successive trinucleotides that code for amino acids are brought into position. The transfer-RNA or tRNA, also an RNA molecule, brings amino acids to the mRNA-ribosoome binding site that correspond to the ribosome-RNA binding triplet. The amino acid recruited by tRNA line up along the previously synthesized amino acids. As this amino acid is linked to the growing polypeptide chain, the tRNA is released and the mRNA moves

further along the ribosome bringing the next codon triplet into position for translation.

Position 1	Position 2 U	Position 2 C	Position 2 A	Position 2 G	Position 3
U	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	U
	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	C
	Leu (L)	Ser (S)	STOP	STOP	A
	Leu (L)	Ser (S)	STOP	Trp (W)	G
C	Leu (L)	Pro (P)	His (H)	Arg (R)	U
	Leu (L)	Pro (P)	His (H)	Arg (R)	C
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	A
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	G
A	Ile (I)	Thr (T)	Asn (N)	Ser (S)	U
	Ile (I)	Thr (T)	Asn (N)	Ser (S)	C
	Ile (I)	Thr (T)	Lys (K)	Arg (R)	A
	Met (M)	Thr (T)	Lys (K)	Arg (R)	G
G	Val (V)	Ala (A)	Asp (D)	Gly (G)	U
	Val (V)	Ala (A)	Asp (D)	Gly (G)	C
	Val (V)	Ala (A)	Glu (E)	Gly (G)	A
	Val (V)	Ala (A)	Glu (E)	Gly (G)	G

Fig. 2.12 Genetic Code. The genetic code is used in translating the genomic mRNA to protein. Amino acids may be abbreviated using a 3-letter or a 1-letter code. Amino acids corresponding to the triplet on the mRNA are abbreviated as follows. Ala (A): alanine; Arg (R): arginine; Asn (N): asparagine; Asp (D): aspartic acid; Cys (C): cysteine; Gln (Q): glutamine; Glu (E): glutamic acid; His (H): histidine; Ile (I): isoleucine; Leu (L): leucine; Lys (K): lysine; Met (M): methionine; Phe (F): phenylalanine; Pro (P): proline; Ser (S): serine; Thr (T): threonine; Trp (W): trytophan; Tyr (Y): tyrosine; Val (V): valine.

This process is repeated in a 5' to 3' direction until a specific termination codon (UAA, UAG or UGA) is reached. The polypeptide chain is then released and the mRNA and ribosome disassociate. There are 64 combinations possible with three bases in every codon triplet. However, there are only 20 amino acids which occur naturally which serve as the building blocks for proteins. As is shown in Fig. 2.12 all amino acids, with the exception of tryptophan, are coded by more than one triplet. This means that the genetic code is *degenerate*. The alternative triplets for a given amino acid only vary by change in the third base of the triplet.

2.3 Gene Expression

Transcription is the process through which a DNA sequence in the nucleus is copied to produce a complementary RNA sequence. This process is done under the aegis of an RNA polymerase enzyme. Thus, the transcription is the mechanism by which genetic information is transferred from the protein coding DNA into mRNA, or messenger RNA, which ultimately leads to the synthesis of proteins. Transcription differs from replication in that only one strand of the DNA, referred to as the template strand, is used make mRNA. Thus, while the DNA exists as a double-stranded molecule, the RNA exists as a single stranded molecule. The complete schematic of the gene expression process is shown on Fig. 2.13.

In humans, a given cell may express anywhere between 1,000 to 10,000 of the possible 25,000 genes. Thus, a kidney cell will express a different set of genes than a liver cell which in turn will express a different set of genes than a brain cell. Some of the genes are expressed in response to the environmental factors while other expression patterns are in response to feedback mechanisms and genetic circuits typical of normal growth and differentiation. The ability a cell to express a different set of genes from the common repertoire is called *differential gene expression*. Furthermore, even when the total number of genes in humans is around 20,000–25,000, eukaryote cells have the ability to further modify the primary transcript via alternative splicing en-route to the synthesis of mature RNA or mRNA. In this process, blocks of mRNA are cut out and rearranged, to produce different arrangements of the original sequence. This increases the number of possible proteins that are synthesized from a single gene sequence!

Proteins are also often modified after translation to become active. For example, although insulin is produced as a 82-amino acid molecule called proinsulin, 31-residues are subsequently removed before it becomes an active hormone. In general the chain of amino acid synthesized by the ribosomes undergoes several levels of folding as shown in Fig. 2.14. The *primary structure* of the protein is simply the sequence of amino acid residues. In some cases proteins may attach with non-protein groups to become active as in the case of hemoglobin where the heme group must be attached before the protein becomes functional. The *secondary structure* of the protein refers to the structure formed by the bonding between different polypeptides. For example, hydrogen bond formed between a group of polypeptides may result in the twisting of the peptide backbone forming an α-helix. The *tertiary* and the *quaternary* structure of a protein refers to its three dimensional form. The three dimensional shape represents the most efficient form and the most favorable arrangement for the protein to perform its function.

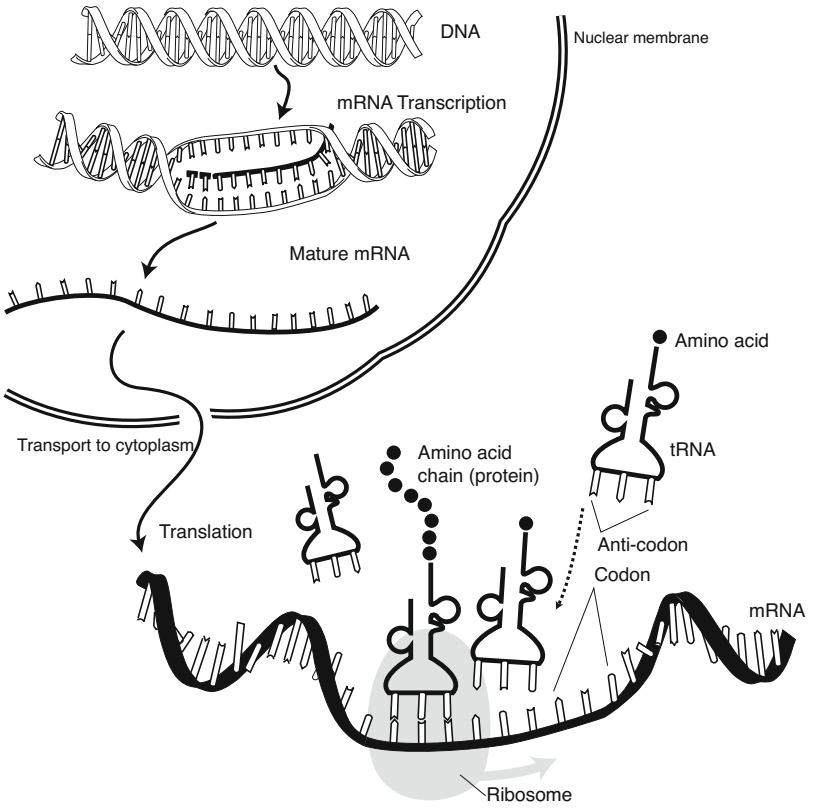


Fig. 2.13 The expression of genes involves the transcription of DNA into a mRNA which in the case of eukaryotes is further spliced before the mature RNA is released into the cellular translation processes convert it to the corresponding protein. *Reprinted from National Human Genome Research Institute's Educational Resources.*

2.4 Gene Linkage

Genetic linkage is defined as the joint co-inheritance of alleles. In the theory that each gene is inherited independent of others, know as an independent assortment of alleles, an organism can pass on an allele without regard to which allele was passed on for a different gene. However, given the independent assortment of genes that are physically close to each other on the chromosome, such an independent assortment may not occur during meiosis. Thus alleles that are on the same chromosome are more likely to be inherited together. These alleles are said to be linked.

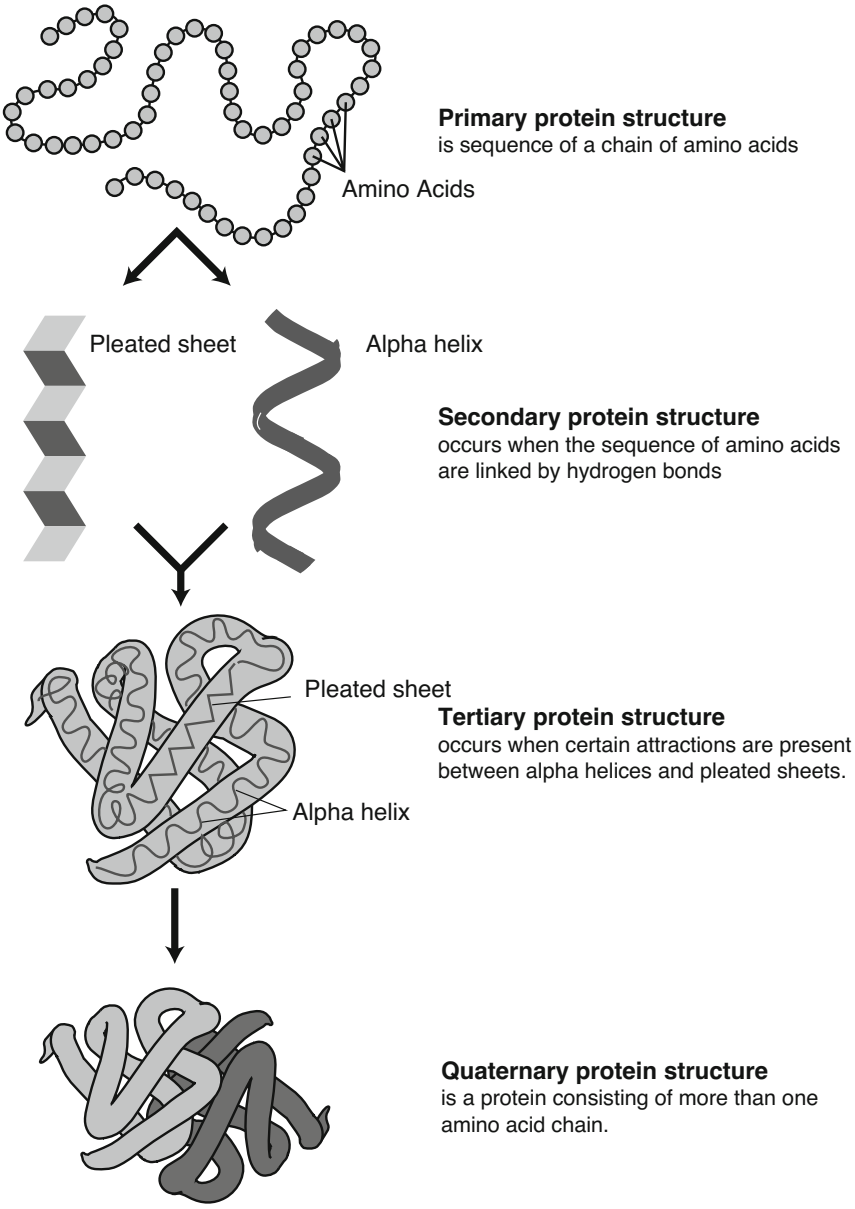


Fig. 2.14 The function of a protein is largely determined by its structure. After the poly-peptide chain of the amino acid molecule has been synthesized, the linear chain undergoes several levels of folding to reach its final stable structure. *Reprinted from National Human Genome Research Institute's Educational Resources.*

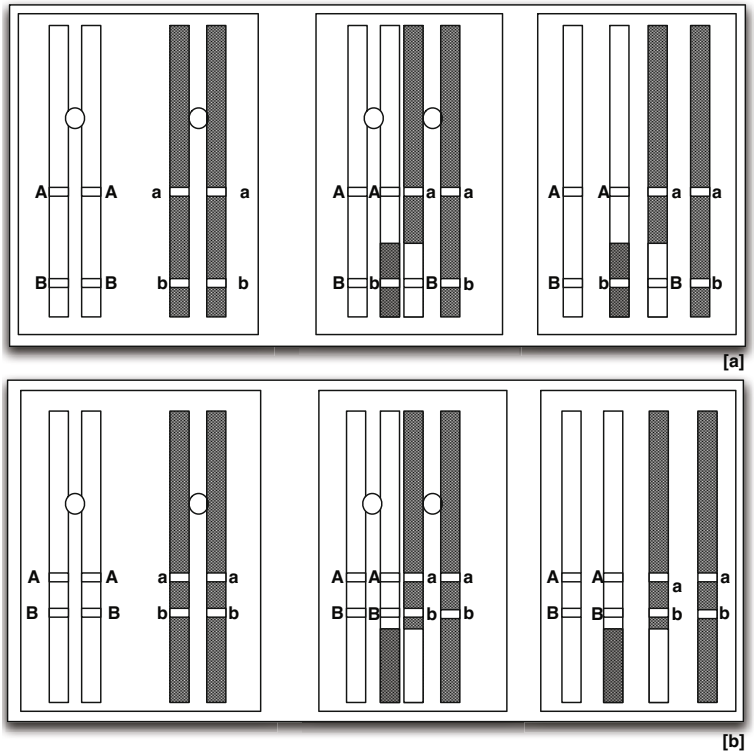


Fig. 2.15 Prior to the first meiotic division where the homologous chromosomes are segregated, portions of one chromosome in the homologous pair may cross over with the other as they line up along before division. Such a cross over results in the production of (a) gametes where genes originally linked in the parent cell are no longer linked in the gametes, or as in (b) continue to be linked in the gametes.

As shown in Fig. 2.15, the two alleles that are close to each other have a greater chance of remaining linked in the gametes during meiosis. Because there is some crossing over of DNA when the homologs segregate during the reduction division in meiosis, alleles on the same chromosome can be separated and go to different cells. There is a greater probability that two alleles will be separated if a cross-over occurs if the alleles are far apart on the chromosome. When two genes are located on the same chromosome, they will usually be inherited as a single unit. When two genes are often inherited together, they are said to be linked and referred to as *linkage groups*. For example, in fruit flies the genes affecting eye color and wing length are linked and inherited together because they appear on the same chromosome.

The probability of the genes getting segregated is proportional to the distance between them. Thus the relative distance between them is calculated using the percentage of offsprings of an organism showing two linked genetic

traits and comparing them to the percentage of the offspring where the two traits are not inherited together. The percentage of the population of descendants that does not show co-inheritance of both traits is thus an indication of the distance between them.

The linkage distance between genes is measured in the units of centimorgans (abbreviated cM) which is a unit of recombinant frequency for measuring genetic linkage. One centimorgan is equal to a 1% chance that a marker at one genetic locus on a chromosome will be separated from a marker at a second locus due to crossing over in a single generation. Another unit of recombination frequency is the map unit or m.u. One m.u. represents a recombination frequency of 1% and thus a map unit is synonymous with centimorgan. In humans, a centimorgan corresponds roughly to a megabase of DNA sequence.

2.5 DNA Sequencing

The process of sequencing a DNA involves determination of the precise order of nucleotides within a DNA molecule. The advent of rapid DNA sequencing methodology is expected to have a positive impact on biological and medical research and discovery since the knowledge of DNA sequences is indispensable for biological research, diagnostics, and therapeutics. It is due to the rapid DNA sequencing with modern DNA sequencing technologies that we are able to sequence complete genome sequencing of numerous organisms including the human genome and other animals, plants, and microbial species.

DNA sequencing techniques involve determining the sequence of fragments of short segments of the target sequence and assembling them together in a manner similar to solving a jigsaw puzzle. The short 500-1500 bp segments are sequenced using Maxim-Gibert or Chain Termination techniques.

Maxim-Gilbert method allows the use of purified samples of double-stranded DNA to be used without further cloning and uses radioactive labeling at one 5' end of the DNA and purification of the DNA fragment to be sequenced. Chemical treatment generates breaks at a small proportion of one or two of the four nucleotide bases in each of four reactions (G, A+G, C, C+T) [1]. Due to the technical difficulties with Maxim-Gilbert, the Chain-Termination method developed Sanger and coworkers in 1977 became the method of choice. Sanger sequencing is the method which prevailed since the 80's [2].

However, the great demand for high speed sequencing is giving rise to the next-generation sequencing methods. The NGS has been applied to applications such as genome sequencing, genome resequencing, transcriptome profiling (RNA-Seq), and DNA-protein interactions (ChIP-sequencing) [3]. It should be noted as we are moving towards personalized medicine, the scope of resequencing – studying the variability of individual genome when compared to the that of species – will become significant.

There is a high demand for sequencing at low-cost with technologies that parallelize the sequencing process and producing thousands or millions of sequences concurrently are coming into the forefront. High-throughput sequencing technologies are expected to lower the cost of DNA sequencing when compared to the traditional chain termination methods and make large volumes of data available for bioinformaticians to explore commonalities and differences between sequences [4, 5].

2.6 Summary

This chapter presented a brief overview of molecular biology and genetics. Coverage of topics includes the the central dogma of molecular biology, gene expression, and protein folding. This chapter is meant to provide an overview of the broad topics that are important for understanding the basis for significant biological features that must be taken into account for biological data analysis techniques and systems. Further details on the topics covered in this chapter and other relevant topics may be found in standard textbooks on molecular biology and genetics.

. Traditional textbooks may be referred for detailed account of cellular processes. Creativity and adaptations in molecular biology are, for example, discussed in [11]. Discussions on Polymerase Chain Reaction (PCR) used for amplification of DNA is provided in [12] and [13].

Computational difficulties relating to the Human Genome Project are discussed by Robbins [14] and Frenkel [15]. Robbins presents the viewpoint that organism's viability is a result of a complex interactions of many "cellular processes."

The "programming metaphor" where the cellular DNA is likened to a computer program is discussed by Atlan [16]. Hofstadter discusses the relationships between molecular biology, mathematical logic and music [17].

Further Readings

1. Maxam, A.M., Gilbert, W.: A new method for sequencing dna. Proc. Natl. Acad. Sci. U S A 74(2), 560–564 (1977)
2. Sanger, F., Coulson, A.R.: A rapid method for determining sequences in dna by primed synthesis with dna polymerase. J. Mol. Biol. 94(3), 441–448 (1975)
3. de Magalhães, J.P., Finch, C.E., Janssens, G.: Next-generation sequencing in aging research: emerging applications, problems, pitfalls and possible solutions. Ageing Res. Rev. 9(3), 315–323 (2010)
4. ten Bosch, J.R., Grody, W.W.: Keeping up with the next generation: massively parallel sequencing in clinical diagnostics. J. Mol. Diagn. 10(6), 484–492 (2008)
5. Tucker, T., Marra, M., Friedman, J.M.: Massively parallel sequencing: the next big thing in genetic medicine. Am. J. Hum. Genet. 85(2), 142–154 (2009)
6. Lewin, B.: Genes VIII. Pearson/Prentice Hall, Upper Saddle River (2004), ergito.com

7. Lewin, B.: Genes IX. Jones and Bartlett Publishers, Sudbury (2006)
8. Watson, J.D.: Molecular biology of the gene. Pearson/Benjamin Cummings, 5th edn. CSHL Press, San Francisco (2004), Watson, J.D., et al.: ill, ports. 29 cm. +e 1 CD-ROM (4 3/4 in.) Systems requirement for accompanying CD-ROM: IBM PC or Macintosh; Windows 98, 2000, NT, XP or MAC OS 9.X, X (2000)
9. Lewin, B., Caparon, M.: Cells. Jones and Bartlett Publishers, Sudbury (2007), Lewin, B., et al.: scientific editors, Leslie Pond, Jamie Kass; [contributors, Michael Caparon et al]. ill (chiefly col.); 29 cm.
10. Doolittle, R.F.: Proteins. Scientific American 253(4), 74–83 (1985)
11. Doolittle, W.F.: Evolutionary creativity and complex adaptations: a molecular biologist's perspective. Jones and Bartlett Publishers, Boston (1994); Doolittle, W.F.: Conference held 1993 Mar. Creative Evolution?! Symposium (1993: University of California, Los Angeles, CA) (1993)
12. Mullis, K.B., Ferre, F., Gibbs, R.: The unusual origin of the polymerase chain reaction. Scientific American 262(4), 56–65 (1990)
13. Mullis, K.B., Ferre, F., Gibbs, R.: The polymerase chain reaction. Birkhauser, Boston (1994), Mullis, K.B., Ferre, F., Gibbs, R.A.: foreword by Watson, J.D ill.
14. Robbins, R.J.: Challenges in the human genome project. IEEE Engineering in Medicine and Biology 11(1), 25–34 (1992)
15. Frenkel, K.A.: The human genome project and informatics. Communications of the ACM 34(11) (1991)
16. Atlan, H., Koppel, M.: The cellular computer dna: program or data? Bulletin of Mathematical Biology 52(3), 335–348 (1990)
17. Hoftadter, D.R.: Godel, Escher, Bach: An Eternal Golden Braid. Basic Books, New York (1976)

2.7 Exercises

Genetic Code. Under the standard code, ATG is the initialization code. In rare cases, translation in eukaryotes can be initiated from codons other than ATG. The standard code currently allows initiation from TTG and CTG in addition to AUG.

```
TTT F Phe      TCT S Ser      TAT Y Tyr      TGT C Cys
TTC F Phe      TCC S Ser      TAC Y Tyr      TGC C Cys
TTA L Leu      TCA S Ser      TAA * Ter      TGA * Ter
TTG L Leu i    TCG S Ser      TAG * Ter      TGG W Trp

CTT L Leu      CCT P Pro      CAT H His      CGT R Arg
CTC L Leu      CCC P Pro      CAC H His      CGC R Arg
CTA L Leu      CCA P Pro      CAA Q Gln      CGA R Arg
CTG L Leu i    CCG P Pro      CAG Q Gln      CGG R Arg

ATT I Ile      ACT T Thr      AAT N Asn      AGT S Ser
ATC I Ile      ACC T Thr      AAC N Asn      AGC S Ser
ATA I Ile      ACA T Thr      AAA K Lys      AGA R Arg
ATG M Met i    ACG T Thr      AAG K Lys      AGG R Arg

GTT V Val      GCT A Ala      GAT D Asp      GGT G Gly
GTC V Val      GCC A Ala      GAC D Asp      GGC G Gly
GTA V Val      GCA A Ala      GAA E Glu      GGA G Gly
GTG V Val      GCG A Ala      GAG E Glu      GGG G Gly
```

Another shorthand notation of representing the standard genetic code where the IUPAC character encoding for the proteins is utilized is as follows:

```
  AAs  = FFLLSSSSYY**CC*WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG
Starts = ---M---------------M---------------M----------------------------
Base1  = TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
Base2  = TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
Base3  = TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
```

1. Consider the following DNA sequence:

   ACCCA TAGGG AGACA TAGTA GATCC ATTAG

   (a) Perform a 6 frame translation of a given DNA strand.
   (b) Compute the length of Open Reading Frames (ORF) in each of 6 frames. Based on your analysis, which strand and which frame within that strand has the highest likelihood for coding.

2. Given that the following DNA sequence is part of an intron-less gene, provide the sequence of the corresponding mRNA sequence as well as the

sequence of protein assuming the first frame is the coding frame. Assume that the organism uses standard genetic code for translation.

```
atgagc aagcac atacaa ctccaa agacgc ccttag
```

3. Given that the following DNA sequence, show the sequence of nucleotides in the six standard frames. Recall, that there are three forward and three reverse reading frames, and that the fourth frame is the first frame on the complementary strand. The sequence reads in your answer must all be from 5'→3' end.

   ```
   aggagt aagccc ttgcaa ctggaa atacac ccattg
   ```

4. Given the following DNA sequence, provide its reverse complement:

   ```
   ACTGAC TCGAAT TGGACC CCTTAA
   ```

5. Given the following DNA sequence, provide the mRNA sequence produced. Assume that the entire sequence belongs to an exon and no bases are spliced out.

   ```
   ATGGAC TCGAAT TGGACC CCTTAA
   ```

6. The length of a bacterial genome is 4.7×10^6 bp. Assuming that the DNA replication rate in bacteria is 300 bps, how long will it take a bacterial culture to grow 100-fold in size.
7. Using the standard genetic code, compute the translation for the following sequence in the first forward frame.

   ```
   CATGGAGCCT CTTGCAGCTT ACCCGCTAAA
   ```

8. Given the following sequence of DNA, determine the largest possible open reading frame. Note that the ORF might occur in the reverse complement strand. So, in order to assign the ORF to a strand you must test out the ORF lengths in all six, i.e. the three forward and three reverse, strands.

   ```
   ATGGAC TCGAAT TGGACC CCTTAA
   ```

Chapter 3
Biological Databases

As computer technology becomes more integrated into biological science, the amount of data that needs to be tracked has been experiencing unprecedented growth. Data sources range from biological sequences to 3D structures, to DNA traces, to proteomic gels. The wide range of biological databases providing essential information related to the life sciences is evident in the snapshot of small subset databanks available to biomedical researchers shown in Fig. 3.1. Databases have become an essential tool for research in biotechnology today.

Life science databases have generally been classified into the following categories:

- DNA and protein sequence databases. These are sometimes referred to as *primary databases*
- Genomics and the databases capturing the sequence of whole genomes
- Protein domain/family and structure databases encompassing protein and other bio-molecular structure. These are sometimes referred to as *secondary databases*
- Databases that track Mutations and polymorphism
- Proteomics databases track information on 2D gel and mass spectroscopy
- Databases tracking information on metabolism and related pathways
- Bibliography databases
- Terminology and Ontological databases
- Other databases providing specialized functions

Nucleic acids (and primary protein sequences) are at the core of molecular biology and are fundamental to the research in bioinformatics as a variety of tools are developed to fully unravel the patterns and functional control signals as well as protein coding regions found in the genomic databases. An important distinction exists between the primary and secondary databases. Generally, the primary sequence data is archival in nature, while the secondary data sets are curated prior to their public release. Thus, in a sense

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_3

AATDB, AceDb, ACUTS, ADB, AFDB, AGIS, AMSdb,
ARR, AsDb, BBDB, BCGD, Beanref, Biolmage,
BioMagResBank, BIOMDB, BLOCKS, BovGBASE,
BOVMAP, BSORF, BTKbase, CANSITE, CarbBank,
CARBHYD, CATH, CAZY, CCDC, CD4OLbase, CGAP,
ChickGBASE, Colibri, COPE, CottonDB, CSNDB, CUTG,
CyanoBase, dbCFC, dbEST, dbSTS, DDBJ, DGP, DictyDb,
Picty_cDB, DIP, DOGS, DOMO, DPD, DPlnteract, ECDC,
ECGC, EC02DBASE, EcoCyc, EcoGene, EMBL, EMD db,
ENZYME, EPD, EpoDB, ESTHER, FlyBase, FlyView,
GCRDB, GDB, GENATLAS, Genbank, GeneCards,
Genline, GenLink, GENOTK, GenProtEC, GIFTS,
GPCRDB, GRAP, GRBase, gRNAsdb, GRR, GSDB,
HAEMB, HAMSTERS, HEART-2DPAGE, HEXAdb, HGMD,
HIDB, HIDC, HlVdb, HotMolecBase, HOVERGEN, HPDB,
HSC-2DPAGE, ICN, ICTVDB, IL2RGbase, IMGT, Kabat,
KDNA, KEGG, Klotho, LGIC, MAD, MaizeDb, MDB,
Medline, Mendel, MEROPS, MGDB, MGI, MHCPEP5
Micado, MitoDat, MITOMAP, MJDB, MmtDB, Mol-R-Us,
MPDB, MRR, MutBase, MycDB, NDB, NRSub, 0-lycBase,
OMIA, OMIM, OPD, ORDB, OWL, PAHdb, PatBase, PDB,
PDD, Pfam, PhosphoBase, PigBASE, PIR, PKR, PMD,
PPDB, PRESAGE, PRINTS, ProDom, Prolysis, PROSITE,
PROTOMAP, RatMAP, RDP, REBASE, RGP, SBASE,
SCOP, SeqAnaiRef, SGD, SGP, SheepMap, Soybase,
SPAD, SRNA db, SRPDB, STACK, StyGene,Sub2D,
SubtiList, SWISS-2DPAGE, SWISS-3DIMAGE, SWISS-
MODEL Repository, SWISS-PROT, TelDB, TGN, tmRDB,
TOPS, TRANSFAC, TRR, UniGene, URNADB, V BASE,
VDRR, VectorDB, WDCM, WIT, WormPep, YEPD, YPD,
YPM, etc .....

Fig. 3.1 A small snapshot of biological databases

the primary databases represent the experimental results, with some interpretation offered by the researchers in the form of the annotations associated with the sequence data. For example, the ubiquitous nucleotide database is GenBank, which contains primary sequence data, as is the Protein Databank (PDB) which is a database of nucleic acid and protein structures.

3.1 Nucleotide Databases

The three collaborative nucleotide databases are quite similar in the content and periodically exchange information to keep their contents synchronized on a daily basis. These databases include sequences submitted directly by scientists and genome sequencing centers and sequences gleaned from literature and patents. The three databases, each of which has the capability to assign a unique accession number to every sequence based on codes preallocated to each center, belong to the International Nucleotide Sequence Database Collaboration are the GenBank at the National Library of Medicine (Bethesda, United States), DNA Data Bank of Japan (DDBJ, Mishima, Japan) and the European Molecular Biology Laboratory (EMBL) nucleotide database from the European Bioinformatics Institute (EBI, Hinxton, UK). For example, based on the accession number assignment scheme, the human globin sequence with the accession number was originally added to the collaborative database by GenBank, while the alpha globin was added to the collaborative database by EMBL. For the sake of brevity, the collaborative nucleotide sequence databases will be henceforth collectively referred to as GenBank.

3.1.1 GENBANK

GenBank is the genetic sequence database maintained by the National Institute of Health (NIH). It is comprised of an annotated collection of all publicly available nucleotide and protein sequences. Generally, the records in the GenBank database are comprised of contiguous stretches of DNA or RNA that have been annotated to add information to the raw sequence segments within that record. Presently, the records in GenBank are based upon the direct submission of DNA sequences by the original authors along with some level of data curation by the GenBank staff facilitating the search and retrieval of the submitted records.

GenBank was designed and created by the National Center for Biotechnology Information (NCBI), a division within the National Library of Medicine within the NIH. These three centers provide separate data entry points and maintain data formats that are somewhat unique to their configuration. However, since all three are part of one international consortium and information between them is exchanged daily, thus making them same data available to the research community at-large.

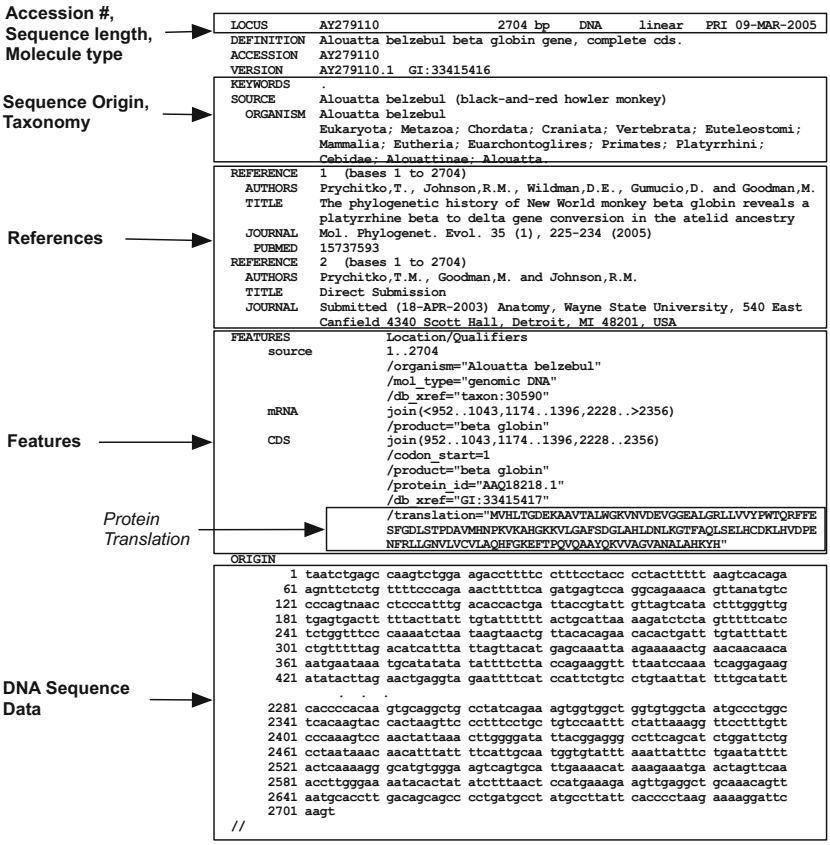


Fig. 3.2 A GENBANK sequence record

A sample entry of the GenBank database is shown in Fig. 3.2. As is demonstrated in this record, the raw sequence data is stored along with an array of ancillary annotations. The data format for this sequence is that utilized by the GenBank flat file system. It is, however, possible to transform this sequence into a variety of other formats, including those used in the EMBL database, as well as the FASTA format which is generally a minimalist format where most of all annotations have been stripped away. Interested readers should complete Exercise 5 for gaining more information on retrieval from GenBank in a variety of formats. Let us next turn to dissecting the entries of the GenBank entry shown in Fig. 3.2.

The Header or Locus Line

The header, or the locus line, of a GENBANK sequence record specifies its locus name. It also provides the length of the DNA sequence, the molecule type that was sequenced to define the sequence data. Additionally, the major database division to which the sequence is categorized, and the date the sequence record was made public are also specified.

The significance of the locus name is mostly historic going back into the days when the sequence records represented a single gene loci and the search for records was facilitated with the use of characteristic locus names such as HUMHBB or SV40 denotes human β-globin gene and simian virus respectively. The locus name has lost its significance these days; however, since many software tools rely on the presence of a unique locus name, the locus name is often replaced with the sequence accession number.

The second item in the header is the length of the sequence. Sequence length ranges from 1 to 350,000 base pairs (bp) in a single record with the sequence records larger than 350 kb (kilobases, or thousand bases) begin allowed when they represent a single gene. NCBI seldom accepts sequences smaller than 50 bp and larger than 350 kb are represented as segmented or "contigged" sets. Current release of GENBANK contain sequence records that are longer than 350 kb.

The third item of the sequence defines the molecule type of the sequence record. The molecule type is DNA or RNA and also indicates the strandedness (single, ss or double ds) of the sequence. The molecule type is intended to represent the original biological molecule that was sequenced to determine the sequence. The acceptable molecule types are DNA, RNA, tRNA, rRNA, mRNA, and uRNA.

The fourth item of the locus line represents the GENBANK division code, which is a three letter code that connotes the taxonomic or database classification of the sequence record. The division codes were once utilized to break up the database into more manageable sized partitions. For example, the human sequences were grouped within the `PRI` or the primate division of GENBANK. Such taxonomic classifications were augmented with database divisions such as Expressed Sequence Tags (EST), Sequence Tagged Sites (STS) and High Throughput Genome sequences (HTG). The CON or contigged division is interesting as it is comprised of sequence records that are virtual and define sequences as they are constructed by piecing together sequence records that contain actual sequence data.

The fifth item of the locus line represents the date field that defines the date the record or an update was made public. Thus, if the record is updated after it was first made public, the date field would specify the date when the updated record was made public. The dates are primarily meant to serve as a guide for the users and are not considered as legally binding for establishing precedence in patent applications.

The Definition, Accession, Version, and Keywords

The definition line (or "def line") attempts to summarize the biology of the sequence record. The definition line is displayed when sequence similarity results are displayed as when, for example, the GENBANK is searched by a tool such as BLAST. In the general format for the definition line shown below, the full genus-species names are used in the definition line in accordance with the agreements between the international collaborative databases.

```
Genus species product name (gene symbol) mRNA or gene, complete cds.
```

The accession number represents the primary key to reference a given record in the database. Accession number exists in one of two formats: the older format, known as the "1+5" format comprised of a single character followed by five digits; the newer format, or the "2+6" format comprises of two letters followed by six digits. All GENBANK accession numbers are recorded on a single line with the word ACCESSION appearing at the beginning of the line. Although most sequences in GENBANK have a single accession number, some sequences have multiple accession numbers recorded on the ACCESSION line. In these instances, the first accession number is considered to be the primary accession number. Sometimes the primary accession number is the replacement for an older accession number which is recorded as the secondary accession number to enable users to link the two accession numbers.

The VERSION line following the ACCESSION number line contains the *accession.version* specifying the version of the sequence. It also contains a GI or GenInfo identifier for the sequence which is internally used to identify each sequence uniquely. Thus, two versions of the same sequence will have different GI numbers. The version number is tied to the sequence data. When sequence data changes, the version number is incremented by one and a new GenInfo identifier is assigned to the sequence. As the GenInfo identifier is the next available integer, there is no numerical relationship between the GI identifier of one version to the next.

The KEYWORD line was meant to include words from a controlled vocabulary. Since sequence authors over the years many have used words that were not a part of this vocabulary, the significance and utility of the keyword line has diminished to a point where its use is discouraged by NCBI.

The Sequence Origin and Taxonomy Lines

The SOURCE line contains the scientific or common name of the organism that is the source of the sequenced data. The taxonomy information following the source lines provides taxonomical lineage information for the organism. The sequence source information and taxonomy information can be derived from source feature in the Sequence Feature Table as described

below. The SOURCE and TAXONOMY information is the hallmark of the older sequence records in GENBANK. There is an ongoing effort where newer records incorporate, and older records are being updated to incorporate, the source feature in the feature table that, in conjunction with the NCBI taxonomy database, can be used to generate this information.

Sequence References

Every GENBANK record must have at least one associated reference or citation. MEDLINE identifiers are included to enable the sequence browsers to link directly to the National Library of Medicine's database of publications. References to unpublished articles is included, as well as scientific credit to the people responsible for determining the sequence. The credit information is usually the last piece of information in the REFERENCES section.

Sequence Features Table

The feature table may be the most important piece of biological information annotated on a sequence record. Features, such as open reading frames or sequences showing similarity to another sequence can be annotated. This enables the feature table to incorporate contributions from researchers doing computational analysis of the sequence databases as well as those annotating sequences by experimental data. The entries in the feature table are annotations associated with (i.e. placed on) a specified span of the sequence whereas the features annotated for the entire sequence are often referred to as descriptors. Each end point of a feature may be specified as a single point, an alternate set of possible end points, a base number beyond which the end point lies, or a region which contains the end point. Feature keys are arranged hierarchically, allowing complex and compound features to be expressed. Both location operators and the feature keys show feature relationships even when the features are not contiguous. The hierarchy of feature keys allows broad categories of biological functionality to be easily annotated. Let us next examine some key features placed on a sequence record.

The Source Feature: All GenBank entries must have the source feature attached with them and the qualifier `organism` must be associated with each source feature. Additional qualifiers may be associated with the feature as shown in the example below where the qualifiers `mol_type` and `db_xref` are associated with the source feature.

```
source      1..2704
            /organism="Alouatta belzebul"
            /mol_type="genomic DNA"
            /db_xref="taxon:30590"
```

The organism qualifier contains the specific species and genus names and may sometimes describe the organism at the level of subspecies. Additional information about the chromosomal localization and tissue source of the sample may be associated with the source. Complete taxonomical information about the source organism is provided by the identifier of the NCBI taxonomical databases. The lineage information for `toxon:30590` defined at NCBI taxonomical database is defined as follows:

```
cellular organisms; Eukaryota; Fungi/Metazoa group; Metazoa;
Eumetazoa; Bilateria; Coelomata; Deuterostomia; Chordata; Craniata;
Vertebrata; Gnathostomata; Teleostomi; Euteleostomi; Sarcopterygii;
Tetrapoda; Amniota; Mammalia; Theria; Eutheria; Euarchontoglires;
Primates; Simiiformes; Platyrrhini; Cebidae; Alouattinae; Alouatta
```

CDS Feature: The CDS feature provides location information for joining the sequence data and making the protein sequence from the coordinates specified on the CDS feature. A series of locations specifying the exons may be "join"-ed into a full coding sequence. In the example below three segments of a nucleotide sequence are joined together to yield a protein sequence.

```
CDS     join(952..1043,1174..1396,2228..2356)
        /codon_start=1
        /product="beta globin"
        /protein_id="AAQ18218.1"
        /db_xref="GI:33415417"
        /translation="MVHLTGDEKAAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFE
        SFGDLSTPDAVMHNPKVKAHGKKVLGAFSDGLAHLDNLKGTFAQLSELHCDKLHVDPE
        NFRLLGNVLVCVLAQHFGKEFTPQVQAAYQKVVAGVANALAHKYH"
```

This protein sequence is assigned an accession number and deposited into a protein database to foster the process of making functional inferences and gene discoveries. The protein sequence accession number follows the "3+5" format, i.e. three letters followed by five digits, followed by the version number of the protein sequence, `AAQ18218.1`. The version number of a protein sequence changes when the protein sequence itself changes. In the example shown the protein sequence is its first version. The translation of the nucleotide sequence into a protein is associated as a qualifier of the CDS feature.

Other features such as **Gene** and **RNA** features are often associated with sequence records. A number of "generic" or miscellaneous feature keys have been added to permit annotation of features that cannot be adequately described by existing feature keys.

Alternative Splicing

The split gene structure of most eukaryote protein coding genes has a very important consequence; that by splicing the gene in different ways variant proteins can be produced. Thus, while there seems to be around twenty thousand (protein coding) genes in the Human Genome, there are many more different proteins; perhaps somewhere between one hundred thousand and one million - with the count depending as much on definitional issues as anything else. The two most basic forms of variation in splicing are shown in Fig. 3.3.

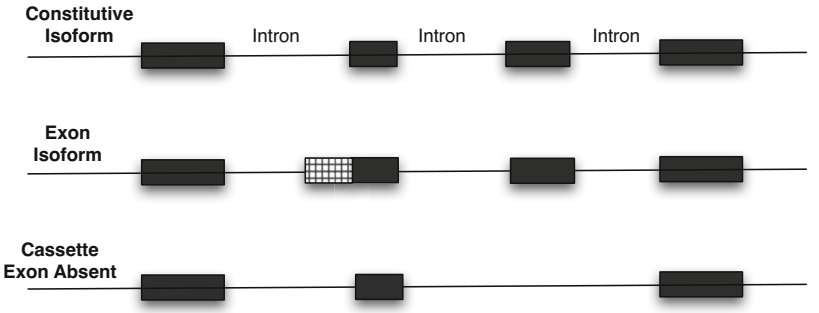


Fig. 3.3 Alternative splicing of the pre-mRNA molecule enables expression of multiple mRNA isoforms from the same gene and thus express multiple protein products through multiple isoforms of individual exons, or using only a subset of exons

Other aspects of alternative splicing, such as mutually exclusive exons, combinatorial splicing, regulation, disease, etc. are beyond the scope of this discussion. Features annotated on GenBank sequences include information on alternatively spliced genes.

Sequence Data

The actual sequence data is provided for the entries that do not belong to the `CON` division. The entries in the `CON` division are virtual and do not contain any sequence data per se but are constructed by joining together of other sequences in the nucleotide database. GenBank, EMBL, and DDBJ have established a special-purpose division, Contig (CON), for exchanging assembly instructions for data in the international DNA sequence databases. Thus, the sequences in the CON division contain no sequence data, but rather instructions for the assembly of sequence data from multiple GenBank records. It also includes instructions for constructing assemblies of non-contiguous sequence shown in Entrez as *segmented sets.*

Now for a bit of historical background on the CON division. In 1995, the International Nucleotide Sequence Database Collaborators (GenBank, DDBJ,

and EMBL) agreed to a 350 kilobases (kb) limit on the size of database sequence records in order to maintain compatibility with existing molecular biology software that was not able to work with large sequences. A new GenBank division was created at that time and called "CON" for designating contigs. The records in the CON division contain the instructions for the assembly of full-length contigs from the sequence data of multiple GenBank records. Although CON division records contain no sequence data, the assembly information they provide makes it possible for database search and retrieval systems to show complete genomic sequences by dynamically assembling the data for display.

However, by 1998, GenBank, DDBJ, and EMBL were routinely accepting submissions from large scale sequencing projects, such as the high-throughput genomic sequences (HTGS), that were longer than 350 kb. To avoid breaking a huge amount of draft sequence into 350 kb chunks, the database collaborators agreed to relax the 350 kb limit in these cases. The 350 kb limit was also relaxed for assemblies of Whole Genome Shotgun (WGS) project data and for large eukaryotic genes. Finally in 2003, the Database Collaborators agreed to remove the 350 kb limit for all sequences as of June 2004, since the increased ability of molecular biology software to analyze long sequences quickly has rendered the limit on sequence length unnecessary.

URL for large sequence: `ftp://ftp.ncbi.gov/genbank/LargeSeqs`

An example of the effect of the removal of the 350 KB limit on GenBank records may be seen in the case of accession `U00096` , the *Escherichia coli* K-12 MG1655 complete genome sequence. Under the 350 KB limit, this accession number referred to a contig record giving a list of short sequences that can be assembled to create the complete genome. With the removal of the 350 KB limit the accession now refers to the complete contiguous sequence for Escherichia coli K-12. The accessions for all 400 parts appear as secondary accessions. The CON division continues to remain as a GenBank division for representing sequences which by their nature are assembles, such as genome scaffold records.

3.2 Protein Sequence Databases

The fundamental building blocks of life are proteins. Enzymes, which are the molecular machines responsible for virtually all of the chemical transformations that cells are capable of, are proteins. In addition, much of the structure of a cell is made up of proteins. Transcription factors are proteins whose job it is to determine which regions of DNA in a cell are transcribed into RNA, and thus which proteins are made in the cell and at what level. That part of the structure which is not made up of proteins is produced by enzymes which are proteins.

Proteins are variable length linear, mixed polymers of 20 different amino acids. Typical proteins contain 200 to 1000 amino acids in each polypeptide

chain[1] and have one to four polypeptide chains. Numerous proteins fall well outside these ranges, however. A human contains on the order of 100,000 different proteins. It is the properties of and the interactions between these 100,000 proteins that make us what we are.

There are a lot of small peptides found in the cell which are not normally referred to as proteins. Examples of such include growth factors and neurotransmitters. Other terms used more or less interchangeably for amino acid polymers are peptides and polypeptides. Actually, the formal definitions of peptide, polypeptide, and protein are overlapping but not identical. The only distinction significant for our purpose is that peptides and polypeptides consist of single covalently linked amino acid polymers whereas proteins can contain one or more different polypeptides linked by noncovalent bonds.

The two protein databases, PIR and SWISS-PROT are different from the nucleotide databases in that they are both curated. This means that the entries in these databases are prepared by a designated group of curators in consultation with external experts. These two protein sequence databases may be considered as secondary sequence databases because they add value by taking the majority of their sequences from the translation of nucleotide sequences in GenBank. A small minority of SWISS-PROT sequences are derived from the scanning journal articles for published protein sequences as well as those that are directly submitted to SWISS-PROT.

3.2.1 Swiss-Prot

Swiss-Prot is an annotated protein sequence database. It was established in 1986 and maintained collaboratively, since 1987, by the group of Amos Bairoch first at the Department of Medical Biochemistry of the University of Geneva and now at the Swiss Institute of Bioinformatics (SIB) and the EMBL Data Library (now the EMBL Outstation - The European Bioinformatics Institute (EBI)). The Swiss-Prot Protein Knowledgebase consists of sequence entries. Sequence entries are composed of different line types, each with their own format. For standardization purposes, the format of Swiss-Prot follows as closely as possible that of the EMBL Nucleotide Sequence Database.

In Swiss-Prot, as in many sequence databases, two classes of data can be distinguished: the *core data* and the *annotation*. For each sequence entry the *core data* consists of the sequence itself, the citation information and taxonomic data providing the description of the biological source of the protein. The *annotation data* consists of the description of function(s) of the protein, the secondary (e.g. alpha helix, beta sheet) and quaternary structure (e.g. homodimer, heterotrimer, etc.), any diseases associated with any number of deficiencies in the protein, and information on its variations and similarities to other proteins. This information is obtained from the publications that

[1] A polypeptide chain comprises of multiple amino acids linked to each other forming the primary protein structure.

report new sequence data, and from articles that periodically update the annotations of families, or groups, of proteins. A panel of external experts are consulted to provide comments and update in a protein group specific manner.

Some of the key design goals of Swiss-Prot database are:

- **Minimal Redundancy:** Multiple reports in literature about the same sequence are merged to minimize redundancy of data. However, if conflicts are found among the various reports of the same sequence, these are indicated within the feature tables of the entry as annotations.
- **Integration:** It is important to provide the users of biomolecular databases with a degree of integration between the three types of sequence-related databases (nucleic acid sequences, protein sequences and protein tertiary structures) as well as with specialized data collections. Swiss-Prot is currently cross-referenced to more than 50 different databases. Cross-references are provided in the form of pointers to information related to Swiss-Prot entries and found in data collections other than Swiss-Prot. This extensive network of cross-references allows Swiss-Prot to play a major role as a focal point of biomolecular database interconnectivity.
- **Extensive Documentation:** This database is extensively documented with specialized document sets and provide various indices for fast reference.

Swiss-Prot Sequence Entry

An annotated image of the parts of a sequence entry is shown in Fig. 3.4. The Swiss-Prot entry name consists of up to 11 uppercase alphanumeric characters. Swiss-Prot uses a general purpose naming convention that can be symbolized as `X_Y`, where X is a mnemonic code of at most 5 alphanumeric characters representing the protein name. Examples: B2MG is for Beta-2-microglobulin; HBA is for Hemoglobin alpha chain; and INS is for Insulin. And Y is a mnemonic species identification code of at most 5 alphanumeric characters representing the biological source of the protein. This code is generally made of the first three letters of the genus and the first two letters of the species. For example, the code `ALLMI` in the entry name `HBB_ALLMI` denotes that the source of the Hemoglobin beta chain sequence (`HBB`) is *Alligator mississippiensis* or the American alligator; similarly, the source of dopamine precursor sequence `DOPO_BOVIN` is *Bos taurus* or bovine.

Other sections of the sequence entry record are described below:

- *Date:* The first DT line indicates when the entry first appeared in the database. The associated comment, *integrated into UniProtKB / database_name*, indicates in which section of UniProtKB, Swiss-Prot or TrEMBL, the entry can be found. The second DT line indicates when the sequence data was last modified. The third DT line indicates when data other than the sequence was last modified.

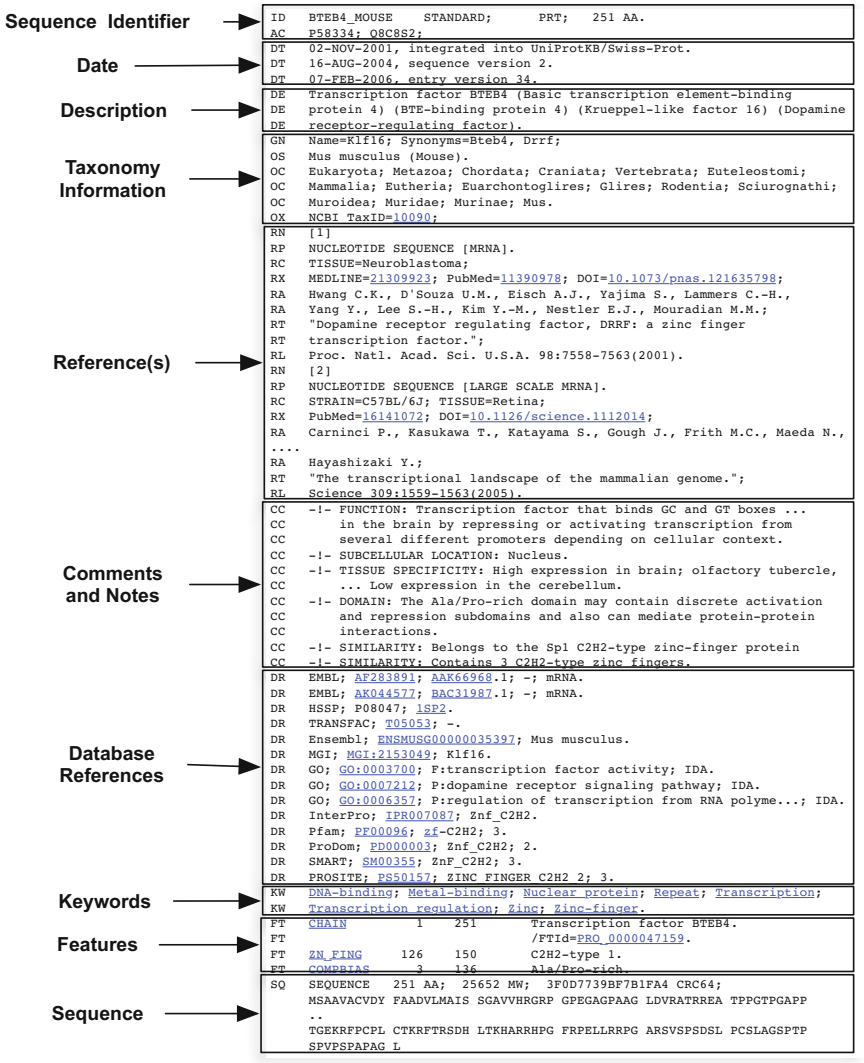


Fig. 3.4 Swiss-Prot sequence Entry

- *Description:* The DE (DEscription) lines contain general descriptive information about the sequence stored. The description always starts with the proposed official name of the protein. Synonyms are indicated between brackets.
- *Taxonomy Information:* The taxonomical and source information is comprised of GN, OS, OG, OC and OX lines. The GN (Gene Name) line

indicates the name(s) of the gene(s) that code for the stored protein sequence. The OS (Organism Species) line specifies the organism which was the source of the stored sequence. The OG (OrGanelle) line indicates if the gene coding for a protein originates from the mitochondria, the chloroplast, the cyanelle, the nucleomorph or a plasmid. The OC (Organism Classification) lines contain the taxonomic classification of the source organism. The taxonomic classification used is that maintained at NCB5 which is also used by the nucleotide sequence databases (EMBL/GenBank/DDBJ). The OX (Organism taxonomy cross-reference) line is used to indicate the identifier of a specific organism in a taxonomic database.

- *Reference(s):* The references section is comprised of RN, RP, RC, RX, RA, RT and RL lines. The RN (Reference Number) line gives a sequential number to each reference citation in an entry. The RP (Reference Position) lines describe the extent of the work relevant to the entry. The RC (Reference Comment) lines store comments relevant to the reference cited. The RX (Reference cross-reference) line indicates the identifier assigned to a specific reference in a bibliographic database. The RA (Reference Author) lines list the authors of the paper (or other work) cited. The RT (Reference Title) lines give the title of the paper (or other work) cited. The RL (Reference Location) lines contain the conventional citation information for the reference.
- *Comments:* The CC lines are free text comments on the entry, and are used to convey any useful information. The comments always appear below the last reference line and are grouped together in comment blocks; a block is made up of 1 or more comment lines.
- *Database References:* The DR (Database cross-Reference) lines are used as pointers to information related to entries and found in data collections other than Swiss-Prot.
- *Keywords:* The KW (KeyWord) lines provide information that can be used to generate indexes of the sequence entries based on functional, structural, or other categories. The keywords chosen for each entry serve as a subject reference for the sequence.
- *Feature Table Data:* The FT (Feature Table) lines provide a precise but simple means for the annotation of the sequence data. The table describes regions or sites of interest in the sequence. In general the feature table lists post-translational modifications, binding sites, enzyme active sites, local secondary structure or other characteristics reported in the cited references. Sequence conflicts between references are also included in the feature table. The FT lines have a fixed format. The column numbers allocated to each of the data items within each FT line are shown in the following table (column numbers not referred to in the table are always occupied by blanks).
- *Sequence:* The SQ (SeQuence header) line marks the beginning of the sequence data and gives a quick summary of its content. The format of the SQ line is:

Table 3.1 Feature Table Columns

Columns	Information
1–2	FT (Code)
6–13	Key Name (Example: ZN_FING). Controlled vocabulary.
15–20	From (end point)
22–27	To (end point)
35–75	Description

```
SQ   SEQUENCE XXXX AA; XXXXX MW; XXXXXXXXXXXXXXXX CRC64;
```

The sequence data line has 60 amino acids per line, in groups of 10 amino acids, beginning in position 6 of the line.

3.2.2 PIR

The Protein Information Resource (PIR), located at Georgetown University Medical Center, USA, was established in 1984 by the National Biomedical Research Foundation (NBRF) as a resource to assist researchers in the identification and interpretation of protein sequence information. The NBRF compiled the first comprehensive collection of macro-molecular sequences in the *Atlas of Protein Sequence and Structure* published from 1965-1978 under the editorship of Dr. Margaret O. Dayhoff.

Dr. Dayhoff and her research group pioneered in the development of computer methods for the comparison of protein sequences, for the detection of distantly related sequences and duplications within sequences, and for the inference of evolutionary histories from alignments of protein sequences. The protein superfamily concept, introduced in 1975, is used to organize the database. Since the last four decades the Protein Information Resource has been a community resource that provides protein databases and analysis tools to support research on molecular evolution, functional genomics, and computational biology.

Today, PIR offers a wide variety of resources mainly oriented to assist the propagation and standardization of protein annotation. Among these are:

- **PIRSF:** provides curated protein families with rules for functional site and protein name. The PIRSF protein classification system is a network with multiple levels of sequence diversity from super-families to subfamilies that reflects the evolutionary relationship of full-length proteins and domains
- **iProLink:** supports text mining in the area of literature-based database curation, named entity recognition, and protein ontology development
- **iProClass:** contains value-added annotation reports for proteins and serves as an integrated resource that provides comprehensive family relationships and structural/functional features of proteins

Additionally, PIR maintains the *PIR-International Protein Sequence Database (PIR-PSD)* and *PIR Non-Redundant Reference Sequence Database (PIR-NREF)*. The `PIR-PSD` is an international database that is both a comprehensive and expertly annotated protein sequence database available in the public domain. The `PIR-NREF` is a database that contains all sequences in PIR-PSD, SwissProt, TrEMBL, RefSeq, GenPept, and PDB. Identical sequences from the same source organism (species) reported in different databases are presented as a single NREF entry with protein IDs and names from each underlying database, in addition to protein sequence, taxonomy, and composite bibliography.

3.2.3 GenPept

The GenPept, or GenBank Gene Products protein database is provided in a format similar to the sequences distributed in GenBank. GenPept is maintained at NCBI and is accessible via Entrez which is a universal web application for accessing all of NCBI's database collection. GenPept is a protein database that is comprised of sequences that are derived from automated translations of coding regions in DNA sequences stored in the nucleotide database, GenBank. It also includes protein sequences gleaned from journal scans. As such, GenPept may be thought of as a proteomic version of GenBank.

3.2.4 UniProt Knowledgebase

Until recently, the EBI/SIB Swiss-Prot, TrEMBL, PIR Protein Sequence Database (PIR-PSD) coexisted as protein databases with differing protein sequence coverage and annotation priorities. In 2002, PIR, along with its international partners, EBI (European Bioinformatics Institute) and SIB (Swiss Institute of Bioinformatics), were awarded a grant from the National Institute of Health, USA, to create UniProt, a single worldwide database of protein sequence and function, by unifying the PIR, Swiss-Prot, and TrEMBL database activities. This led to the creation of the UniProt consortium. The primary mission of the consortium is to support biological research by maintaining a high quality database that serves as a stable, comprehensive, fully classified, richly and accurately annotated protein sequence *knowledgebase*, with extensive cross-references and querying interfaces freely accessible to the scientific community.

The UniProt is maintained collaboratively by the Swiss Institute for Bioinformatics (SIB) and the European Bioinformatics Institute (EBI) and consists of two sections: Swiss-Prot - a section containing manually-annotated records with information extracted from literature and curator-evaluated computational analysis, and TrEMBL - a section with computationally analyzed records that await full manual annotation. The UniProtKB/Swiss-Prot

Protein Knowledgebase is a curated protein sequence database that provides a high level of annotation, a minimal level of redundancy and a high level of integration with other databases. Together with UniProtKB/TrEMBL, it constitutes the UniProt Knowledgebase, one component of the Universal Protein Resource (UniProt), intended to be *portal* allowing easy access to all publicly available information about protein sequences – a central database of protein sequences with accurate, consistent, rich sequence and functional annotation.

3.3 Biological Patterns Databases

This section provides a brief background of the various databases that store biological patterns. Most of these databases contain evolving information, and have thus gone through several revisions since they were first introduced in mid-1980s. From their earlier emphasis on the specification of motif consensus, their present focus is towards an integration of functional information. As DNA level patterns play a vital role in the process of differential gene regulation, versions of the TRANSFAC database described below that contain certain specialized patterns are available only in the version of the database that users must subscribe to access.

3.3.1 PROSITE

The PROSITE database and tools are available at `http://www.expasy.org/`. The PROSITE database is a compilation of sites and patterns found in protein sequences that are described as regular expressions or profiles. The information about the biologically significant patterns and profiles contained in the database enables one to establish the family of protein (if any) to which a new sequence belongs, or which known domain(s) it contains. Sequence patterns stored in PROSITE database are particularly useful when the pairwise alignment does not yield a significant match between a pair of protein sequences. Furthermore, sequence alignment programs fail to make distinctions between an amino acid at an important active site and an amino acid with no critical role. Thus, in comparison to the alignment based functional characterization, PROSITE offers some distinct advantages.

The PROSITE uses two kinds of descriptors to characterize patterns, namely, regular expressions and weight matrices or profiles. A regular expression entry in PROSITE is shown in Fig. 3.5. A regular expression yields a boolean result. i.e. the pattern either matches or does not match. The regular expression does allow the representation of adequate variability in short patterns such a enzymatic catalytic sites, prosthetic group attachment sites, metal ion binding sites, or regions involved in molecule binding. The regular expression based patterns offer intelligibility. For example, a PROSITE regular expression for a S-100/ICaBP type calcium binding protein signature

is shown in Fig. 3.5. The S-100 are small dimeric acidic calcium and zinc-binding proteins abundant in the brain. In most cases the function of these proteins is not yet known, although it is becoming clear that they are involved in cell growth and differentiation, cell cycle regulation and metabolic control. The pattern shown unambiguously pick up proteins belonging to this family by detecting the EF region of high Calcium affinity.

```
PROSITE: PDOC00275

-Consensus pattern: [LIVMFYW](2)-x(2)-[LK]-D-x(3)-[DN]-x(3)-
                    [DNSG]-[FY]-x-[ES]-[FYVC]-x(2)-[LIVMFS]-[LIVMF]
```

Fig. 3.5 A regular expression pattern entry in PROSITE

It may be noted that the PROSITE uses an extended notation for representing regular expression, where a term like `[LIVMFYW](2)` denotes the occurrence of two residues from the set {`L,I,V,M,F,Y,W`}. The residue x denotes a wild-card, and "-" serve a connectives in the regular expression. Thus, a motif `LL-LD-K-D-LDL-D-LDL-N-F-D-E-F-DN-L-L` will match the above regular expression. Note that the dashes are only shown to enhance clarity, and do not denote the indel characters as often used in alignment depiction.

As shown in Fig. 3.6, the profile table in PROSITE is a position specific amino acid weights and gap cost matrix. The initial section of such a weight matrix is shown. Heme-binding peroxidases carry out a variety of biosynthetic and degradation functions using hydrogen peroxide as the electron acceptor. Since the entire model has 575 states, only the initial set of states of the model identified by PROSITE accession `PS50292` shown is an animal heme peroxidase superfamily profile. This profile was generated by creating a multiple sequence alignment of 192 entries from Swiss-Prot, TrEMBL and TrEMBL-NEW, which are linked to and directly accessible from this entry in PROSITE.

The quantitative behavior of a profile allows acceptance of a mismatch at a highly conserved position if the rest of the sequence exhibits a sufficiently high level of similarity. Thus, the detection of poorly conserved domains such as immunoglobulin domains, `SH2` or `SH3` is possible. Moreover, the profile based descriptions of the patterns span a much larger region compared to the regular expressions, and thus are able to quantitatively characterize the entire protein family.

```
ID   PEROXIDASE_3; MATRIX.
AC   PS50292;
DT   NOV-2002 (CREATED); NOV-2002 (DATA UPDATE); NOV-2002 (INFO UPDATE).
DE   Animal heme peroxidase superfamily profile.
MA   /GENERAL_SPEC: ALPHABET='ABCDEFGHIKLMNPQRSTVWYZ'; LENGTH=575;
MA   /DISJOINT: DEFINITION=PROTECT; N1=6; N2=570;
MA   /NORMALIZATION: MODE=1; FUNCTION=LINEAR; R1=2.2956; R2=0.01870166; TEXT='-LogE';
MA   /CUT_OFF: LEVEL=0; SCORE=359; N_SCORE=9.0; MODE=1; TEXT='!';
MA   /CUT_OFF: LEVEL=-1; SCORE=224; N_SCORE=6.5; MODE=1; TEXT='?';
MA   /DEFAULT: M0=-8; D=-50; I=-50; B1=-1000; E1=-1000; MI=-105; MD=-105; IM=-105; DM=-105;
MA   /I: B1=0; BI=-105; BD=-105;
MA   /M: SY='P';M=-2,-10,-24,-12,-8,-14,-10,-15,-9,-9,-12,-7,-7,2,-10,-11,-3,-3,-8,-24,-14,-10;
MA   /M: SY='P';M=-6,-7,-25,-6,-5,-15,-19,-13,-6,-6,-9,-5,-7,2,-8,-8,-6,-4,-6,-26,-12,-8;
MA   /M: SY='T';M=-6,-4,-20,-7,-5,-12,-17,-11,-7,-6,-9,-6,-1,-11,-6,-5,0,4,-4,-28,-11,-7;
MA   /M: SY='C';M=-7,-16,65,-24,-23,-14,-24,-22,-17,-23,-12,-9,-14,-32,-22,-23,-6,-6,-5,-40,-20,-23;
MA   /I: I=-9; MD=-19;
```

Fig. 3.6 Position specific amino acid weights and gap costs corresponding to a PROSITE entry

3.3.2 TRANSFAC: Transcription Factors and Regulation

The development path for the TRANSFAC database has been geared by the objective of providing a biological context for understanding the function of regulatory signals found in genomic sequences. The aim of this compilation of signals was meant to provide all relevant data about regulating proteins and allow researchers to trace back transcriptional control cascades to their origin [5, 16]. The TRANSFAC database contains information about regulatory DNA sequences and the transcription factors binding to and acting through them. At the core of this database are its components describing the transcription factor (FACTOR), its corresponding binding site (SITE) and the regulation of the corresponding gene (GENE). The GENE table is one of the central tables in this database. It is linked to several other databases including S/MARtDB, TransCOMPEL, LocusLink, OMIM, and RefSeq.

Sites are experimentally proven for their inclusion in the database. The experimental evidence of the transcription factor and the DNA-binding site is described, and the cell type from which the factor is derived is linked to the respective entry in the CELL table. A set of weight matrices are derived from the collection of binding sites. These matrices are recorded in the MATRIX table. Moreover, as determined by their DNA-binding domain, the transcription factors are assigned to a certain class, and hence link to the CLASS table is established.

This databases is accessible from `http://www.gene-regulation.com/`. As an example, consider the following somewhat edited entry from the SITES table in TRANSFAC shown in Fig. 3.7. The entry provides a wide variety of information about the transcription factor, such as the binding sequence motif (SQ) and the first (SF) and the last position (ST) of the factor binding site. The accession number of the binding factor itself is provided (BF) - this is in fact a key for the FACTOR table in TRANSFAC. The source of the factor

```
AC   R00026
TY   D
DE   CA-ACT (cardiac alpha-actin); G000193.
SQ   CCAAATAAGG.
SF   -113
ST   -84
BF   T00765 SRF (504 AA); Quality: 4; Species: mouse, Mus musculus.
OS   human, Homo sapiens
OC   eukaryota; animalia; metazoa; chordata; vertebrata; tetrapoda; mammalia;
OC   eutheria; primates
SO   0003 3T3
SO   0042 C2 myoblasts
SO   0069 F9
MM   gel retardation
DR   EMBL: M13483; HSACTCA (377:386).
DR   EPD: EP16033; HS_ACTC.
RN   [1]
RX   MEDLINE; 89093119.
RA   Boxer L. M., Miwa T., Gustafson T. A., Kedes L.
RT   Identification and Characterization of a Factor That Binds to Two Human
RT   Sarcomeric Actin Promoters
RL   J. Biol. Chem. 264:1284-1292 (1989).
```

Fig. 3.7 TRANSFAC table for storing sites

is identified (SO). The specific type of cells where the factor was found to be active are identified, 3T3, C2 myoblasts, and F9 in this case. Additional information about these cells is accessible under the CELL table with the accession numbers of 0003, 0042 and 0069 respectively. External database references and their corresponding accession numbers are provided under the (DR) field, as well as publication titles (RT) and citation information (RL).

The table MATRIX table shown in Fig. 3.8 within the TRANSFAC represents the binding site data as a weight matrix or a profile. If a binding site contributes to the construction of a matrix, is indicated by including a MX field in the site's record. Most sites, however, do not have such a reference, as the matrices are typically built from artificially generated consensus sequences. For example, let us look at the somewhat edited matrix record shown in Fig. 3.8. Each of the 14 9-bp binding sites used in the construction of this matrix are artificially generated sequences, possibly through the consensus of multiple experiments.

The above database of transcription factor binding sites is utilized; either directly by running a search of the consensus sequences as a simple text search, or through the analysis of a matching program that looks for the high likelihood of occurrence of the pattern specified by the profile weight matrix characterizing the binding site.

```
AC   M00395
NA   HOXA3
DE   HOXA3 (homeobox cluster protein)
BF   T00378 HOXA3; Species: mouse, Mus musculus.
PO      A      C      G      T
01      1      9      3      1      C
02      3      5      5      1      N
03      1      1      1     11      T
04      9      0      2      3      A
05      6      2      2      4      N
06      3      1      3      7      N
07      4      1      4      5      N
08      1      1      7      5      K
09      1      3      5      5      N
BA   14 sites selected from random oligonucleotides
BS   R09080 Start: 4; Length: 9; Gaps:; Orientation: p.
BS   R09081 Start: 3; Length: 9; Gaps:; Orientation: p.
BS   R09082 Start: 7; Length: 9; Gaps:; Orientation: p.
BS   R09083 Start: 3; Length: 9; Gaps:; Orientation: p.
BS   R09084 Start: 4; Length: 9; Gaps:; Orientation: p.
BS   R09085 Start: 8; Length: 9; Gaps:; Orientation: p.
BS   R09086 Start: 5; Length: 9; Gaps:; Orientation: p.
BS   R09087 Start: 3; Length: 9; Gaps:; Orientation: p.
BS   R09088 Start: 4; Length: 9; Gaps:; Orientation: p.
BS   R09089 Start: 5; Length: 9; Gaps:; Orientation: p.
BS   R09090 Start: 3; Length: 9; Gaps:; Orientation: p.
BS   R09091 Start: 4; Length: 9; Gaps:; Orientation: p.
BS   R09092 Start: 3; Length: 9; Gaps:; Orientation: p.
BS   R09093 Start: 5; Length: 9; Gaps:; Orientation: p.
CC   14 bp random sequence, 4 rounds of selection
```

Fig. 3.8 TRANSFAC matrix table

3.4 Genome Viewer

The University of California Genome Browser which includes genome tracks. Each track provides a different type of feature including genes, SNP, CpG islands, etc. This browser is accessible at `http://www.ucsc.edu`. Fig. 3.9 illustrates the result of searching for globin genes in the human genome.

The National Center for Genome Resources has a genome viewer. This is accessible at `http://www.ncbi.nlm.nih.gov/mapview/`. The NCBI genome browser is a comprehensive collection of genomes. The browser is well connected with the other NCBI resources. The National Center for Biotechnology Information maintains information on complete genomes. Genome level browsers provide information on complete genomes for humans, other eukaryotes including plants, as well as prokaryotes. Genome level resources are accessible at `http://ncbi.nlm.nih.gov/genome`. This site lists the entire set of genome level information that is available. In addition to an overall guide, genome level information is available for each organism under the categories of mapview, BLAST, and projects.

As an example, select the mapview category for the human genome. Typing in *collagen* in the search box produces the following display. As illustrated in Fig. 3.10, the map of the human chromosomes is annotated with markers indicates the specific locations where genes related to collagen are encoded on

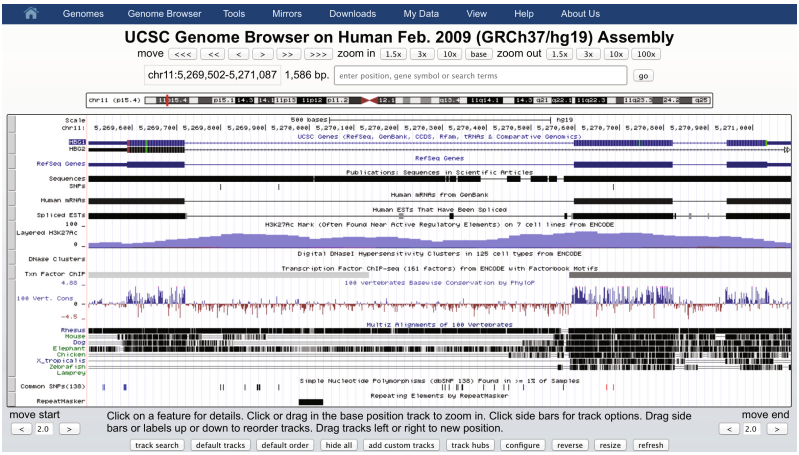


Fig. 3.9 UCSC Genome Browser

the genome. Detailed information for each of these genes is also in the lower panel (not shown in the illustration). The locations of genes on the various human chromosomes is shown and is hyperlinked to the related resources.

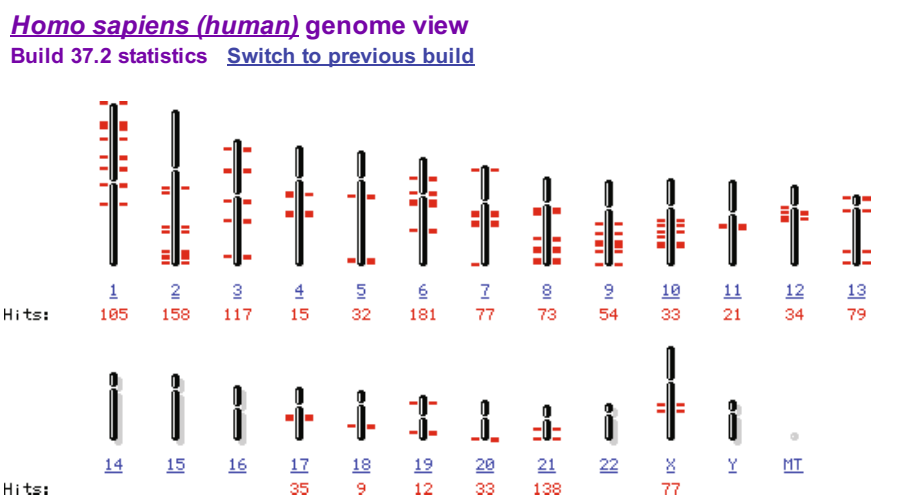


Fig. 3.10 Map View for the Human Genome where *collagen* associated genes are highlighted

3.5 Gene Ontology Database

The Gene Ontology database aims to provide a controlled vocabulary for describing many functions, processes and components. The GO project has developed three structured controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a *species-independent manner*. The use of GO terms facilitates uniform queries across collaborating databases. The ontologies are structured as directed acyclic graphs where a child, or more specialized term, can have many parent (general) terms. For example, the biological process term hexose biosynthesis has two parents, hexose metabolism and monosaccharide biosynthesis. This structure enables queries that involve general as well as specialized terms. One can thus look for all of the gene products in the human genome that are involved in hexose metabolism, which in turn will include genes involved in hexose biosynthesis. Each entry in GO has a unique numerical identifier of the form GO:nnnnnnn, and an associated term such as *cell*, *fibroblast growth factor receptor binding*, *hexose metabolism*, etc. Each term is also assigned to one of the three ontologies, molecular function, cellular component or biological process. The terms in an ontology are linked by two relationships, *IS-A* and *PART-OF* representing a simple class-subclass relationship and the part-subpart relationship respectively. The Gene Ontology Consortium makes the GO database available for download at `http://www.geneontology.org/`. Feature annotations in GenBank also contain references to GO terms where additional details on a given biological term may be found. Due to the growing significance of gene ontology in inference of biological functions, it is important to review capabilities that MATLAB provides in processing gene ontology related information.

The Gene Ontology project, or GO, provides a controlled vocabulary to describe gene and gene product attributes in any organism. It can be broadly split into two parts. The first is the ontology itself–actually three ontologies, each representing a key concept in Molecular Biology: the molecular function of gene products; their role in multi-step biological processes; and their localization to cellular components. The second part is annotation, the characterization of gene products using terms from the ontology.

3.5.1 Go Terms

Each GO term consists of a unique alphanumerical identifier, a common name, synonyms (if applicable), and a definition. Terms are classified into only one of the three ontologies. Each ontology is structured as a directed acyclic graph (DAG). Sometimes a term may have multiple meaning based on species; when GO uses a 'sensu' tag to differentiate among them. The Gene Ontology project also provides mappings of its terms to other classification systems covering the same areas of biology.

GO consortium reviews and may add any new terms suggested by the members of the research community. Similarly, the consortium may deprecated, or marked as "obsolete" when they loose their significance or have become misleading. The ontology file is freely available from the GO website; the terms can be searched and browsed online using the GO browser AmiGO.

3.5.2 Associations

A number of organizations, including model organism databases and large multi-species protein databases, perform analyses of protein sequences and issue tables of associations between putative gene products and GO terms. These are freely available from the GO website and can be downloaded individually or viewed online using AmiGO.

In many older genetic sequence databases, annotations bear little or no indication of their provenance so that a user cannot readily ascertain the nature and strength of the evidence behind them, which leads to what is known in the field as the 'transitive annotation problem.' Some gene is characterized by actual wet lab experiments, and its sequence deposited in a major public database with annotation from those experiments. Other sequences that have not been characterized in the lab are annotated based on their sequence similarity to this one, and these other sequences in turn form the basis for yet more annotations, and so forth. Thus a user cannot know how many steps of sequence similarity stand between the annotation for some genetic sequence and any actual wet-lab data.

A GO association has metadata indicating:

- Who made the assertion that this GO term applies to the putative product of this protein sequence,
- When this assertion was made,
- One or more three-letter Evidence code(s) denoting the type of evidence on which this assertion is based.

Any automatic program output that has not been curated by a human being gets the evidence code IEA meaning Inferred from Electronic Annotation. The use of a code other than IEA implies that a human curator has checked this annotation. For instance TAS for Traceable Author Statement means a curator has read a published scientific paper and the metadata for that association bears a citation to that paper. On the other hand, ISS for Inferred from Sequence Similarity means a human curator has reviewed the output from a sequence similarity search and verified that it is biologically meaningful. The Gene Ontology project, or GO, provides a controlled vocabulary to describe gene and gene product attributes in any organism. It can be broadly split into two parts.

The first is the ontology itself–actually three ontologies, each representing a key concept in Molecular Biology: the molecular function of gene products;

their role in multi-step biological processes; and their localization to cellular components.

The second part is annotation, the characterization of gene products using terms from the ontology. A number of organizations, including model organism databases and large multi-species protein databases, perform analyses of protein sequences and issue tables of associations between putative gene products and GO terms. These are freely available from the GO website and can be downloaded individually or viewed online using AmiGO.

Associations in GO are supported by evidence. Table 3.2 shows the set of codes used to annotate the evidentiary support of an association of a gene product to a GO term.

Table 3.2 Evidence Codes for GO Associations

IC	Inferred by curator
TAS	Traceable author statement
IDA	Inferred from direct assay
IMP	Inferred from mutant phenotype
IGI	Inferred from genetic interaction
IPI	Inferred from physical interaction
ISS	Inferred from sequence or structural similarity
IEP	Inferred from expression pattern
NAS	Non-traceable author statement
RCA	Inferred from reviewed computational analysis
IEA	Inferred from electronic annotation
NR/ND	Not recorded, No biological data available

In many older genetic sequence databases, annotations bear little or no indication of their provenance so that a user cannot readily ascertain the nature and strength of the evidence behind them, which leads to what is known in the field as the 'transitive annotation problem.' Some gene is characterized by actual wet lab experiments, and its sequence deposited in a major public database with annotation from those experiments. Other sequences that have not been characterized in the lab are annotated based on their sequence similarity to this one, and these other sequences in turn form the basis for yet more annotations, and so forth. Thus a user cannot know how many steps of sequence similarity stand between the annotation for some genetic sequence and any actual wet-lab data.

A GO association has metadata indicating:

‡ Who made the assertion that this GO term applies to the putative product of this protein sequence,
‡ When this assertion was made,
‡ One or more three-letter Evidence code(s) denoting the type of evidence on which this assertion is based.

Any automatic program output that has not been curated by a human being gets the evidence code IEA meaning Inferred from Electronic Annotation. The use of a code other than IEA implies that a human curator has checked this annotation. For instance TAS for Traceable Author Statement means a curator has read a published scientific paper and the metadata for that association bears a citation to that paper. On the other hand, ISS for Inferred from Sequence Similarity means a human curator has reviewed the output from a sequence similarity search and verified that it is biologically meaningful.

3.5.3 MATLAB Interface to GO

The latest version of Gene Ontology database, available from `http://www.geneontology.org` is downloaded over the Web into MATLAB using the function *geneont* when the `LIVE` parameter is *true*. The ontology file is located at `http://www.geneontology.org/ontology/gene_ontology.obo` In the sample code below, the gene ontology object is created as a structure variable `GO`. Information about this ontology is provided by the MATLAB function *get* which displays the number of terms in the database. In the listing below, the Gene Ontology object contains 41,211 terms.

```
>> GO = geneont('LIVE', true);

% Display information about the object GO

>> get(GO)
                  default_namespace: 'gene_ontology'
                     format_version: '1.0'
                       data_version: ''
                            version: ''
                               date: '14:06:2014 00:36'
                          saved_by: 'jenkins-slave'
                  auto_generated_by: ''
                           subsetdef: {18x1 cell}
                             import: ''
                      synonymtypedef: ''
                            idspace: ''
    default_relationship_id_prefix: ''
                         id_mapping: ''
                             remark: [1x129 char]
                            typeref: ''
                 unrecognized_tag: {3x2 cell}
                              Terms: [41211x1 geneont.term]
```

Searching

Ontology objects constructed as describe above may be searched using search terms which may be specified as regular expressions. For example, the gene ontology object `GO` that was created may be searched for the terms that contain the term `'ribosome'`. The function *regexpi* searches for the occurrence of property `name` for the occurrence of this regular expression. The result is a an array `comparison` which is equal in size to the array `Terms` and which contains either an empty set or the location of match in the corresponding cell of the array `Terms`. The indices of the non-empty cells in the array `comparison` is stored in the array `indices`. The set of terms corresponding to these indices is next retrieved.

```
>> comparison = regexpi(get(GO.Terms,'name'), 'ribosome');
>> indices = find(~cellfun('isempty',comparison));
>> terms_with_ribosmome = GO.Term(indices)
  33x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

Ancestors, Decedents and Relatives

After a GO object is constructed, one can use the various MATLAB functions to traverse the ancestors, descendants and relatives of any node. Further, the nodes of an ontology object thus traversed may in turn may be used to construct a sub-ontology. This is illustrated in the following sample sub-ontologies created created by ancestors, descendants and relatives of `GO:46680`.

```
% Get the ancestors for a Gene Ontology term.
>> ancestors = getancestors(GO, 46680)

ancestors =

        8150
        9636
       10033
       14070
       17085
       42221
       46680
       50896
```

```
% Create a sub Gene Ontology.

>> subontology = GO(ancestors)

Gene Ontology object with 6 Terms.

% Get the descendants for a GO term. The maximum depth is set to 5.
>> descendants = getdescendants(GO, 46680, 'Depth', 3);

% Create a sub Gene Ontology.
subontology = GO(descendants)

Gene Ontology object with 8 Terms.

% Get relatives for a GO term. Includes both ancestors and descendants

relatives = getrelatives(GO, 46680, 'Height', 3, 'Depth', 2);
subontology = GO(relatives)
Gene Ontology object with 6 Terms.

>> so = get(subontology);
>> firstTerm = so.Terms(1)
            id: 9636
          name: 'response to toxic substance'
      ontology: 'biological process'
    definition: [1x202 char]
       comment: ''
       synonym: {3x2 cell}
          is_a: 42221
       part_of: [0x1 double]
      obsolete: 0
```

Matrix of Relationships

The MATLAB function *getmatrix* converts a gene-ontology object into a matrix of relationship values between nodes. A value of 0 in this matrix indicates no relationship, a value of 1 indicates an "is_a" relationship, and a value of 2 indicates a "part_of" relationship. The function also returns a column vector listing the GO identifiers corresponding to the rows and columns of the matrix returned. In the listing below, an gene ontology object is created using the terms that contain the regular expression discussed earlier:

```
>> ribont = GO(terms_with_ribosmome)
>> [MATRIX, ID] = getmatrix (ribont);
```

The matrix of relationships, `MATRIX`, and the term identifiers `ID`, may be used in conjunction with the *biograph* object to display this ontology. Notice that the GO term identifiers which are are returned as numbers by the *getmatrix* are converted into text strings by the function *num2goid* below.

```
>> bg = biograph (MATRIX, num2goid(ID));
>> view (bg)
```

Annotations

Annotations are uploaded into MATLAB from a file. Gene Ontology Consortium makes several annotation files available on their web site. A list of current gene annotations is available from:

`http://www.geneontology.org/GO.current.annotations.shtml`

Download the file containing GO annotations for the gene products of *homo sapiens*. Assume that this file is downloaded to `gene_association.goa_human.gz` to your MATLAB Current Directory. The file may be unzipped and read into structure `HumanGenes` by the following commands:

```
% Uncompress the file using the gunzip function.

>> gunzip('gene_association.goa_human.gz')

% Read the file into MATLAB.
>> HumanGenes = goannotread('gene_association.goa_human');

% Create a structure with GO annotations and get a list of genes.
>> S = struct2cell(HumanGenes);
>> genes = S(3,:)'
```

3.5.4 Example

Characterizing the Ontology Database: Load in the gene ontology and develop a histogram of the various categories of terms. I.e. , a plot depicting the number of terms in each of the three ontologies, molecular function, biological process, and cellular component.

```
GO = geneont('LIVE', true);
onto = get(GO.Terms, 'ontology');
mole_func = regexpi(onto, 'molecular function');
bio_proc = regexpi(onto, 'biological process');
cell_comp = regexpi(onto, 'cellular component');

index_mol = find(~cellfun('isempty', mole_func));
index_bio = find(~cellfun('isempty', bio_proc));
index_cell = find(~cellfun('isempty', cell_comp));

mole_func_num = length(index_mol);
bio_proc_num = length(index_bio);
cell_comp_num = length(cell_comp);

GeneOnt= [mole_func_num bio_proc_num cell_comp_num];
bar(GeneOnt);
```

Characterizing the Annotations: Associations in GO are supported by evidence. Table 3.2 shows the set of codes used to annotate the evidentiary support of an association of a gene product to a GO term.

```
HumanGenes = goannotread('gene_association.goa_human');
S = struct2cell(HumanGenes);
evidence = S(7,:);  %Evidence code is in the 7th cell
evidence = evidence';   %become column vectors
index_IC = find(~cellfun('isempty', regexpi(evidence, 'IC')));
index_TAS = find(~cellfun('isempty', regexpi(evidence, 'TAS')));
index_IDA = find(~cellfun('isempty', regexpi(evidence, 'IDA')));
index_IMP = find(~cellfun('isempty', regexpi(evidence, 'IMP')));
index_IGI = find(~cellfun('isempty', regexpi(evidence, 'IGI')));
index_IPI = find(~cellfun('isempty', regexpi(evidence, 'IPI')));
index_ISS = find(~cellfun('isempty', regexpi(evidence, 'ISS')));
index_IEP = find(~cellfun('isempty', regexpi(evidence, 'IEP')));
index_NAS = find(~cellfun('isempty', regexpi(evidence, 'NAS')));
index_RCA = find(~cellfun('isempty', regexpi(evidence, 'RCA')));
index_IEA = find(~cellfun('isempty', regexpi(evidence, 'IEA')));

IC = length(index_IC);
TAS = length(index_TAS);
IDA = length(index_IDA);
IMP = length(index_IMP);
IGI = length(index_IGI);
IPI = length(index_IPI);
ISS = length(index_ISS);
IEP = length(index_IEP);
NAS = length(index_NAS);
RCA = length(index_RCA);
IEA = length(index_IEA);

EvidCodes = [IC TAS IDA IMP IGI IPI ISS IEP NAS RCA IEA];
bar(EvidCodes);
```

The result of performing the analysis in the example above is shown in Fig. 3.11. Sometimes it is also desirable to link Annotations with Ontology Structure. In order to achieve this goal, the list of relatives for a gene of interest are first determined. Next, these relatives are partitioned based on ontology. For each ontology (MF, BP or CC) the gene or product names that are related to gene may be graphically depicted.

3.6 Other Databases

There are more than 1500 different databases related to life science research. Generally these are all accessible through the web (`http://www.expasy.org/links.html`). These database also vary in size ranging anywhere from 100 KB to 100 GB and can track primary,

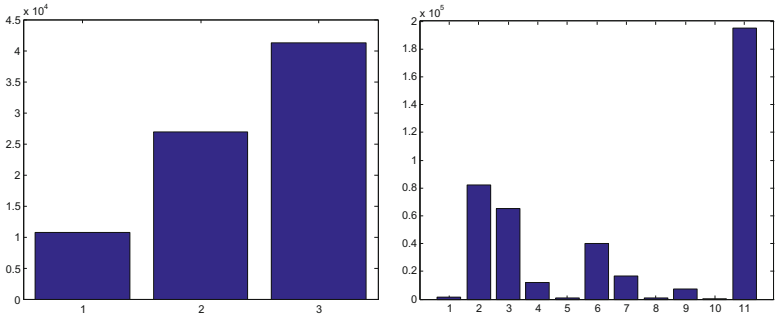


Fig. 3.11 Gene Ontology database distribution of entries. Bar charts representing the (a) distribution of the count of the three node types in the database, and (b) distribution of the count of the various evidence codes in the database.

secondary or functional information. Some databases, particularly the primary databases are updated daily while others which contain curated annotations may be updated monthly or even annually.

A comprehensive coverage of biological databases is beyond the scope of this text. Brief information on Reference Sequence Database, Gene Ontology, and Protein Structure Databases is presented below. Gene Ontology is an important step towards developing a unifying framework for analysis, interpretation and interchange of data related to life sciences. The Gene Ontology database is not a necessarily a secondary database as the tasks related to developing a hierarchical vocabulary are not necessarily dependent upon the availability of a primary database. On the other hand, the structure database provides information on the 3D structure of biomolecules. This necessarily is a secondary database as the structure information is another facet of the primary sequence information about the biomolecule.

3.6.1 RefSeq: NCBI Reference Sequences

The history of biological sequences and annotations has been driven by individual investigators' laboratories across the world depositing sequencing and annotation information into public databases which are in turn made accessible to the community at large. One problem with such world-wide collaborative approach is that a single biological entity may be represented by many different entities in the various databases. Sometimes, there isn't enough information included with the entries to establish reliability and consistency between the multiple entries that aren't in agreement. Further,

for some entries, the source of database entry – whether it was experimental or in-silico – is not easy to establish. The NCBI initiated the RefSeq project to address these issues and to provide a *reference sequence* for each type of sequence in the central dogma – nucleotide sequences, mRNA sequences, and protein sequences.

`RefSeq` provides a non-redundant representation for each type of biological entity. Therefore there is a single entry in `RefSeq` for each biological entity, for both DNA, RNA and protein sequences. `ReqSeq` entities undergo an ongoing curation process and therefore represent a *reliable* resource on the current state of knowledge and information about a biological entity. `ReqSeq` entries are explicitly linked to one another.

`RefSeq` entries are distinguished from other entries in `GenBank` by their use of characteristic accession numbers following a "2+6" format, with the first two characters of `ReqSeqs` being `NT`, `NM` or `NP` designating whether the reference sequence is a nucleotide, mRNA or protein sequences. For example, `NT_234671` is nucleotide reference sequence. Other distinguishing accession numbers are the `XM` and `XP` accession numbers representing the model mRNA and model protein sequences respectively that are derived from genome annotations.

In `RefSeq` database, for both protein or nucleotide `RefSeq` databases, will return a single entry in response to a query for a given gene or gene product. However, if there are several products corresponding to a gene corresponding to alternative splicing, entries in `RefSeq` will be included for each splice variant. As an example, there are three `RefSeq` entries (`NM_005368`, `NM_203377`, and `NM_203378`) corresponding to the three distinct splice variants of myoglobin, which is a gene responsible for binding the oxygen molecule in muscles. Corresponding to the three variants are the three transcripts stored in the protein `RefSeq` database. These are respectively identified by the accession numbers, `NP_005359`, `NP_976311`, and `NP_976312`.

3.6.2 ESTs and UniGene

Included within the various GenBank divisions is dbEST or the database of Expressed Sequence Tags containing information about the cDNA, or complementary DNA, sequences from many organisms. The cDNA sequence is derived by taking the RNA expressed in a tissue sample and converting it to a stable form of complementary DNA which is usually sequenced in a "single pass" without requiring the need for sequence assembly and are generally between 300–800 bp in length. The single pass cDNA is thus tag of the expressed sequence and is commonly referred to as an EST or Expressed Sequence Tag. All cDNA sequences, and thus all ESTs, are derived from a specific

tissue sample such as human liver or mice brain. A summary of ESTs contained in the dbEST division of GenBank is provided at
`http://www.ncbi.nlm.gov/dbEST/dbEST_summary.html`.

The human genome is thought to contain about 22,000 genes. The UniGene or Unique Gene Project aims at creating gene-oriented clusters that are obtained by partitioning of Expressed Sequence Tags into non-redundant sets such that there is a single unique gene cluster assigned to each gene for an organism. The redundancy of information expressed in UniGene is reflected in the number of members in that cluster. For example, a gene expressed in many tissues will have thousands of ESTs in that cluster, whereas a gene that is rarely expressed will have only a few ESTs in the corresponding UniGene cluster.

3.6.3 Structure Databases

The Protein Data Bank (PDB) is a web accessible database that contains information about the three-dimensional structures of large biological molecules. PDB contains structural information about both proteins and nucleic acids. Since structure and function are closely related, the PDB aims at bridging the gap between the function of large biomolecules and the role that they universally play in number of organisms including bacteria, yeast, plants, mice, and in healthy as well as diseased humans. The key to understanding the function of a molecule is its shape. The Protein Data Bank is maintained by the Research Collaboration for Structural Bioinformatics or RCSB. The RCSB is comprised of Rutgers University, the San Diego Supercomputing Center, the Center for Advanced Research in Biotechnology and the University of Wisconsin. The database provides software for viewing 3D structures as well. This resource is accessible at `http://www.pdb.org/`.

3.7 Summary

The journal *Nucleic Acid Research* publishes an issue each year in January which presents a survey of all major databases in molecular biology. The 2014 annual issue listed over 1500 databases [9]. Daily exchanges of data occur between GenBank , EMBL and DDBJ – the three collaboration databases collectively known as the International Nucleotide Sequence Database Collection (INSDC). The EMBL database provides third party annotations as well as automated procedures for uploading sequences. DDBJ, the DNA Databank of Japan, has classified all genes in the collaboration and captured Expressed Sequence Tags (ESTs) from the honeybee. A complete list of resources available at the NCBI is provided by Wheeler [10].

Barker provides a review of the protein sequence organization and features of the Protein Information Resource (PIR) [11] and describes the capability of feature-based or annotation-based retrieval in PIR [12]. Boeckmann describes

Swiss-Prot in [13] and discusses how protein entries in Swiss-Prot provide an interdisciplinary overview of relevant information by combining experimental results and computed features and offers detailed annotation of human protein sequences. The UniProt Consortium [14], formed in 2002, offers the ability to store and interconnect all available information on proteins that have been manually curated in a concerted effort by the European Bioinformatics Institute (EBI), the Protein Information Resource (PIR) and the Swiss Institute of Bioinformatics (SIB). Thus, the UniProt represents a unification for the protein databases similar to as the consortium of nucleotide databases does for nucleotide databases.

Hulo [15] provides information on PROSITE and protein signatures (patterns represented as extended regular expressions or as scoring matrices) that are linked to the protein family, domain or functional site identified by the signature. Information on the resources offered by the TRANSFAC as well as the various pieces of biological data that it interrelates is provided by Matys [16].

A comprehensive review of the Protein Data Bank (PDB) is provided by Berman [17]. The paper by Kouranov discusses the worldwide initiative in structural genomics and the online tools offered by RCSB [18]. The ability to search the PDB using a tool called PAST is discussed by Toubig [19]. A comprehensive review of the Gene Ontology (GO) project is provided in [20] and [21]. The hierarchical tree of GO terms may be viewed graphically by a web tool called WEGO as described by Ye [22].

Further Readings

1. Benson, D.A., Clark, K., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Sayers, E.W.: Genbank. Nucleic Acids Res 42(Database issue), D32–D37 (2014)
2. Pruitt, K.D., Maglott, D.R.: Refseq and locuslink: Ncbi gene-centered resources. Nucleic Acids Res 29(1), 137–140 (2001)
3. Sigrist, C.A., de Castro, E., Cerutti, L., Cuche, B.A., Hulo, N., Bridge, A., Bougueleret, L., Xenarios, I.: New and continuing developments at prosite. Nucleic Acids Res 41(Database issue), D344–D347 (2013)
4. Sigrist, C.A., Cerutti, L., de Castro, E., Langendijk-Genevaux, P.S., Bulliard, V., Bairoch, A., Hulo, N.: Prosite, a protein domain database for functional characterization and annotation. Nucleic Acids Res 38(Database issue), D161–D166 (2010)
5. Huntley, R.P., Sawford, T., Martin, M.J., O'Donovan, C.: Understanding how and why the gene ontology and its annotations evolve: the go within uniprot. Gigascience 3(1), 4 (2014)
6. The UniProt Consortium. Activities at the universal protein resource (uniprot). Nucleic Acids Res (June 2014)
7. Shoop, E., Casaes, P., Onsongo, G., Lesnett, L., Petursdottir, E.O., Donkor, E.K.Y., Tkach, D., Cosimini, M.: Data exploration tools for the gene ontology database. Bioinformatics 20(18), 3442–3454 (2004)

8. Roncaglia, P., Martone, M.E., Hill, D.P., Berardini, T.Z., Foulger, R.E., Imam, F.T., Drabkin, H., Mungall, C.J., Lomax, J.: The gene ontology (go) cellular component ontology: integration with sao (subcellular anatomy ontology) and other recent developments. J Biomed Semantics 4(1), 20 (2013)
9. Fernández-Suárez, X.M., Rigden, D.J., Galperin, M.Y.: The 2014 nucleic acids research database issue and an updated nar online molecular biology database collection. Nucleic Acids Res 42(Database issue), D1–D6 (2014)
10. Wheeler, D.L., Barrett, T., Benson, D.A., Bryant, S.H., Canese, K., Chetvernin, V., Church, D.M., Dicuccio, M., Edgar, R., Federhen, S., Feolo, M., Geer, L.Y., Helmberg, W., Kapustin, Y., Khovayko, O., Landsman, D., Lipman, D.J., Madden, T.L., Maglott, D.R., Miller, V., Ostell, J., Pruitt, K.D., Schuler, G.D., Shumway, M., Sequeira, E., Sherry, S.T., Sirotkin, K., Souvorov, A., Starchenko, G., Tatusov, R.L., Tatusova, T.A., Wagner, L., Yaschenko, E.: Database resources of the national center for biotechnology information. Nucleic Acids Res 36(Database issue), D13–D21 (2008)
11. Barker, W.C., Garavelli, J.S., McGarvey, P.B., Marzec, C.R., Orcutt, B.C., Srinivasarao, G.Y., Yeh, L.S., Ledley, R.S., Mewes, H.W., Pfeiffer, F., Tsugita, A., Wu, C.: The pir-international protein sequence database. Nucleic Acids Res 27(1), 39–43 (1999), P41 lm05798/lm/nlm Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, P.H.S. England
12. Barker, W.C., Garavelli, J.S., Huang, H., McGarvey, P.B., Orcutt, B.C., Srinivasarao, G.Y., Xiao, C., Yeh, L.S., Ledley, R.S., Janda, J.F., Pfeiffer, F., Mewes, H.W., Tsugita, A., Wu, C.: The protein information resource (pir). Nucleic Acids Res 28(1), 41–44 (2000), P41 lm05978/lm/nlm Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, P.H.S. England
13. Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.C., Estreicher, A., Gasteiger, E., Martin, M.J., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., Schneider, M.: The swiss-prot protein knowledgebase and its supplement trembl in 2003. Nucleic Acids Res 31(1), 365–370 (2003), Journal Article England.
14. UniProt Consortium. The universal protein resource (uniprot). Nucleic Acids Res 35(Database issue), D193–D197 (2007), niProt Consortium1 1 u01 hg02712-01/hg/nhgri 1r01hgo2273-01/phs Journal Article Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't England
15. Hulo, N., Bairoch, A., Bulliard, V., Cerutti, L., Castro, E.D., Langendijk-Genevaux, P.S., Pagni, M., Sigrist, C.J.: The prosite database. Nucleic Acids Res 34(Database issue), D227–D230 (2006), Journal Article Research Support, Non-U.S. Gov't England
16. Matys, V., Kel-Margoulis, O.V., Fricke, E., Liebich, I., Land, S., Barre-Dirrie, A., Reuter, I., Chekmenev, D., Krull, M., Hornischer, K., Voss, N., Stegmaier, P., Lewicki-Potapov, B., Saxel, H., Kel, A.E., Wingender, E.: Transfac and its module transcompel: transcriptional gene regulation in eukaryotes. Nucleic Acids Res 34(Database issue), D108–D110 (2006), Journal Article Research Support, Non-U.S. Gov't England
17. Berman, H., Henrick, K., Nakamura, H., Markley, J.L.: The worldwide protein data bank (wwpdb): ensuring a single, uniform archive of pdb data. Nucleic Acids Res 35(Database issue), D301–D303 (2007), GR062025MA/Wellcome Trust Lm05799/lm/nlm Historical Article Journal Article Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, Non-P.H.S. England

18. Kouranov, A., Xie, L., de la Cruz, J., Chen, L., Westbrook, J., Bourne, P.E., Berman, H.M.: The rcsb pdb information portal for structural genomics. Nucleic Acids Res 34(Database issue), D302–D305 (2006), Gm63208/gm/nigms Journal Article Research Support, N.I.H., Extramural Research Support, U.S. Gov't, Non-P.H.S. England
19. Taubig, H., Buchner, A., Griebsch, J.: Past: fast structure-based searching in the pdb. Nucleic Acids Res 34(Web Server issue), W20–W23 (2006), Journal Article England
20. Gene Ontology Consortium. The gene ontology (go) project in 2006. Nucleic Acids Res 34(Database issue), D322–D326 (2006), Gene Ontology Consortium Hg02273/hg/nhgri Journal Article Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't England
21. Harris, M.A., Clark, J., Ireland, A., Lomax, J., Ashburner, M., Foulger, R., Eilbeck, K., Lewis, S., Marshall, B., Mungall, C., Richter, J., Rubin, G.M., Blake, J.A., Bult, C., Dolan, M., Drabkin, H., Eppig, J.T., Hill, D.P., Ni, L., Ringwald, M., Balakrishnan, R., Cherry, J.M., Christie, K.R., Costanzo, M.C., Dwight, S.S., Engel, S., Fisk, D.G., Hirschman, J.E., Hong, E.L., Nash, R.S., Sethuraman, A., Theesfeld, C.L., Botstein, D., Dolinski, K., Feierbach, B., Berardini, T., Mundodi, S., Rhee, S.Y., Apweiler, R., Barrell, D., Camon, E., Dimmer, E., Lee, V., Chisholm, R., Gaudet, P., Kibbe, W., Kishore, R., Schwarz, E.M., Sternberg, P., Gwinn, M., Hannick, L., Wortman, J., Berriman, M., Wood, V., de la Cruz, N., Tonellato, P., Jaiswal, P., Seigfried, T., White, R.: The gene ontology (go) database and informatics resource. Nucleic Acids Res 32(Database issue), D258–D261 (2004), Gene Ontology Consortium Hg02273/hg/nhgri Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, P.H.S. England
22. yE, J., Fang, L., Zheng, H., Zhang, Y., Chen, J., Zhang, Z., Wang, J., Li, S., Li, R., Bolund, L., Wang, J.: Wego: a web tool for plotting go annotations. Nucleic Acids Res 34(Web Server issue), W293–W297 (2006), Journal Article Research Support, Non-U.S. Gov't England

3.8 Exercises

1. The International Nucleotide Sequence Database Collection (INSDC) synchronizes their sequence records on a daily basis. For the GenBank sequence listed in this chapter, find the corresponding sequence records in EMBL and DDBJ. Comment on the formats used by the three databanks and if all the features in the GenBank entry are integrated into the other databases.
2. Obtain information provided on BRC2 protein, implicated in susceptibility to breast cancer, from the SWISS-PROT database. How many different sequences of this protein are provided? Comment on how they are different.
3. Perform the same analysis on the haemoglobin protein as you did in the question 2 above. This time use the PIR database. Also, run these searches on the UniProt. Do you find any significant enhancements in the information provided by UniProt?
4. Given a sequence entry below from the GENBANK shown in Fig. 3.12, provide a short description for each of the following:

(a) What is the accession number for this sequence?
(b) What is the length of this sequence?
(c) What is the type of molecule (DNA, RNA, Protein) and its topology corresponding to this sequence?
(d) Which organism is this sequence derived from? Is this organism a prokaryote?
(e) Is there a coding region in this sequence? If so, what is its location?
(f) In addition to the coding region, is there another feature annotated on this sequence. If so, what feature is it, and where on the sequence is it located?
(g) Is this sequence rich in A+T or is it rich in G+C? Justify.
(h) If a researcher wants to read further about the gene reported in this sequence, what would be a good place to start their research? Be specific.

5. Visit the Entrez home page and search the core nucleotide database for the accession number `M34058`. This is a Human beta-globin gene from a thalassemia patient. Save this sequence in FASTA formats. Comment on the annotations from the GenBank entry that are retained in the FASTA version of the sequence record.
6. Use MATLAB help on `getgenbank` and use the partial sequence access feature to retrieve the sequence of the coding segment spanning from location 139 through 4287 from the sequence identified by accession number `M10051`.
7. Visit the full genome section of Entrez and download the sequences of HIV-1 and HIV-2. Save these files on your local computer in GenBank format. Use MATLAB to read these files in into MATLAB and create two

```
LOCUS       AF162693        2088 bp    mRNA    linear   INV 02-AUG-1999
DEFINITION  Caenorhabditis elegans thioredoxin reductase homolog mRNA,
            complete cds.
ACCESSION   AF162693
VERSION     AF162693.1
KEYWORDS    .
SOURCE      Caenorhabditis elegans
  ORGANISM  Caenorhabditis elegans
            Eukaryota; Metazoa; Nematoda; Chromadorea; Rhabditida;
            Rhabditoidea; Rhabditidae; Peloderinae; Caenorhabditis.
REFERENCE   1  (bases 1 to 2088)
  AUTHORS   Buettner C., Harney J.W., Berry M.J.
  TITLE     "The Caenorhabditis elegans homologue of thioredoxin reductase
            contains a selenocysteine insertion sequence (SECIS) element that
            differs from mammalian SECIS elements but directs selenocysteine
            incorporation"
  JOURNAL   J. Biol. Chem. 274(31):21598-21602(1999).
  MEDLINE   99348283
   PUBMED   10419466
FEATURES             Location/Qualifiers
     source          1..2088
                     /mol_type="mRNA"
                     /db_xref="taxon:6239"
                     /organism="Caenorhabditis elegans"
     CDS             1..1578
                     /db_xref="GOA:Q17745"
                     /db_xref="HSSP:1H6V"
                     /db_xref="UniProt/Swiss-Prot:Q17745"
                     /codon_start=1
                     /protein_id="AAD46625.1"
                     /translation="MYIKGNAVGGLKELKALKQDYLKEWLRDHTYDLIVIGGGSGGLA
                      ...
                     ICLRNEEEKVVGFHILTPNAGEVTQGFGIALKLAAKKADFDRLIGIHPTVAENFTTLT
                     LEKKEGDEELQASGCXG"
                     /product="thioredoxin reductase homolog"
                     /transl_except="(pos:1570..1572,aa:OTHER)"
                     /note="start codon not determined experimentally;
                     selenocysteine"
     misc_feature    1766..1813
                     /note="SECIS element"
BASE COUNT      636 a    363 c    468 g    621 t
ORIGIN
        1 atgtatatca aaggaaatgc tgttggtggt ctcaaagaac ttaaagctct gaagcaggat
       61 tatttgaaag aatggcttcg tgatcatacc tacgacctga ttgtcattgg aggaggatct
      ...
     1981 tatccagtgt ttaagtatgt cctctcactt ttttttcaat ttcaatcccg tacttcttcc
     2041 ctatttttgg gtaaatccaa gccttttttca ctctctctct taaatgc
```

Fig. 3.12 A Sample GenBank Sequence

separate sequence objects. Open these two sequence objects using `seqviewer` tool and compare the proteins annotated on each sequence. Comment on the difference and similarity between these two sequences.

8. Use the function `getgenpept` to download and save the protein sequence identified by accession number `AAD51968`. Save the downloaded sequence locally in *GenBank* and *EMBL* format. What is the accession number of DNA sequence from which this GenPept entry is derived. Next, review the saved sequence to familiarize yourself with the two formats. Comment on the similarity and differences between the two formats.
9. Use MATLAB to download a PDB data entry identified by 1MBS representing the structure of extracellular matrix. Open the structure up in molviewer tool. How many atoms and bonds does this structure contain. Now review the alpha-helices in the structure by turning on

rockets, *strands*, and *trace* options under the structures selection. Can you count the number of alpha helices from this view.

10. Use the function `getgeodata` to download gene expression information identified by accession number `GSE11287`. Unlike a single sample, this data entry contains a number of samples – or a series – from an experiment. How many samples does this series contain? What is the size of each sample? Provide MATLAB code to extract out a each of the sample from the series.
11. You are interested in determining if enzyme tyrosine kinase has a characteristic signature. Describe your plan for solving this problem. Does this protein in fact has such a signature?
12. List the transcription factor binding sites for the heat shock protein in humans by searching the TRANSFAC database. Provide any additional information you find.
13. A GenBank entry provides information about Taxonomy. For the nucleotide sequence listed in this chapter provide a graphical representation of the taxonomy by running it through the NCBI taxonomy browser.
14. A GenBank record also provides information on the GO terms that are applicable to the sequence record. Use the weGO tool to view the terminology tree for a GO term.
15. The PDB captures the atom positions for a biomolecule. What are some of the tools that are provided for viewing this structural information.
16. Answer the following questions for the Ontology graph shown below in Fig. 3.13. Assume that the arrows represent an "is_a" relationship.

    (a) What are the descendants for `GO:0030684`?
    (b) What are the ancestors for `GO:0009547`?
    (c) What are the relatives for `GO:000313`?

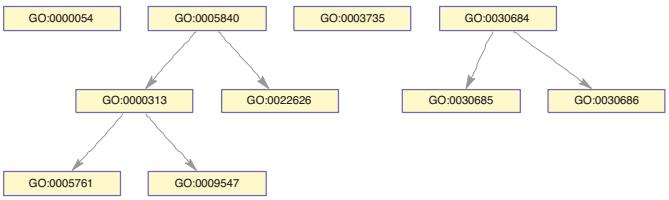


Fig. 3.13 A Sample GO Graph

17. Load in the gene ontology and develop a histogram of the various categories of terms. That is, a plot depicting the number of terms in each of the three ontologies, molecular function, biological process, and cellular component.

18. Associations in GO are supported by evidence. Table 3.2 shows the set of codes used to annotate the evidentiary support of an association of a gene product to a GO term.
19. Find the list of relatives for the interferon beta gene. Partition the relatives based on ontology. For each ontology (MF, BP or CC) draw a graph showing the gene or product names that are related to beta interferon.
20. Develop a program that takes as input two gene names and outputs an terms that are annotated on both. It should list the terms common in each of the three ontologies. Are there annotations on *interferon beta* are common with the annotations on *interferon gamma*? Test your program on at least three other pairs of genes.
21. Use the PubMed to search for articles that describe gene detection algorithms using neural networks. Name some journals that publish articles in bioinformatics.

Chapter 4
Processing Biological Sequences with MATLAB

MATLAB incorporates a number of bioinformatics function packaged together in its Bioinformatics Toolbox. The Bioinformatics Toolbox provides functions for creating objects such as biograph, the gene ontology and trees for phylogenetic analysis. In addition, this toolbox provides a number of functions for downloading sequences from DNA, protein and structure databases. Sequence analysis functions are provided for performing statistical analysis on the sequence data as well as for comparing DNA and protein sequences using the standard alignment methods (Needleman-Wunsch and Smith-Waterman) for global and local alignments. Capabilities for finding multiple sequence alignments, physlogenetic analysis, visualization and modeling sequence profiles using Hidden Markov methods are included within in the Bioinformatics Toolbox. The toolbox provides extensive functionality for retrieving, writing and and analyzing microarray data. The richness of tools included makes it possible to apply the concepts covered in the subsequent chapters within the framework of the same application. This provides an overview of the basic functions that enable reading, writing and display of biological sequence data.

4.1 Sequence Acquisition

The commands for downloading a file from the nucleotide databases GenBank and EMBL and the protein database GenPept are described below. In each of the commands below, the parameter supplied is the accession number of the sequence (DNA or protein) being retrieved. We previously looked at functions that return a sequence object *structure* that contains as attributes information gathered from the sequence record and typically includes to the raw sequence data as its `Sequence` member.

```
dnaSeq = getgenbank('M10051');
seqObj = getembl ('X00558');
pepSeq = getgenpept('AAA59174');
```


G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology*,
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_4

In this chapter we will look into some of the following, and some additional functions for processing sequences in MATLAB.

Function	Description	Example
`nt2aa`	Converts a nucleotide sequence to an amino acid sequence	aaSeq = nt2aa(dnaSeq)
`aa2nt`	Converts an amino acid sequence to a nucleotide sequence	ntSeq = aa2nt(aaSeq)
`dna2rna`	Converts a DNA to a RNA sequence	rnaSeq = dna2rna(dnaSeq)
`rna2dna`	Converts a RNA sequence to DNA sequences	dnaSeq = rna2dna(rnaSeq)
`seqcomplement`	Complementary sequence	seqC = seqcomplement(seq)
`seqrcomplement`	Reverse-complementary sequence	seqRC = seqrcomplement(seq)
`seqreverse`	Sequence orientation reversed	seqR = seqreverse(seq)
`aacount`	Counts frequency of amino acids	aaCnt = aacount(aaSeq)
`basecount`	Counts frequency of nucleotides	ntCnt = basecount(dnaSeq)
`dimercount`	Counts frequency of 2-mers	diCnt = dimercount(dnaSeq)
`codoncount`	Counts frequency of 3-mers	cdnCnt = codoncount(dnaSeq)
`nmercount`	Counts frequency of n-mers	nmerCnt = nmercount(dnaSeq, n)
textttntdensity	Nucleotide ensity profile sequence	ntdensity(dnaSeq)
`codonbias`	Compute bias in the usage of codons	codonbias(dnaSeq)
`cpgislands`	Locate stretches of CG dimers or CpG islands	cpgisland(dnaSeq)
textttseqshowwords	Find specific words in sequence	seqshowwords(seq)
`seqwordcount`	Counts words in a sequence	wCnt = seqwordcount(seq)
`seqshoworfs`	Show location of Open Reading Frames (ORFs)	seqshoworfs(dnaSeq)

MATLAB stores the sequence information as a structure constructed upon parsing of the information contained in the sequence file. As an example, the components of the structures `dnaSeq` and `seqObj` are listed below. Note that the structure of the sequence retrieved from each of these databases is different:

```
>>dnaSeq
dnaSeq =
                LocusName: 'HUMINSR'
      LocusSequenceLength: '4723'
     LocusNumberofStrands: ''
            LocusTopology: 'linear'
        LocusMoleculeType: 'mRNA'
     LocusGenBankDivision: 'PRI'
    LocusModificationDate: '06-JAN-1995'
               Definition: 'Human insulin receptor mRNA, complete cds.'
                Accession: 'M10051'
                  Version: 'M10051.1'
                       GI: '186439'
                 Keywords: 'insulin receptor; tyrosine kinase.'
                   Source: 'Homo sapiens (human)'
           SourceOrganism: [4x65 char]
                Reference: {[1x1 struct]}
                  Comment: [14x67 char]
                 Features: [51x74 char]
                      CDS: [1x1 struct]
                 Sequence: [1x4723 char]
```

```
>> seqObj
seqObj =
            Identification: [1x1 struct]
                 Accession: 'X00558'
           SequenceVersion: 'X00558.1'
               DateCreated: '13-JUN-1985 (Rel. 06, Created)'
               DateUpdated: '18-APR-2005 (Rel. 83, Last updated, Version 4)'
               Description: 'Rat liver apolipoprotein A-I mRNA (apoA-I)'
                   Keyword: 'apolipoprotein; lipoprotein; signal peptide.'
           OrganismSpecies: 'Rattus norvegicus (Norway rat)'
    OrganismClassification: [3x75 char]
                 Reference: {[1x1 struct]}
                   Feature: [23x75 char]
                 BaseCount: [1x1 struct]
                  Sequence: [1x877 char]
>>
```

Each of the components of the sequence objects contains detailed information accessible by drilling down into the component. The datatype for each component of the structure is also identified. For example, information in `seqObj` indicates that the sequence retrieved is a mRNA sequence of rat liver apolipoprotein that was originally deposited into the databank in 1995 (in Release 6 of the databank), and later revised in 2005 (in release 83 of the databank). As an example of drilling down, the features annotated on this sequence may be viewed by typing in the name of the component – *Feature* in this case – desired to be viewed:

```
>> seqObj.Feature

ans =
Key             Location/Qualifiers
source          1..877
                /organism="Rattus norvegicus"
                /mol_type="mRNA"
                /db_xref="taxon:10116"
sig_peptide     33..86
CDS             33..812
                /product="preproapolipoprotein A-I"
                /db_xref="GOA:P04639"
                /db_xref="HSSP:P02647"
                /db_xref="InterPro:IPR000074"
                /db_xref="InterPro:IPR013326"
                /db_xref="UniProtKB/Swiss-Prot:P04639"
                /protein_id="CAA25224.1"
                /translation="MKAAVLAVALVFLTGCQAWEFWQQDEPQSQWDRVKDFATVYVDAV
                KDSGRDYVSQFESSTLGKQLNLNLLDNWDTLGSTVGRLQEQLGPVTQEFWANLEKETDW
                LRNEMNKDLENVKQKMQPHLDEFQEKWNEEVEAYRQKLEPLGTELHKNAKEMQRHLKVV
                AEEFRDRMRVNADALRAKFGLYSDQMRENLAQRLTEIRNHPTLIEYHTKAGDHLRTLGE
                KAKPALDDLGQGLMPVLEAWKAKIMSMIDEAKKKLNA"
mat_peptide     105..812
                /product="apolipoprotein A-I"
```

```
polyA_signal    858..863
polyA_site      877..877
```

MATLAB provides the capability to view the sequence component of this object using the function, `seqdisp`. Only a portion of the entire display created by this function is shown.

```
>> seqdisp (seqObj.Sequence)

ans =
  1  AGCTCCGGGG GAGGTCGCCC ACATCCTTCG GGATGAAAGC TGCAGTGTTG GCTGTGGCCC
 61  TGGTCTTCCT GACAGGTTGC CAAGCTTGGG AGTTCTGGCA GCAAGATGAG CCCCAGTCCC
             .....
781  TCGATGAGGC CAAAAAGAAG CTGAACGCTT AGTGAGGCGC CCGTCACCAC TCCCCACCCC
841  TGAATTGGCT TTCTTACAAT AAACGTTTCC AAAGTGG
>>
```

4.2 Operations on Nucleotide Sequences

In the following illustration, we begin with a DNA sequence which is used for initializing a string variable in MATLAB which is then complemented and also reverse complemented. The reverse complementation is obtained by reading the string from 3' to 5' (right to left) and proceeding with the complementation. Protein translations are then performed and the sequence composition of the protein sequence is displayed. An "*" character designates the occurrence of a stop codon, and is shown as the residue "Others" in computing compositions.

```
ntSeq = 'ACAGTGCCCCCCTATATGGCCACCAGGTAG'

ntSeq =
ACAGTGCCCCCCTATATGGCCACCAGGTAG

>> length (ntSeq)
ans =

    30

%Find the base frequencies
>> basecount (ntSeq)
ans =

    A: 6
    C: 6
    G: 5
    T: 4

%Display the original sequence
>> seqdisp (ntSeq)
```

```
ans =

 1  ACAGTGCCCC CCTATATGGC CACCAGGTAG

%Display the complement of this sequence
>> seqdisp (seqcomplement (ntSeq))

ans =

 1  TGTCACGGGG GGATATACCG GTGGTCCATC

%Display the reverse complement of this sequence
>> seqdisp (seqrcomplement (ntSeq))

ans =

 1  CTACCTGGTG GCCATATAGG GGGGCACTGT

%Transform this sequence to a amino-acid, i.e. protein sequence

>> aaSeq = nt2aa (ntSeq)

aaSeq =

TVPPYMATR*

%Obtain amino acid counts
>> aacount (aaSeq)

ans =

         A: 1
         R: 1
         N: 0
         D: 0
         C: 0
         Q: 0
         E: 0
         G: 0
         H: 0
         I: 0
         L: 0
         K: 0
         M: 1
         F: 0
         P: 2
         S: 0
         T: 2
         W: 0
         Y: 1
         V: 1
    Others: 1
```

The result of applying MATLAB nucleotide density detector function `ntdensity` on a DNA sequence is shown in Fig. 4.1. The result of applying the detection of `CpG` islands is shown in Fig. 4.2.

```
% Download the sequence from GenBank
>> seqObj = getgenbank('M10051');

% Plot the nucleotide density profile

>> ntdensity(seqObj.Sequence)

% Plot the CpG profile

>> cpgisland(seqObj.Sequence, 'PLOT', true)
```

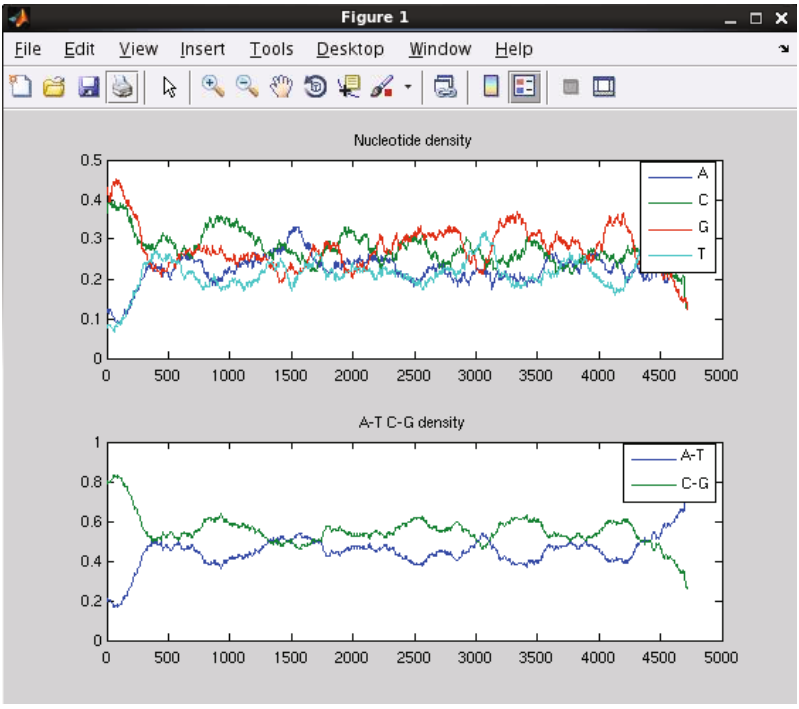


Fig. 4.1 The display of nucleotide density computed by sliding a window across the nucleotide sequence. The lower panel displays the G+C content of the DNA sequence. Generally, G-C rich sequences are rich in gene.

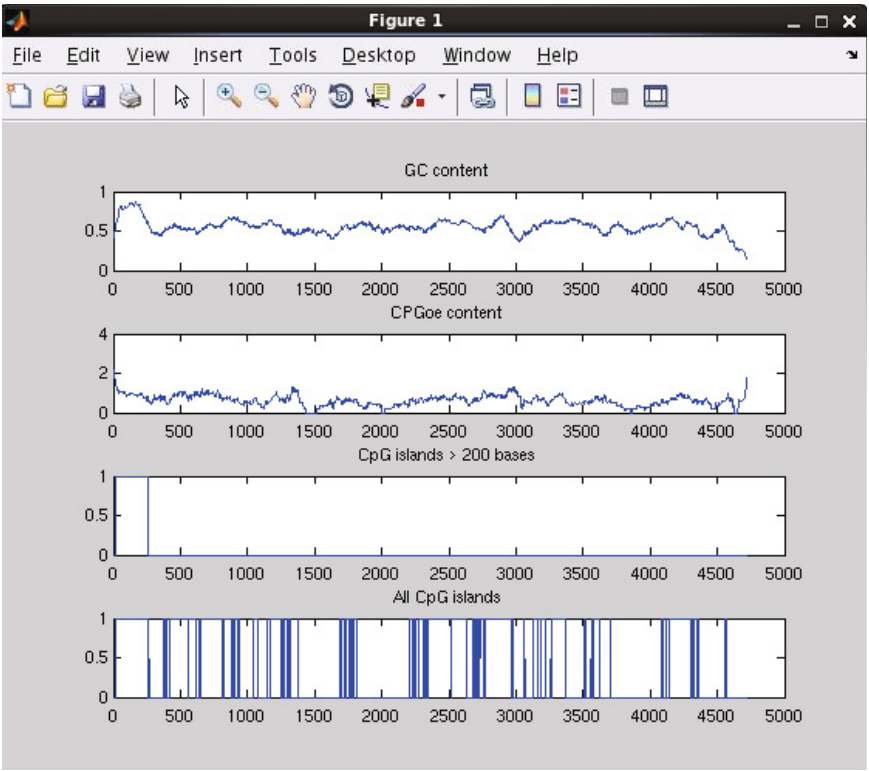


Fig. 4.2 The result of applying CpG island analysis on a DNA sequence

4.3 Joining Exons

Previous chapter discusses the example of joining exons to create a coding segment in an eukaryotic gene. In the example shown below, seven exons from four different GenBank sequences are combined to form a coding segment. Cell array is used to store the accession number and the segment of sequence joined to form the coding region, `CDS`.

```
% Create a cell array with the accession number and coordinates of the exons
% the form the gene

join ={{'AF018429',[282:561]}, {'AF018429',[1034:1172]} ...
        {'AF018430',[560:651]},  ...
        {'AF018431',[1:45]}, ...
        {'AF018432',[658:732]}, {'AF018432',[884:954]}, {'AF018432',[1391:1447]}}

% We create an empty cell array named exons with each element is set to cell with
% zero value. So exons looks like {[0] [0]  [0]}. This is a way to tell MATLAB to
% treat exons as a cell array
```

```
exons = num2cell(zeros(1,length(join)))

% Next we extract out the individual exons. Note that if the array exons had not been
% initialized to a cell array, the assignment statement offs(i) = { .. } below will
% error out.

for i=1:length(orfs)
    seqObj = getgenbank(join{i}{1});
    orfs(i) = { seqObj.Sequence(join{i}{2}) }
end

% Finally, all our exons are in the cell array. The command below takes the elements
% of the cell array and converts it into a matrix yielding the CDS

CDS = cell2mat(orfs)
```

4.4 An Example

In this example sequence identified by accession number `EU919427` is downloaded from GenBank into a local file. This sequence is read using the MATLAB function *genbankread* and the sequence data is extract from the sequence object. MATLAB functions *nt2aa* are used to convert the DNA sequence into an amino acid sequence. Appropriate frame shifts are performed to generate amino acid translations in the three forward and three reverse frames. The number of stop codons and the largest distance between the stop codons is plotted for all three frames.

4.4.1 Download Sequence

Download `EU919427` into a genbank formatted file. A keyword search on nucleotide database accessible over the web from National Center for Biotechnology Information (NCBI) may be performed. The site `http://www.ncbi.nih.gov` provides access to these databases. A keyword search using the term "beta globin" was performed. After the appropriate database entry is located, the display of the record should be changed to the desired format. The display options allow changing format to GenBank, ASN, XML, FASTA, etc. In this example, we save the file in the GenBank format, which is the default option selected. The view should be changed to show the sequence as "Text," since we do not want to save the sequence data saved along with any embedded hypertext tags.

As we would like to experiment with the detection of Open Reading Frames, we look for a DNA sequence. For this purposes of illustration, a synthetic sequence construct of BMPR2-R899X (Bmpr2) gene located. The display is changed to text only and the sequence is saved as a file named `EU919427_SynthethicGene.gb`.

4.4.2 Read That Downloaded File

Read that downloaded file into a sequence object using *genbankread*. The `genbankread` function is used to read a local sequence stored in GenBank format on the local filesystem. Other functions for reading files in EMBL and FASTA are provided by functions `emblread` and `fastaread`. The function shown reads the saved sequence file into a sequence object seqObj.

```
>> seqObj = genbankread ('EU919427_SynthethicGene.gb')

seqObj =

               LocusName: 'EU919427'
      LocusSequenceLength: '4794'
            LocusTopology: 'linear'
        LocusMoleculeType: 'DNA'
     LocusGenBankDivision: 'SYN'
    LocusModificationDate: '26-AUG-2008'
              Definition: 'Synthetic construct BMPR2-R899X (Bmpr2) gene ...'
               Accession: 'EU919427'
                 Version: 'EU919427.1'
                      GI: '197205403'
                  Source: 'synthetic construct'
          SourceOrganism: [2x38 char]
               Reference: {[1x1 struct]  [1x1 struct]}
                Features: [34x74 char]
                     CDS: [1x1 struct]
                Sequence: [1x4794 char]
```

4.4.3 Process Sequence

As a first step, the sequence data is extracted from the sequence object and copied into a string variable `seq`:

```
>> seq = seqObj.Sequence
```

Next the function *nt2aa* is used to convert the sequence into the amino acid translations in the three forward and three reverse frame as follows:

```
>> fwd = seq;
>> rev = seqrcomplement (seq);

% frames 1, 2 ,and 3

>> frm1 = fwd (1:length(fwd));
>> aa1 = nt2aa (frm1);
>> frm2 = fwd (2:length(fwd));
>> aa2 = nt2aa (frm2);
>> frm3 = fwd (3:length(fwd));
>> aa3 = nt2aa (frm3);
```

```
% frames 4, 5 ,and 6
>> frm4 = rev (1:length(rev));
>> aa4 = nt2aa(frm4);
>> frm5 = rev (2:length(rev));
>> aa5 = nt2aa(frm5);
>> frm6 = rev (3:length(rev));
>> aa6 = nt2aa(frm6);
```

As an alternative approach to creating six frame sequences as character strings, two cell arrays named `frm` and `aa` may be created using the following commands written up as follows:

```
for i = 1:3
 frm{i} = fwd (i:length(fwd));
  aa{i} = nt2aa(frm{i});
end;

for i = 4:6
 frm{i} = rev (i-3:length(rev));
 aa{i} = nt2aa(frm{i});
end;
```

4.4.4 Extracting Stop Codons

Next the number of stop codons in the forward strand – i.e. first, second, third frame – and the reverse strand – i.e. fourth, fifth, sixth frames – is computed as follows:

```
% stop codons show up as the '*" character in amino acid sequence.
stpcnt(1)= length (find (aa1=='*'))
stpcnt(2)= length (find (aa2=='*'))
stpcnt(3) = length (find (aa3=='*'))
stpcnt(4) = length (find (aa4=='*'))
stpcnt(5) = length (find (aa5=='*'))
stpcnt(6) = length (find (aa6=='*'))
```

The length of the **longest uninteruppted** strand in first, second, and third (forward) and the fourth, fifth, and sixth (reverse) frames is found using the *diff* function. The difference between the locations of the successive stop codons is returned by the *diff*, the *max* of which is returned the largest span of uninterrupted stop codons

```
maxlen(1) = max(diff(find (aa1=='*')))
maxlen(2) = max(diff(find (aa2=='*')))
maxlen(3) = max(diff(find (aa3=='*')))
maxlen(4) = max(diff(find (aa4=='*')))
maxlen(5) = max(diff(find (aa5=='*')))
maxlen(6) = max(diff(find (aa6=='*')))
```

Alternatively, the above commands may be performed by the following loop on the cell arrays `frm` and `aa` discussed earlier:

```
for i = 1:6
  stpcnt(i) = length(find(aa{i} == '*'));
  maxlen(i) = max(diff(find(aa{i} == '*')));
end;
```

4.4.5 Charting Results

In this case we will produce two bar graphs. The first series plotted in red represents the count of stop codons, and the second series plotted in blue represents the length of longest uninterrupted sequence in that frame. This data, is stored in arrays `stpcnt` and `maxlen`. The MATLAB *bar* command plots a bar graph where the second argument provides the two groups of data as the columns of the 2×6 matrix.

```
bar ([1:6], [stpcnt' maxlen']);
grid on;
legend ('Stop Count', 'Longest Inter-Stop Length');
ylabel ('Count & Base Pairs', 'fontsize', 16);
xlabel ('Frames', 'fontsize', 16);
```

As a result of issusing the command above, the bar graph shown in Figure 4.3 is plotted.

4.5 Restriction Site Detection

Restriction Enzymes are proteins that cut DNA at a specific location. Two restriction enzymes, `Eco-RI` and `Hind-III` are commonly utilized in laboratories. `Eco-RI` looks for a substring `GAATTC` and cuts the DNA sequence at that site; `Hind-III` cuts at substring `AAGCTT`. Some of the cuts made by restriction enzymes leave an overhang, or *sticky ends*, which biologists often use in their experiment design for reformation of DNA and for cloning and sequencing. Some restriction enzymes cut at leaving a blunt end.

As an example, Fig. 4.4 lists the recognition sites for enzymes `Eco-RI` and `Hind-III`. The restriction enzyme `Eco-RI` recognizes `GAATTC` and cuts between `G` and `A`, and the restriction enzyme recognizes `AAGCTT` and cuts

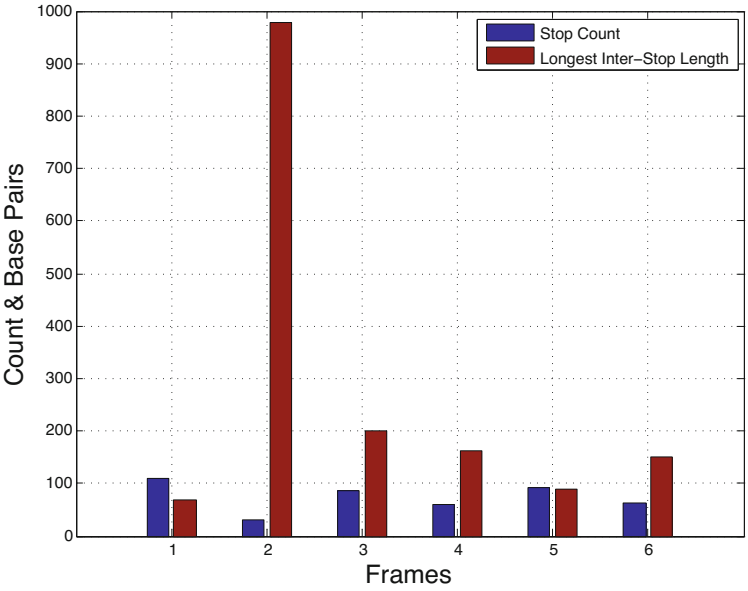


Fig. 4.3 MATLAB bar graph depicting the number of stop codons and the length of the longest uniterrupted sequence length in each of the six frames

between `A` and `A`. Both these enzymes recognize the occurrence of a biological chromosome – where the recognition site read from $5' \rightarrow 3'$ is the same as the site read on the reverse complementary strand read from $3' \rightarrow 5'$.

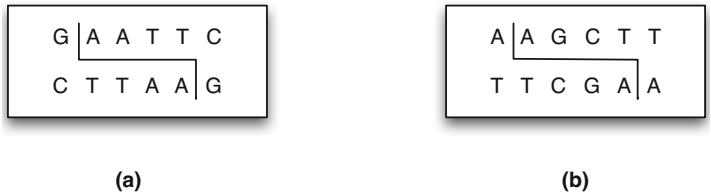


Fig. 4.4 The motifs recognized by restriction enzymes (a) EcoRI and (b) HindIII. Both of these enzymes, like most restriction enzymes, cut a DNA sequence at a biological palindromic site.

Restriction maps are computed to plan out biological experiments. A restriction map provides a visual representation of where a sequence of DNA may be cut by one of or a subset of more than a thousand of known restriction enzymes. Splicing out a sequence of gene using a set of restriction enzyme digestion, and amplification using a Polymerase Chain Reaction (PCR), is often utilized for further study of genes or sequence of interest. The site

`http://www.neb.com/rebase` provides a list of all restriction enzymes in a number of formats.

Let's consider the following program. Here a sequence is downloaded from EMBL. The first 1000 bases of the downloaded sequence is written locally into a file in FASTA format. The function for writing a sequence into FASTA format is `fastawrite`. This function takes the *filename* to be written, the one line annotation, and the sequence to be written into the file. FASTA format is popular for sequence analysis because its primary focus is the DNA sequence itself. It does not really focus on the annotations such as taxonomy, references, and features. Therefore, for the purposes of sequence analysis and speed, FASTA format is preferred.

```
>> seqEmbl=getembl('U15422')

seqEmbl =
            Identification: [1x1 struct]
                 Accession: 'U15422'
           SequenceVersion: 'U15422.1'
               DateCreated: '18-FEB-1995 (Rel. 42, Created)'
               DateUpdated: '14-NOV-2006 (Rel. 89, Last updated, Version 6)'
               Description: [2x75 char]
                   Keyword: '.'
           OrganismSpecies: 'Homo sapiens (human)'
    OrganismClassification: [3x75 char]
                 Organelle: []
                 Reference: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
    DatabaseCrossReference: [10x42 char]
                  Comments: []
                   Feature: [421x75 char]
                 BaseCount: [1x1 struct]
                  Sequence: [1x40573 char]
              RetrieveURL: 'http://www.ebi.ac.uk/cgi-bin/dbfetch?
                                  db=EMBL&id=U15422&style=raw'

>> fastawrite('U15422-Subseq.fasta', 'Subsequence U15422 (1:1000) - Fasta format', ...
                  seqEmbl.Sequence(1:1000))

>> fastawrite('U15422.fasta', 'U15422 - Fasta format', seqEmbl.Sequence)
```

The contents of FASTA sequence file U15422.fasta are shown in Fig. 4.5.

We next utilize MATLAB function to read in using function `fastaread`. The sequence that is read in and searched for the occurrence of `Eco-RI` and `Hind-III` cut sites using the regular expression specified. The MATLAB function `regexpi` finds the index of the location where the regular expression occurs in the sequence. Then we add `1` to the cut locations because the actual cut site is offset by `1` from the reported location of the regular expression.

```
>> seq = fastaread('U15422.fasta')

seq =
      Header: 'U15422 - Fasta format'
    Sequence: [1x40573 char]
```

```
>> cutPattern = '(GAATTC|AAGCTT)';
>> cutLocations = regexpi (seq.Sequence, cutPattern);
>> cutLocations = cutLocations + 1;

>> xvals = 1:length(seq.Sequence);
>> yvals = zeros(1, length(xvals));
>> for i = 1:length(cutLocations)
     yvals(cutLocations(i)) = 1;
   end;

>> subplot(3,1,1), bar (xvals, yvals), ...
                  title('Bar Graph'), set(gca, 'YTick', 0:1);

>> subplot(3,1,2), stem (xvals, yvals), ...
                  title('Stem Graph'), set(gca, 'YTick', 0:1);

>> subplot(3,1,3), plot(xvals, yvals), ....
                  title('Plot Graph'), set(gca, 'YTick', 0:1);
```

```
>Subsequence U15422 (1:1000) - Fasta format
gatctctctaattacagggcccacagggccagcagtatagataggtcacccactccacccagctgggtgt
ggggaggagccgccctgtctggaggtgagaggggcccagaggtcccacaggtccttgcagggcccaagca
agtctgggtcaagacactgatcctgtctcatgatccttcccccgactcctcacctgaacacctctggacg
tgtggggtcttttccaggtttgcttattattactattaatttaaaaaggaatagggctggtcatggtggt
ccacgcctataatcccagcactttgggaggctgaggtggaaggattgcttgagcccaggagtttgagacc
agcctgggaaacatggtgagaacccgtttctatttttttgtttttttaatttttaagaggaaaaaacagaaa
aagtagtagcgtaaatgcaacgtgggattctagattgaaccttgtgtctttaccagccctctaggtgatt
ctggtgcatgctaaaatttgagaactgctagtccactgaatatattccagagaaattctagctacagcag
cactgtttataatggcagaaaattggaaacagcccaagtgtccatcaagagggaatgaaggccaggcatg
gtggctcacatctctaatcccagcacttcaagaagctgatgtgggtggatcacttgaggtcaggagttcg
agaccagcctggccaacatggccaaactccatctctactaaaaacacaaaaattagctgggcgtggtggt
gcatgtctgtagtcccagcaatttaggaggctgaggcatgagaaccacttgagcctgggaggcggaggtt
tcagtgagccgagattgtgccactgcactccagcctgggcaacagagtgagactctgtctcaaaaaaacc
cccacaaaaaacaagaggcaattaagacagttacaaggtaggcacataaggattactatgtaacataata
tgtaacttaccactatgtaa
```

Fig. 4.5 A sample FASTA created from sequence extracted U15422 and saved to a file U15422.fasta

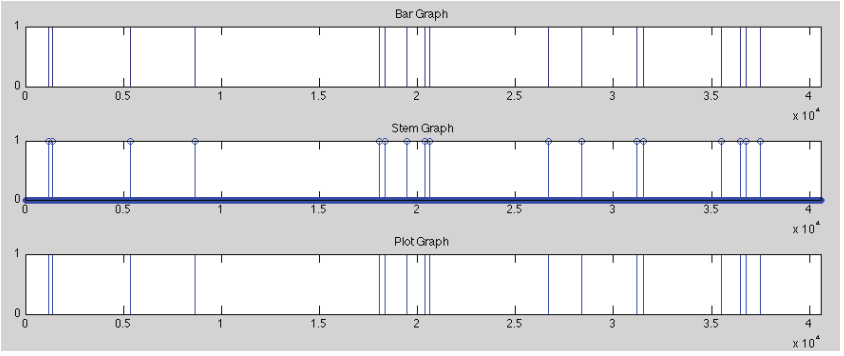


Fig. 4.6 The location of restriction cut sites is shown with three possible plot types

The remaining commands create an array of `x` and `y` values to be plotted. These sites may be plotted using a bar graph, a stem graph, or a plot graph. These three graphs are illustrated in Fig. 4.6.

Restriction enzyme sequences are specified using the IUPAC ambiguity codes.

Further Readings

1. Rivard, S.R., Mailloux, J.-G., Beguenane, R., Bui, H.T.: Design of high-performance parallelized gene predictors in matlab. BMC Res Notes 5, 183 (2012)
2. Sreeskandarajan, S., Flowers, M.M., Karro, J.E., Liang, C.: A matlab-based tool for accurate detection of perfect overlapping and nested inverted repeats in dna sequences. Bioinformatics 30(6), 887–888 (2014)
3. Cai, J.J., Smith, D.K., Xia, X., Yuen, K.-Y.: Mbetoolbox: a matlab toolbox for sequence data analysis in molecular biology and evolution. BMC Bioinformatics 6, 64 (2005)

4.6 Exercises

1. Perform the following steps.
   - Download NM_005368 into a genbank file (from NCBI)
   - Read that downloaded file into a sequence object using `genbankread`.
   - Extract out the sequence from this sequence object.
   - Use *nt2aa* to convert the sequence into aa in the three forward and three reverse frames.
   - Find the count stop codons in the first, second, and third (forward) and the fourth, fifth, and sixth (reverse) frames.
   - Produce a plot of the count of stop codons in each of the six frames.
   - Find the length of longest uninterrupted span in first, second, and third (forward) and the fourth, fifth, and sixth (reverse) frames.
   - Produce another graph that plots the longest uninterrupted span in each of the six frames.
2. Develop a MATLAB pipeline that packages the above steps. The pipleline function should ask for a set of accession numbers and perform the ORF detection steps outlined above.
3. Use MATLAB bioinformatics toolbox function to download the GenBank sequence with accession number U15422. View the DNA sequence object constructed and describe the annotated features. Can you determine which gene, if any, does this sequence contain?
4. Extract out the raw sequence data from the above sequence object and perform a display of the nucleotides.
5. Next, compute the basic statistics for the sequence data you obtained in Question 4. This could include the sequence composition, dinucleotide and trinucleotide frequency counts. Compare these observed frequencies with what is expected by pure chance. How close are the sequence statistics to what you expect by pure chance?
6. Consider the forward and reverse complement of the sequence in Question 4. Perform a count of open reading frames in each of the six frames thus obtained. Which frame do you think is the coding frame. Does your answer correspond to the sequence annotations.
7. Analyze the sequence object and determine if it contains any references to protein database. Download the referenced protein sequence and study its ontological references.
8. Compare the protein annotations with the amino acid sequences you prediction. Comment on the correspondence between the two.
9. For the following DNA sequence, write the MATLAB code for computing and plotting
   - the single nucleotide frequencies,
   - the di-nucleotide frequencies, and
   - the tri-nucleotide frequencies

```
ATTCG CTACC GTTCA CACGA TTTGA
CTTCG TTATC GTTCA TACGA TTTGA
CCTTA CGCGC GTTCG CTGGA TATCA
```

Next, use the inbuilt MATLAB function(s) to compute the mono-, di-, and tri-nucleotide frequencies. Compare results of using the inbuilt function to those obtained with your program.

10. Consider the following DNA sequence:

ACCCA TAGGG AGACA TAGTA GATCC ATTAG

(a) Write a MATLAB program to perform a 6 frame translation of the sequence in both the forward and reverse complement orientation.
(b) Compute and plot a graph of the length of Open Reading Frames (ORF) in each of 6 frames.
(c) Based on your analysis, which of the frames within has the highest likelihood for coding for proteins.

Part II
Information Retrieval from Biological Databases

Taking advantage of data stored in heterogeneous biological databases can be a difficult, time consuming task for a multitude of reasons. These reasons include the vast volume of biological data, the growing number of biological databases, the rapid rate in the growth of data, the overabundance of data types and formats, the wide variety of bioinformatics data access techniques, database heterogeneity, and the interdisciplinary nature of bioinformatics. In the course of their work, scientists may query a biological database, visually scan the results of this query, locate the relevant data, and subsequently use this data in a query submitted to another biological database.

Performing manual multi-database queries in this manner can be a time-consuming process, especially when large amounts of data are involved, as is often the case in biological research. Therefore, a need exists for automated data integration from multiple, heterogeneous databases. Automating biological database data integration can also speed up the discovery of new medications and the introduction of these drugs to market, with potentially wide ranging benefits to mankind.

This retrieval algorithms provide a solution to this problem of automating and approximating matches to the query and involve the use of a number of enabling technologies. The ultimate goal is to create a federation database that allow high level querying of multitude of databases using a user-friendly client applications.

Chapter 5
Sequence Homology

The rationales behind the comparison of sequences may be manifold. Above all, the theory of evolution tells us that gene sequences may have derived from common ancestral sequences. Thus, the comparison of biological sequences in this context is based on counting the number of mutations, insertions, and deletions of bases necessary to transform one DNA sequence into another.

One way to visualize the similarity between two protein or nucleic acid sequences is to use a similarity matrix, known as a dot plot which is a simple visual representation of the similarity between two sequences. Dot plots provide a preliminary idea of the similarity of the two sequences based on the number and length of matching segments between the sequences, with identical sequences having a diagonal line in the center of the matrix.

Sequence alignment is a formalized method for establishing similarity between biological sequences and serves as the basis for searching biological databases. It is a very useful method in itself with many applications. For example, one might compare a new sequence with a previously established and characterized sequence with the objective of learning the functional and structural properties of the new sequence through analogy based on what is already known about the established sequence.

This chapter begins with a discussion on dot plots and presents the algorithms for alignment of sequences and their applications in comparing biological sequences and for retrieving similar sequences from biological databases is discussed. Also elaborated are variations of the core sequence alignment techniques using insertion and deletion cost and similarity models, as well as different types of alignment procedures, such as global, local and semi-global or fit alignments, as well as their biological significance.

5.1 Information Retrieval from Biological Databases

Databases are used to obtain primary structures, such as the sequence information or some information associated with the molecular sequence, through

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_5

the process of annotation, curation and linking within the context of other biological databases. Fundamentally, there are two basic methods for searching sequence databases: (1) particular keyword label (e.g., "cytochrome c" or "dopamine precursor") with the databases allowing usual types of logical connectives between the keywords and their characteristics to better identify the target being sought; or (2) search engines can be used to hunt for sequences that are similar to one another – homology based searches. Keyword based searching of biological databases using the Entrez tool is discussed in this chapter.

5.1.1 Entrez

A search engine is a system for retrieving information from a set of documents via a direct search of the documents or by searching an index constructed from the documents. Most search engines operate via a combination of search algorithms and user-supplied keywords. Keyword searching is a method for retrieving information for a user based on the weighted values of each keyword searched for in a set or index of documents. These keywords need to be identified by the user themselves and should be aimed at uniquely identifying the items the user wishes the search engine to retrieve. These keywords are then passed through the search engines algorithms in order to generate the search results. Multiple keywords can be used and combined with Boolean operators in order to further refine the results of a search.

The Boolean operators used by most search engines are AND, OR, and NOT. The AND operator is used to indicate that the two keywords being combined by the AND operator must both be present in the search results, OR indicates that either keyword the Boolean operation is performed on may be present in the search results, and NOT indicates that the 2nd keyword it is performed on should not exist in the items returned by the search results. There are other Boolean operators that can be used by a search engine based on what sort of documents it is expected to search, but these are the most common operators used by popular search engines such as Google, Yahoo!, and Lycos.

Search for phrases may be performed by including the phrase in quotation marks. An example of phrase based retrieval is a search for the term, "Matrix Attachment Region." This is a special case of the AND query, where the terms must all occur as for a query, `Matrix AND Attachment AND Region`. However, in a phrase query the terms must all occur in that order and contiguously.

5.1.2 Search Example

An excellent example of keyword and Boolean searching as it applies to Bioinformatics can be seen in Entrez, The Life Sciences Search Engine. This search

Fig. 5.1 Entrez interface allows a cross database querying capability. In this example, the search was run to encompass all the databases accessible at NCBI. The query was specified using logical connectives. The number of hits in each database is shown on the left.

engine is freely accessible online at `http://www.ncbi.nlm.nih.gov/Entrez/`. As illustrated in Fig. 5.1, Entrez simultaneously searches for keywords in numerous biological databases such as nucleotide, protein, and gene databases as well as databases focusing on published academic work such as PubMed and OMIM. An example of keyword and Boolean searching in Entrez is a search for articles pertaining to dysentery. Dysentery may have two different causes, amoebic dysentery, caused by the amoeba Entamoeba

Histolytica, and Shigellosis, caused by bacteria of the genus Shigella. A search for "Dysentery AND Human NOT Amoebic returns results from every database Entrez searches relating to dysentery and humans while excluding references to amoebic dysentery. The correct use of Booleans and keywords can narrow the results of a search from millions of hits to just a handful. Boolean searches work well in cases where exact matches are necessary, and can be used to conduct rapid searches of large document bases. For searches where exact matches will not work, a similarity search based on a method such as alignment must be used. A good situation for a boolean search is a keyword search for academic papers on a subject, while a good example of a similarity search is determining how related biological sequences are by supplying a biological sequence as a keyword and then applying a similarity score to the potentially related sequences.

5.1.3 Obtaining Sequences Using Matlab

One of the most useful features available in the Bioinformatics Toolbox is the ability to obtain nucleotide and protein sequences from GenBank. Using the *getgenbank* function allows a user to retrieve sequences from the GenBank database via the Internet. By providing an accession number, a call to this function will return data pertaining to the matchin nucleotide or protein sequence. In the following example, the nucleotide sequence for a protein associate with Huntington's Disease and assigning the data to the sequence variable. By default, this sequence is returned in the GenBank file format.

Listing 5.1

```
>>n_sequence = getgenbank('NM_003949')

n_sequence =

                LocusName: 'NM_003949'
      LocusSequenceLength: '4100'
     LocusNumberofStrands: ''
            LocusTopology: 'linear'
       LocusMoleculeType: 'mRNA'
     LocusGenBankDivision: 'PRI'
    LocusModificationDate: '16-MAR-2008'
               Definition: [1x92 char]
                Accession: 'NM_003949'
                  Version: 'NM_003949.3'
                       GI: '120431739'
                  Project: []
                 Keywords: []
                  Segment: []
                   Source: 'Homo sapiens (human)'
```

```
SourceOrganism: [4x65 char]
     Reference: {1x10 cell}
       Comment: [35x67 char]
      Features: [91x74 char]
           CDS: [1x1 struct]
      Sequence: [1x4100 char]
     SearchURL: [1x70 char]
    RetrieveURL: [1x104 char]
```

end-listing-5.1

Similarly, protein sequences can be retrieved from GenBank using the getgenpept function. In the following example, the protein for Huntington's diesease is retreived in the FASTA file format.

```
>>p_sequence = getgenpept('AAB38240', 'FileFormat', 'FASTA')

p_sequence =

    Header: [1x68 char]
  Sequence: [1x3144 char]
```

5.1.4 Benchmarks

Goodness of retrieval is often measured by two parameters: *precision* and *recall*. **Precision** measures the number of relevant records result set as a fraction of all the records in the retrieved. **Recall** measures the number of relevant records that were retrieved as a fraction of all relevant records that the database contains irrespective of all the non-relevant records found in the result set. For example, if there are `T` records in the database that are relevant to the user query, and the result set contains `S` records out of which `R` are relevant, the parameters precision and recall may be defined as follows:

$$Recall = \frac{R}{T}$$

$$Precision = \frac{R}{S}$$

As far as boolean keyword based querying of database is concerned, the goodness of retrieval is essentially determined by the keywords and connectors chosen. Often it is an iterative process where the initial set of keywords and connectors are refined. Over-inclusive connected will tend to reduce the precision while enhancing the recall, while under-inclusive connected will have the opposite effect of increasing precision at the cost of reduction in recall.

5.2 Dot Plots

Dot plots are a simple technique for visualizing similarity between two nucleic acid or protein sequences. This technique utilizes a two-dimensional matrix which has the sequences being compared laid out along the vertical (row) and horizontal (column) axes of the matrix. When the residues of both sequences match at the same location on the plot, a dot is drawn at the corresponding position – a black dot being placed if the nucleotides or the residues are identical. Thus, matching sequence segments appear as runs of diagonal lines across the matrix.

An estimation of similarity of the two sequences is gleaned from the length, and the count of matching segments shown in the dot-plot matrix. Identical proteins have a have a diagonal line running across the center of the matrix. When either sequence is changed through insertions and deletions, disruptions in this diagonal of the dot plot are observed. When sequences share a local regions of similarity, or share common repetitive sequences, additional off-diagonal matches are observed in the dot plot matrix.

To enhance selectivity of matches reported by dot-plots, and to reduce the noise attributed to matching sequence segments arising simply by chance, a threshold *tuple-size* is sometimes utilized. For example, a match based on tuple-size of `3` requires three residues or nucleotides to match between the sequence along the x-axis and the sequence along the y-axis before a dot is placed on the matrix. This is effective because the chance matches go down with a tuple size of `3`.

The two sequences can be compared by taking the reverse complement of DNA sequence to determine if the matching strands are complementary to each other. The direction of the sequences on the axes will ultimately are combined to form lines. The closeness of the sequences in similarity will determine how close the line is to the diagonal line – the position of the line being affected by events such as frame shifts, direct repeats, and inverted repeats. Frame shifts include insertions, deletions, and mutations. The presence of one of these features, or the presence of multiple features, will cause for multiple lines to be plotted.

The following code sample illustrates the use of MATLAB function `seqdotplot` utilized for comparing two sequences `seq1` and `seq2`. The MATLAB function provides for a window size, or tuple size, as well as the level of identity required within the window. In the example shown below, we have used a window size of 6 with at least 5 characters required to be matched within the window before a dot is placed in the dot plot matrix.

Listing 5.2

```
>> seq1 = 'ACCTGACTGGGCTTAGACCTTGAACTTGAACACTTGCCTT';
>> seq2 = 'TTTTTACCTGACTGGGCTTGGGGGAGACCTTGAACTTGAACACTTGCCTT';
>> seqdotplot(seq1, seq2, 6, 5)
```

end-listing-5.2

It may be observed that the second sequence has the first sequences embedded within it. The second sequence has a string `TTTTT` preceding the start of the first sequence, and also embedded is a string `GGGGG` located 14 bp downstream. The resulting dot plot shows that the sequence are matching. This is above evident in the dot plot produced as shown in Fig. 5.2 shown. Notice that there are off diagonals dots resulting from random matches though the line formed by the string of dots dominates around the diagonals.

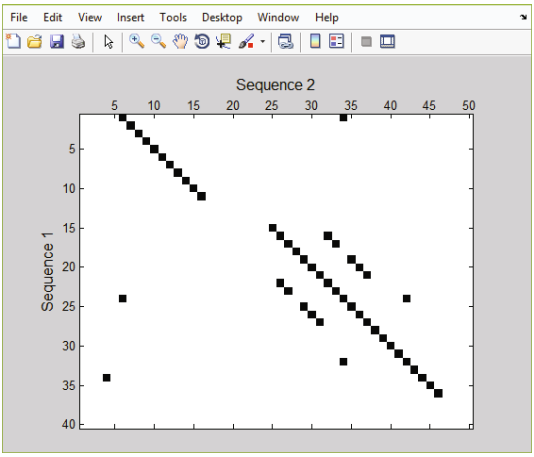


Fig. 5.2 Dot plot for two DNA sequences

In the example shown next, DNA sequences of two prions identified by accession numbers AB060288 and AB060290 are compared using the dot plot technique. The sequences of the GenBank entries are downloaded and compared. The dot plot illustrates a strong similarity between the two sequences. The code sample also illustrates some additional functionality of the `seqdotplot` function. This function can return a count of the number of matches found, as well as a sparse matrix containing the locations of matches.

Listing 5.3

```
>> seq1 = getgenbank('AB060288', 'sequence', true);
>> seq2 = getgenbank('AB060290', 'sequence', true);
>> seqdotplot(seq1, seq2, 11, 7);
>> matchCount = seqdotplot(seq1, seq2, 11, 7);
>> [matchCount matchLocations] = seqdotplot(seq1, seq2, 11, 7);
```

end-listing-5.3

The resulting dot plot in Fig. 5.3 shows that the sequence are similar as there is a strong diagonal line spanning the dot plot matrix.

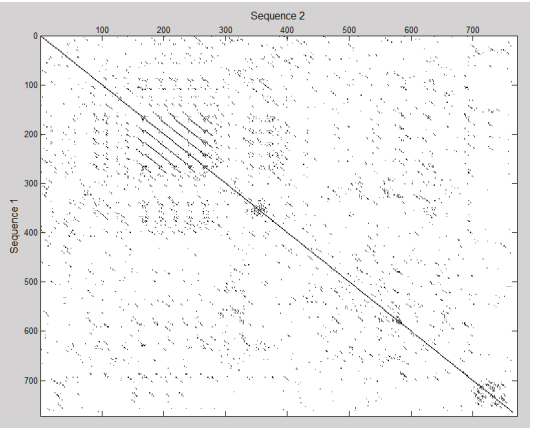


Fig. 5.3 Dot plot for DNA sequences of two prion proteins downloaded from GenBank

5.3 Sequence Alignment

Sequence alignment is probably the most primitive operation in computational biology; it essentially involves placing one sequence above the other and comparing the aligned vertical pairs to establish correspondence between the sequences at each position. Sequence alignment serves as the basis for a wide variety of more complex manipulations, such as finding parts of sequences that are alike and those that differ.

The format for representing the alignment between two DNA sequences is shown below. In this example, an alignment from base index 10–25 in the upper sequence with the bases 20–35 on the lower sequence is shown. The numbers at each end of the alignment correspond to the sequence index in the original sequence. The matching pair in the two sequences is shown as a (—). The (:) is often used to identify similar but nonidentical pairs. In our example, the IUPAC ambiguity code `N` s used which pairs with `G, C, T,` or `A`. A mismatch occurs between the bases `G` and `T` at positions 16 and 26 of the upper and lower sequences respectively.

```
10 AANCGTGATCGATGC 25
   ||:||| ||||||||
20 AATCGTTATCGATGC 35
```

5.3.1 Edit Distance

The fundamental underpinning of sequence alignment is the concept of edit distance, a metric designed to measure the difference between two strings. The edit distance formalization focuses on editing one of the strings and transforming it into the other using a series of character level edit operations. The permitted set of edit operations is limited to insertion of a character into the first string, deletion of a character from the first string, or the substitution (replacement) of a character in the first string by a character in the second string. Naturally there a number of ways to edit the source string and transform it to the target string. The set of operations required two such transformations of the string `"PASTRY"` into the string `"FACTORY"` may be listed as follows using the notation `D` to denote a deletion from the first string, `I` to denote an insertion of character into the first string, and `R` to denote a replacement of a character in the first string by a character in the second string. Note that the character `M` denotes a character match between the first and the second string that requires no string edits.

```
Edit Operations:    RMRMIMM      DIMDIMIMM
First String:       PAST RY      P AS T RY
Second String:      FACTORY       FA CTORY
```

The first set of edit operations shown above is comprised of two replaces and one insert. Namely, one way to transform the string `PASTRY` into `FACTORY` is by replacing `P` by `F`, replacing `S` by `C`, and inserting an `O` after the `T`. This, under the unit cost model where every edit operation has a cost of 1, has a cost of 3. The second set of edit operations shown requires two delete operations and three insert operations accumulating a total cost of 5. The string edit distance between two strings is defined to be the minimum number of string edit operations needed to transform the first string into the second string. It is left as an exercise to show using the dynamic programming method discussed in section 5.4 that the string edit distance between `PASTRY` and `FACTORY` is 3.

5.4 Dynamic Programming Algorithm

Dynamic programming is an efficient programming technique for solving certain combinatorial problems. It is particularly important in bioinformatics as it is the basis of sequence alignment algorithms for comparing protein and DNA sequences. In the bioinformatics application Dynamic Programming yields enormous efficiency in comparison to a purely recursive algorithm and thus converts what would be an $O(2^N)$ algorithm to an $O(N^2)$. Using dynamic programming it is possible for an algorithm to evaluate all possible ways of aligning one sequence against another in polynomial time,

even though the number of possible alignments grows exponentially with the length of the two sequences.

A dynamic programming based optimization algorithm lets us achieve a minimization of edit operations without explicitly enumerating all possible alignments of two sequences. While performing the alignment score optimization, one may either define a distance or a similarity measure as the basis for scoring an alignment. The difference lies more in the interpretation of the values. A distance function will define the distance between matching characters as zero, and assign some positive values for mismatches and gaps and then aim at minimizing this distance. A similarity function on the other hand will assign a high (positive) value to matches and a low (negative) values for gaps and mismatches and then maximize the resulting score. The basic process of comparing sequences is the same in either case. In 1981, Smith and Waterman showed that for global alignment, i.e. when a score is computed over the entire length of both sequences, the two concepts are in fact equivalent.

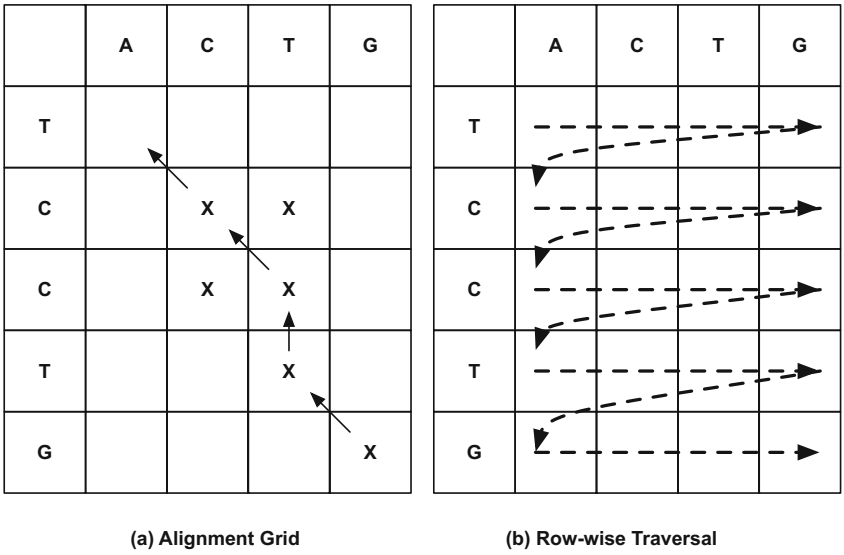


Fig. 5.4 (a) The dynamic programming grid that is utilized for computation of the alignment. The X marks designate the matching characters. The alignment is path that passes through the Xs and utilizes only vertical, horizontal and diagonal movements. One possible alignment is shown. (b) An example of row wise traversal that is commonly utilized for computing the grid cell values.

The alignment algorithm is based on a grid. If the first sequence is placed horizontally and the second vertically on the grid as shown in Fig. 5.4. An alignment corresponds to a path through this grid. Each position in the grid

determines a position in the first and second sequence. For example, imagine putting an 'X' in the grid corresponding to each position in the alignment where an element from the first sequence is aligned against an element from the second sequence. Next join the these X's up in order - only horizontal vertical and diagonal steps are allowed. The path through the grid shows which elements of the first and second sequence were matched up and therefore determines the alignment.

Using paths in a grid to represent alignments provides a method of computing best alignments. A score can be placed in each cell of the grid, the score for the best partial alignment ending at that position in the two sequences. All the scores for the best alignments ending at (i,0) or ending at (0,j) are zero. The score at (i,j) can be calculated from the scores at (i-1,j), (i,j-1) and (i-1, j-1), so by filling in the scores in the grid the score for the best alignment can be found. Once the best score is found the path that leads to it and hence the alignment can be traced back through the grid rapidly. In computing the scores each cell takes constant time to compute and so the overall algorithm has time complexity O(mn) where m and n are the lengths of the two sequences.

In the recursive formulation for completing the score in the grid, one writes a routine that calculates the score at (i,j) recursively while steeping though all the elements of the grid. and the nature of the traversal is such that the calls for computing the score values for (i-1,j), (i,j-1) and (i-1, j-1) have been completed prior to the invocation for cell (i,j). before returning its result.

5.4.1 Distance-Based Alignment

Albeit the score computation formula for distance value $D_{j,j}$ is defined using a recurrence relation formulation, its computation proceeds iteratively. The computation of the distance value for cell (i,j) is computed by reusing the results stored in the grid for cells (i-1,j), (i,j-1) and (i-1, j-1). This implies that the score value computation may proceed in a normal left-to-right, top-to-bottom scan of the score grid.

Let us assume that we are performing an optimization based on minimization of distance between the two sequences. As described in section 5.4.2, such an optimization could be based on the maximization of similarity between the two sequences as well if a similarity-based scoring matrix is used. However, for now, let us assume that we are aligning the sequences to minimize their distance metric. Let us assume the following unit cost model where distance between matching characters defined to be "0" and the distance between mismatching characters and distance for an is assumed to be "1." Also, the distance associated with the insertion or deletion of a character, designated as _, is assumed to be "1".

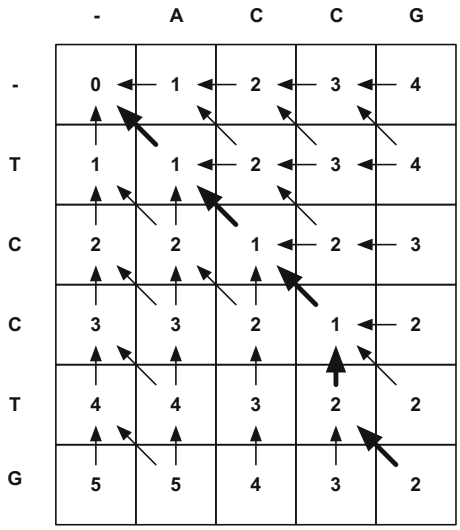


Fig. 5.5 Array values for computing global alignment using distance based unit cost model. The distance attributed to insertion (deletion) and substitution operation is equal to "1". The back-pointers maintained at each cell enable the traversal of the optimal path through the array values computed using dynamic programming. For the optimal alignment the string-edit distance between the sequences is computed to be 2.

$$d(a,b) = \begin{cases} 0 & : \quad if\ a = b \\ 1 & : \quad if\ a \neq b \\ 1 & : \quad if\ a = \_\ \ or\ \ if\ b = \_ \end{cases} \tag{5.1}$$

The formulation for computing the score for cell (i,j) is defined in Eq. 5.2 below:

$$D_{i,j} = Min \begin{cases} D_{i-1,j-1} + d(a_i, b_j) \\ D_{i,j-1} + d(\_, b_j) \\ D_{i-l,j} + d(a_i, \_) \end{cases} \tag{5.2}$$

The result of building the grid is shown in Fig. 5.5. This demonstrates the computation of Global Alignment using Needleman-Wunsch global alignment algorithm. Each cell shows the possible optimal predecessors that an alignment path may take along any of the multiple optimal paths. The process of constructing the optimal global alignment begins from the lowest-rightmost corner, i.e. cell (m,n), and traces back to the cell (0,0). The paths corresponding to the optimal alignment are shown. The optimal alignment is not

unique, even though there is a unique minimum alignment score for a pair of sequences. In the example shown in Fig. 5.5 however, only the singleton alignment shown yields the minimum distance of 2.

Example 5.1

In MATLAB, one has the ability to define a custom scoring matrix for a given alignment algorithm to use. Computing a distance alignment is simply a matter of using the Needleman-Wunsch implementation, supplied by the MATLAB Bioinformatics Toolbox. To use MATLAB to determine `nwalign` function in MATLAB adds scores in cells $S_{i,i}$ (i.e. adds the score one matches) and subtracts the scores in cells $S_{i,j}$, where $i \neq j$ (i.e. subtracts scores where a mismatch occurs). The code to produce a scoring matrix for use in distance-based alignment of nucleotide sequences is provided below.

Once the scoring matrix has been constructed, aligning the two sequences is simply a matter of passing two sequences, as either strings or sequence constructs, along with the scoring matrix and some other parameters to the function `nwalign`. The scalar result of this operation is the global distance score for the two sequences.

Listing 5.4

```
  scoring_matrix = [
       0 -1 -1 -1;
      -1  0 -1 -1;
      -1 -1  0 -1;
      -1 -1 -1  0;
  ];

seq1 = 'acgtacgtacgt';
seq2 = 'tcgttcgttttt';

nwalign(seqs1, seqs2, 'scoringmatrix', scoring_matrix,
                'extendgap', 1, 'gapopen', 1, 'alphabet', 'nt');
```

end-listing-5.4

One can take the absolute value of the result to determine the distance between the sequences. In the above example, the result is -5, meaning the distance between the sequences is 5 as can be seen from the alignment.

```
ACGTACGTACGT
 ||| |||   |
TCGTTCGTTTTT
```

End of Example

5.4.2 Similarity-Based Alignment

Similar to the recursive formulation for completing the distance values in the grid, the computation of the similarity score for cell (i,j) is computed such that similarity values for cells (i-1,j), (i,j-1) and (i-1, j-1) have been completed in the prior steps. The example shown in Fig. 5.6 performs an optimization based on maximization of similarity between the two sequences using the following unit scoring model.

$$s(a,b) = \begin{cases} 1 & : \ if\ a = b \\ -1 & : \ if\ a \neq b \\ -1 & : \ if\ a = \_\ or\ if\ b = \_ \end{cases} \tag{5.3}$$

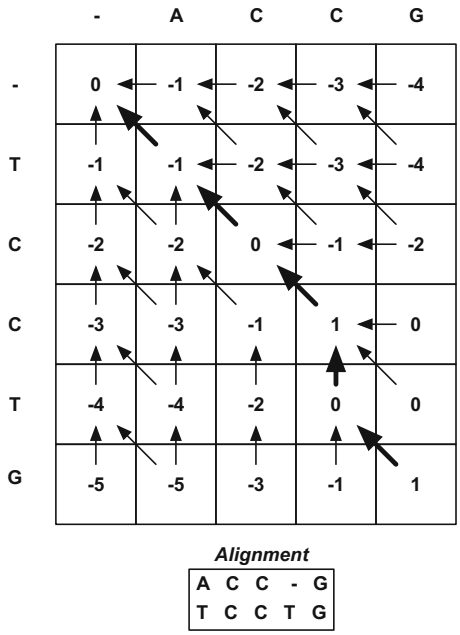


Fig. 5.6 The computation of the array values in this similarity based global alignment is done using a unit scoring model where the scores of insertion (deletion) and substitution operation is "-1". Each match operation is assigned a score of "+1". Again, back-pointers maintained at each cell to traverse the optimal path through the array values computed using dynamic programming. For the optimal alignment the similarity score between the sequences is 3.

Fundamentally, the operations of calculating the similarity scores will remain the same as shown in Eq. 5.4 below. Under this formulation, the score for cell (i,j) denoted as $S(i, j)$ is computed by iteratively computing the array values in a left to right, top to bottom traversal.

$$S_{i,j} = Max \begin{cases} S_{i-1,j-1} + s(a_i, b_j) \\ S_{i,j-1} + s(\_, b_j) \\ S_{i-l,j} + s(a_i, \_) \end{cases} \tag{5.4}$$

Each cell in Fig. 5.6 again shows the possible predecessors that are its optimal predecessors in that an alignment path may take any of the multiply optimal paths. The paths corresponding to the optimal alignment, backtracking from the cell (m,n) to cell (0,0), is shown. The optimal alignment is the same as that shown in Fig. 5.5 with the similarity score in this case of 3.

5.5 Longest Common Subsequence

This section presents another important problem as a special case of the sequence alignment problem that emerges when the scoring matrix is chosen such that the mismatches have a score of zero while the matches account for a score of `+1` . This problem is referred to as the *longest common subsequence problem* . Before describing the algorithm let's clarify the difference between a subsequence and a substring.

Given a string S, a subsequence is defined as a subset if the characters of S arranged in their original "relative" order. Thus a subsequence may be defined as using a set of indices $i_1 < i_2 < i_3 \ldots < i_k$, for some $k \leq n$. The subsequence is defined by the string $S(i_1)S(i_2)S(i_3) \ldots S(i_k)$. Thus, whereas the substring is comprised of contiguous subset of characters from the sequence, there is no such contiguity requirement for a subsequence. For example, considering a string "BIOINFORMATICS", we have one example of a subsequence to be "FORTS" which is not a substring. A substring, "INFORM", on the other hand is a subsequence as well.

The problem of finding the longest common subsequence (LCS) between two sequences S_1 and S_2 is defined as problem of finding the longest subsequence that occurs in both the strings. Considering the sequence alignment array, the LCS may be computed by using a scoring matrix that assigns a score of zero for all mismatches or gaps and a score of +1 for matches.

Example 5.2

Similar to computing distance-based alignments, similarity alignments can be implemented in MATLAB by using a custom scoring matrix and Bioinformatics Toolbox's `nwalign` function. The scoring matrix for similarity-based alignment, as previously discussed, possesses a score of 1 for each match and a score of -1 for each mismatch (e.g. the cells along the diagonal consists of 1 with -1 everywhere else). Once this scoring matrix is constructed, simply pass it and the two sequences to compare to the `nwalign` function. The scalar result returned is the similarity value of the two sequences.

In the following listing, two sequences are aligned and their similarity is stored in the variable `similarity`.

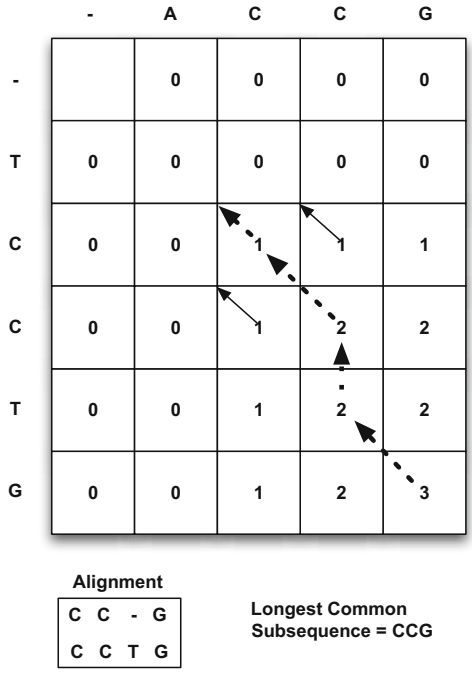


Fig. 5.7 The similarity matrix values with the cost model defined to find longest common subsequences. Unlike the global alignment unit cost model, the similarity score for gap indel and substitution is "0" in this case (instead of "-1"). The match score is still "+1".

Listing 5.5

```
scoring_matrix = [
    1 -1 -1 -1;
   -1  1 -1 -1;
   -1 -1  1 -1;
   -1 -1 -1  1;
]

seq1 = 'acgtacgtacgt';
seq2 = 'tcgttcgttttt';

similarity = nwalign(seqs1, seqs2, 'scoringmatrix', scoring_matrix,
                     'extendgap', 1, 'gapopen', 1, 'alphabet', 'nt')
```

end-listing-5.5

The previous code sample would produce a similarity score of 2.

MATLAB toolbox also provides a function `swalign`, or Smith-Waterman Alignment, that performs a similarity based alignment of sequences.

End of Example

5.5.1 Insertion, Deletion and Substitution Operations

The concept of gap costs is motivated by the evolution model where an alignment defines an *evolutionary*distance between two DNA or protein sequences. For computing similarity scores generally assigns `0` to a match, some negative number to a mismatch and a larger negative number to an insertion and deletion, or *indel*, events. By adding these values along an alignment one obtains a score for this alignment. Sometimes a distance function for two sequences can be defined by looking for the alignment which yields the minimum score.

The treatment of gaps deserves special care. One mostly uses a gap penalty function which charges a gap open penalty for every gap that is introduced and penalizes the length with a gap extension penalty which is charged for every inserted or deleted letter in that gap. Clearly, this results in an affine linear function in the gap length, frequently written as $g(k) = \alpha + \beta\dot{k}$, defining the cost for inserting a gap of length k in the sequence.

$$S_{i,j} = Max \begin{cases} S_{i-1,j-1} + s(a_i, b_j), \\ Max_{1 \le k \le j}\{S_{i,j-k} - g(k)\}, \\ Max_{1 \le l \le i}\{D_{i-l,j} - g(l)\} \end{cases} \tag{5.5}$$

5.6 Alignment Types

As shown in Fig. 5.8, three types of alignments are used in practice. These are **global**, **local** and **fit or semiglobal** alignments. Global alignment finds the optimal score of matching the entire span of one sequence to the entire span of another. This type of alignment is used when the sequences being compared are approximately equal in length or are known to be spanning the same domain so that a sequence level homology between them is expected *a-priori*. The example alignments in Sections 5.4.1 and 5.4.2 were all global alignments.

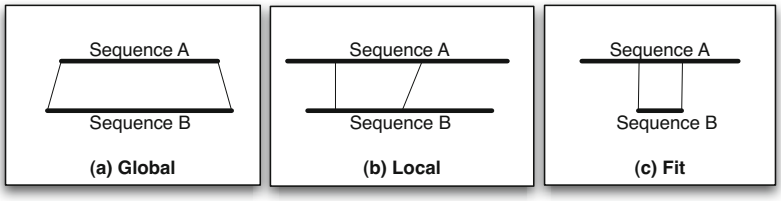


Fig. 5.8 Different types of alignments. (a) In global alignment the comparison of the entire span of the two sequences, A and B, are compared to compute the alignment score; (b) Local alignment aims at finding maximally similar sub-sequences from the two sequences; (c) Fit alignment, used for pattern detection, is a match between a sub-sequence of one sequence and an entire span of another.

As shown in Fig. 5.8 (b), a local alignment between two sequences attempts to find a subsequences from the two sequences that match. This is by far the most common type of alignment sought between two sequences. Local alignment is the basis for sequence retrieval utilized by GenBank retrieval tool BLAST. Theoretically, a local alignment is identical to the longest common subsequence problem discussed in Section 5.5.

The fit alignment, also known as **semiglobal** alignment, shown in Fig. 5.8 (c) is a hybrid of the global and local alignments as one of the two sequences is aligned globally where mismatches at its ends result in a penalty. The sequence that is aligned across its full span is typically a pattern that we seek to approximately match somewhere in the other sequence. Since the pattern can occur anywhere in the target, mismatches at the ends do not incur a penalty.

As an example, let's consider a fit alignment of the Sequence B=`CAATA` in a Sequence A = `TCCAGGACATAAGGA`. Although the CAT-Box sequence does not exactly occur in the target sequence A, fit alignment allows us to look for this match approximately. A dynamic programming array is set up as shown in Fig. 5.9.

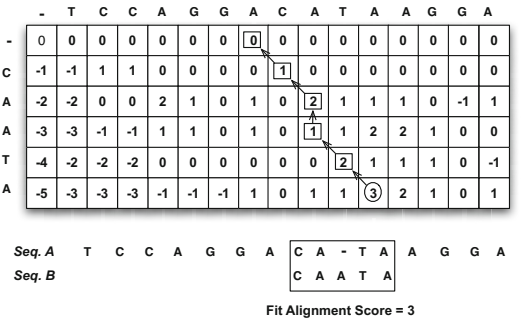


Fig. 5.9 The similarity matrix values for fitting a pattern into a sequence. A gap penalty for the pattern sought, which must match fully, is set to "-1" while the end gaps for the target sequences where the occurrence of the pattern is sought do not incur any penalty. The alignment is backtracked from the highest similarity value in the last row and is traced all the way up to the first row.

A unit cost similarity model is used for completing the elements of the dynamic programming array where the top sequence is the background in which the target pattern is sought. The pattern sought is `"CAATA"` and it is found in the larger sequence at the position shown. The location of the pattern's occurrence is backtracked from the highest value accumulated in the last row of the dynamic programming array. Furthermore, the backtracking continues until the entire pattern span is covered and first character of the pattern represented by the first row is reached.

5.6.1 Needleman-Wunsch in Matlab

The Needleman-Wunsch Algorithm is a global alignment algorithm that uses dynamic programming to align nucleotide or protein sequences. This algorithm is implemented in the MATLAB Bioinformatics Toolbox by the nwalign function. This function accepts the same parameters as swalign and returns results in the same format. The following example globally aligns two protein sequences.

- Matlab's `nwalign` performs global alignments.

```
[SCORE, ALIGNMENT, STARTAT] = NWALIGN(SQ1, SQ2,
            'ALPHABET', A, 'SCORINGMATRIX', matrix, 'GAPOPEN', penalty,
            'EXTENDGAP', penalty,  'SHOWSCORE', true)
```

  Where

  - `ALPHABET` specifies whether the sequences are amino acids (`'AA'`) or nucleotides (`'NT'`). The default is `AA`.
  - `SCORINGMATRIX` defines the scoring matrix to be used for the alignment. The default is `BLOSUM50` for `AA` or `NUC44` for `NT`.
  - `GAPOPEN` defines the penalty for opening a gap in the alignment. The default gap open penalty is 8.
  - `EXTENDGAP` defines the penalty for extending a gap in the alignment. If `EXTENDGAP` is not specified, then extensions to gaps are scored with the same value as `GAPOPEN`.
  - `SHOWSCORE` displays the scoring space and the winning path.

- Parameters for local alignment routine, `SWALIGN` described in the next section, are essentially the same. Also, with the exception of `SQ1` and `SQ2`, the two sequences to be aligned, the other parameters are optional.
- The return parameter `STARTAT` is for consistency between local and global alignment functions. It provides the starting location for the alignment in the two sequences. This will always be position 1 in case of global alignments.
- Several MATLAB functions provide the capability for specifying scoring matrices. Common scoring matrices are: `blosum (N)`, `dayhoff`, `gonnet`, `nuc44` and `pam (N)`.

5.6.2 Smith-Waterman in Matlab

The Smith-Waterman Algorithm is a local alignment algorithm that can be used to align nucleotide or amino acid sequences. Matlab supports this algorithm by using the swalign function. This function returns a vector in the format [score, alignment, start], where score is the bitscore, alignment is a representation of the alignment, and start is a vector representing where the alignment begins in each sequence. The following example aligns two nucleotide sequences.

Listing 5.6

```
>> [score alignment start]=swalign('ATCGATACGGAG','CGATACCGGCG',
'Alphabet', 'nt')

score =

   21.3333

alignment =

CGATA-CGGAG
||||| ||| |
CGATACCGGCG

start =

     3
     1
```

end-listing-5.6

The function can be used with any popular scoring matrix (defaults to NUC44 for nucleotide and BLOSUM50 for amino acid), or a custom scoring matrix. Additionally, one can specify a cost for creating or extending a gap. This example aligns two amino acid sequences. If the Alphabet parameter is not passed, the function will attempt to determine what type of sequences are used. The example below illustrates the use of Smith-Waterman local alignment function for finding local homologs between two protein sequences.

Listing 5.7

```
>> [score alignment start]=swalign('QHKATPCCM', 'DVQTATPECM',
'SCORINGMATRIX', 'PAM40', 'GAPOPEN', 4, 'EXTENDGAP', 8)

score =

   17.5000

alignment =

QHKATP-CCM
|  ||| | |
Q-TATPEC-M

start =

     1
     3
```

end-listing-5.7

5.6.3 BLAST in Matlab

The Basic Local Alignment Search Tools (BLAST) is a tool used to rapidly perform local alignments of a query sequence (or a set of query sequences) against a database of sequences. The National Center for Biotechnology Information (NCBI) hosts BLAST and a number of databases against which to search so that users can search via the Internet. The Bioinformatics Toolbox further supports a wrapper which allows one to query NCBI's BLAST server from within MATLAB via the blastncbi function. This function works in tandem with the getblast function, which is used to fetch the results of the query from the server.

When using BLAST one must first determine which program to use. A BLAST program determines the type of the query sequence and the result sequences (i.e. amino acid or nucleotide). For instance, blastn uses a nucleotide query sequence and searches against nucleotide sequences.

Another useful search parameter is the specification of a database to search. Each database only supports certain BLAST programs. For instance, the nr (non-redundant) database can only be searched using a nucleotide program.

Each of these parameters can be used with the *blastncbi* function which initiates a query on NCBI's BLAST server. This function simply returns a vector where the first element is the request ID and the second element is the request time of execution. The request ID is a unique key used to identify and fetch your results from the server. The request time of execution is the time, in minutes, anticipated until the completion of the results are available.

The following example initiates a BLAST query for a nucleotide sequence in the *nr* database using the *blastn* program.

Listing 5.8

```
>> [rid rtoe] = blastncbi('GCGCGTCTGTTCTGTGGAACAGGAGGCAGTTGTTTTCCGTCCGGCT', ...
                       'blastn', 'Database', 'nr')

rid =
TVR2WPE501R

rtoe =
2
```

end-listing-5.8

Once the query has been submitted, you need to retrieve the results from NCBI. This can be done using the *getgenbank* function, which returns the results of the query, if it has finished. In an attempt to assure that the query has completed, use the `WaitTime` parameter and set it to the request time of execution returned from *blastncbi*. `WaitTime` specifies the time, in minutes, to wait for results and defaults to 0.

Listing 5.9

```
>> results = getblast(rid, 'WaitTime', 16)

results =
            RID: 'TVR2WPE501R'
      Algorithm: 'BLASTN 2.2.29+'
          Query: 'Length=46'
       Database: 'Nucleotide collection (nt)'
           Hits: [1x50 struct]
     Statistics: [1x959 char]
```

end-listing-5.9

If more than one result is returned, the user can iterate through a list. MATLABãlso provides the function `blastread` to read data from NCBI BLAST report.

5.7 More Alignment Functions in MATLAB

This example illustrates the use of functions provided in MATLAB that extracts portions of the sequence using the features annotated on the sequence. The extracted sequences are then aligned and gap characters are inserted into the extracted sequence based on the alignment information generated by *nwalign*.

Initially two nucleotide are retrieved sequences from the GenBank database. The two sequences are for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1) identified by accession numbers `AF509094` and `DQ094287` respectively.

Listing 5.10

```
hsn01 = getgenbank('AF509094');
hsn02 = getgenbank('DQ094287');
```

end-listing-5.10

Next, the `CDS` feature annotated on the sequence are extracted as the coding region from the two nucleotide sequences.

Listing 5.11

```
hsn01_cds = featuresparse(hsn01,'feature','CDS','Sequence',true);
hsn02_cds = featuresparse(hsn02,'feature','CDS','Sequence',true);
```

end-listing-5.11

These nucleotide sequences are converted into amino acid sequences using the MATLAB function *nt2aa*, and aligned using *nwalign*. The resulting alignment is stored in the variable `alignment`:

Listing 5.12

```
[sc,alignment] = nwalign(nt2aa(hsn01_cds), ...
                         nt2aa(hsn02_cds), 'extendgap',1);
```

end-listing-5.12

A portion of the `alignment` matrix is shown below. This matrix has three rows where the first and third row correspond to the sequences and second row uses a standard notion for marking the alignment. It should be noted the deletion characters, `-`, have been inserted into the original sequences where appropriate.

Listing 5.13

```
QKQEIKMNPNQKIMTIGSICMVIGMISLVLQIGNMISIWASHSI   . . .
      |||||||:||||||||  |::||:||:|||||||:||||
------MNPNQKIITIGSICMVTGIVSLMLQVGNMISIWVSHSI   . . .
```

end-listing-5.13

The function *seqinsertgaps* is used to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences. This function inserts three gap characters in the nucleotide sequence corresponding to a single gap character in the amino acid sequence. Thus the DNA sequence alignments are forced to occur at codon boundaries.

Listing 5.14

```
hsn01_aligned = seqinsertgaps(hsn01_cds,alignment(1,:))
hsn02_aligned = seqinsertgaps(hsn02_cds,alignment(3,:))
```

end-listing-5.14

By the way of an example, aligned can be used as input to other functions such as *dnds* used for calculating synonymous and non-synonymous substitutions rates of the codon-aligned nucleotide sequences. By setting Verbose to true, codons considered in the computations and their amino acid translations are displayed.

Listing 5.15

```
[dn,ds] = dnds(hsn01_aligned, hsn02_aligned, 'verbose', true)
```

end-listing-5.15

seqpdist: MATLAB includes a general purpose pair-wise similarity computation function. As an example, the following code snippet reads all thirty-two (32) sequences from a FASTA library file `pf00002.fa`. These sequences are stored in an array `seqs`.

Listing 5.16

```
seqs = fastaread('pf00002.fa');
```

end-listing-5.16

The function *seqpdist* next an alignment scoe for every possible pair of sequences:

Listing 5.17

```
dist = seqpdist(seqs,'Method','alignment-score',...
                'Indels','pairwise-delete',...
                'ScoringMatrix','pam250');
```

end-listing-5.17

5.8 Further Readings

Linking of the various tables available in the Entrez web resource is described [1]. As whole genome sequences became available, they were also made available by the NCBI through this resource [2]. Enterez further expanded its score and included Conserved Domain Database (CDD) structure information [3]. Entrez's integration with Online Mendelian Inheritance in Man (OMIM) database utilized in medical genetics [4, 5]. Further enhancements to the gene centered information in Entrez are described in [6].

The first contribution to the field of sequence alignment is considered to be the method for global alignment proposed by Needleman and Wunsch [7]. This algorithm used a fixed penalty for a gap regardless of the length of the gap inserted and was later shown to run in cubic time. Quadratic time algorithms have been proposed and a fairly comprehensive review of dynamic programming methods is provided by Miller and Pearson in [8].

A classic text on sequence alignment is by Sankoff and Kruskal [9]. The seminal paper that perhaps revolutionized the field with the proposed method for local alignment was the paper by Smith and Waterman [10]. This was a very influential paper and has since then most local alignment methods are also referred to as "Smith Waterman" methods. The paper does not specify the implementation details or provide any complexity analysis of the algorithm. Huang provided a quadratic time linear space implementation of this local alignment algorithm [11]. Yamaguchi provided a hardware implementation of the algorithm using FPGA arrays [12].

Various types of alignments are reviewed in the work by Waterman [13] as well as consensus based methods are provided in [14].

Further Readings

1. Gibney, G., Baxevanis, A.D.: Searching ncbi databases using entrez. Curr. Protoc. Hum. Genet., Chapter 6:Unit6.10 (October 2011)
2. Tatusova, T.A., Karsch-Mizrachi, I., Ostell, J.A.: Complete genomes in www entrez: data representation and analysis. Bioinformatics 15(7-8), 536–543 (1999), Journal Article England
3. Marchler-Bauer, A., Lu, S., Anderson, J.B., Chitsaz, F., Derbyshire, M.K., DeWeese-Scott, C., Fong, J.H., Geer, L.Y., Geer, R.C., Gonzales, N.R., Gwadz, M., Hurwitz, D.I., Jackson, J.D., Ke, Z., Lanczycki, C.J., Lu, F., Marchler, G.H., Mullokandov, M., Omelchenko, M.V., Robertson, C.L., Song, J.S., Thanki, N., Yamashita, R.A., Zhang, D., Zhang, N., Zheng, C., Bryant, S.H.: Cdd: a conserved domain database for the functional annotation of proteins. Nucleic Acids Res 39(Database issue), D225–D229 (2011)
4. Hamosh, A., Scott, A.F., Amberger, J.S., Bocchini, C.A., McKusick, V.A.: Online mendelian inheritance in man (omim), a knowledgebase of human genes and genetic disorders. Nucleic Acids Res 33(Database issue), D514–D517 (2005)
5. McKusick, V.A.: Mendelian inheritance in man and its online version, omim. Am. J. Hum. Genet. 80(4), 588–604 (2007)
6. Maglott, D., Ostell, J., Pruitt, K.D., Tatusova, T.: Entrez gene: gene-centered information at ncbi. Nucleic Acids Res 39(Database issue), D52–D57 (2011)
7. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48(3), 443–453 (1970), Journal Article England
8. Pearson, W.R., Miller, W.: Dynamic programming algorithms for biological sequence comparison. Methods Enzymol 210, 575–601 (1992), Lm04969/lm/nlm Lm05110/lm/nlm Comparative Study Journal Article Research Support, U.S. Gov't, P.H.S. United states
9. Sankoff, D., Kruskal, J.B.: Time Warps, String Edits, and Macromolecules: The Theory and Practice of String Comparison. Addison-Wesley, Reading (1983)
10. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147(1), 195–197 (1981), Journal Article England
11. Huang, X.Q., Hardison, R.C., Miller, W.: A space-efficient algorithm for local similarities. Comput. Appl. Biosci. 6(4), 373–381 (1990), Dk01589/dk/niddk Dk27635/dk/niddk Lm05110/lm/nlm Comparative Study Journal Article Research Support, U.S. Gov't, P.H.S. England Cabios
12. Yamaguchi, Y., Maruyama, T., Konagaya, A.: High speed homology search with fpgas. Pac. Symp. Biocomput., 271 (2002), UI - 21926494 NOT IN FILE
13. Waterman, M.: Mathematical Methods for DNA Sequences. CRC Press, Boca Raton (1989)
14. Waterman, M.S., Jones, R.: Consensus methods for dna and protein sequence alignment. Methods Enzymol 183, 221–237 (1990), Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, Non-P.H.S. Research Support, U.S. Gov't, P.H.S. United states

5.9 Exercises

1. Circle the list of boolean operators that you can use with Entrez:

   o AND
   o OR
   o NOT

2. Which of the following connectors would you use to retrieve DNA sequences related to Cystic Fibrosis.

   o Cystic AND Fibrosis
   o Cystic OR Fibrosis
   o Cystic OR NOT Fibrosis
   o "Cystic Fibrosis"

3. Explain the difference between two search terms: (a) New AND Mexico (b) "New Mexico". Which of the terms (a) or (b) is likely to yield more results. Explain.
4. The C-reactive protein in humans is associated with an increased risk of thrombosis and is also associated with fibrosis. Which of the following search terms should you use to retrieve the most accurate set of results.

   o C-reactive protein, thrombosis, fibrosis
   o C-reactive protein, human, increased risk of thrombosis, fibrosis
   o C-reactive protein, human
   o C-reactive protein, human, thrombosis, fibrosis

5. Appropriately combine the query terms in Question 4 and run a search on the Entrez server. Evaluate your results. Can you comment on the precision of your result set for each of the query ran.
6. Explain why it is difficult to estimate the information retrieval *recall rate* in biological databases.
7. Conduct a search of Entrez on Matrix and Scaffold Attachment Regions (S/MARs). Write a 1-page summary describing why these regions are important, how many sequences related to S/MARs are currently available in the GenBank. What other databases amongst those housed at NCBI contain information on S/MARs.
8. Which of the following is not an alignment between sequences s = `ATTACG` and t = `TTAG`?

```
   (a)          (b)          (c)          (d)
AT-TACG      --ATTACG--      ATACG       ATTACG
|| :|||                     |||:         ||| |
TTAG---      TT------AG      TTAG-       -TTA-G
```

9. Assume that the score for a match is a +1. The penalty for a mismatch is -2 and for every gap is -1. What is the score for each of the following alignments?

```
   (a)          (b)          (c)          (d)
AT-TACG        ATTAC        ATACG        ATTACG
:| :|||        :| ||        |||:        | || |
TTAG---        GT-AC        TTAG-        ACTA-G
```

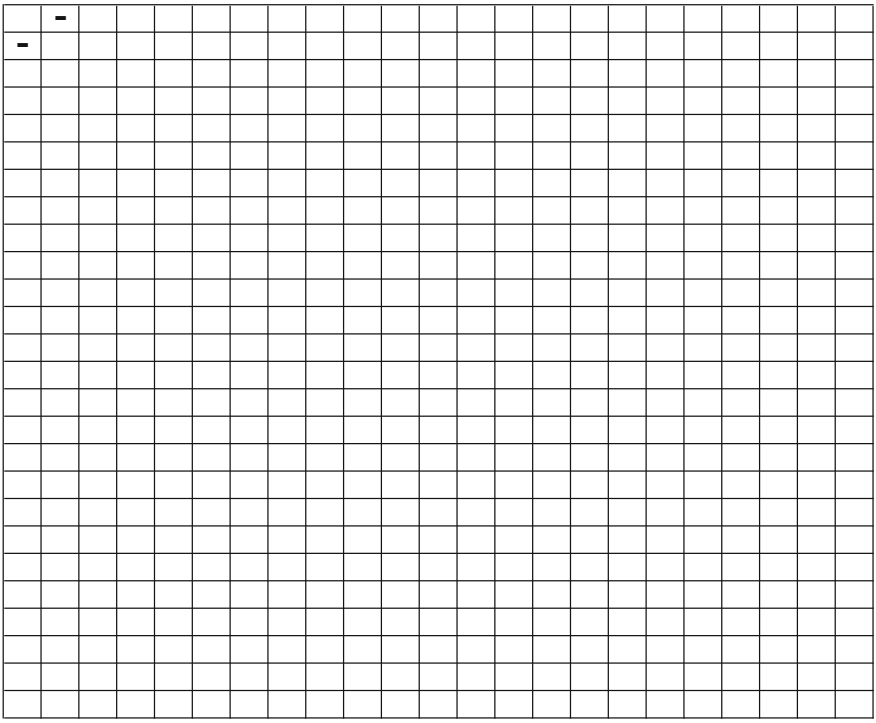
10. Find all optimal global alignments between strings s = `AAAGGC` and t = `AAGGC` assuming a unit cost model.
11. Which of the following represents a local alignment between sequences s = `AATACG` and t = `TTTACT`?

```
(a)          (b)          (c)          (d)
TAC          AATACG       AATACG-      -TAC-
|||          |||          ::|||        |||
TAC          --TAC-       TTTAC-T      TTACT
```

12. Find the optimal local alignment between strings s = `AAAGGCATT` and t = `ATAGAGCCAGTT` assuming a unit cost model.
13. Find the optimal location of occurrence of the substring s = `ATTA` in the sequence t = `ATGGTATATAAGCCGAT`.
14. Given any two sequences and considering a similarity based cost model what can you say about which score, local similarity or global similarity, will be higher. Justify.
15. Given that the scoring scheme is S(a,a) = +1, and S(a,-) = S(-, a) = S(a,b) = -1, find the global alignment between sequences below. Clearly show the alignment grid and list all the optimal alignments.

```
ACCCGGGTTGC
ATCCTGGGC
```

16. What values would you change in the scoring matrix so that indels are *preferred* over substitutions in the final alignment.
17. Nucleotide similarity scores under the unit cost model are defined as $s(u, u) = +1$, $s(u, v) = -1$, $s(u, -) = -1$, and $s(-, u) = -1$. Compute the local alignment between U=`ATTAGGAATTAA` in sequence V = `CCACCATTTAATTT`. Use the empty dynamic programming grid provided in Question 15 above.
18. What values in the scoring matrix will result producing alignments that are generally gapless and *prefer substitutions* to insertion and deletion.
19. A human mitochondrion sequence approximately 2,500 bp long having a GenBank accession number of `BD190340` needs to be compared with



a mitochondrian sequence of approximately 6400 bp long having a GenBank accession number of `XM_508037`. Compare the two sequences using a dot plot analysis. Identify the regions of homology between the two sequences in the dot plot and compare your result by alignment with Smith Waterman function.

20. The following MATLAB code segment is used for retrieving a partial sequence from GenBank. GenBank contains large reference sequences created by evaluating a consensus from many smaller sequences. Generally, the accession number for reference nucleotide and protein sequences use a prefix of `NC_` and `NP_` respectively.

```
>> human = getgenbank('NC_000011', ...
                      'partialseq', [5225224, 5227311], ...
                      'sequenceonly', true);
>> chimp = getgenbank('NC_006478', ...
                      'partialseq', [4976170, 4978435], ...
                      'sequenceonly', true);
```

The code snippet above extract the DNA sequence of beta globin gene in human and chimp from their corresponding reference sequences. Compare

these two sequences using a (a) dot plot, (b) Smith-Waterman, and (c) Needleman-Wunch methods. Comment on your results.

21. Implement a function for performing semi-global alignment in MATLAB. You are allowed to use any of the existing functions (i.e. nwalign, swalign) provided in MATLAB for your implementation. Your function should include an argument for passing the scoring matrix.
    Run your code and the result of finding the following pattern in a DNA sequence. Try running the search with at least two scoring matrices of your choice.
    `Pattern: 'TTAATA'`
    Sequence to be searched:
    `GGAGTAGGATGGGATTAGCCATACCACTATTACCCAGAATAA`
    `AACCTATTGGGATTTACCCTATTATTATATTATTTCCGAGAG`

Chapter 6
Protein Alignments

Protein sequence data has begun playing a significant role in biology and biochemistry. In the past, protein sequence determination was usually one of the last steps in the characterization of a protein. Given the advancements of the Human Genome Project, one can first sequence all the genes and use sequence analysis to infer function using sequence alignment methods.

With the development of high performance sequence comparison methods, researchers have begin making discoveries purely based on sequence comparison. For example, in a surprising discovery made using computational techniques, a tumor suppressor gene in humans was related to yeast and *E. coli* DNA repair enzymes. This discovery, albeit the result of a similarity search, helped researchers better study the nature of mutations in oncogenes. As whole genomes from a variety of organisms become available, protein sequence comparison will become indispensable for understanding biological function.

The power of protein sequence comparison stems from the fact that a large portion of protein sequence information is preserved throughout the evolutionary process. The history of a protein can often be traced back 12 billion years, and may be compared with other homologous proteins that share a common ancestor. Homologous proteins always share a common three-dimensional folding structure and often share common active sites or binding domains, and frequently share common functions.

Principally, there are two reasons for comparing and aligning protein sequences:

- To obtain an accurate alignment
- To search a database with a newly discovered protein sequence and identify its functions via analogous known proteins

Underlying principles and techniques for sequence comparison as applied to proteins are discussed in this chapter.

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_6

6.1 Scoring Matrices

All algorithms to compare protein sequences rely on some scheme to score the equivalence of of each of the 210 possible pairs of amino acids. (i.e. 190 pairs of different amino acids + 20 pairs of identical amino acids). Most scoring schemes represent the 210 pairs of scores as a 20×20 matrix of similarities where identical amino acids and those of similar character (e.g. I, L) give higher scores compared to those of different character (e.g. I, D). Since the first protein sequences were obtained, many different types of scoring scheme have been devised. The most commonly used are those based on observed substitution and of these, the 1976 Dayhoff matrix for 250 PAMs has until recently dominated.

6.1.1 Identity Matrix

This is the simplest scoring scheme; amino acid pairs are classified into two types: identical and non-identical. Non-identical pairs are scored 0 and identical pairs given a positive score (usually 1). The scoring scheme is generally considered less effective than schemes that weight non-identical pairs, particularly for the detection of weak similarities. The normalized sum of identity scores for an alignment is popularly quoted as "percentage identity", but this value can be useful to indicate the overall similarity between two sequences, there are pitfalls associated with the measure, particularly when percentage identity is the statistic used for an reporting the similarity between two sequences. However, the expected value of percentage identity is strongly dependent upon the length of alignment; for example, an alignment of length 200 showing 30% identity is more significant than an alignment of length 50 with the same identity.

6.1.2 Chemical Similarity Scoring

The aim with chemical similarity scoring schemes is to give greater weight to the alignment of amino acids with similar physiochemical properties. This is desirable since major changes in amino acid type could reduce the ability of the protein to perform its biological role and hence the protein would be selected against during the course of evolution. Researchers classified amino acids on the basis of polar or non-polar character, size, shape and charge and gives a score of 6 to inter-conversions between identical rare amino acids (e.g. F, F). The score was reduced to 0 for substitutions between amino acids of quite different character (e.g. F, E).

6.1.3 Observed Substitutions

Scoring schemes based on observed substitutions are derived by analyzing the substitution frequencies seen in alignments of sequences. This is something of a chicken and egg problem, since in order to generate the alignments, one really needs a scoring scheme but in order to derive the scoring scheme one needs the alignments! Early schemes based on observed substitutions worked from closely related sequences that could easily be aligned by inspection. More recent schemes have had the benefit of the earlier substitution matrices to generate alignments on which to build. Long experience with scoring schemes based on observed substitutions suggests that they are superior to simple identity, genetic code, or intuitive physiochemical property schemes. Two examples of such scoring matrices are PAM and BLOSUM. These are discussed in more detail below.

6.1.4 PAM Scoring Matrix

The PAM or Accepted Point Mutation scheme for scoring amino acid pairs was that developed by Dayhoff and coworkers. They presented a method for estimating the matrix M from the observation of 1572 accepted mutations between 34 super-families of closely related sequences. The system arose out of a general model for the evolution of proteins. Dayhoff and coworkers examined alignments of closely similar sequences where the the likelihood of a particular mutation (eg. A-D) being the result of a set of successive mutations (eg. A-x-y-D) was low.

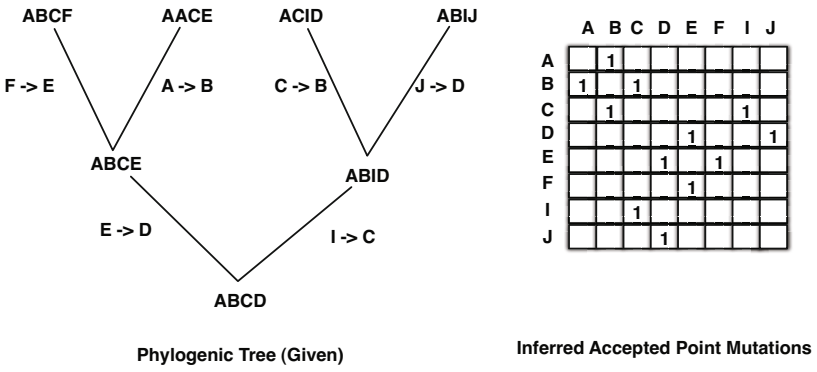


Fig. 6.1 The phylogenetic tree is used as a basis for computing the mutation frequencies thus defines the accepted point mutations. By constructing a phylogenetic tree, the PAM matrix construction takes into account the inferred accepted mutation frequencies through the evolutionary history of related proteins.

Consider for example the phylogenetic tree and the corresponding accepted point mutations shown in Figure 6.1. The underlying assumption is that the likelihood of amino acid X replacing another amino acid Y is the same as that of Y replacing X. Correspondingly, when a "1" is entered into the cell XY, a "1" is also entered into the cell YX. In general, when such mutations are being accumulated, the cell entries will be incremented. Furthermore, the counting process is continues up the phylogenetic tree and is not simply limited to the leaves of the tree. This enables the computation of the accepted point mutations to go back in evolutionary time history and thus produces a sharper rendition of the which amino acid mutations are relatively more acceptable. Thus, the replacement of amino acid I with C and the replacement of E with D is not observed in the leaves of the phylogenetic tree. In order to record these and other acceptable point mutations that occurred in the past, one has to construct the phylogenetic tree and use the augmented set of mutations as the basis recording the frequency counts.

A	Ala																			
R	Arg	30																		
N	Asn	109	17																	
D	Asp	154	0	532																
C	Cys	33	10	0	0															
Q	Gln	93	120	50	76	0														
E	Glu	266	0	94	831	0	422													
G	Gly	579	10	156	162	10	30	112												
H	His	21	103	226	43	10	243	23	10											
I	Ile	66	30	36	13	17	8	35	0	3										
L	Leu	95	17	37	0	0	75	15	17	40	253									
K	Lys	57	477	322	85	0	147	104	60	23	43	39								
M	Met	29	17	0	0	0	20	7	7	0	57	207	90							
F	Phe	20	7	7	0	0	0	0	17	20	90	167	0	17						
P	Pro	345	67	27	10	10	93	40	49	50	7	43	43	4	7					
S	Ser	772	137	432	98	117	47	86	450	26	20	32	168	20	40	269				
T	Thr	590	20	169	57	10	37	31	50	14	129	52	200	28	10	73	696			
W	Trp	0	27	3	0	0	0	0	0	3	0	13	0	0	10	0	17	0		
Y	Tyr	20	3	36	0	30	0	10	0	40	13	23	10	0	260	0	22	23	6	
V	Val	365	20	13	17	33	27	37	97	30	661	303	17	77	10	50	43	186	0	17
		Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V

Fig. 6.2 Accepted mutation frequencies (× 10) observed in 34 super-families of proteins grouped into 71 evolutionary trees. The table represents a total of 1,572 exchanges. Thirty-five (35) exchanges of the total 190 possible are never observed. Fractional exchanges result from ambiguity in ancestral sequences.

Dayhoff constructed such a frequency matrix of using 34 super-families grouped into 71 evolutionary trees. In order to minimize the effect of successive accepted mutations at a given site, the sequences within a tree were less that 15% different from one another. As shown in Figure 6.2, 35 of the 190 possible exchanges were never observed. The data values in the table have been multiplied by 10. The total number of mutations observed was 1,572 with 83 being the largest number of mutations observed between any two amino acids, *Asp* and *Glu* in this case.

Mutability and Occurrence Probability of Amino Acids

A mutation matrix, denoted by M, describes the probabilities of amino acid mutations for a given period of evolution.

$$Pr(Amino\ Acid_i \rightarrow Amino\ Acid_j) = M_{ij} \tag{6.1}$$

Implicit in this equation is the relative ease with which an amino acid changes – a parameter that is defined as the probability that a given amino acid will change in a given small evolutionary interval. This parameter is called the *relative mutability* of the amino acid. This parameter is computed as the ratio of the number of times that the amino acid is replaced and the number of times it is observed in all the alignments. All the sequences from the various trees are combined and treated in an aggregate manner for computing the relative mutability. The number of changes are computed by considering the pairwise alignments as shown in Figure 6.3.

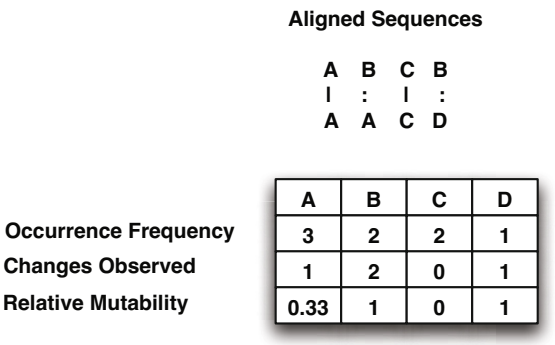


Fig. 6.3 The relative mutability of the various amino acids is computed as the ratio of the number of changes observed in the amino acid and the number of times the amino acid occurs and is thus exposed to mutations

The relative abundance of each amino acid observed in the accepted point mutation is also captured. Dayhoff captured the relative mutability a scale such that the mutability of `Ala` was assigned a value of 100. The relative mutability is converted to a probability value by scaling with a coefficient λ which is related to the definition of PAM-1. If we denote the relative mutability of an amino acid i as μ_i, the scaling coefficient is defined such that $(\mu_i \times \lambda)$ is the probability that the amino acid i will change in the evolutionary period of interest. Furthermore, if we let the normalized frequency of the amino acid i be f_i, the probability that a given amino acid will change is given by Eq. 6.2.

$$\sigma = \sum_{i}^{20} f_i \mu_i \lambda \tag{6.2}$$

The summation σ is the probability that a change will be observed at any position in the given sequence. In accordance to the stipulation of PAM-1, this probability is to equal 0.01 as PAM-1 corresponds to an evolutionary distance where one change occurs per 100 residues. This enables us to calculate the value for λ as shown in Figure 6.4 and which equals $\frac{0.01}{75.2} = 1.329 \times 10^{-4}$.

		Mutability (μ_i)	Normalized Frequency (f_i)	$\mu_i \times f_i$
A	**Ala**	100	0.087	8.700
R	**Arg**	065	0.041	2.665
N	**Asn**	134	0.040	5.360
D	**Asp**	106	0.047	4.982
C	**Cys**	020	0.033	0.660
Q	**Gln**	093	0.038	3.534
E	**Glu**	102	0.050	5.100
G	**Gly**	049	0.089	4.361
H	**His**	066	0.034	2.244
I	**Ile**	096	0.037	3.552
L	**Leu**	040	0.085	3.400
K	**Lys**	056	0.081	4.536
M	**Met**	094	0.015	1.410
F	**Phe**	041	0.040	1.640
P	**Pro**	056	0.051	2.856
S	**Ser**	120	0.070	8.400
T	**Thr**	097	0.058	5.626
W	**Trp**	018	0.010	0.180
Y	**Tyr**	041	0.030	1.230
V	**Val**	074	0.065	4.810

Fig. 6.4 The sum of the product of relative mutability (μ_i) and normalized observed amino acid frequency (f_i) is used for computing the coefficient λ that provides the scaling factor for converting the mutability to the probability that an amino acid changes during a given evolutionary distance such as PAM-1 where one (1) point mutation is accepted per 100 amino acids

PAM

The PAM is a mutation probability matrix. Each element of the mutation probability matrix (M_{ij}) gives the probability of the amino acid in row i mutating to the amino acid in column j after a particular evolutionary time. In the PAM model of evolution, amino acids mutate randomly and independently from one another but according to some predefined probabilities. A 1-PAM mutation matrix describes an amount of evolution which will change, on the average, 1% of the amino acids.

The probability of changing an amino acid a to amino acid b is computed as a conditional probability that a will change to b given that a will change. Again, the probability that the amino acid a changes to amino acid b is the fraction $\frac{f_{ab}}{f_a}$ where frequency f_{ab} is obtained from Figure 6.2 and $f_a = \sum_{j,j\neq a} f_{aj}$. Thus, the element M_{ab} of PAM-1 is computed using Eq. 6.3:

$$\begin{aligned} M_{ab} &= P(a \rightarrow b) \\ &= P(a \rightarrow b | a \ changed) \times P(a \ changed) \\ &= \frac{f_{ab}}{f_a} \times \mu_a \times \lambda \end{aligned} \tag{6.3}$$

The non-diagonal elements of the PAM-1 matrix having been thus computed, the diagonal elements are computed as: $M_{aa} = 1 - \sum_{b\neq a} M_{ab}$. The PAM-1 matrix computed in this manner is shown in Figure 6.5.

A	**Ala**	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
R	**Arg**	1	9914	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
N	**Asn**	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
D	**Asp**	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	4	3	0	0	1
C	**Cys**	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Q	**Gln**	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
E	**Glu**	10	0	7	56	0	35	9864	4	2	3	1	4	2	0	3	4	2	0	1	2
G	**Gly**	21	1	12	11	1	2	7	9935	1	0	1	2	2	1	3	21	3	0	0	5
H	**His**	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
I	**Ile**	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	32
L	**Leu**	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
K	**Lys**	2	37	25	6	0	12	7	2	2	4	1	9926	19	0	3	8	11	0	1	1
M	**Met**	1	1	0	0	0	2	0	0	0	5	8	4	9875	1	0	1	2	0	0	4
F	**Phe**	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
P	**Pro**	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
S	**Ser**	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9841	38	5	2	2
T	**Thr**	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
W	**Trp**	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1	0
Y	**Tyr**	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9946	1
V	**Val**	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9902
		Ala **A**	**Arg** **R**	**Asn** **N**	**Asp** **D**	**Cys** **C**	**Gln** **Q**	**Glu** **E**	**Gly** **G**	**His** **H**	**Ile** **I**	**Leu** **L**	**Lys** **K**	**Met** **M**	**Phe** **F**	**Pro** **P**	**Ser** **S**	**Thr** **T**	**Trp** **W**	**Tyr** **Y**	**Val** **V**

Fig. 6.5 The mutation probability matrix for the evolutionary distance of 1 PAM. Elements of the matrix are multiplied by 10,000. The elements of the matrix, M_{ij} give the probability that the amino acid in row i will replace the amino acid in column j after the specified evolutionary distance which in this case is one (1) accepted point mutation per 100 amino acids.

The PAM model is an empirical one that scales probabilities of change from one amino acid to another in terms of a unit which is defined as an expected 1% change between two amino acid sequences. If we have a probability or frequency vector p capturing the frequency of occurrence of amino acids, the product $M \times p$ gives the probability vector, i.e. the expected frequency of p after a random evolution equivalent to 1-PAM unit. Similarly, after k units

of evolution (what is sometimes called k-PAM evolution) a frequency vector p will be changed into the frequency vector $M^k p$.

This PAM model of evolution is symmetric, and thus we cannot distinguish the probability of amino acid i evolving to j from the probability of j evolving to i. This implies a simple relation for the entries in any M^k:

$$f_i(M^k)_{ji} = f_j(M^k)_{ij}$$

PAM matrices for different values of evolutionary time spans are provided in MATLAB. The default ordering of amino acids is `A R N D C Q E G H I L K M F P S T W Y V B Z X *` with * denoting the deletion character. The PAM-n matrix may be requested by providing a value of $10 \leq n \leq 500$ as the parameter to function `pam`. However, the value of n must be multiple of 10. The order in which the values are reported may be adjusted by defining a *Order* property. Note that the scores for deletion character are not included in the second command listed below.

Listing 6.1

```
PAM50 = pam(50);
PAM250 = pam(250,'Order','CSTPAGNDEQHRKMILVFYW');
```

end-listing-6.1

Dayhoff Matrix

A Dayhoff matrix is computed from a 250-PAM mutation matrix. The Dayhoff matrix is a log-odds scoring matrix that is typically used for the standard dynamic programming methods of sequence alignment. The Dayhoff matrix entries are related to M^{250} matrix by:

$$D_{ij} = 10 \log \frac{(M^{250})_{ij}}{f_i}$$

Thus the D_{ij} matrix is a log odds ratio of the observing the amino acid i after it was mutated from an amino acid j (numerator) and the random occurrence of the amino acid i (denominator).

Aligning sequences by dynamic programming using Dayhoff matrices is equivalent to finding the alignment which maximizes the probability that the two sequences evolved from an ancestral sequence. More precisely, we are comparing the logarithm of the probabilities of two events, namely the event a) that the two sequences are independent of each other, and the event (b) that the two sequences have evolved from some common ancestral sequence after some amount, t, of evolution. The event (a) has the probability equal to the product of the individual frequencies of the two amino acids, i.e.

$$Pr(\textit{independent alignment of } i \textit{ and } j) = f_i f_j$$

And the probability of the competing event (b) is:

$$\begin{aligned} Prob(event\ (b)) &= Pr(i\ and\ j\ have\ a\ common\ ancestor\ x) \\ &= \sum_x f_x Pr(x \to i) Pr(x \to j) \\ &= \sum_x f_x (M^t)_{ix}\ (M^t)_{jx} \\ &= \sum_x f_j (M^t)_{ix}\ (M^t)_{xj} \\ &= f_j (M^{2t})_{ij} = f_i (M^{2t})_{ji} \end{aligned} \tag{6.4}$$

The entries of the Dayhoff matrix are the logarithm of the quotient of these two probabilities of event (b) and event (a). Since dynamic programming maximizes the sum of the similarity measure, the sum of the logarithms or the product of these quotients of probabilities is maximized. As a result, dynamic programming finds the alignment which maximizes the overall probability the amino acids in the alignment of having evolved from a common ancestor. In this manner a maximally likely alignment, measured against the null hypothesis of being independent, is obtained.

The resulting measure of similarity is a sum of Dayhoff entries which is 10 times the logarithm of this probability. For example, the matching described by with a similarity of 238 means that the probability of both sequences coming from a common ancestors 10^{24} times the probability that they are independent.

MATLAB function `dayhoff` provides this matrix for use in computation of protein similarities. The command below returns Dayhoff's original PAM-250 substitution matrix into the variable df. The order of amino acids in the matrix is `A R N D C Q E G H I L K M F P S T W Y V B Z X *`, with * designating the deletion character.

Listing 6.2

```
df = dayhoff;
```

end-listing-6.2

6.1.5 BLOSUM Matrix

Although Dayhoff's method was pioneering in the field, nowadays we are able to estimate M by somewhat more accurate methods. One of these methods is the one utilized for computing the BLOSUM (BLOcks SUbstitution Matrix) scoring matrices family. The derivation of BLOSUM matrices is somewhat simpler than the PAM matrices. Similar to the construction of PAM matrices, the BLOSUM matrix also begins with the data set that can be *trusted*.

This alternative approach has been developed by Henikoff & Henikoff where the BLOSUM matrices were developed by starting with a set of

proteins from the public databases that had been grouped into related families. From these they obtained blocks of aligned sequences where a block is defined as an ungapped alignment of a relatively conserved region of a family of proteins. Given the alignment shown the similarity matrix computation proceeds by taking the log odds value of the probability of observing a pair of amino acids in the alignment to the occurrence of the pair of amino acid occurring purely by chance. For example, consider the following alignment block defined over amino acids A, B, C and D.

A	B	B	C	A
B	B	A	D	C
A	B	D	D	D
A	A	A	B	C
A	B	A	D	C
A	B	A	D	C

This sample block comprises of an observation of 30 amino acids, out of which 11 are A, 8 are B, 5 are C, and 6 are D. Thus the observed proportions of the amino acids A, B, C and D are $\frac{11}{30}$, $\frac{8}{30}$, $\frac{5}{30}$, and $\frac{6}{30}$ respectively. These background probabilities are used computing the amino acid pairings by pure random chance based solely on the relative abundance of the amino acids.

Next, we compute the pairing probabilities from the given alignment block. Recall that each block is an ungapped alignment and therefore we need not concern ourselves with the *indel* character. Each column in the block represents an observation of $\binom{6}{2} = 15$ possible alignment pairs. Since we have a block comprising of 5 columns, there are a total of 75 alignment pairs. Their distribution is as follows:

Amino Acid Pair	Probability or Observed Proportion
A $\leftrightarrow$ A	$\frac{16}{75}$
A $\leftrightarrow$ B	$\frac{14}{75}$
A $\leftrightarrow$ C	$\frac{4}{75}$
A $\leftrightarrow$ D	$\frac{5}{75}$
B $\leftrightarrow$ B	$\frac{10}{75}$
B $\leftrightarrow$ C	$\frac{1}{75}$
B $\leftrightarrow$ D	$\frac{5}{75}$
C $\leftrightarrow$ C	$\frac{6}{75}$
C $\leftrightarrow$ D	$\frac{8}{75}$
D $\leftrightarrow$ D	$\frac{6}{75}$

The computation of the scoring matrix proceeds by comparing the observed pairings of the amino acids in the block with the expected pairings that

would have been observed by chance alone. Letting $p_{x,y}$ be the probability of observing an alignment amino acids x and y within a block, with p_x and p_y being the probabilities of these amino acids' abundance, the estimated likelihood of the observation being a significant is $\frac{p_{xy}}{p_x \cdot p_y}$. A log-likelihoods is used in practice. For the block we are using in our example, these log-likelihoods are as follows:

Amino Acid Pair	Observed Proportion	Expected Proportion	$2 \times \log_2 \left(\frac{\text{observed proportion}}{\text{expected proportion}} \right)$
A $\leftrightarrow$ A	$\frac{16}{75}$	$\frac{11}{30} \cdot \frac{11}{30}$	1.332
A $\leftrightarrow$ B	$\frac{14}{75}$	$\frac{11}{30} \cdot \frac{8}{30}$	1.866
A $\leftrightarrow$ C	$\frac{4}{75}$	$\frac{11}{30} \cdot \frac{5}{30}$	-0.393
A $\leftrightarrow$ D	$\frac{5}{75}$	$\frac{11}{30} \cdot \frac{6}{30}$	-0.275
B $\leftrightarrow$ B	$\frac{10}{75}$	$\frac{8}{30} \cdot \frac{8}{30}$	1.814
B $\leftrightarrow$ C	$\frac{1}{75}$	$\frac{8}{30} \cdot \frac{5}{30}$	-3.474
B $\leftrightarrow$ D	$\frac{5}{75}$	$\frac{8}{30} \cdot \frac{6}{30}$	0.644
C $\leftrightarrow$ C	$\frac{6}{75}$	$\frac{5}{30} \cdot \frac{5}{30}$	3.052
C $\leftrightarrow$ D	$\frac{8}{75}$	$\frac{5}{30} \cdot \frac{6}{30}$	3.356
D $\leftrightarrow$ D	$\frac{6}{75}$	$\frac{6}{30} \cdot \frac{6}{30}$	2.0

The elements of the BLOSUM matrix are found by calculating twice the logarithm (to the base 2) of the ratio of the proportion of times each amino acid combination occurs in any column to the proportion expected under random allocation. This value, shown in the last column is rounded to the nearest integer. For the alignment block shown above the substitution matrix, without the overabundance correction described below, would thus be as follows:

	A	**B**	**C**	**D**
A	1	2	0	0
B	2	2	-3	1
C	0	-3	3	3
D	0	-1	3	2

Since the algorithms that construct the aligned blocks employ substitution matrices, there is a circularity of computation in the procedure as the aligned blocks are needed for the computation of substitution matrices. This circularity is resolved by first using a unitary matrix for bootstrapping the alignment process, using local multiple alignments of more distantly related sequences.

BLOSUM is really a family of matrices. Commonly utilized scoring matrices are BLOSUM-50, BLOSUM-62 and BLOSUM-80. In order to understand the differences between these matrices, a significant shortcoming of the derivation of the matrix must be considered. The substitution matrix derived using the procedure above is influenced heavily by the population of sequences in

the block. Thus, if there are many very closely related proteins in a block, the contribution of the block will be biased towards closely related proteins. Considering the two alignment blocks shown in Fig. 6.6, the substitution matrix derived from block with multiple instances of `ABAA` will quite different from the matrix derived form the block where this sequence is only represented once.

Fig. 6.6 An over-representation of a sequence in a block (left) will yield substitution matrix that is significantly different from a block where sequences are uniformly distributed

Henikoff and Henikoff overcame this problem by treating multiple sequences in a block that are sufficiently close to each other as a *single* sequence. This allowed them to define the various levels of the BLOSUM matrices by differentially weighting the degree of similarity between sequences.

	A Ala	R Arg	N Asn	D Asp	C Cys	Q Gln	E Glu	G Gly	H His	I Ile	L Leu	K Lys	M Met	F Phe	P Pro	S Ser	T Thr	W Trp	Y Tyr	V Val
A **Ala**	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R **Arg**	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N **Asn**	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D **Asp**	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C **Cys**	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q **Gln**	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E **Glu**	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G **Gly**	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H **His**	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I **Ile**	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L **Leu**	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K **Lys**	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M **Met**	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F **Phe**	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P **Pro**	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S **Ser**	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T **Thr**	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W **Trp**	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y **Tyr**	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V **Val**	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Fig. 6.7 The Block Substitution Matrix or BLOSUM matrix. This is BLOSUM 62 of the family of matrices. The index 62 in its name characterizes that sequences that are more than 62% similar in the protein blocks used in its derivation are given a cumulative weight of a single sequence. Generally, the smaller the index of the BLOSUM matrix, the longer the evolutionary time span of its sensitivity. Thus, a BLOSUM 30 matrix will detect more distant relationships than the BLOSUM 80 matrix.

For example, a BLOSUM62 matrix is calculated from protein blocks such that if two sequences are more than 62% identical, then the contribution of these sequences is weighted to sum to one. In this way the contributions of multiple entries of closely related sequences is reduced. Similarly, other matrices are obtained by varying the clustering threshold – the BLOSUM-80 matrix derived using a threshold of 80%identity. A researcher interested in capturing distant relationships would thus utilize a lower-indexed matrix, BLOSUM-50 for instance, as its derivation would have assumed that sequences that 50% or more identical are treated as identical, leaving the other more distant proteins to define the terms of the substitution matrix. The more commonly used BLOSUM-62 matrix is shown in Fig. 6.7.

6.1.6 Matrices Derived from Structure

The most reliable protein sequence alignments may be obtained when all the proteins have had their tertiary structures experimentally determined. Comparison of three dimensional structures also allows much more distantly related proteins to be aligned accurately. Analysis of such alignments should therefore give the best substitution matrices. Researchers have derived substitution frequencies from a number of proteins that have been structurally aligned after grouping them based on function.

6.1.7 Choosing the Right Scoring Matrix

The general consensus is that matrices derived from observed substitution data (e.g. the Dayhoff or BLOSUM matrices) are superior to identity, genetic code or physical property matrices. It is recommended that a mutation data matrix for the distance of 250 PAMs as a result of a study using a dynamic programming procedure to compare a variety of proteins known to be distantly related.

Recently, Altschul has examined Dayhoff style mutation data matrices from an information theoretical perspective. For alignments that do not include gaps he concluded that a matrix of 200 PAMs was most appropriate when the sequences to be compared were thought to be related. However, when comparing sequences that were not known in advance to be related such as when searching a database scanning, a 120 PAM matrix is a better choice. When using a local alignment method, PAM40, PAM120 or PAM250 could be used. The lower PAM matrices will tend to find short alignments of highly similar sequences, while higher PAM matrices will find longer, weaker local alignments.

Henikoff and Henikoff have compared the BLOSUM matrices to multiple PAM matrices by evaluating how effectively the matrices can detect known

members of a protein family from a database when searching with the ungapped local alignment program BLAST. They conclude that overall the BLOSUM 62 matrix is the most effective. However, all the substitution matrices investigated perform better than BLOSUM 62 for a subset of protein families. This suggests that no single matrix is the complete answer for all sequence comparisons. It is probably best to complement the BLOSUM 62 matrix with comparisons using 250 PAM.

6.2 Further Readings

An introductory procedure for constructing Point Acceptance Matrix (PAM) is presented by Dayhoff in [6] and further details are offered in [7]. Altschul provides a comprehensive guidelines for deciding which PAM matrix (PAM 120, PAM 200, etc.) should be chosen under what circumstances [8]. Statistical significance for the basis for the choice of PAM matrices is discussed in [9]. The procedure for constructing the BLOSUM matrix is described by Henikoff in [10] and [11].

Further Readings

1. Ma, J., Wang, S.: Algorithms, applications, and challenges of protein structure alignment. Adv. Protein Chem. Struct. Biol. 94, 121–175 (2014)
2. Stamm, M., Staritzbichler, R., Khafizov, K., Forrest, L.R.: Alignme–a membrane protein sequence alignment web server. Nucleic Acids Res (April 2014)
3. Ritchie, D.W., Ghoorah, A.W., Mavridis, L., Venkatraman, V.: Fast protein structure alignment using gaussian overlap scoring of backbone peptide fragment similarity. Bioinformatics 28(24), 3274–3281 (2012)
4. Léonard, S., Joseph, A.P., Srinivasan, N., Gelly, J.-C., de Brevern, A.G.: mulpba: an efficient multiple protein structure alignment method based on a structural alphabet. J. Biomol. Struct. Dyn. 32(4), 661–668 (2014)
5. Arriagada, M., Poleksic, A.: On the difference in quality between current heuristic and optimal solutions to the protein structure alignment problem. Biomed. Res. Int. 459248 (2013)
6. Dayhoff, M.O., Orcutt, B.C.: Methods for identifying proteins by using partial sequences. Proc. Natl. Acad. Sci. U S A 76(5), 2170–2174 (1979), Journal Article Research Support, U.S. Gov't, P.H.S. United states
7. George, D.G., Barker, W.C., Hunt, L.T.: Mutation data matrix and its uses. Methods Enzymol 183, 333–351 (1990), Gm37273/gm/nigms Journal Article Research Support, U.S. Gov't, P.H.S. United states
8. Altschul, S.F.: Amino acid substitution matrices from an information theoretic perspective. J. Mol. Biol. 219(3), 555–565 (1991), Comparative Study Journal Article England

9. Karlin, S., Altschul, S.F.: Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proc. Natl. Acad. Sci. U S A 87(6), 2264–2268 (1990), Gm10452-26/gm/nigms Gm39907-02/gm/nigms Journal Article Research Support, U.S. Gov't, Non-P.H.S. Research Support, U.S. Gov't, P.H.S. United states
10. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. U S A 89(22), 10915–10919 (1992), Comparative Study Journal Article Research Support, U.S. Gov't, P.H.S. United states
11. Henikoff, S., Henikoff, J.G., Alford, W.J., Pietrokovski, S.: Automated construction and graphical presentation of protein blocks from unaligned sequences. Gene. 163(2), GC17–GC26 (1995), Gm29009/gm/nigms Comparative Study Journal Article Research Support, U.S. Gov't, P.H.S. Netherlands

6.3 Exercises

1. Consider the row corresponding to amino acid `R` in a hypothetical scoring matrix utilized for protein. Assume that the values for the columns corresponding to amino acids `A` and `W` are -10 and -1 respectively. Which (check all that apply) of the following inferences can you draw:

   - o Over evolutionary time, `R` is more likely to mutate to `A` than it is to `W`.
   - o Over evolutionary time, `R` is more likely to mutate to `W` than it is to `A`.
   - o `R` is functionally similar to `W`.
   - o `R` is functionally similar to `A`.

2. Assuming that the penalty for a gap is `-4`, compute the score for the following alignment using the PAM-1 matrix shown in Fig. 6.5.

```
  (a)                    (b)
ADAD-RQQ               KA-LMAR
|||:  :|               :| :: :
ADACN-DQ               VAKKN-S
```

3. PAM-1 is a unit of evolutionary time where the amino acids have mutated such that there is 1% change between the evolved sequence and the original sequence. By this analogy would a PAM-100 represent a time span where the evolved sequence changed from the original by 100%? And, what percent change do you expect for evolutionary duration corresponding to PAM-250? Explain your answer.
4. Repeat problem 2 using the BLOSUM-62 matrix shown in Fig. 6.7.
5. Using the scores in BLOSUM-62 matrix, and assuming the score of indel is -4, provide the score for the following alignments. Which alignment is relatively better.

```
HARF-L          HARF--L
 |:: |           | |  |
-AFHIL          -A-FHIL
```

   Is there a indel score value that would make the two alignment scores relatively comparable?

Chapter 7
Multiple Sequence Alignment

Multiple Sequence alignment (MSA) is a generalization of Pairwise Sequence Alignment to multiple sequences. Thus, instead of aligning two sequences, the objective in MSA is to align k sequences simultaneously such an overall functional is optimized. The motivation behind doing a MSA is that it allows us to extract consensus evident in a widely diverse set of sequences. The similarities we observe across a wider range of sequences can help us better understand the evolutionary history of sequences as well as help infer a functional relationship amongst a group of biological sequences. Particularly for protein sequences, MSA can provide insight into the secondary/tertiary structure of proteins and discover critical consensus motifs and common blocks representative of protein domains or the functional units. Generally however, before performing the MSA step, typically we already know that the set of sequences being aligned are related, and our objective is to discover those regions and strength of relatedness.

7.1 Scoring Multiple Sequence Alignment

In a **multiple sequence alignment**, homologous residues amongst a set of sequences are alignment together in a column. Scoring alignments is a prerequisite for generally comparing alignments and in ultimately finding an optimal alignment that minimizes the alignment score. Generally, the computation of the score of a multiple sequence alignment is obtained as a sum of the scores of the individual columns in the alignment. For an alignment comprising of N columns, the alignment scores is computed as:

$$\sum_{i=1}^{N} S(m_i) + \Delta$$

The essential concept of adding the individual scores of the N columns of the multiple alignment where the score of aligning the ith. column is denoted as $S(m_i)$, Although the score of individual columns takes gaps into consideration,

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_7

there is an overall gap introduction cost included in the term Δ in the above equation. However, the cost of gaps in the overall multiple alignment was to be considered to the extent that individual column scores account for it, the term Δ may be excluded from the MSA scoring equation above.

Given the formalization above, the problem of finding an optimal multiple sequence alignment is thus reduced to scoring individual columns. The Sum of Pair or SP measure used to define the score of a column of the alignment. The column score $S(m_i)$ is defined using the pairwise scores:

$$S(m_i) = \sum_{k<l} s(m_i^k, m_i^l)$$

The similarity score is between residues m_i^k and m_i^l is based on appropriate similarity model. For example, a unit cost model may be utilized in the case of DNA sequences and BLOSUM or PAM matrices in case of protein sequences. Unlike in alignment of two sequences, the pairwise scores in a multiple sequence alignment could require the scoring $s(-,-)$. This similarity score is generally considered zero.

Example 7.1

Assume the we have the following MSA. And, assuming the unit model where s(a,a) = 1 and s(a,b) = 0.

A	A	–	G	–
C	–	T	C	T
A	–	T	G	–

The pair wise similarity for columns 1, 3 and 4 is 1 each, and other columns have a pairwise score of zero. The total MSA score is thus 3.

End of Example

Even though the sum of pairs scores is utilized for the computing the heuristically best alignment, the problem of finding the optimal multiple sequence alignment is formulated below.

7.2 Mathematical Formulation for the MSA Problem

A Global Multiple Sequence Alignment of $k > 2$ sequences $X = \{x^1, x^2 \ldots x^k\}$ is obtained by inserting gaps ("_") into the $\{x^1, x^2 \ldots x^k\}$ and thereby transforming them to $\{x'^1, x'^2 \ldots x'^k\}$. The MSA is then defined by a block $k \times L$ where all the k transformed sequences have been arranged in a matrix and have a uniform length of L. The restriction in the MSA matrix is that none of the columns is comprised entirely of gaps.

The evaluation of the MSA is typically done using a Sum of Pair scoring or SP scoring. A scoring matrix like PAM or BLOSUM may be used for scoring the alignment between any pair of sequences. These scores are added to yield the overall MSA scores.

7.3 MSA-Dynamic Programming

The computation of a MSA can, theory, be a generalization of the dynamic programming approach used for pairwise alignment discussed in Section 5.4. In case of three sequences, that have been aligned up to characters x_i, y_j and z_k, the optimal match score may be written in terms of the previously computed match scores and the similarity costs defined by a scoring matrix. In this manner the multiple sequence alignment problem may be posited as an optimization problem where a three dimensional array is computed using the following recursive formula in Eq. 7.1.

$$M_{i,j,k} = Max \begin{cases} M_{i-1,j-1,k-1} + S(x_i, y_j, z_k) \\ M_{i,j-1,k-1} + S('-', y_j, z_k) \\ M_{i-1,j,k-1} + S(x_i, '-', z_k) \\ M_{i-1,j-1,k} + S(x_1, y_j, '-') \\ M_{i,j,k-1} + S('-', '-', z_k) \\ M_{i-1,j,k} + S(x_i, '-', '-') \\ M_{i,j-1,k} + S('-', y_j, '-') \end{cases} \tag{7.1}$$

In general the number of terms to be considered in the recursive formulation with n sequences will be $2^n - 1$. The time complexity for the computation is there $\Theta(2^n \times L^n)$. Thus, the MSA problem is intrinsically an hard NP-complete problem and thus a close approximation to the optimal solution will be acceptable in practice.

7.4 Progressive Alignment Methods

Progressive alignments are a commonly utilized methods for developing multiple sequence alignments. Essentially the process of developing a progressive alignment is based on constructing successive pairwise alignments starting with a standard pairwise alignment of two sequences closest within the group of sequences to be aligned. Subsequently, the next sequence closest sequence is chosen from the group and aligned with the first alignment. This process continues until all sequences have been aligned.

Progressive alignment methods are heuristic algorithms where the optimization process is governed by the objective for minimization of overall pairwise scores. Most algorithms implementing progressive alignment methods use a *guide tree* for establishing an order in which the sequences are merged into the progressively growing multiple alignment. A guide tree is formed by taking all the sequences and applying the principles of agglomerative clustering to construct a binary tree whose leaves represent sequences and internal nodes represent alignments.

7.4.1 Constructing the Guide Tree

The construction of the guide tree for a set of N sequences essentially proceeds as follows:

1. The pairwise similarity (or distance) score matrix is computed.
2. Each of the N sequences is considered to be a singleton group. The inter-group similarity (or distance) is identical to the pair wise similarity computed in the previous step.
3. Groups are merged such that each successive merge step chooses the most similar groups and recomputes the new group's similarity (or distance) to all of the other groups.
4. The merging process stops when all sequences belong to one large group containing all N sequences.
5. The order in which the sequence and groups are merged provides the *guide tree*.

The following example illustrates the application of the above process.

Example 7.2

Consider the following set of sequences:

```
S1 = ATTTACGCCT
S2 = TTAAGCCAT
S3 = TTAATTAACC
S4 = ATTTTCCGGA
S5 = AATTTACCGCCT
```

Assuming a unit gap cost model, the pairwise distance values are computed as follows:

	S1	**S2**	**S3**	**S4**	**S5**
S1	0	4	6	5	2
S2		0	5	7	6
S3			0	8	7
S4				0	5
S5					0

Based on the distance values, the closest sequences in the set are sequences S1 and S5 which are merged into a new sequence S6. The distances of this merged sequence, S6, is next computed by taking the average distance of S1 and S5 to each of the remaining sequences and recoding that into the distance matrix shown below.

	S6	**S2**	**S3**	**S4**
S6	0	5	6.5	5
S2		0	5	7
S3			0	8
S4				0

The smallest distance between any two sequence pairs in the above array is now 5. We can choose to merge any of three pairs that have this minimum distance. Let us choose to merge sequences S6 and S4 into a new group S7.

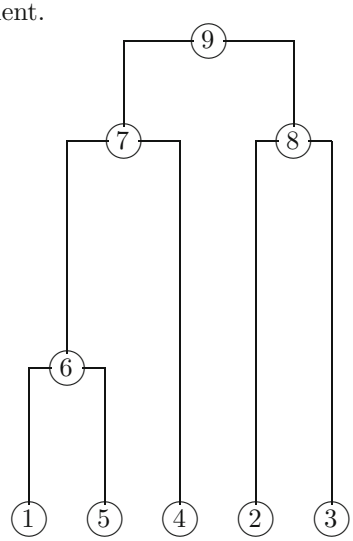
The distance of S7 will be next computed to the remaining sequences. For computing the distance between groups, a weighted average with the members of the group must be used so that all the individual sequence distance are acounted for. Thus, since group S6 already contains two sequences, the distance between S7 and S2 is computed as $\frac{2\times d(6,2)+1\times d(4,2)}{2+1}$ or 6.67. Distance between S7 and S3 is similarly computed. Upon completion of this step, the new distance matrix looks as follows:

	S7	**S2**	**S3**
S7	0	6.67	7
S2		0	5
S3			0

The closest sequence pair to merge next is that of S2 and S3. The merged group is labeled S8. Since each of these contains a single sequence, their distance to the S7 may simply be averaged and results in the distance matrix shown below:

	S7	**S8**
S7	0	6.33
S8		0

The final step is simply a merge of groups S7 and S8 to yield the final group S9 that comprises of the entire sequence set. The essential purpose of this process is the construction of the guide tree. Guide tree provides the order in which the sequences will be merged into the progressively growing alignment.



End of Example

7.4.2 Constructing MSA with the Guide Tree

Sequence alignments are performed as dictated by the guide tree. Upon the alignment of the original seed pairwise alignment, any stage of following the guide tree will result in requiring one of the two possible alignments to be performed. Either a sequence might need to be aligned to a group of sequences, or a group of sequences might need to be aligned to another group of sequences.

When a single sequence is required to be aligned to a group of sequences, dynamic programming algorithm is applies to compute the score (or distance) of the new sequence and all the sequences in the group. The highest scoring alignment is used to determine how the new sequence is subsequently aligned to the group. And when a group of sequence is to be aligned with another group, the highest pairwise score between each member of the two groups is used to establish how the two groups align with each other.

As progressive alignments are formed, the gaps introduced in a pairwise alignment are replaced with a special character, such as an `X`. This allows the gaps to progress till the end when all `X`s in the alignment constructed are replaced with the gap character `-`. Underlying principle in progressive alignment may therefore be stated as "once a gap, always a gap." The dynamic programming alignment algorithms must also be adjusted to accommodate the special symbol `X` such that there is no cost associated with aligning an `X` with anything including other `X` characters.

Example 7.3

Continuing with the previous example, where the guide tree was constructed using the pairwise distances between sequences, the process of progressively building the multiple sequence alignment is illustrated in this example.

As the first sequence pair to be merged, S1 and S5, their alignment between them is changed such that the indel characters are replaced with `X`s.

```
S1: -ATTTA-CGCCT          S1: XATTTAXCGCCT
     ||||| |||||
S5: AATTTACCGCCT          S5: AATTTACCGCCT
```

According to the guide tree, the next sequence to be aligned is S4. The procedure for aligning this sequence to this group will attempt to align this sequence with each of the sequences in the group. Effectively, we therefore attempt an alignment of S4 with S1 and S5 and the alignment that scores better determines how the sequence is aligned to the group. The each of these alignments and their distances with the unit cost model are shown below:

```
S1: XATTTAXCGCCT          S5: AATTTACCGCCT
     |||| :||                   |||| |||
S4: -ATTTTCCG-GA          S4: -ATTTTCCG-GA
Distance = 4              Distance = 5
```

Recall that in the computation of the distance values, the alignment of the character X with any other character does not add to the overall alignment distance measure. Based on the alignments above, the sequence S4 would therefore be aligned to the S1 for the purposes of including it into the group. Since this alignment introduces gaps in S1, these gaps are correspondingly added to each of the other sequences in the group to keep the MSA consistent. The three sequence MSA at this stage looks as follows:

```
S1: XATTTAXCGCCT
S5: AATTTACCGCCT
S4: XATTTTCCGXGA
```

Similar process is used to create a an alignment of sequences S2 and S3 computed as follows with the neutral character X replacing the delete characters.

```
S2: TTAAGCCA-T              S2: TTAAGCCAXT
    ||||    |
S3: TTAATTAACC              S3: TTAATTAACC
```

The next step in the progressive alignment process involves the merging of the two sequence groups, {S1, S4, S5} and {S2, S3}. For an alignment of two groups, all sequence pairs in the two groups are tried and the best scoring alignment between the groups is used to anchor the alignment of the merged groups. In our example, one of the six possible pairwise alignments shown in Fig. 7.1 will be chosen to anchor the entire group's alignment.

Given that the sequence S1 from the first group is closest to the sequence S2 from the second group, their alignment is used to anchor the alignment of the merged group. Since these sequences are previously aligned to the other members of their respective groups, the new gaps inserted in S1 are also inserted in S4 and S5. Correspondingly, any new gaps inserted in S2 are also inserted into S3. These steps and the resulting multiple sequence alignment is shown in Fig. 7.2.

End of Example

7.5 Profiles

For construction of the multiple sequence alignment, it is sometimes advantages to align the new sequence (or groups) with an existing group after consideration of the overall distribution of the residues used in those groups. By considering the pairwise alignments of the closest members to seed the alignment, the accuracy of the alignment is somewhat compromised. The profile approach, while following the guide tree, builds the multiple sequence alignment using the alignments between two sequences, or between a sequence and a profile, or between a profile and a profile.

A sequence profile represents column-wise characteristics of a set of aligned sequences. In particular, it gives position-dependent weights for all 20 amino

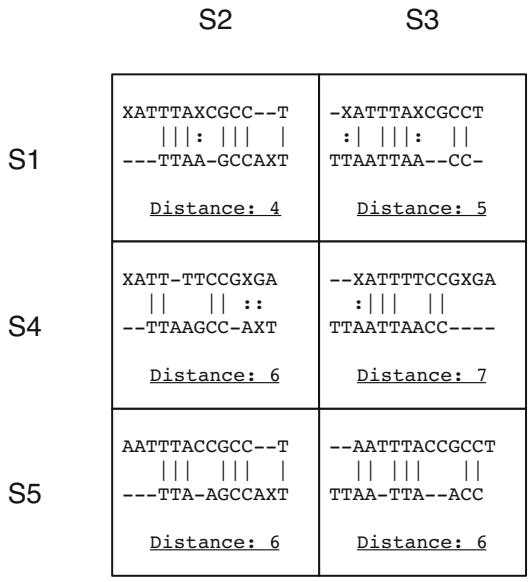


Fig. 7.1 Pairwise alignments of each group members. The alignment of S1 and S2 is used to anchor the alignment of the two groups.

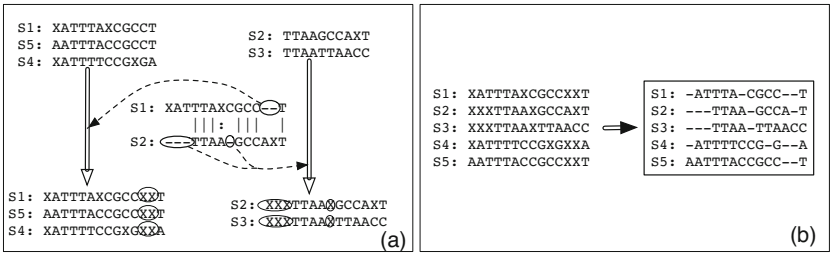


Fig. 7.2 Based on the gaps inserted in the two anchor sequences: (a) Remaining members of the group are updated to align with the anchor. Upon completion of the merge, (b) final alignment is generated by replacing the neutral character `X` with a `-`.

acids (or the four bases) and as for insertion and deletion events at any sequence position. Multiple alignments of related sequences properly translated into position-specific score matrices (PSSMs), profiles or hidden Markov models (HMMs) contain conservation patterns and statistical information about the alignment groups. The following is an example of profile computed from an alignment.

While the pair-wise scoring is quite successful in aligning two related families, profiles are successful in detecting weak similarities between conserved protein regions.

7.5.1 Constructing MSAs with Aligned Blocks

In the progressive alignment algorithm described above, we performed the alignment of two groups of sequences by considering the pairwise alignments between the members of the groups and using the closest pair as an anchor for aligning the two groups. However, rather than using a pairwise anchor, by the process described in this section, an alignment of the *entire group of sequences* in one aligned group may be performed *against the entire group* of aligned group.

The scoring model utilzed in this method associates a linear gap cost set at $s(-,a) = s(a,-) = -g$, and $s(-,-) = 0$. Scores between other non-gap residues are obtained from the appropriate scoring matrix such as a unit cost matrix for DNA and PAM or BLOSUM matrices for proteins. If the first group of sequences are labeled from 1 to n, and the second group labeled $(n+1)$ through N, the score of an alignment between two groups is defined as follows:

The formulation of the sum of pair scores is thus defined:

$$\sum_i S(m_i) = \sum_i \sum_{k<l\leq N} s(m_i^k, m_i^l) \tag{7.2}$$

$$= \sum_i \sum_{k<l\leq n} s(m_i^k, m_i^l) + \sum_i \sum_{n<k<l\leq N} s(m_i^k, m_i^l) + \sum_i \sum_{k\leq n, n<l\leq N} s(m_i^k, m_i^l) \tag{7.3}$$

We can split the terms in those terms that comprise the scores arising out of individual groups and those that arising out of the comparison of the two groups. The first two terms in the equation correspond to the the scores of the two intra-group comparisons, while the third term is the inter-group comparison term. Since the insertion of a gap within either of the groups introduces a column of gaps, and s(-,-)=0, the terms that affect the overall similarity score is essentially the inter-group comparison term.

The process of a compring an alignment group against another alignment group us based essentially on setting up the dynamic programming array with the entire columns of sequences along the rows and columns of the grid and columns are scored against other columns to compute the alignment.

Let the two aligned groups be denoted as Group I comprising of sequences $x^1, x^2 \ldots x^N$ and $y^1, y^2 \ldots y^M$. The similarity score between columns i in Group I and column j in Group II is calculated recursively by the equation below. As shown below, similar to the dynamic programming array calculation, the value used is the maximum of the composite similarity between the two columns and the score of inserting a column of gaps in the alignment block of one of the groups.

$$S_{i,j} = Max \begin{cases} S_{i-1,j-1} + \sum_{k \leq N, l \leq M, k \leq l} s(x_i^k, y_j^l)) \\ S_{i,j-1} + \sum_{k \leq N, l \leq M, k \leq l} s(\_, y_j) \\ S_{i-l,j} + \sum_{k \leq N, l \leq M, k \leq l} s(x_i, \_) \end{cases}$$

Example 7.4

Given the following two aligned blocks, S1 and S2, compute their alignment between these blocks using the method described.

```
S1:          S2:
ATTA         ATTA-
CTTG         AT-AG
```

End of Example

Such a method of using a progressive refinement of aligned blocks, rather than anchoring the alignment of blocks based on pairwise comparison, is utilized by CLUSTAL tool for multiple sequence alignments.

7.5.2 Modeling MSA as Profiles

Sequence profiles provide a numerical representation of multiple sequence alignment data. Essentially, the frequency of each nucleotide (or amino acid) is captured for each column of the multiple sequence alignment. Thus, the profile representation for a multiple sequence alignment with L columns is a of $5 \times L$ matrix for DNA sequence alignments and a $21 \times L$ matrix for protein alignment alignments. The $5^{th.}$ and the $21^{st.}$ row corresponds to the frequency of a gap character.

Example 7.5

Consider the following MSA block:

```
-ATA-CCCT
ACTACGCCT
-ATTCG-GA
```

Its frequency profile is computed by determining the normalized frequency of each of the four nucleotides and the gap character. It workes out to be as follows:

0.33	0.67	0	0.67	0	0	0	0	0.33
0	0.33	0	0	0.67	0.33	0.67	0.67	0
0	0	0	0	0	0.67	0	0.33	0
0	0	1	0.33	0	0	0	0	0.67
0.67	0	0	0	0.33	0	0.33	0	0

The first four rows are respectively the normalized frequencies of nucleotides, A, C, G and T, while the fifth row provides the frequency of the gap character.

End of Example

Alignment between profiles may also be performed in a similar manner as the alignment between the groups of aligned sequences discussed in Eq. 7.4 above. However, in the case of profile, the number of rows in is dependent on the the size of the alpabet. The alphabet is denoted as Σ and comprises of the four nucleotides and the indel character, and the twenty amino acid and the indel character in case of DNA and protein sequences respectively.

$$S_{i,j} = Max \begin{cases} S_{i-1,j-1} + \sum_{k\leq|\Sigma|,l\leq|\Sigma|,k\leq l}(p_i^k \dot{p}_j^l)s(k,l) \\ S_{i,j-1} + \sum_{k\leq|\Sigma|,l\leq|\Sigma|,k\leq l}(p_-^k \dot{p}_j^l)s(\_,l) \\ S_{i-l,j} + \sum_{k\leq|\Sigma|,l\leq|\Sigma|,k\leq l}(p_i^k \dot{p}_-^l)s(k,\_) \end{cases} \tag{7.4}$$

7.6 Progressive Alignment in MATLAB

MATLAB provides the `nwalign` function for performing global alignment. MATLAB uses a default scoring matrix for nucleotide sequences. This is a 15 by 15 scoring matrix that takes into account the IUPAC ambiguity code. The following code segment is used to compute the pairwise similarity scores and generate a guide tree.

```
seq = {'ATTTACGCCT', 'TTAAGCCAT', 'TTAATTAACC',
                        'ATTTTCCGGA', 'AATTTACCGCCT'};
dist = seqpdist (seq, 'Alphabet', 'nt');
links = linkage(dist);
dendrogram (links);
```

The commands above utilize the MATLAB function *seqpdist* to compute the distance between the elements of the sequence array. A linkage analysis is next performed. Although several options are provided for performing linkage analysis which essentially is a process of clustering, default parameter are used. The result of this linkage analysis is visualized using the *dendrogram* command. The result is the guide tree shown in Fig. 7.3.

7.6.1 Profiles in MATLAB

The MATLAB bioinformatics toolbox also provides the function for computing the sequence profile.

```
msa = ['-ATA-CCCT'; 'ACTACGCCT'; '-ATTCG-GA'];
prf = seqprofile (msa, 'Alphabet', 'nt', 'Gaps', 'all')
```

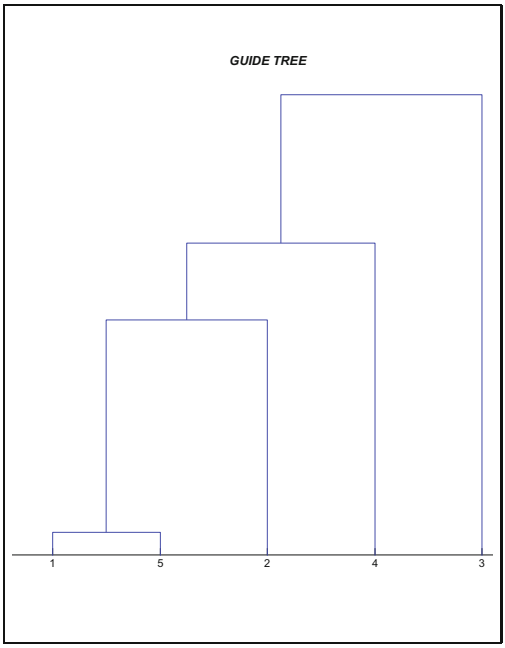


Fig. 7.3 Guide tree constructed by comparing sequences and performing the linkage analysis using default parameters in MATLAB

The value of profile variable *prf* in the above illustration is computed by MATLAB function *seqprofile*. The arguments to this function specify that the alignment is defined on the nucleotide alphabet and that all the gaps should be considered for computing the profile.

7.6.2 MSA in MATLAB

Multiple sequence alignment (MSA), as the name implies, is an alignment of two or more sequences. The Bioinformatics Toolbox includes progressive MSA via the function *multialign*. Simply pass *multialign* an array of sequences to have them aligned. You can further view this alignment in an interactive, color-coded display with the *multialignviewer* function. This function will open up a viewer in a window that will show you the consensus string for the alignment as illustrated in Fig. 7.4. Also, you can introduce or remove gaps by dragging letters.

Listing 7.1

```
>> seqs = {'AATTCCCGG' 'ATTCCGG' 'TTCCGG' 'AATTCTGG' 'AATTCC'};
>> ma = multialign(seqs)

ma =

AATTCCCGG
-ATTCC-GG
--TTCC-GG
AATTCT-GG
AATTCC---

>> multialignviewer(ma)
```

end-listing-7.1

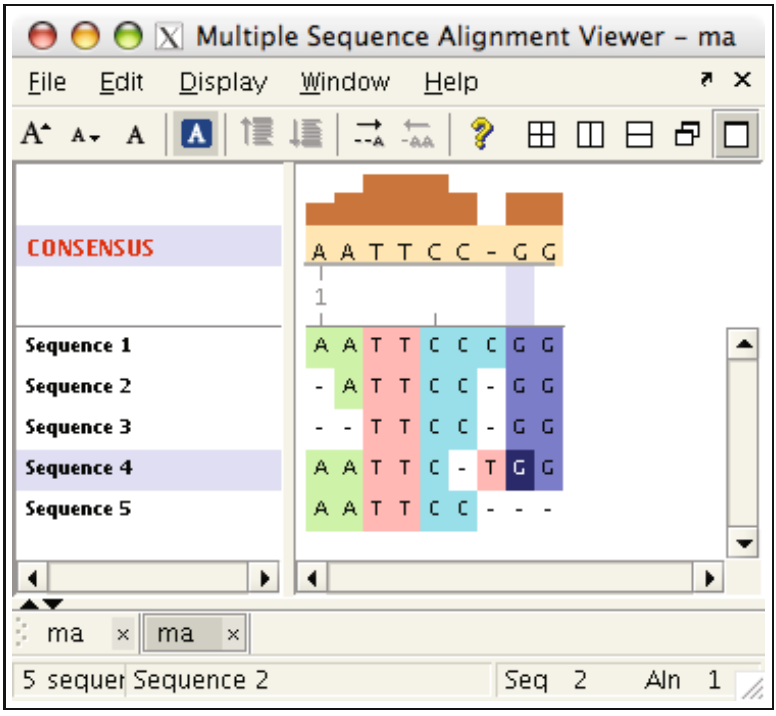


Fig. 7.4 Multiple alignment viewer

7.6.3 PILEUP

PILEUP Is another software for generating multiple sequence alignments by progressively refining pairwise alignments.

Further Readings

1. Bawono, P., Heringa, J.: Praline: a versatile multiple sequence alignment toolkit. Methods Mol. Biol. 1079, 245–262 (2014)
2. Pais, F. S., Ruy Pde, C., Oliveira, G., Coimbra, R.S.: Assessing the efficiency of multiple sequence alignment programs. Algorithms Mol. Biol. 9(1), 4 (2014)
3. Kaya, M., Sarhan, A., Alhajj, R.: Multiple sequence alignment with affine gap by using multi-objective genetic algorithm. Comput Methods Programs Biomed 114(1), 38–49 (2014)
4. Modzelewski, M., Dojer, N.: Msarc: Multiple sequence alignment by residue clustering. Algorithms Mol. Biol. 9(1), 12 (2014)
5. Li, Z., Natarajan, P., Ye, Y., Hrabe, T., Godzik, A.: Posa: a user-driven, interactive multiple protein structure alignment server. Nucleic Acids Res (May 2014)

7.7 Exercises

1. Consider the following sequence set:

```
ATG
GTG
ATG
TTG
ATG
ATG
GTG
ATG
```

2. These sequences respresent the translation start sites obtained as a result of multiple sequence alignments of a family of genes. Show the profile matrix resulting from these sites. Also, develop the position scoring weight matrix.
3. Assume a unit cost and linear gap model and the following sequences:

```
S1 = CCGGCTTCGCGACG
S2 = CGGTTGCCGAGC
S3 = AATGCGCTCCGGCCC
S4 = TTTCCAATCGGCC
```

   (a) Construct a guide tree for the above four sequences.
   (b) Apply the pairwise alignment technique utilizing 'once a gap always a gap' and using a `X` for the gap neutral character. Recall that the cost for aligning a gap character with anything, including another gap character, is 0. Compare your results with the `multialign` function in MATLAB.
   (c) Research and comment on the algorithm used in multialign. Is it similar to CLUSTALL or PILEUP. Support your answer.
   (d) Compute the SP score for a multiple sequence alignment.

4. Given the set of four sequences being aligned,

```
S1 = 'ATTGGCACCA'
S2 = 'GGTTCCA'
S3 = 'AAATTGGACC';
S4 = 'TATTCCACCA'
```

   The similarity scores have been computed, as follows:

	S1	S2	S3	S4
S1		-2	2	6
S2			-7	-2
S3				1
S4				

(a) Compute the guide tree. Show the values of the simailarity matrix after each merge operation.
(b) Assume that the similarity scores between the sequences continue to be the same as the process of progressive alignment proceeds, can you come up with what the multiple alignment would look like?
(c) Compute the SP score for your answer in Exercise 4b.

5. Given the set of four sequences being aligned,

```
S1 = 'ATTGGCACCA';
S2 = 'ATTTGGACCA';
S3 = 'TGGTTCCA';
S4 = 'ATTCCACCAC';
```

The distance scores have been computed under the **unit** model, as follows:

	S1	S2	S3	S4
S1		2	4	3
S2			5	4
S3				6
S4				

(a) Compute the guide tree. Show the values of the distance matrix after each merge operation.
(b) Assume that the distance scores between the sequences continue to be the same as the process of progressive alignment proceeds, determine what the multiple alignment would look like.
(c) Compute the SP score from your answer in Exercise 5b and compare it to your answer in Exercise 4b.

Chapter 8
Alignment Tools

Sequence alignment serves as the basis for comparing two sequences. The score obtained from the comparison of two sequences, the query and the candidate sequence in the database, is in turn used to retrieve the candidate sequences that are related to the query as evidenced by the value of their similarity score. While performing real time searches the sheer size the database often precludes the applicability of obvious and direct approaches and necessitates the development of algorithms that can yield *approximate* scores in reasonable amount of turnaround time. Consequently, the practical search algorithms are not guaranteed to produce the best or the optimal matches, but rather, offer a high probability that the matches returned will be true matches of the query and the retrieved database sequence. Often a *p-value* is associated with the retrieved sequence that is indicative of the probability that the match detected by the retrieval procedure could have occurred simply by chance. Alternatively, an *Expect* or an *E-value* may be returned by such a procedure indicative of the probability with which the reported score can be expected simply by chance.

This chapter provides an overview of the dot-plot algorithm which forms the basis for BLAST, the database retrieval tool that is probably utilized most commonly for searching genomic and protein sequence databases. The overview of dot-plot is followed by a somewhat detailed discussion of the BLAST algorithm and score calculation methodology. A overview of FASTA algorithm that employs a procedure similar to BLAST is described next.

8.1 Dot Plots

Dot plots are perhaps one of the simplest methods for estimating similarity between two sequences. To compute the dot-plot alignment, the two sequences are placed along the orthogonal axes of a grid. A certain number of residues or nucleotides (typically a single character) are compared and a dot is placed on the grid if the characters in the two sequences are in agreement.

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_8

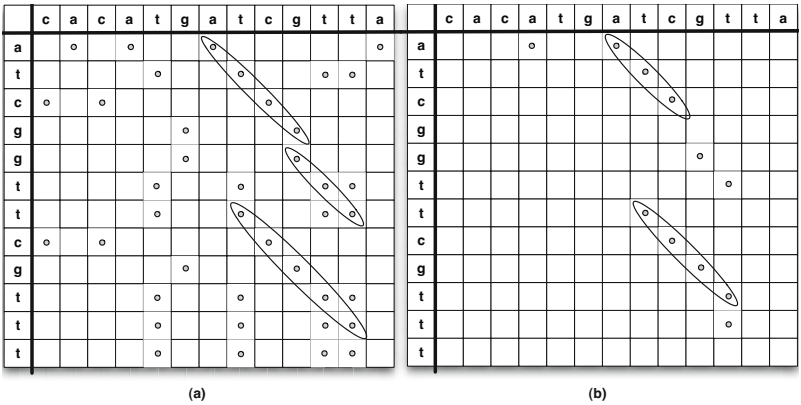


Fig. 8.1 A dot plot alignment for two DNA segments with the word sizes of (a) one and (b) two. A dot is placed on the grid where there is a single or two nucleotide match(es) between the two sequences. A putative local alignment is said to occur when the score of of dots along a diagonal exceeds a threshold. While the scores for DNA sequence dot plot matches are relatively easy to extend from single nucleotide to two nucleotides, the scores for dot plot matches derived from a BLOSUM or PAM matrix in protein dot-plot alignments will not afford such an extension.

Consider for example Fig. 8.1 depicting the dot plot alignments for two segments of DNA. In Fig. 8.1 (a) a single nucleotide match in the two sequences results in placing a dot on the grid. The local alignments are extracted by considering long matches along the diagonals. The matches along the diagonals that do not obtain a high enough score are discarded. If the threshold match score between two nucleotides is assumed to be ≥ 3, the local alignments that result have been annotated on the dot-plot.

When the sequences being compared are very similar to each other, a match of a single nucleotide will make the dot plot grid extremely dense and the string of matches along the diagonals will not be easy to visually discriminate and extract. As dot plot based alignment is typically utilized as a visual tool, the method utilized for enhancing the selectivity of the dots is the use of word size. As illustrated in Fig. 8.1 (b) a word size of two, where a dot is placed on the cells in the grid where a dinucleotide match exists between the two sequences results in the elimination of lot of spurious short alignments.

As discussed in the next section, BLAST uses the dot-plot concept in the seeding phase of the local alignment generation. BLAST further scores a seed by utilizing the PAM or the BLOSUM matrix for protein sequence alignments. In this manner a local alignment along the diagonal in a protein sequence grid may be picked for extension even when the match is not exact asa long as the alignment score is higher than a preset threshold.

8.2 BLAST

BLAST or the Basic Local Alignment and Search Tool is probably the most commonly utilized program for searching Biological Databases. As shown in Table 8.1 is really a collection of five programs. These are the BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX. These programs are characterized by the type of query sequence (DNA or Protein) utilized and the type of database that is searched.

Table 8.1 BLAST programs

Program	Query	Database	Search Description
BLASTN	DNA	Genomic	Straight search for matching nucleotide strings in a genomic database.
BLASTP	Protein	Protein	Search using a scoring matrix (such as PAM, BLOSUM) for a protein query string in a protein database.
BLASTX	Protein	Genomic	Search a protein query sequence against a translation of genomic database using a scoring matrix (e.g. PAM, BLOSUM).
TBLASTN	DNA	Genomic	Search a translated DNA query against a translated genomic database using a scoring matrix (such as PAM, BLOSUM).
TBLASTX	DNA	Protein	Search a translated DNA query sequence against a protein database using a scoring matrix (such as PAM, BLOSUM).

BLAST uses heuristics to enhance the speed of Smith-Waterman local sequence alignment. It proceeds in three basic steps, namely, seeding, extension and evaluation. In the seeding step, a dot plot is used to find areas in the two sequences where an alignment may be found. These *seed* word hits are extended through the process of a random walk where the core are extended such that the local match score is allowed to drop by a preset maximum to allow the algorithm to escape out of a local maxima in the hope of finding a higher maximum score around the seed. Finally, the third step entails an evaluation of the alignments thus achieved by seeding and extension. The process of alignment evaluation is statistical in nature and those alignments which have a very small probability of occurring by chance are reported as HSPs or High Scoring Pairs and sorted by the bit value corresponding to their significance.

8.2.1 Seeding

The basic assumption of BLAST algorithm is that sequences that have a significant alignment share a common word. A *word* is simply defined as a

sequence of characters. For example, the DNA sequence `ACCTATGA` has the following four-nucleotide words occurring in it: `ACCT`, `CCTA`, `CTAT`, `TATG` and `ATGA`. As a first step towards aligning sequences, BLAST determines the locations of all the common words. These are called *word hits* and serve as the seeds of alignment. BLAST allows the word hits to be approximate as well in order to increase the sensitivity of the searches. This is important, particularly in the case of amino acids where the searches often utilize a scoring matrix, such as BLOSUM-62 or PAM-250, with the identical word hits in protein sequences restricting the significant alignments to a very small subset.

8.2.2 Extension

After a seeding of the alignment is achieved through word hits, an extension of the word hit clusters is next performed by the BLAST algorithm. This process proceeds as a random walk shown in Fig. 8.2.

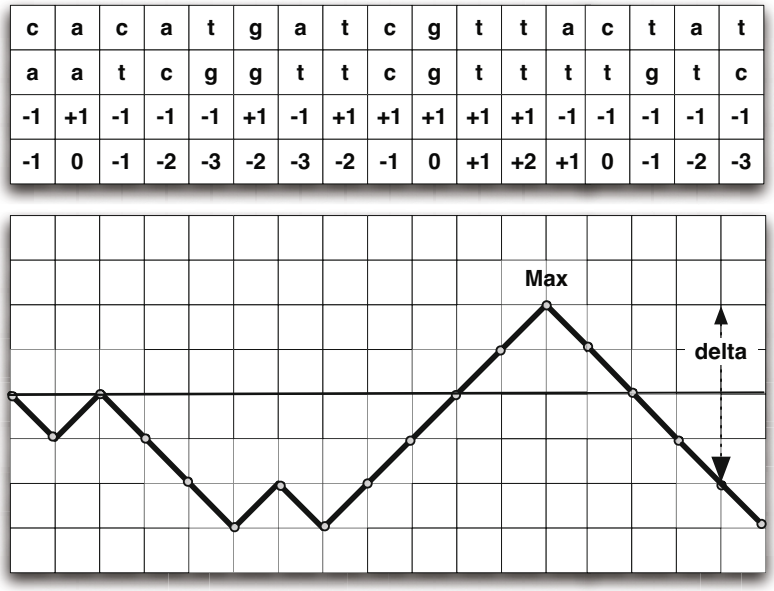


Fig. 8.2 The process of extending word hits. The maximum score is allowed to drop at most by a preset amount *delta* before the extension process is terminated. The alignment is trimmed back to the location with the maximum score.

The concept of gap extension through a random walk using DNA sequences is explained by the example in Fig. 8.2. The maximum value that the score is allowed to drop by, *delta* has been set to 4. The seed extension proceeds

in both the directions using a process similar to the extension shown in the illustration. The random walk is characterized by peaks and valleys. The nucleotide match in the two sequences results in the incrementing the score, while a mismatch reduces the accumulated score. A plot of the accumulated score is maintained. As the score reaches a maximum, it is allowed drop at most by the preset value *delta* before the extension process ceases. Upon ceasing the extension process, the alignment extension is trimmed to the position where the accumulated score peak was reached. The peak value in the illustration was reached at the value of +2.

If the value of *delta* was set to 3 instead of 4, the extension process would have ceased at position 2 with a peak score of 0 because a valley immediately following this peak would have resulted shown in the threshold drop necessary to terminate the gap extension process. Thus, a low value of delta will generally yield shorter alignments and may in fact cause the algorithm to run slower requiring the system to manage a large number of short alignments. If an excessive large value of delta is used there may be wasted computation. The alignment would be extended appropriately nonetheless as the extension will be clipped to retain alignments of the sequences corresponding to the peak accumulated score reached starting from the beginning of the walk before reaching the valley that caused the termination of the gap extension process. A small value of *delta* could completely miss a neighboring peak and consequently a larger value is chosen in practice.

8.2.3 Evaluation

The extension of the seeds into both the directions results in the generation of alignments. The evaluation step focuses on determining if these alignments are statistically significant. Karlin-Altschul statistics are applied to determine if the observed alignment is statistically significant and, if so, it is labeled as a High-Scoring Segment Pair or *HSP*. The basis for Karlin-Altschul statistic is Eq. 8.1 which states that the expected number E of alignments is inversely exponentially related to the normalized score of λS and is directly proportional to the search space defined as product of the length of the query sequence (m) and the database (n). The parameter k is a constant in the equation.

$$E = kmne^{-\lambda S} \tag{8.1}$$

This statistic has been used to provide a way of calculating how long a sequence must be before it can produce an alignment with sufficient expect. The minimum length, $\mathcal{L}$, is usually referred to as the *expected HSP length* and is defined by Eq. 8.2.

$$\mathcal{L} = \ln \frac{kmn}{H} \tag{8.2}$$

The quantity H, the relative entropy of the scoring scheme is defined in Eq. 8.3 where the summation is over the n residues (example, 20 amino acids, or 4 nucleotide bases) for protein sequences and the quantity S_{ij} is the log-likelihood score assigned to the alignment of residue i with j as discussed in Section 6.1.5.

$$H = \sum_{i=1}^{n} \sum_{j=i}^{n} q_{ij} \lambda S_{ij} \tag{8.3}$$
$$where\ S_{ij} = \log \frac{q_{ij}}{p_i p_j}$$

Thus, the expected HSP length is a function of the query length and varies from search to search. This length $\mathcal{L}$ is also taken into consideration for taking into account the edge effects and computing the *effective* lengths of query and database sequences used for determining the search space size. Specifically, the length of each sequence in the database as well as the query sequence is reduced by $\mathcal{L}$. The corrected values of query sequence length m_{eff} and database sequences n_{eff} as given below are used in the computation in Eq. 8.1 where the search space size used is $m_{eff} \times n_{eff}$.

$$m_{eff} = max\ [(m - \mathcal{L}),\ \frac{1}{k}]$$
$$n_{eff} = n\ -\ (|N|\ \times\ \mathcal{L})$$
$$where\ |N| \text{ is the number of sequences in database}$$
$$\text{and } n \text{ is the size (length) of the database}$$

Let us begin by rewriting Eq. 8.1 as follows with the corrected values of the sequence and the database being used in the equation:

$$\begin{aligned} E &= k \cdot m_{eff} \cdot n_{eff} \cdot e^{-\lambda S} \\ &= m_{eff} \cdot n_{eff} \cdot e^{\ln k} \cdot e^{-\lambda S} \\ &= m_{eff} \cdot n_{eff} \cdot e^{-(\lambda S - \ln k)} \end{aligned} \tag{8.4}$$

The raw score, S, obtained using the appropriate scoring matrix used for alignment is converted to a *nat* score (S_{nat}) or a *bit* score (S_{bit}) using the formulation in Eq. 8.5.

$$\begin{aligned} S_{nat} &= (\lambda S - \ln k) \\ S_{bit} &= \frac{(\lambda S - \ln k)}{\ln 2} \end{aligned} \tag{8.5}$$

In this manner a normalized bit score for the alignment is created and the expected value is correspondingly computed using Eq. 8.6.

$$E = m_{eff} \cdot n_{eff} \cdot 2^{-S_{bit}} \tag{8.6}$$

8.2.4 BLAST Reports - Example

An example of the BLAST report is shown in Fig. 8.3. This example shows the result of conducting a search on DNA sequence databases. The query used was a segment of the DNA sequence for the human heat-shock protein. As illustrated, the classical BLAST report consists of a set of sequences in the database that have a high similarity to the query sequence. For each of the sequences, the BLAST report also contains a local alignment of the query and the database sequence.

The final section of the BLAST report contains information on the statistical parameters used for computing the alignment scores, i.e. the parameters used for evaluating the *significance* of the alignment. The application of the theory in Section 8.2.3 is presented below:

The values for parameters $\lambda,\ k$ and $H = 1.37,\ 0.711$ and 1.31 respectively.

The actual length of Query and Database m and $n = 811$ and 5957977502 respectively.

The number of sequences in Database $= 46285$

The expected HSP Length $\mathcal{L} = \dfrac{\ln kmn}{H} = \dfrac{\ln(0.711 \times 811 \times 5957977502)}{1.31} = 22.03$

The effective length of Query $= m_{eff} = 811 - 22 = 789$

The effective length of Database $= n_{eff} = 5957977502 - 22 \times 46285$

5956959232

The effective size of search space $= m_{eff} \times n_{eff} = 789 \times 5956959232 = 4700040834048$

The raw score of Alignment $= S = 291$

The bit score of Alignment $= S_{bit} = \dfrac{(\lambda S - \ln k)}{\ln 2}$

$= \dfrac{(1.37 \times 577 - \ln 0.711)}{\ln 2} \approx 577$

The Expect or E − value of Alignment $= E = m_{eff} \times n_{eff} \times 2^{-S_{bit}}$

$= 4700040834048 \times 2^{-577} \approx 7 \times e^{-162}$

The value of λ in the report has been rounded to 1.37. The exact value of λ is 1.3743. If this exact value of λ is utilized, the *Expect* or *E-Value* obtained will be exactly equal to $7 \times e^{-162}$.

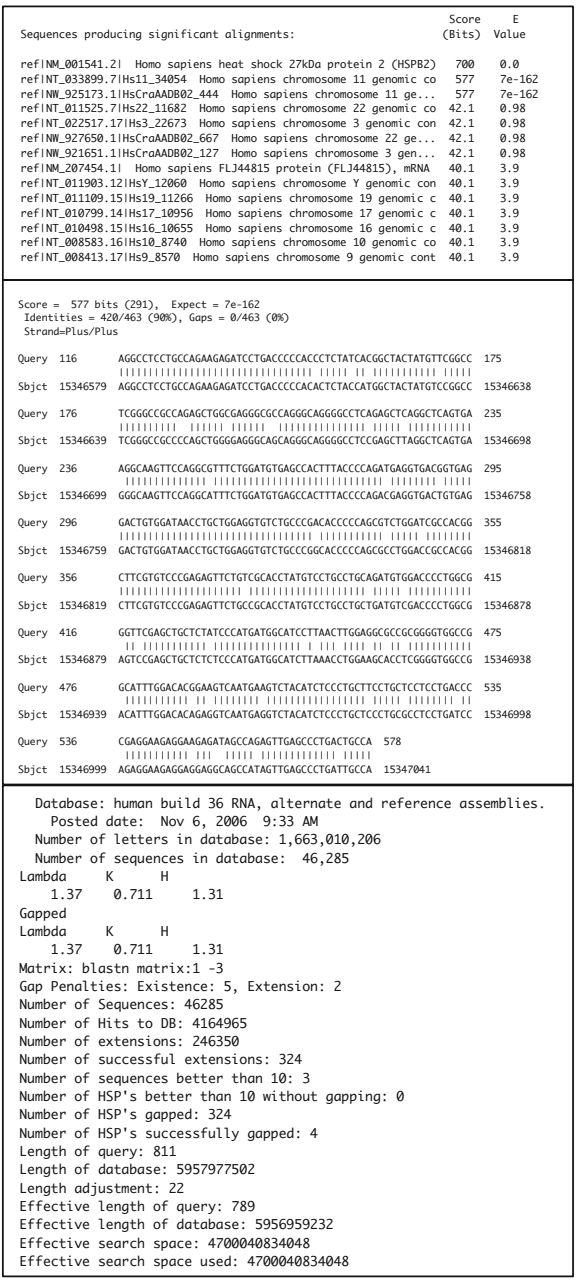


Fig. 8.3 The BLAST report provides a summary of the alignment score as well as the details of individual alignments. The footer of the report provides the database and query statistics used for computing the normalized score (in bits) and the *Expect* or E value which is indicative of the probability that match was found purely by chance.

8.2.5 P-Value for a Score

In addition to providing an E-value as an estimate of the significance of an alignment, some BLAST variants also provide a P-value. The relationship between these two values is given in Eq. 8.7.

$$\begin{aligned} P &= 1 - e^{-E} \\ E &= -\ln(1 - P) \end{aligned} \tag{8.7}$$

The significance of these two numbers is somewhat distinct although for small values of E, the two numbers are almost identical. While the E-value of a given score is an estimate of the number of chance alignments expected with that score, the P-value for a score provides an estimate of observing that score purely by chance.

8.3 FASTA

FASTA is another commonly utilized alignment program for computing local alignments. FASTA search begins by breaking the search sequence into words. Genomic sequences for example are broken into words of 4–6 nucleotides while peptide sequences are broken into words of 1–2 residues. A hash table is next constructed which notes down the location of each word in the database sequence. The word positions of the query sequences are offset to obtain the best possible correlation matches with the word locations in the database.

8.4 Further Readings

The BLAST program is described in Altschul [1].

The extensions of the BLAST program to include gaps is described in [2]. BLAST introduced the idea of using a small window of identity between the two sequences that can lead to growing an alignment. This was further utilized by Chao, Zhang, Ostell and Miller [3] for comparison of very large sequences. Applications of and modifications to the traditional BLAST algorithm are discussed in [4, 5, 6, 7].

Steps for using the FASTA tool for comparison of sequences is described in [8, 9, 10]. FASTA Algorithm was described by Lipman and Pearson in [11]. A comparison of FASTA and pure dynamic programming was done by Pearson in [12]. FASTA's applications for searching protein libraries was described in [13].

Further Readings

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. 215(3), 403–410 (1990), Lm04960/lm/nlm Lm05110/lm/nlm Journal Article Research Support, U.S. Gov't, P.H.S. England
2. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped blast and psi-blast: a new generation of protein database search programs. Nucleic Acids Res 25(17), 3389–3402 (1997), Lm05110/lm/nlm Journal Article Research Support, U.S. Gov't, P.H.S. Review England
3. Chao, K.M., Zhang, J., Ostell, J., Miller, W.: A local alignment tool for very long dna sequences. Comput. Appl. Biosci. 11(2), 147–153 (1995), R01 lm05110/lm/nlm Journal Article Research Support, U.S. Gov't, P.H.S. England Cabios
4. Cameron, M., Williams, H.E., Cannane, A.: Improved gapped alignment in blast. IEEE/ACM Trans. Comput. Biol. Bioinform. 1(3), 116–129 (2004)
5. Zhang, H.: Alignment of blast high-scoring segment pairs based on the longest increasing subsequence algorithm. Bioinformatics 19(11), 1391–1396 (2003)
6. Mount, D.W.: Using the basic local alignment search tool (blast). CSH Protoc (2007), pdb.top17 (2007)
7. Healy, M.D.: Using blast for performing sequence alignment. Curr. Protoc. Hum. Genet. Chapter 6:Unit 6.8 (January 2007)
8. Mount, D.W.: Recommended steps for a fasta search. CSH Protoc. (2007), pdb.ip40 (2007)
9. Mount, D.W.: Using a fasta sequence database similarity search. CSH Protoc (2007), pdb.top16 (2007)
10. Pearson, W.R.: Blast and fasta similarity searching for multiple sequence alignment. Methods Mol. Biol. 1079, 75–101 (2014)
11. Lipman, D.J., Pearson, W.R.: Rapid and sensitive protein similarity searches. Science 227(4693), 1435–1441 (1985), So7-rr05431/rr/ncrr Journal Article Research Support, U.S. Gov't, P.H.S. United states
12. Pearson, W.R.: Comparison of methods for searching protein sequence databases. Protein Sci 4(6), 1145–1160 (1995), Lm04969/lm/nlm Comparative Study Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, P.H.S. United states a publication of the Protein Society
13. Pearson, W.R.: Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the smith-waterman and fasta algorithms. Genomics 11(3), 635–650 (1991), Lm04969/lm/nlm Comparative Study Journal Article Research Support, U.S. Gov't, P.H.S. United states

8.5 Exercises

Multiple Choice Questions

1. You want to run a BLAST search with the following query sequence: `ACCTTTTAGGGTTAGAG`. Which of the following will programs represent a valid choice. Circle all that apply.

   o BLASTN
   o BLASTP
   o TBLASTN
   o BLASTX
   o TBLASTX

2. You want to run a BLAST search with the following query sequence: `NSLNKVIPSPPTHSLG`. Which of the following will programs represent a valid choice. Circle all that apply.

   o BLASTN
   o BLASTP
   o TBLASTN
   o BLASTX
   o TBLASTX

3. What is the appropriate interpretation of *bit score*:

   o It measures of similar bit sub-strings between the query and a hit.
   o It measures the number of bits needed to represent the probability that the similarity between the query and a hit is purely by chance.
   o It measures t the similarity value.
   o It measures the number of identical bits between a binary representation of the query and the hit.

Short Answer Questions

1. The Bit score provided for your sequence Q and a database sequence DBSeq1 is 30 and with another DBSeq2 is 10. Which of the two matches, DBSeq1 or DBSeq2, is more significant?
2. Assuming that the effective length of the query Q is 100 and the effective length of the database is 10^9, compute the corresponding E-values for the two hits in Problem 1. Also compute the P-values from the E-values. Compare the significance of an alignment with respect to the bit score, the E-value and the P-value.
3. You are searching a query in two different databases, D_1 and D_2. The effective lengths of the two databases is 10^6 and 10^9 respectively. The bit score for the highest scoring match from D_1 is 30, while the bit score for the highest scoring match from D_2 is 60. Which of these two matches is more significant. Explain.

4. Compute the dot plot between the sequences S=`CCTTGATAACCATTATA` and sequence T=`CATATCCATTAAGGTGG`. Assume word size of 2. Extend the seeds in either direction using a delta value of 3.
5. Given the value of k and λ as 0.7 and 1.4 respectively, convert the raw alignment score in the above problem to a bit score. What is the expectation that an alignment with this bit score will be observed by chance.
6. Assume that the effective lengths of the sequences being compared are equal to the their actual lengths. What is the expected value of this alignment? Recall that the expected value of an alignment is the probability that a match score will be observed purely by chance.
7. Develop a dot plot for comparing the sequences S1=`ATTGGAAGAGTAAG` and S2=`CCTCCGAAGAACGTAACCG`. Use a word size of 3.
8. Extend the dot plot in Ex. 7 with a delta value of 2. Recall that the delta value is the most a similarity score is allowed to drop before the attempt at extension is aborted.
9. Compare and contrast the types of alignments that are generated by increasing the match word size and reducing delta, and those that generated by a smaller word size and a larger delta.
10. Given that the values of k and H are 0.7 and 1.3 repectively, and a qeury of length 500nt is compared against a database of length 10 million nucleotides, what is the expected length of a HSP? What the expected HSP length be if the database was 10 billion nuclotides?
11. Using the geometric distribution formulation estimate the significance of the maximal local alignment block in the following alignment:

```
ACCCATAGGATTA
|:  ||||||
AT–CATAGG———
```

Chapter 9
Biolinguistic Methods

The number of biological sequences in the genomic databases, such as the GenBank, have exponentially increased during the past decade. Sequence retrieval systems are required to quickly and efficiently find sequences, that are related to a query sequence. Several comparison algorithms that generally rely upon the existence of local string similarities between the query and the database sequences have been widely utilized and accepted as the basis for biosequence retrieval from DNA sequences databases. This chapter describes a new method for sequence comparison based on *k-mer* word frequency profiles. In this algorithm, the distribution of the *k-mer* words found on the two sequences, captured by their frequency profiles, are treated as the signatures of sequences. This representation enables us to perform a comparison of sequence similarity using Shannon's entropy based divergence measures.

Due the relative importance of finding database matches, the significance of *sensitive* searches from databases has become critical. High speed and efficient retrieval algorithm have become the main goal of bioinformatics research. It plays an important role in virtually every branch of molecular biology and is crucial for explaining the functionality of sequences determined from the genome project. In the past decade, several sequence comparison or alignment techniques, such as dynamic programming-based alignment, BLAST and FASTA based searching tools described in earlier chapters have been developed and widely used in genomic databases. From a fundamental standpoint, these algorithms are based on applying the string edit distance to genome sequence retrieval. This approach, however, may sometimes cause biologically significant relationships to be lost amongst a deluge of strong sub-string matches, which must therefore be evaluated from statistical perspective.

Biolinguistic methods aim at overcoming some of these difficulties. In this approach the *k-mer* word frequency distribution is considered to carry information about the characterization of a sequence. The sequence word-based profiles are computed as the probabilities of the words occurring in the sequence. In this manner, the profile of a sequence is treated as its scale-space

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_9

signature. Sequence comparison performed using the scale-space profile and is based on matching the *global* properties of the sequences. Correspondingly, Shannon's entropy based divergence measures are applied to compute the distance between sequence profiles. As this method is not based on string edit distances, it has been shown to be more sensitive than the existing retrieval methods.

Statistical comparison of genomes between different bacterial strains also demonstrates biases observed in the usage of di-, tri- and tetra-nucleotides within the whole genomes. Viral genomes also have distinctive signatures of short oligonucleotide abundance that pervades the entire genome and distinguishes it from other genomes. Researchers have computed the dinucleotide biases as the odds ratio $\rho_{XY} = \frac{f_{XY}}{f_X f_Y}$ where f_X is the frequency of nucleotide X, and f_{XY} is the frequency of dinucleotide XY and successfully utilized it for estimating the distance $\delta(f, g)$ between two genomes f and g. This distance measure is defined as:

$$\delta(f, g) = \frac{1}{16} \sum_{ij} |\rho_{ij}(f) - \rho_{ij}(g)| \tag{9.1}$$

As we discuss later, the metric of relative dinucleotide abundance defined in Eq. 9.1 is similar to the variational distance between probability density function defined in section 9.2.2. Experimental results show that the closely related organisms have more similar profiles (or genome signatures) than do distantly related organisms.

9.1 Sequence Profiles

Biological sequences are broadly categorized into genomic and proteomic sequences. As discussed earlier, the genomic or DNA sequences are text strings that are comprised of 4 different nucleotides (A,C,G,T). Similarly, protein sequences are text strings defined on a 20 character alphabet of amino acids, the building blocks of all proteins. The sequence comparison algorithms, such as Smith-Waterman, FASTA, BLAST, or those based on Suffix Arrays all utilize the string comparison as the core computational unit for measuring the distance between sequences. The biological sequences, however, are really a mosaic of sequence level patterns that come together in a synergistic manner to coordinate and regulate the synthesis of proteins. The biolinguistic method described in this chapter utilizes pattern-directed properties of biological sequences for comparing sequences for effectuating retrieval of sequences from biological databases.

In the profile based searching technique, a *basis* pattern set forms the set of features using which the sequences may be compared. The biological sequences being compared are transformed into a *pattern-profile* which is representative of the relative frequency with which each pattern in the basis set occurs in the sequence. Although, the set of patterns to be utilized as

the feature basis may be provided based on biological criteria relevant to the sequences being compared, a set comprised of all possible words of a given size k, can also be considered as a basis set of features. The set of *all* possible k-mer words may be utilized as basis feature set. The frequency profiles obtained by measuring the relative abundance of each word, and representing it as a probability density function, is referred to as its *k-mer profile*.

Thus, the biological sequences to be compared are transformed into *k-mer* profiles. The *k-mer* profile is treated as the signature of the biological sequence. Subsequently, statistical and information-theoretic approaches are utilized to evaluate the proximity of the two sequence signatures. It may be noted that a sequence representation in the transformed k-mer space is no longer dependent on sequence size, but only dependent on the scale k used for enumerating the words whose frequencies are captured for representation of the sequence.

In this manner each sequence is represented by a set of *k-mer* words, with the probability of each word being dependent upon the *content* of the sequence. For a sequence with length N and *k-mer* word, the number of words contained in that sequence is $(N - k + 1)$. Fig. 9.1 illustrates the granularity of information content. As the word size increases, the information starts to become more specific to the sequence, and the random chance of similarities in profiles of two sequences drops exponentially (as the number of words increases exponentially at the rate of 4^k). The $x - axis$ in a profile represent the category assigned to each word. For example, a *3-mer* is comprised of $4^3 = 64$ words or categories, etc.

The *k-mer* profiles for all the sequences in the database are pre-computed and stored in the database. Only the profile of the query sequence needs to be computed at search time. The search for the database sequences that are related to the query sequence then proceeds by finding the *distance* of each database sequence profile from the profile of the query sequence. Some computational techniques for measuring the distance between two sequences are described below.

9.2 Comparing k-mer Profiles

The comparison of k-mer profiles may be accomplished by statistical as well as vector space approaches.

9.2.1 Vector Space Comparison

Vector space methods essentially rely on computation of the similarity using cosine similarity. The two profiles are treated as vectors on a *k-mer* space. Their similarity is computed as the cosine of the angle between them as shown in Eq. 9.2.

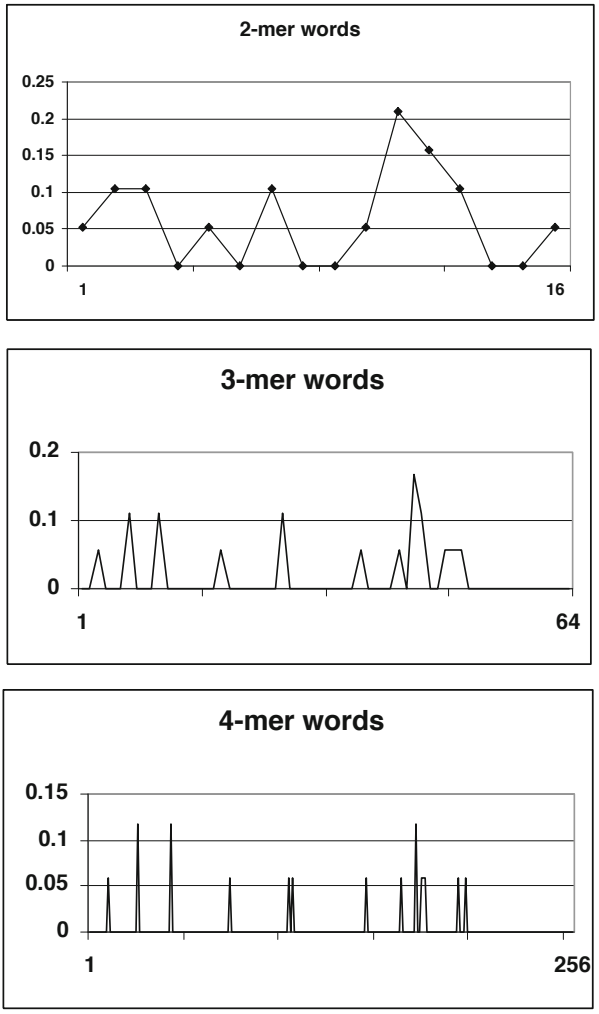


Fig. 9.1 Consider the DNA sequence `ACGGCAGGTACGGTAAGGTT`. Its 2-mer, 3-mer and 4-mer word profiles are shown above. The profiles are $p.d.f$s, where $p(x_i)$, $1 \leq x_i \leq 4^k$ represents the probability of finding the word x_i in the sequence. x_i is a category index obtained by hashing the *k-mer* work.

$$cos(p_i, p_j) = \frac{\sum_{x \in W} p_i(x) p_j(x)}{\sqrt{\sum_{x \in W} p_i^2} \sqrt{\sum_{x \in W} p_j^2}} \tag{9.2}$$

The vector space model is utilized extensively in information retrieval applications where the documents are represented on a hyperspace of terms. The dot product between term frequency vectors are normalized using the magnitude of term vectors being compared. In this manner the *relative* frequencies of the individual terms are taken into account for the purpose of comparison. If the two vectors, regardless of their magnitude, are oriented along the same direction, the cosine of the included angle will be a unity. This is precisely what cosine similarity measures.

In a similar manner if a dictionary of all possible *n-mer* words (example, the set of all codons, or triplets are 3-mers under this definition) is created and biological sequences represented thereon, the relative word frequency usage can similarly serve as a basis for their comparison. Generally, a comparison based on larger word size will tend to be more selective, or tend to reduce the false positive rate. While a comparison on a smaller word size will be tend to be more sensitive, and similarly reduce the false negative rate.

The example below illustrates the use of profiles as a basis for comparing sequences. It is probably instructive to mention that in practice a *1-mer* profile will never be used to compare sequences. Moreover, for comparing protein sequences, the set of all possible words will be become intractable. So, methods for reduction in alphabets, such as a categorization and groupings of amino acids using charge and hydrophobicity often need to be utilized.

Example 9.1

As a simple example, consider the cosine similarity computations using a one-mer document frequency vectors computed over the following three sequences:

```
S1 = ACCTGGTATCCATTGCCA
S2 = CCTTAATTGGGTT
S3 = TTCCGGTAGCGATACAATTAAC
```

There are a total of four one-mers for DNA sequences. Consequently, the one-mer profiles for these sequences may be computed by considering the frequencies of the four nucleotides and using Laplace's rule to convert those into probabilities. Recall that Laplace rule eliminates the occurrence of zero probabilities by adding a one to all the frequencies observed.

Word	f_1	f_2	f_3	p_1	p_2	p_3
A	4	2	7	$\frac{5}{22}$	$\frac{3}{17}$	$\frac{8}{26}$
C	6	2	5	$\frac{7}{22}$	$\frac{3}{17}$	$\frac{6}{26}$
G	3	3	4	$\frac{4}{22}$	$\frac{4}{17}$	$\frac{5}{26}$
T	5	6	6	$\frac{6}{22}$	$\frac{7}{17}$	$\frac{7}{26}$
	18	13	22	1.0	1.0	1.0

The dot product computed must be normalized with the magnitude of the probability density vectors. The magnitude of the three vectors are computed as follows:

$$|p_1| = \sqrt{\left(\frac{5}{22}\right)^2 + \left(\frac{7}{22}\right)^2 + \left(\frac{4}{22}\right)^2 + \left(\frac{6}{22}\right)^2} = 0.5102 \tag{9.3}$$

$$|p_2| = \sqrt{\left(\frac{3}{17}\right)^2 + \left(\frac{3}{17}\right)^2 + \left(\frac{4}{17}\right)^2 + \left(\frac{7}{17}\right)^2} = 0.5359$$

$$|p_3| = \sqrt{\left(\frac{8}{26}\right)^2 + \left(\frac{6}{26}\right)^2 + \left(\frac{5}{26}\right)^2 + \left(\frac{7}{26}\right)^2} = 0.5073$$

Pairwise cosine similarity between these 1-mer profiles can be computed as:

$$cos(p_1, p_2) = \frac{(\frac{5}{22})(\frac{3}{17}) + (\frac{7}{22})(\frac{3}{17}) + (\frac{4}{22})(\frac{4}{17}) + (\frac{6}{22})(\frac{7}{17})}{|p_1||p_2|} = 0.919 \tag{9.4}$$

$$cos(p_1, p_3) = \frac{(\frac{5}{22})(\frac{8}{26}) + (\frac{7}{22})(\frac{6}{26}) + (\frac{4}{22})(\frac{5}{26}) + (\frac{6}{22})(\frac{7}{26})}{|p_1||p_3|} = 0.973$$

$$cos(p_2, p_3) = \frac{(\frac{3}{17})(\frac{8}{26}) + (\frac{3}{17})(\frac{6}{26}) + (\frac{4}{17})(\frac{5}{26}) + (\frac{7}{17})(\frac{7}{26})}{|p_2||p_3|} = 0.924$$

Based on the cosine similarity values for the one-mer frequencies, sequences S1 and S3 are thus the closest neighbors.

End of Example

9.2.2 Divergence Measures

Profile based divergence measure is a distance metric that falls between the value of 0.0 and 1.0. Several techniques have been proposed for measuring divergence between two probability density functions. These include the

variational distance $V(p_1, p_2)$, and the divergence $I(p_1, p_2)$ and divergence $J(p_1, p_2)$, as defined below. Divergence I and J require that the probability density functions satisfy the condition of absolute continuity. The $L(p_1, p_2)$ divergence measure utilized in our research is derived from the *relative entropy* or the K-divergence, and is symmetric.

$$V(p_i, p_j) = \frac{1}{2} \cdot \sum_{x \in W} |p_i(x) - p_j(x)| \tag{9.5}$$

$$I(p_i, p_j) = \sum_{x \in W} p_1(x) ln \frac{p_i(x)}{p_j(x)} \tag{9.6}$$

$$J(p_i, p_j) = I(p_i, p_j) + I(p_j, p_i) \tag{9.7}$$

$$K(p_i, p_j) = \sum_{x \in W} p_i(x) ln \frac{p_i(x)}{\frac{p_i(x)+p_j(x)}{2}} \tag{9.8}$$

$$L(p_i, p_j) = \frac{K(p_i, p_j) + K(p_j, p_i)}{2} \tag{9.9}$$

where, W represents the different words in a profile.

The variation distance V serves as an upper bound on both K- and L-divergence measures. The divergence measure may also be used for measuring the similarity between sequences A and B. If this is desirable, a similarity measure may be defined as follows:

$$S(A, B) = \ln \frac{1.0}{L(A, B)} \tag{9.10}$$

Example 9.2

Divergence measures may be computed for the data presented in section 9.2.1 for computing cosine similarities.

For example, the variational distance between sequence S1 and S2, represented by 1-mer profiles p_1 and p_2 respectively, can be computed using their profiles as follows:

$$V(p_1, p_2) = \frac{1}{2}\left[\left|\frac{5}{22} - \frac{3}{17}\right| + \left|\frac{7}{22} - \frac{3}{17}\right| + \left|\frac{4}{22} - \frac{4}{17}\right| + \left|\frac{6}{22} - \frac{7}{17}\right|\right] = 0.193 \tag{9.11}$$

Other divergence measures are similarly computed. The following table summarizes the results:

Sequence Pair	J Divergence	L Divergence	Variational Distance
(S1, S2)	0.167	0.021	0.193
(S1, S3)	0.053	0.007	0.091
(S2, S3)	0.157	0.019	0.186

As is observed by the divergence measures as well, the consensus of all the measures is indicative that sequences S1 and S3 are the closest neighbors.

End of Example

9.3 Processing Profiles in MATLAB

MATLAB string searching capabilities are utilized in the developing the following program for generating a dictionary of words for processing:

```
% Function generates a 3 mer dictionary of words and finds
% profile for the sequence passed.

function [prof freq mer3] = profile3mer (seq)
n = ['A', 'C','G','T'];
freq = zeros(64,1);
for i=1:4
    for j=1:4
        for k=1:4
            b1 = n(i); b2 = n(j); b3 = n(k);
            pat = [b1 b2 b3];
            ind = (k-1)*4^0 + (j-1)*4^1 + (i-1)*4^2 + 1;
            mer3(ind,:) = pat;
        end;
    end;
end;
%
% loop through the patterns
%

for i = 1:size(mer3,1)
    pat = mer3(i,:);
    freq(i) = length(findstr (seq, pat));
end;

% increase freq vector by 1 as per Laplace rule
freq = freq + 1;
sumf = sum(freq);
for i=1:size(mer3,1)
    prof(i) = (freq(i) / sumf);
end;
```

The following MATLAB functions illustrate the computation of cosine similarity and divergence measures.

Cosine Similarity computation are performed by the following function:

```
function sim = cosine_sim (pi, pj)
mag_pi = norm (pi);
mag_pj = norm (pj);
sim = dot (pi, pj) / (mag_pi * mag_pj);
```

The function for computing **K divergence** shown below:

```
function kdiv = k_divergence (pi, pj)
pavg = (pi + pj)/2;
pi_by_pavg = pi ./ pavg;
kdiv = sum (pi .* log (pi_by_pavg));
```

K-divergence is an asymmetric divergence measure. That is, $K(p_1, p_2) \neq K(p_2, p_1)$. The **L-divergence** measure, based on K-divergence is a symmetric measure and defined by the following MATLAB function.

```
function ldiv = l_divergence (pi, pj)
ldiv = (k_divergence (pi,pj) + k_divergence (pj,pi))/2;
```

Finally, the upper bound on the divergence measure the variational distance.

Variational Distance is defined in the following MATLAB function:

```
function v = variational (p1, p2)
v = 0.5 * sum (abs(p1-p2));
```

9.3.1 MATLAB Program

We can use functions defined above to compute the cosine similarity and L-divergence and variational distances based on a 3-mer profile using a following sample program.

```
S1 = 'ACCTGGTATCCATTGCCA';
S2 = 'CCTTAATTGGGTT';
S3 = 'TTCCGGTAGCGATACAATTAAC';

S = {S1, S2, S3};

% Compute frequencies

fprintf ('------------3-MER STATISTICS----------------\n');
fprintf ('Seq-Pair      Cos-Sim   Var-Dist L-Div \n');

% Compute cosine similarity
for i = 1:(length(S)-1)
    [prf1 f m ] = profile3mer (S{i});
    for j = i+1:length(S)
        [prf2 f m ] = profile3mer (S{j});
        cos_sim (i,j) = cosine_sim (prf1, prf2);
        var_dist (i,j) = variational (prf1, prf2);
        l_div (i,j) = l_divergence (prf1, prf2);
```

```
        fprintf('(S%-d and S%-d) = %5.3f \t %5.3f \t %5.3f \n', ...
                  i, j, cos_sim(i,j), var_dist (i,j), l_div (i,j));
    end;
end;
fprintf ('-----------------------------------------\n');
```

The above program generates the following output:

```
------------3-MER STATISTICS----------------
Seq-Pair       Cos-Sim    Var-Dist L-Div
(S1 and S2) = 0.913        0.129    0.019
(S1 and S3) = 0.875        0.181    0.030
(S2 and S3) = 0.912        0.157    0.021
-----------------------------------------
```

9.3.2 Mutual Information

Mutual information is a measure of the information in one set of sequences that may be deduced from another set of sequences. We may think of this as an increase in our ability to estimate the probability density of sequence y (sequence at the input of an imaginary communication channel), if we are given the density function for sequence x (the sequence at the output of the imaginary communication channel).

$$I(x,y) = H(x) + H(y) - H(x,y) \tag{9.12}$$

$$H(x) = -\sum_{i} p_x(i) log_2 p_x(i) \tag{9.13}$$

$$H(y) = -\sum_{j} p_y(j) log_2 p_y(j) \tag{9.14}$$

$$H(x,y) = -\sum_{i \in x} \sum_{j \in y} p_{x,y}(i,j) log_2 p_{x,y}(i,j) \tag{9.15}$$

or,

$$I(x,y) = -\sum_{i \in x} \sum_{j \in y} p_{x,y}(i,j) log_2 \frac{p_{x,y}(i,j)}{p_x(i) p_y(j)} \tag{9.16}$$

H(x), H(y) is the entropy of sequence x and y respectively. $p_{x,y}(i,j)$ is the joint probability for i ∈ x and j ∈ y, where i and j are words in these sequences. If sequences x and y are independent, $p_{x,y}(i,j) = p_x(i)p_y(j)$, and the mutual information $I(x,y) = 0$. When two identical sequences appear at two sides of the channel separately, the mutual information reaches a maximum value. Hence, we can use mutual information as the measure of similarity of two sequences. The normalized similarity for mutual information is:

$$S(X,Y) = \frac{2 \cdot I(X,Y)}{I(X,X) + I(Y,Y)} \tag{9.17}$$

The difficult part here is to determine the joint probability of words occurring in the sequences x and y. This requires the computation of the alignment between the two sequences.

9.4 Sequence Comparison

The similarity of word frequencies is defined using an information theoretical divergence metric. This similarity measure is an absolute metric and does not require its significance be evaluated relative to a score population. As discussed above, the dominant patterns observed among DNA sequences constitute a useful indicator of their similarity. As a first step towards the comparison of patterns, the frequency of *k-mer* words found in a sequence may serve as its signature, or profile of the patterns occurring within the sequences. We refer to this frequency profile of words as the scale-space representation of a sequence. This is appropriate since the size of the profile is dependent upon a scale, which in this case corresponds to the word size chosen as the basis of representation.

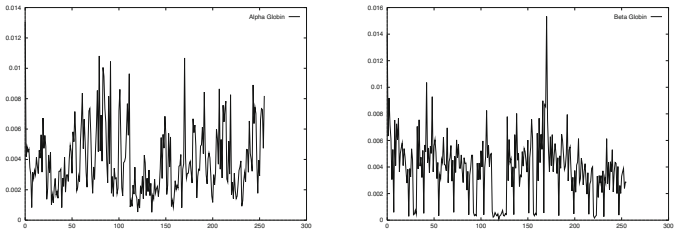


Fig. 9.2 The *4-mer* scale-space representations of α-globin and β-globin gene sequences. These plots represent the frequencies of each of the 256 possible words that may be formed using 4-nucleotide from a 4-character DNA alphabet. The *x-axis* corresponds to a word and the *y-axis* specifies the frequency of its occurrence in the sequence. The graphs for the two globin sequences depict that the 4-letter word that occurs most frequently in α-globin also occurs most frequently in β-globin.

The sequences being compared are represented by the profile of their *k-mer* word frequencies. The k-mer words in case of DNA sequences are formed by k-nucleotide strings. Example *4-mer* profiles for the α– and β–globin sequences are shown in Fig. 9.2. Since it is possible to construct 256 words using 4-characters (or nucleotides) of the DNA alphabet comprised of symbols {`A`, `C`, `T`, `G`}, the frequency values for each of these words constitutes the *4-mer* profile. One advantage of such a representation is its independence of the length of the sequence. For example, the sequence lengths are very different for the *4-mer* representations shown in Fig. 9.2. The α-globin gene cluster is

approximately 250,000 bases or ∼250 kb long, while the β-globin sequence is only ∼80 kb. In their scale-space representation their similarity is measured using the distribution of their word populations. This is significant since the scale-space representation compares the overall properties of the entire sequence extent. This is in contrast to the search for local similarity that the existing methods utilize for genomic databases searching.

One significant advantage of the scale-space representation is that the result of comparing two sequences is a metric that falls between `0` and `1`. By treating the scale-space representation of two sequences as probability density functions (*PDF*), the divergence measures for the comparison of *pdf*s is utilized for their comparison. Two such measures are the variation distance, $V(p_1, p_2)$ and the *L-divergence*, $L(p_1, p_2)$ can be used to compute the *LDSS* or L-Divergence based Similarity Scores (LDSS). This is derived from divergence values for the word size W by: $LDSS \equiv 1.0 - L(p_1^W, p_2^W)$. Such a measure is utilized in the example shown below.

9.4.1 Biolinguistic Retrieval from GBPRI Database

A sequence database comprising of the *GB-PRI* subsection of was created to test the performance of the profile based methods for genome sequence retrieval. This subset is comprised of ∼114,000 sequences. Sequences shorter than 1 kb were excluded from the profile based search as probability density estimations are not reliable for sequences shorter than 1 kb. The strength of the profile based algorithm is primarily aimed at enhancing the biological significance of comparing large sequences, where the chances of a large number of sub-string matches is high. Fig. 9.3 depicts the background frequency of each *3-mer* word in the database. This is the relative frequency of the trinucleotides in the `GB-PRI` database. As we describe later, the relative prevalence of the words in the database may be utilized for assigning a *significance* to each word. In general, the significance of the word is in inverse proportion to the frequency of that word occurrence in the database. This helps to correctly bias the profile similarity scores toward agreements on the *rarer* words, i.e. words for which a random match of two profiles is not likely.

9.4.2 Retrieval Results

The *3-mer* and *4-mer* sequence profiles are pre-computed and stored in database. The sequence retrieval procedure between a query and database sequences proceeds as follows:

i Compute the query sequence profile with *k-mer* words.
ii For each *k-mer* DB sequence profile, compute the distances between query profile and DB sequence profiles.
iii Sort the divergence values.
iv Retrieve the sequences with the smallest x% distance scores.

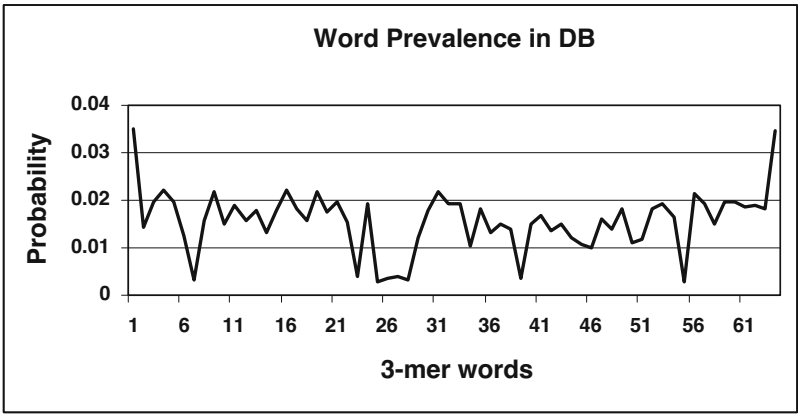


Fig. 9.3 Word Prevalence in DB for 3-mer word. The first word is `AAA` and last word is `TTT`. They both have high probabilities (about 0.035). Some words have low probabilities (about 0.003). In the weighted profiles based comparison, a match on the frequency of low probability words is assigned a higher level of significance.

Fig. 9.4 shows scores obtained for the top 1% matches with eight different randomly selected human DNA query sequences. The length of these sequence also varied from 100 bp to 100 kb. The score of top 1% matches based on *3-mer* and *4-mer* profiles are shown in this figure. It is clear from this figure that the divergence values produced using the *4-mer* profiles are roughly one half an order of magnitude higher than those produced with *3-mer* profiles. This needs to be taken into consideration for comparison of divergence scores obtained for different values of k.

9.4.3 Retrieval Evaluation

Experiments conducted with profile based retrieval system aimed to demonstrate that the average score of a query sequence from the sample of primate sequences (`GB-PRI`) is in close correspondence with the evolutionary distance of the organism to which the query belongs, and the primates.

Eight sequences were randomly selected each from humans, bacteria, arabidopsis, and yeast. The lengths of the randomly selected query sequences were approximately 100 bp, 500 bp, 1000 bp, 5 kb, 10 kb, 20 kb, 50 kb and 100 kb. The divergence scores for the sequences from the four species were compared, and generally the closest neighbor to `GB-PRI` were the human sequences for all sequence sizes. The results shown in Fig. 9.5 illustrate the average divergence of query sequence and its top 1% neighbors in the database. Similar results obtained by considering the top 5% neighbors of a query sequence are shown in Fig. 9.6.

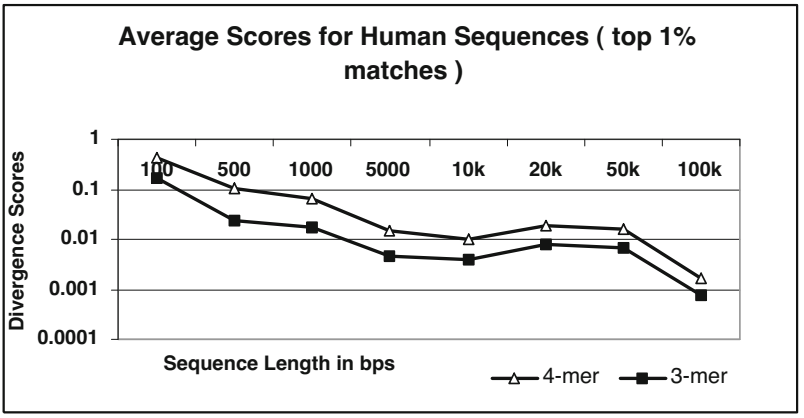


Fig. 9.4 The variation of divergence scores with profile word size. The larger the value of the word size, the higher the divergence scores. The growth in divergence scores is exponential with the increase in word size. In plot above, the average of the top 1% scores obtained by comparing `GB-PRI` and a set of eight randomly selected human sequences are shown for word sizes of 3 and 4.

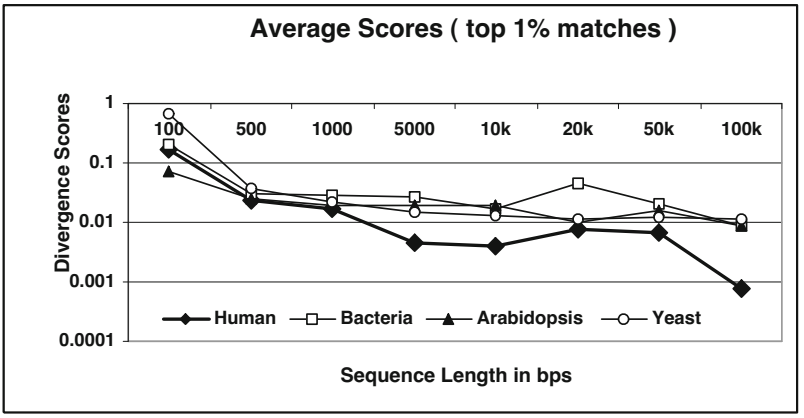


Fig. 9.5 With the increment of sequence length, the divergence scores are going to be decreased. The average scores for the top 1% matches are shown here. It may noted that even though the average match scores decrease with the sequence length, the trends (the relative score of human is the lowest, as humans are the closest neighbor to primates) continue to persist. Generally, however, the query sequences are seldom as large as 100 kb.

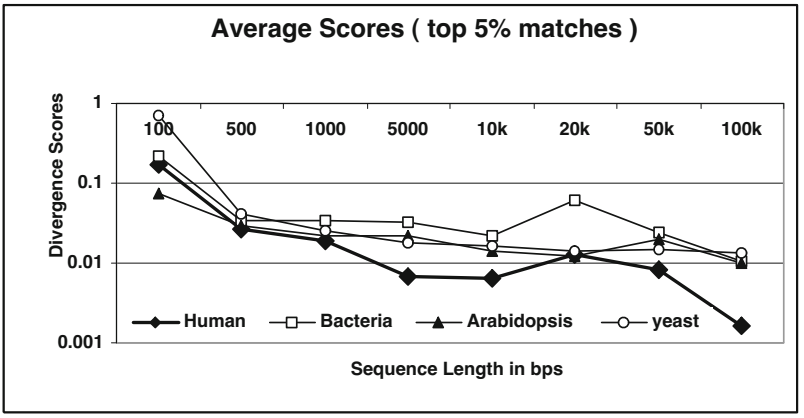


Fig. 9.6 The average scores for top 5% neighbors for randomly selected query sequences belonging to the four species studied

From the above plots, it is evident that the bacterial sequences are generally the farthest away from the primate section of `GenBank`. In Fig. 9.7 the ratio of the average score for bacteria's top 1% neighbors and human's top 1% neighbors are plotted. This clearly shows that the divergence scores for humans tend to an order of magnitude smaller than those of bacteria. In this manner, the profile based genome retrieval algorithm clearly demonstrates the power of this method to discriminate the sequence neighbors.

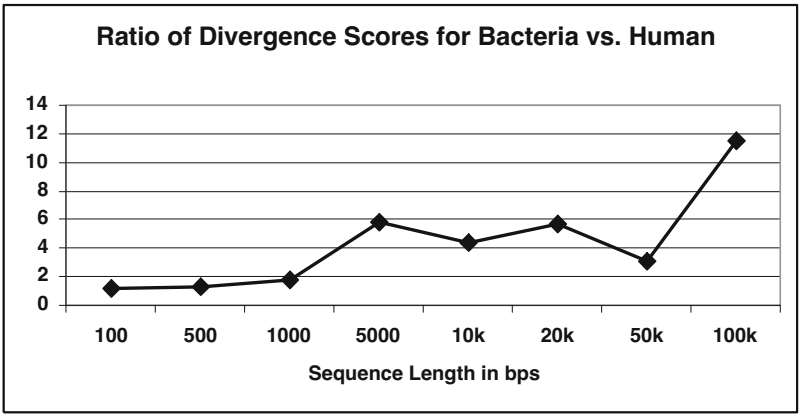


Fig. 9.7 For top 1% matches, the divergence scores between bacteria and human. This figure shows that the distance of a bacterial sequence from the database can be an order of magnitude more than distance of the human sequence from the database.

The experiments indicate that the sequences with closest overall distance scores belonged to humans, and the farthest overall distance scores belonged to bacteria. This demonstrates the biological validity of the profile based retrieval, as the humans are evolutionary neighbors of primates. The power of this method stems from the observation that as query sequence size was increased the separation between the two scores also increased. Thus, the profile based method is not prone to generating false positive matches as the chance of random local alignments will tend to increase for larger query sequences.

9.5 Weighted Profiles

The profile or the distribution of *k-mer* words in a sequence, tends to become noisy and cause the divergence measure to be influenced by the frequency values that are an artifact of the background (random) word distribution. Fig. 9.3 shows that some words have higher probabilities and are expected to vary in their frequency distributions thereby causing the divergence to be skewed toward random fluctuations. The weighted profile approach is designed to alleviate the role of frequently occurring words that are expected to be subjected to large variations from one sequence to the next in the database. The weighted profile is computed by considering the database word frequencies. The frequently occurring words in the database are assigned an attenuating weight to limit their variations. The detailed definition of the weights assigned to individual words is as follows:

Let $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_n\}$, be the database frequency profile, where λ_i is the probability of word $e_i \in \Omega$ occurring in the DB.

$$\lambda_i = \frac{\sum_{j=1}^{m}(\|S_j\| - k + 1) \times P_{S_j}(i)}{\sum_{j=1}^{m}(\|S_j\| - k + 1)} \tag{9.18}$$

where $S_j \in DB$, $P_{S_j}(i)$ is the probability of word $e_i \in \Omega$ occurring sequence S_j. The vector Λ can also be considered as the expected probability of words occurring in the entire genome database. The weights of words for the DB is defined as: $\mathcal{W} = \{w_i \mid i = 1, 2, ..., n\}$ with each w_i is the weight for word $e_i \in \Omega$ defined as:

$$w_i = -log(\frac{m}{n_i}) \tag{9.19}$$

where, m is the total number of sequences in the database, the n_i is the number of sequences for which the probability of finding the word e_i is greater than the expected value λ_i. The weighted word profile for sequence S_j, j = 1, 2, ..., m is defined as:
$\Phi = \{\phi_i \mid i = 1, 2, .., n\}$, with each

$$\phi_i = \frac{w_i \times P_{S_j}(i)}{\sum_{t=1}^{n} w_t \times P_{S_j}(t)} \tag{9.20}$$

and, $P_{S_j}(i)$ is the probability of word e_i occurring on sequence S_j, j=1, 2, ..., m and i=1, 2, ..., n. The L-divergence similarity score(LDSS) σ with weighted profile between sequence S_1 and S_2 using weighted profile is defined:

$$\sigma = \frac{\sum_{i=1}^{n} \phi_1(i) log \frac{\phi_1(i)}{\frac{\phi_1(i)+\phi_2(i)}{2}} + \sum_{i=1}^{n} \phi_2(i) log \frac{\phi_2(i)}{\frac{\phi_1(i)+\phi_2(i)}{2}}}{2} \tag{9.21}$$

9.6 Summary

Biolinguistic methods utilize information theory and the concept of entropy. Ideas of using entropy in genetic patterns and switches has also been utilized by researchers and used in the visualization of patterns using Logos..

Researchers have utilized a framework for using cross-entropy as a dissimilarity measure. Divergence methods for comparing two probability density functions are commonly studied. Some of these concepts are applicable to applications in searching biological databases as well as for comparing properties of proteins in establishing their phylogeny.

Further Readings

1. Tribus, M.: Thermostatics and Thermodynamics. D. Van Nostrand Company, Inc., Princeton (1961)
2. Schneider, T.D.: Information and entropy of patterns in genetic switches. In: Erickson, G.J., Smith, C.R. (eds.) Maximum-Entropy and Bayesian Methods in Science and Engineering, Dordrecht, The Netherlands, pp. 147–154. Kluwer Academic Publishers (1988)
3. Pierce, J.: An introduction to information theory: Symbols, Signals and Noise, 2nd edn. Dover Publications, Inc., New York (1980)
4. Rao, C., Nayak, T.: Cross entropy, dissimilarity measures, and characterization of quadratic entropy. IEEE Trans. Inform. Theory IT-31(5), 589–593 (1985)
5. Lin, J.: Divergence measures based on Shannon entropy. IEEE Trans. Inform. Theory 37(1), 145–151 (1991)
6. Singh, G.: Searching biological databases using biolinguistic methods. In: Arora, D., Berka, R., Singh, G. (eds.) Applied Mycology and Biotechnology: Genes, Genomics and Bioinformatics, vol. 5. Elsevier Science (2006)
7. Singh, G., Singh, H.: Functional proteomics with biolinguistic methods. IEEE Engineering in Medicine and Biology 24(3), 73–80 (2005)

9.7 Exercises

Consider the following set of sequences in DNA database:

```
Seq-1: AGACTGTTACCCAGAAAACTTACAAATTGTAAATGAGAGGTTAGTGAAGAT
Seq-2: GGATCCAGCCTGACCTTGTAAAATAGCCTAACGTGTGTTCCCTAG
Seq-3: CTTAAGACATCAAACAATGTATGTTGAGTTTAACAAGGGAACACAACAAGATG
Seq-4: CTGCTCTAGGAAAAAATGCCTAGATACAAATAAAGACTTT
Seq-5: CTCAGCTTTTGTTTGTCTTGGAAAGTTGTTATTTTTCCCTCATTTCTGAAGGTCA
```

1. Develop a tri-gram frequency profile for the sequence collection.
2. Reduce the frequency profile to a probability density profile.
3. Using cosine similarity to find the nearest neighbor for the query sequence below. Compare your results with those obtained with variational distance:

```
>Query
Query: AAGATAACACATACAGAAAATGTGAGAAAA
```

4. Comment on the relative advantages of each method.
5. Using variational distance find the nearest neighbor for the query sequence.
6. Repeat problem 5 using divergence measure discussed in this chapter.
7. Compare your results with those obtained with BLAST.

Part III
Biological Sequence Analysis

Special sequences of regulatory importance such as introns, promoters, enhancers, Matrix Association Regions (MARs), and repeats are found in non-coding DNA. Many of these regions contain patterns that represent functional control points for cell specific or *differential* gene expression, while others, such as *repetitive* DNA patterns, serve as a biological clock. These and numerous other examples indicate that *patterns* in the eukaryotic DNA may play a vital role in its viability. Other examples of these patterns include the $A + T$ or $G + C$ rich regions, telomeric repeats of sequence AGGGTT, rare occurrence or absence of dinucleotides TA and GC, and tetranucleotide CTAG, and the GNN periodicity in gene coding regions. There is evidence that suggests that some deviations from conserved patterns are deleterious to the viability of an organism. Therefore, the DNA is not a homogeneous string of characters, but is instead comprised of a mosaic of sequence level motifs that come together in a synergistic manner to coordinate and regulate the synthesis of proteins.

Computational models of biological sequences and patterns are of fundamental importance in bioinformatics since they serve as basis for detection of patterns and estimating the significance of their occurrence. Patterns at the various levels of abstraction drive research in genomics and proteomics. Patterns detected at a coarse level of granularity, such as splice sites, binding sites, and functional domains, drive the definition and detection of patterns at higher levels of abstraction such as introns, exons, repetitive DNA, and locus control regions. The higher level patterns provide information about the structure of proteins and their role in the metabolic pathways of cellular functions.

The modeling of sequences and patterns are two complementary objectives. Sequence models provide a basis for establishing the significance of patterns observed, while the pattern models help us look for specific motifs of functional significance. We therefore must consider both of these issues. For example, the probability of finding a pattern such as `TATAAT` in a sequence that is *rich* in the bases `A` and `T` will be more than finding the same pattern in a sequence that is rich is bases `G` and `C`. It is therefore necessary to consider models for both sequences and patterns to accurately estimate the significance of patterns discovered in biological sequences.

A common measure of the significance of a pattern detected in a sequence is *bits*, which is computed as $\log_2(p)$ where p is the probability of finding a given pattern in the background sequence. In the above example, the significance of finding an `AT`–rich pattern in a `GC` rich sequence would therefore be higher than finding such a pattern in a sequence that is `AT`-rich. Thus, the background phenomenon is important to consider when establishing the significance of biological patterns detected in a sequence.

The following sections are divided to discuss both of these aspects, i.e. modeling sequence and modeling patterns.

Notation: The alphabet from which sequences and patterns are drawn is denoted as Σ. For example, the DNA alphabet is Σ_{DNA} ={A,C,T,G}. The cardinality of the alphabet is denoted by $|\Sigma|$. The cardinality of a DNA alphabet is 4, i.e. $|\Sigma_{DNA}| = 4$. A string of zero or more characters drawn from this alphabet is denoted by Σ^*. Σ^* also used to denote the entire set of strings that may be generated from the alphabet.

Chapter 10
Sequence Models

The two main sequence models are Independent Identically Distribution (IID) and Markov Chain (MC). Sequence models are needed to represent the background stochastic processes in a manner that enables one to analytically justify the significance of observations. To provide an analogy, the determination of the sequence model is similar to determining the probability of obtaining a head (H) while tossing a coin. (For a fair coin, this probability would be $\frac{1}{2}$). In general however, we may estimate this probability by studying the strings of the heads and tail sequences that a given coin has produced in the past. Similarly, given the DNA sequence(s), we may induce the underlying model that represents the maximally likely automaton that produced the sequence.

Let us continue our analogy further. After the coin model has been induced, it would be possible to predict the probability of observing coin tossing pattern such as "three heads in a row", etc. Similarly, after inducing a DNA sequence model, it would be possible to deduce the expected frequency of occurrence of a DNA sequence patterns. This is helpful in classifying patterns in terms of their relative abundance in a sequence specific manner. Thus, sequence mode M is defined on the sample space Σ^* and assigned to every sequence $x \in \Sigma^*$ a probability that is a function of x and the sequence model parameters M.

10.1 Independent Identical Distribution (IID)

The simplest of all the sequence models is the Independent Identically Distribution or the IID model. In this model, each of the four nucleotides is considered to occur independently of each other. Furthermore, the probability of occurrence of a given nucleotide at a given location is identical to the probability of its occurrence at another location. For example, assume that the sequence is defined using an IID random variable which can take on the possible values defined by the DNA alphabet = {A, C, T, G}. In this case, defining the individual probability values (p_A, p_C, p_T, and p_G) specifies the

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_10

complete model for the sequence. The values may in turn be computed simply by considering the prevalence of each base in the given sequence. In statistical terminology, the Maximally Likely or ML estimator for the probability of occurrence of a given base X is simply $\frac{n_X}{L}$ where n_X is the frequency of occurrence of the base X in a sequence of length L.

In general, the maximal likely estimator for the parameters may be used. Using the ML estimation, the probability of each base α may be estimated as:

$$P(\alpha) = \frac{n_\alpha(L)}{|L|} \tag{10.1}$$

This simply counts the relative frequency of a nucleotide α in a sequence of length L. This estimator has the advantage of simplicity, and usually works well when $|L|$ is large. Given that the Model M_{IID} has been induced from the sequence data, the probability of an occurrence of a pattern x may be computed using the following:

$$P(x|M_{IID}) = \prod_{i=1,\ldots,n(x)} P(x_i) \tag{10.2}$$

where $P(x_i)$ is the probability of nucleotide x_i at position i along the pattern. The model assumes that the parameters (probability of each of the four nucleotides) are independent of the position along the pattern.

Example 10.1

Consider the following DNA sequence:

```
SEQ = AACGT CTCTA TCATG CCAGG ATCTG
```

In this case the IID model derived from the sequence given the alphabet Σ = {A, C, G, T}, the sequence model parameters are $\{\frac{6}{25}, \frac{7}{25}, \frac{5}{25}, \frac{7}{25}\}$, corresponding to the maximally likely estimation of the occurrence of each of the four bases. These are thus the IID parameters for the background sequence. The probability of finding the pattern `CAAT` on this sequence would be equal to $(p_C \times p_A \times p_A \times p_T)$ or $(\frac{7}{25} \times \frac{6}{25} \times \frac{6}{25} \times \frac{7}{25}) = 0.0045$.

End of Example

10.2 Markov Chain Model

In a Markov Chain (MC) model the value taken on by a random variable at a given location along the sequence is dependent upon the value(s) taken by the random variable at prior locations. The number of historical locations (or states, as they are formally called) that influence the value of the random variable at a given location is specified by the the degree of the Markov process being used in modeling. In a first-order Markov chain models, the probability of observing a nucleotide at location i is only dependent upon the nucleotide located at location $(i-1)$. Similarly, the probability of occurrence

of a nucleotide will be dependent upon the identity of nucleotides at locations $(i-1)$ and $(i-2)$ in a second order Markov chain model, etc.

The first-order MC model is characterized by $|\Sigma| + |\Sigma|^2$ parameters, corresponding to the individual nucleotide frequencies as well as dinucleotide frequencies. In addition to the $|\Sigma|$ parameters capturing the probability of occurrence of the symbols at position 1, the $|\Sigma|^2$ parameters capture the conditional probabilities of finding a base β at position *(i)* given that the base α was found at position *(i-1)*. These values are computed by using Bayes rule, i.e. by finding the abundance of the dinucleotide $\alpha\beta$ as a fraction of the abundance of the nucleotide α.

This sequence model M is defined on the sample space Σ^* where every sequence x of length $n(x)$ on Σ^* is assigned the probability:

$$P(x|M) = P_1(x_1) \prod_{i=2,\ldots,n(x)} P_2(x_i|x_{i-1}) \tag{10.3}$$

where P_1 is a probability function on Σ that models the distribution of α's at the first position in the sequence and P_2 is the conditional probability function on Σ x Σ that models the distribution of β's at position $i > 1$ on the alphabet symbol α at position *i-1.*

The parameter estimation using the Maximally Likely estimator proceeds in a manner analogous to the IID model estimation. The transition probabilities are however estimated using Bayes theorem:

$$P_2(\beta|\alpha) = \frac{P(\alpha\ \beta)}{P(\alpha)} \tag{10.4}$$

It may be pointed out that the sequence model permits the parametrization of the sequence in a position-invariant manner. This will be extended to a position dependent Markov Chain model when we describe the modeling of patterns – where each position in the pattern will have an associated set of $|\Sigma|^2$ conditional probability parameters.

Example 10.2

Consider once again the same 25-nucleotide sequence:

```
SEQ = AACGT CTCTA TCATG CCAGG ATCTG
```

While considering the first-degree Markov chain models, the 4-parameters corresponding to individual nucleotide frequencies, and the 4^2 parameters corresponding to the dinucleotide frequencies need to be computed. The alphabet $\Sigma = \{\text{A, C, G, T}\}$, the sequence model parameters are the same as before: $\{\frac{6}{25}, \frac{7}{25}, \frac{5}{25}, \frac{7}{25}\}$.

In order to compute P_2, the $\Sigma \times \Sigma = \Sigma^2$ conditional probability values, the dinucleotide frequencies and probabilities are computed from the sequence data. The dinucleotide frequencies and the probabilities are shown below in Table 10.1 with the parenthesized numbers representing the probabilities:

Table 10.1 Joint Probability

	A	**C**	**G**	**T**
A	$p_{AA} = \frac{1}{24}$	$p_{AC} = \frac{1}{24}$	$p_{AG} = \frac{1}{24}$	$p_{AT} = \frac{3}{24}$
C	$p_{CA} = \frac{2}{24}$	$p_{CC} = \frac{1}{24}$	$p_{CG} = \frac{1}{24}$	$p_{CT} = \frac{3}{24}$
G	$p_{GA} = \frac{1}{24}$	$p_{GC} = \frac{1}{24}$	$p_{GG} = \frac{1}{24}$	$p_{GT} = \frac{1}{24}$
T	$p_{TA} = \frac{1}{24}$	$p_{TC} = \frac{4}{24}$	$p_{TG} = \frac{2}{24}$	$p_{TT} = \frac{0}{24}$

The probabilities for single nucleotides are determined by considering all nucleotides except the first as the conditional probabilities are computed for nucleotides 2 through n. The resulting probabilities are $p_A = \frac{5}{24}$, $p_C = \frac{7}{24}$, $p_G = \frac{5}{24}$, and $p_T = \frac{7}{24}$. The conditional probabilities are next computed using the Bayes theorem. For example, the probability of finding `C` at position $(i + 1)$ given that an `A` has been found at position i is $P(C|A) = \frac{p_{AC}}{p_A} = \frac{1/24}{5/24} = \frac{1}{5}$. The conditional probabilities for the example sequence are shown in Table 10.2.

Table 10.2 Conditional nucleotide probabilities for the 25-nt example sequence

$\downarrow i \mid \overrightarrow{(i-1)}$	**A**	**C**	**G**	**T**
A	$\frac{1}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{3}{7}$
C	$\frac{2}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{3}{7}$
G	$\frac{1}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{1}{7}$
T	$\frac{1}{5}$	$\frac{4}{7}$	$\frac{2}{5}$	0

Using these model parameters, the probability of finding the pattern `CAAT` on this sequence using the first order Markov model of the underlying sequence would be equal to $P(C).P(A|C).P(A|A).P(T|A)$ or $(\frac{7}{25}).(\frac{1}{7}).(\frac{1}{5}).(\frac{1}{5})$ = 0.002. This is in contrast to the IID sequence model probability of the pattern being 0.0045.

End of Example

10.3 Matrix Association Regions

Matrix Attachment Regions (MARs) are eukaryotic DNA elements that may help regulate differential gene expression. The annotation of MAR sequences on a genetic map can provide insights into regions of locus control; however, computational MAR detection is made difficult by a lack of conservation sequences between MARs. One cannot strictly detect MARs based on a given pattern but must make a prediction based on a series of highly co-occurring motif sequences. Such subsequences have been previously identified for the use in the MAR-Wiz computational tool for the detection of MARs.

Next we look into the significance of these patterns by computing the patterns' chi-square value. The chi-square value allows us to determine which motifs occur more frequently in MARs than would be expected in an arbitrary nucleotide sequence.

10.3.1 Introduction

Matrix Attachment Regions (MARs), also known as Scaffold Attachment Regions (SARs), are short nucleotide sequences (approximately 100 to 1000bp in length) which may assist in DNA replication or transcription. These sequences exist in eukaryotic DNA and attach the DNA to the nuclear matrix, between which genes reside on chromatin loops. [1, 2]. While AT-rich, MARs are believed to lack conservation sequences [3], which results in computational MAR detection being a difficult task.

Previously, MARs motifs have been computationally identified for the software tool MAR-Wiz [4, 5]. These in silico analyses identified 39 motifs, which are represented as regular expressions. These motifs which were determined to occur more frequently in MARs than in arbitrary nucleotide sequences. MAR-Wiz utilizes these motifs to compose rules, which take two forms. The first is simply an OR-type rule that is matched when any motif that comprises that rule is matched. The second type of rule is an AND-type rule that requires two or more rules to occur in sequence. MAR-Wiz determines the unexpectedness of these motifs and rules and uses these with a sliding-window approach to estimate the location of MARs. For each window position, the unexpectedness of the window is given as the sum of the unexpectedness of every rule found within the window.

It is important to test the efficacy of these motifs to determine their suitability for detecting MARs. Since MARs are AT-rich, many AT-rich motifs will be more likely to occur by chance within a MAR than in an arbitrary nucleotide sequence. This motif, however, may not occur more frequently in MARs than in another sequence that is, itself, AT-rich. Therefore, motifs could exist in this set that are apt at detecting AT-rich sequences, but not necessarily MARs. As illustrated below, chi-square based analysis may be used to determine the suitability of statistically establishing significance of motifs.

10.3.2 Selecting Statistically Significant Motifs

In order to determine the effectiveness of current predicted motifs at detecting MARs, we compared the actual number of occurrences of each motif in each MAR sequence to the number that would be expected to occur by chance.

MARs, used to study the performance of the set of motifs, were retrieved from GenBank. A keyword-based search limited to genomic DNA in

eukaryotes was conducted using Entrez for the term "Matrix Attachment Region." The results were manually inspected to identify sequences in which a non-putative MAR had been identified as a feature. These 61 subsequences were added to our library for further analysis.

Computationally predicted motifs, in the form of regular expression, were identified from literature [4]. The regular expression were translated to be comprised strictly of A, C, G, and T; as opposed to allowing for IUPAC ambiguity codes. The probabilities of these motifs occurring randomly were then calculated as a function of the probability of each of the four bases. We used an independent and identically-distribution (iid) model for the motifs (the probability of a nucleotide occurring at a position is independent of nucleotides at any other position). For example, the probability of a simple motif occurring, such as motif m_1 ($ATTTA$) is simply the product of the probability of each base occurring. This can be given by a function:

$$P(m_1) = P_a \cdot P_t \cdot P_t \cdot P_t \cdot P_a \tag{10.5}$$

Slightly more complicated is the probability of motif m_{31} (YR) which is the sums of the probabilities for the nucleotides included in Y (C or T) and R (A or G). So, the probability of Y is the same as the sum of the probabilities of C and T. Similarly, the probability of R is the sum of the probabilities of A and G. So, overall the probability for motif $m_3$1 is shown as:

$$P(m_{31}) = (P_c + P_t) \cdot (P_a + P_g) \tag{10.6}$$

For multiple occurrences of a given base or ambiguity code, such as an example motif such as $A\{3\}$ the probability is simply equal to the probability of the base or ambiguity code, raised to the power of the number or occurrences (in this case, P_a^3). Since $A\{3\}$ is equivalent to AAA it can be shown that the probability of AAA is equal to P_a^3).

$$P(AAA) = P_a \cdot P_a \cdot P_a = P_a^3 \tag{10.7}$$

So, the probability for motif m_{12} ($TAN\{3\}TGN\{3\}CA$), is the product of the probabilities of TA, $N\{3\}$, TG, $N\{3\}$, and CA.

$$P(m_12) = P_t \cdot P_a \cdot ((P_a + P_c + P_t + P_g)^3) \cdot P_t \cdot P_g \cdot ((P_a + P_c + P_t + P_g)^3) \cdot P_c \cdot P_a \tag{10.8}$$

Clearly, over any arbitrary, large dataset, the probability of each nucleotide will approach 1/4. This function, however, allows us to determine with what frequency we should expect a given motif to occur in any given MAR. For example, the probabilities for A and T are higher in MARs–given the AT-richness of MARs–and, as such, we'd expect a high number of motifs that are themselves AT-rich. This probability will allow us to more effectively determine whether a motif is more likely in a MAR than any arbitrary,

non-MAR sequence, for which the proportions of each nucleotide are similar to those in MARs.

Motifs were discarded from our set of statistically verified motifs if they occurred less times than expected. For motifs that occurred more frequently than expected, we calculated their chi-square value χ^2. Motifs were then listed in order of increasing chi-square value. These results can be seen in Fig. 10.2.

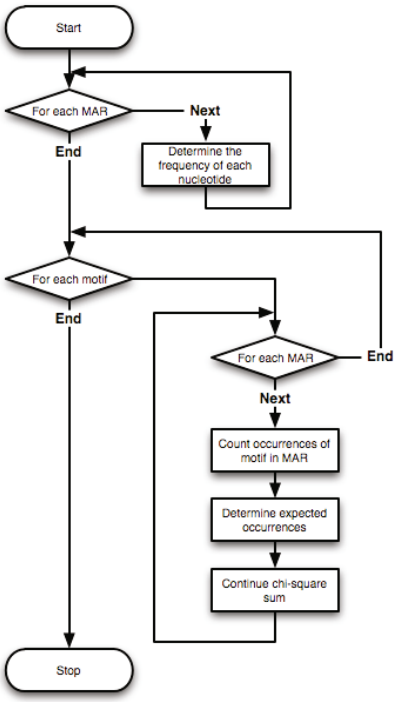


Fig. 10.1 The algorithm used to determine the predicted and actual occurrences and chi-square value for each MAR

$$\sum_{m \in M} \frac{(X_m - \mu_m)^2}{\mu_m} \tag{10.9}$$

10.3.3 Removing Motifs and Rules from MAR-Wiz

Since those motifs that occurred less often than expected have been removed from our set, patterns with high chi-square also those that occur much more often than anticipated. Our ordered list, therefore, ranks motifs from those of least significance at detecting motifs to most significant.

Motif	Motif	Expected	Observed	Chi-square
m_1	$ATTA$	1499.41	1073	-
m_2	$ATTTA$	500.616	453	-
m_3	$ATTTTA$	168.645	204	79.5232
m_4	$TGTTTTG$	13.811	32	157.086
m_5	$TGTTTTTTG$	1.47226	1	-
m_6	$TTTTGGGG$	1.38835	7	247.13
m_7	$AAAAN_7AAAN_7AAAA$	0.83117	15	1452.63
m_8	$TTTTN_7TTTN_7TTTT$	0.83117	15	1452.63
m_9	$TTTAAA$	168.645	238	171.335
m_{10}	AAA	4531.99	4019	-
m_{11}	$AAAA$	1499.41	1587	161.154
m_{12}	TAN_3TGN_3CA	42.7963	42	-
m_{13}	TAN_3CAN_3TG	42.7963	58	106.791
m_{14}	TGN_3TAN_3CA	42.7963	50	85.3869
m_{15}	TGN_3CAN_3TA	42.7963	42	-
m_{16}	CAN_3TAN_3TG	42.7963	20	-
m_{17}	CAN_3TGN_3TA	42.7963	58	106.791
m_{18}	$RNYNNCNNGYNGKTNYNY$	3.91252	4	120.861
m_{19}	$GTNWAYATTNATNNR$	1.51401	9	810.318
m_{20}	$NCNNCYNGKTNYNY$	15.6643	25	61.8074
m_{21}	$WWWWWWN\{8\}S\{0,4\}WWWWWW$	766	1686.3	-
m_{22}	$AATATTTTT$	6.79283	18	310.776
m_{23}	$AATAAAYAAA$	3.39608	19	536.527
m_{24}	$ATATTT$	168.645	242	185.817
m_{25}	$WTTTAYRTTTW$	6.79151	12	145.28
m_{26}	$ATTTCASTTGTAAAA$	0.00196501	8	639940
m_{27}	$WWCAAWG$	110.488	109	-
m_{28}	$CTTTTAGCWWW$	0.648685	1	15.9582
m_{29}	$TGTTTATGNTTTCCGAAANNNAAAA$	$3.19983 * 10^-7$	0	-
m_{30}	$TAATTA$	168.645	124	-
m_{31}	YR	33089	29794	-
m_{32}	$AYCYRTRCAYYW$	1.33738	0	-
m_{33}	$AATAAYAA$	28.6543	41	145.895
m_{34}	$AWWRTAANNWWGNNNC$	3.89067	5	33.6584
m_{35}	$WAWTTDDWWWDHWGWHMAWTT$	0.310064	1	358.399
m_{36}	$WADAWAYAWW$	132.913	198	221.735
m_{37}	$WWDAWAYAWW$	265.827	312	171.351
m_{38}	$TWWTDTTWWW$	184.447	245	181.134
m_{39}	$TTWTWTTWTT$	18.9432	72	1022.41

Fig. 10.2 Table of chi-square values for motifs

Firstly, we tested to see if this set—with 13 motifs removed—would adequately detect MARs. We removed these motifs from the set used by MAR-Wiz and tested the results against results generated by MAR-Wiz using the original rule set. These tests were run on sequences for which MAR-Wiz successfully identified MARs that had been previously located experimentally. Initially, for some sequences, results produced using the modified rule set differed from the results produced using the original rule set. We then found

that adding only motif m_{21} back into the rule set corrected this issue—this motif will be discussed later. After re-adding motif m_{21}, for all input test sequences, the results were manually inspected and found to be nearly identical to those produced by MAR-Wiz using the original rules.

We then removed motifs one at a time, in order of increasing chi-square value, starting with the lowest chi-square value. When all the motifs that composed a rule were discarded, the rule was discarded as well. After each removal, we again manually inspected the results of MAR-Wiz using the newest rule set and compared it to the output from the original rule set. This continued until we could no longer remove the motif with the lowest chi-square value and retain similar results to the original results. Figs. 10.3 and Fig 10.4 show the similarity between the results using the original and new rules for a specific sequence.

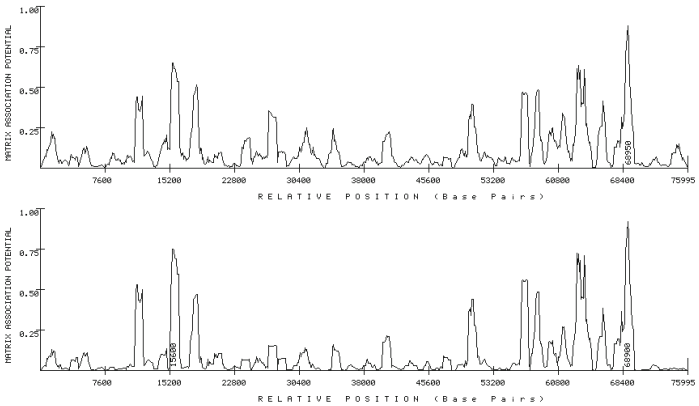


Fig. 10.3 TOP: A graph indicating the potential for MARs in human beta globin using the original rules. Bottom: A graph indicating the potential for MARs in the same sequence, but with the new rules set.

The original set of rules consisted of 39 motifs comprising more than 10 rules. The resulting set contained only 13 motifs comprising 5 rules.

Surprisingly, over all MARs, only 26 of 39 motifs were actually observed in our MAR library more than expected. Furthermore, the Chi-square value of each motif was determined for each MAR in which the number of observed occurrences was greater than the number expected. Of those 26 motifs that occurred more frequently than expected with 61 degrees of freedom (the number of MARs in our library), 23 had a P-value of less than 0.05 and 21 had a P-value less then 0.005 (Fig. 10.5). The respective thresholds for P-values are 80.23, 89.59, and 93.18 for 0.05, 0.01, and 0.005.

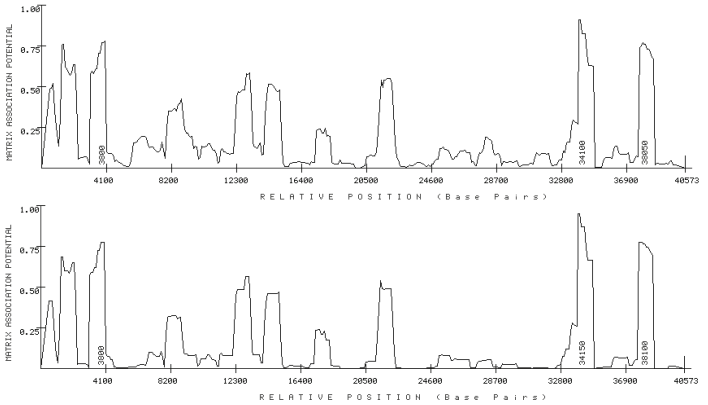


Fig. 10.4 TOP: A graph indicating the potential for MARs in human protamine using the original rules. Bottom: A graph indicating the potential for MARs in the same sequence, but with the new rules set.

Specifically of interest is motif 39 ($TTWTWTTWTT$) which occurs 72 times, but was expected to occur less than 19 times. Furthermore, motifs 3, 9, 24, 36, 37, and 38 all occur in every MAR and do so with a greater frequency than would be expected.

In general, it appears that these 13 patterns that did not occur more frequently than expected are poor as being a positive indicator of a MAR. Specifically, short and simple patterns such as $ATTA$ seem to not effective signs of MARs as these sequences ought to occur many times within AT-rich regions. In actuality, however, most of these patterns do not occur with a much greater frequency than anticipated and some even occur less often than expected.

Furthermore, many long motifs, while occurring more frequently than expected, occurred so few times as to be of little use in predicting MARs. Motifs that occurred once or twice within the entire MAR library were removed from the rule set. One would anticipate the removal of these infrequently occurring motifs as they seem to only indicate specific MARs and would not serve very well as a general indicator.

Motif 21 seems to be an exception to this rule as it is anticipated to occur with high frequency, occurs less than expected in reality, and is still significant as an indicator of MARs. In our experiments, motif m_{21} comprises a MAR-Wiz rule all by itself. In literature, this rule was expressed as the separation of two instances of the same motif, $WWWWWW$, separated by eight to twelve other nucleotides, and was the only such rule to be represented by the disjunction of conjunctions [5]. In our experiments, we replaced this by simply using one motif via the regular expression, $WWWWWWN\{8\}S\{0,4\}WWWWWW$.

Both the rule from literature and our regular expression are equivalent. Our regular expression is modified so that it cannot match both a string and some

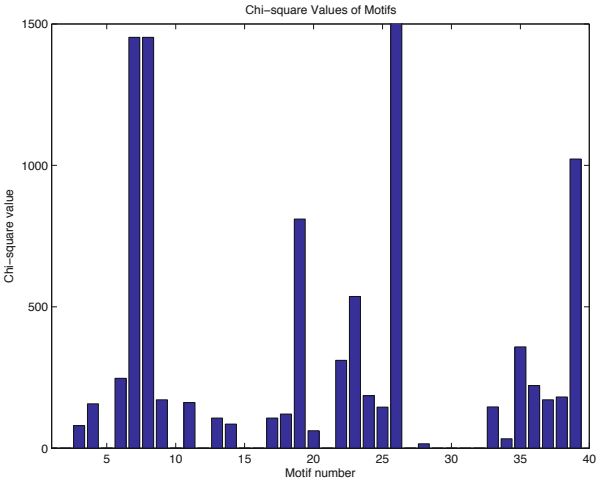


Fig. 10.5 This graph shows the chi-square values for each motif. The graph is only defined for motifs where the occurrences exceeded the expectation. Note: motif m_{26} has a chi-square value of 639,940 and has been clipped by the graph window.

initial substring for that string, as allowing for such an possibility would result in an inflated probability for this regular expression. While only occurring 766 times as opposed to the expected 1,686 and only presenting in 41 out of 61 MARs, the removal of m_{21} from the rule set drastically changed the output for certain motifs. Since it only occurs in roughly two-thirds of MARs, we cannot claim it to be a necessary condition for the existence of a MAR. As such, we cannot statistically or computationally explain the significance of this motif.

Other than motif m_{21}, the 13 motifs remaining in the rule set are the most statistically significant motifs acquired from literature. That is, they are they seem to be good indicators of a MARs presence as they occur more frequently in MARs than would be expected in other nucleotide sequences of even similar nucleotide distribution. We believe the resulting set is a core set of rules that can be used to computationally detect MARs.

Further Readings

1. Strick, R., Laemmli, U.K.: SARS are cis dna elements of chromosome dynamics: Synthesis of a SAR repressor protein. Cell 83, 1137–1148 (1995)
2. Lodish, H., Berk, A., Matsudaira, P., Kaiser, C.A., Kreiger, M., Scott, M.P., Zipursky, S.L., Darnell, J.: Molecular Cell Biology, 5th edn., pp. 427–429. W.H. Freeman and Company, New York (2004)
3. Lewin, B., Cassimeris, L., Lingappa, V.R., Plopper, G. (eds.): Cell, pp. 259–260. Jones and Bartlett Publishers, Sudbury (2007)

4. Singh, G.B., Krawetz, S.A.: Data mining algorithm for discovering matrix attachment regions (mars). In: Proceedings of SPIE, vol. 4057, pp. 330–341 (April 2000)
5. Singh, G.B.: Discovering matrix attachment regions (mars) in genomic databases. SIGKDD Explorations 1(2), 42–48 (2000)
6. Durbin, R.: Biological sequence analysis: probabalistic models of proteins and nucleic acids. Cambridge University Press, Cambridge (1998)
7. Zhang, Y.-Q., Rajapakse, J.C.: Machine learning in bioinformatics. Wiley series on bioinformatics. Wiley, Hoboken (2009)
8. Robin, S., Rodolphe, F., Schbath, S.: DNA, words and models. Cambridge University Press, Cambridge (2005)
9. Trent, R.J.: Clinical bioinformatics, vol. 141. Humana, Totowa (2008)

10.4 Exercises

1. What is the probability of finding a pattern `ATTACG` in a DNA sequence where all bases are equally likely. How many occurrences of this pattern do you expect to see in a sequence that is 10^8 base pairs in length.
2. Assume that Pr(A)=Pr(T)=0.2, and Pr(G)=Pr(C) = 0.3. Using Poisson probability distribution to calculate the probability of finding *two* occurrences of the pattern ACCTGACC in a sequence window of 500 bp long?
3. Repeat problem 2 for probability of finding *two or more* occurrences of the pattern ACCTGACC in a sequence window of 500 bp long? What is the significance of this observation?
4. Assume that the background sequence is given as shown below:

   ```
   CCTTA ATTAC CAAGG CATTA CCGAT
   ```

   a. Construct an IID model for the sequence and compute the probability of finding the pattern CAAT.
   b. Compute the probability of finding CAAT under a first order Markov model. *You only need to compute the conditional probabilities that you require for this purpose.*
   c. Compute the log likelihood ratio:

$$LLR = \log \frac{Pr(CAAT|\text{Markov})}{Pr(CAAT|\text{IID})}$$

5. Given the sequence:

   ```
   ATATTATGCCGTATAACCGGTT
   ```

   Construct its IID model and a first order Markov chain model. Using these models, estimate the probability of finding sequence, `ATTA` in the sequence. Clearly state the significance of your answers in each case.
6. Compute the probability of finding a pattern ATTA given that you have no *a-priori* knowledge of the background sequence data.
7. Assume that the background sequence is given as shown below. Construct an IID model for the sequence and compute the probability of finding the pattern ATTA.

   ```
   ATTTT CTGGG ATATC CGGAG GATAT GGGAC CCTAG
   ```

8. Assuming the same background sequence as given in the previous question, compute the first order Markov model for the background. What is the liklihood of finding the pattern `ATTA` assuming the background behaves as a first order Markov chain?

9. Which of the answers of the liklihood value computed above is most reliable and why? Justify with an new *example* of your choice.
10. What is the liklihood ratio of finding the pattern `ATTA` in a sequence with an IID model characterized by $p_A = p_C = p_G = p_T = \frac{1}{4}$ and your answer for Exercise 8. Express this ratio as a log-likelihood. What is the significance of this value?

Chapter 11
Subsequence Pattern Models

Having described some methodologies for modeling sequences, we now turn our attention toward describing the statistical modeling procedures for DNA patterns. There are a growing number of well-established patterns that we may wish to model and search for in DNA sequences. Often these patterns of functional significance are discovered after an alignment of sequences belonging to a particular family. Such a multiple sequence alignment is often interspersed with gaps of varying sizes. However, there are sections in the final alignment that are free of gaps in all of the sequences. These fixed-size ungapped aligned regions represent the types of patterns that we need to model to identify patterns in an anonymous segment of DNA. The statistical technique based on Hidden Markov models may be employed for developing such a closed form representation of a set of patterns. This section describes the statistical modeling procedures for DNA patterns. Thus, our attention is now focused somewhat more on modeling the motifs that are associated with certain biological functions. In the previous section our goal was to characterize the "haystack" of data in which these biological "needles" of information are hidden. In contrast, our goal in this section is to model the nuggets themselves. The statistical techniques that may be employed for developing a closed form representation of a set of patterns are described below.

11.1 Regular Expressions

Regular expressions are utilized in bioinformatics for representation of patterns in a condensed format. One of the most useful features in the Perl programming language is its powerful string manipulation tools, including its ability to manipulate regular expressions – the natural representational style for biological patterns. The Perl programming language is popular with biologists because of its practical applications to DNA and protein sequences.

For example, a compact regular expression based representation of the TATA-box could be devised. Consider, for example, that the sequence level

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_11

motifs for TATA-box are TATAAT or TATTAT or TATAA. One could conveniently represent and look for these variations using a single regular expression: $TAT(AAT|TAT|AA)$. Special characters, `'^'` and `$+` are used to denote the start and the end of the sequence being searched. For example, `^MSE` denotes the peptide sequence that begins with `MSE`, and similarly the `PDZ` binding sites at the end of a sequence record are denoted by $[ST]X[VIL]$\$. Some meta-characters are symbols that represent more than one character. Within the context of regular expressions, these are "*" and "+" which represent matching with zero or more, and one of more characters respectively. Thus, the regular expression `AAAAN+AAAA` matches two "A" tracks of length 4 that are separated by any number of bases.

11.2 Weight Matrices

A DNA sequence matrix is a set of fixed-length DNA sequence segments aligned with respect to an experimentally determined biologically significant function. A DNA sequence motif can be defined as a matrix of depth 4 utilizing a cut-off value. The 4-column/mononucleotide matrix description of a genetic signal is based on the assumption that the motif is of fixed length, and that each nucleotide is independently recognized by a trans-acting mechanism. If a set of aligned signal sequences of length L correspond to the functional signal under consideration, then $F = [f_{bi}], (b \in \Sigma), (j = 1 \ldots L)$ is the nucleotide frequency matrix, where f_{bi} is the absolute frequency of occurrence of the $b^{th\cdot}$ type of the nucleotide out of the set $\Sigma = \{A, C, G, T\}$ at the $i^{th\cdot}$ position along the functional site. A method for converting the frequency matrix into a weight matrix has been proposed. This method is based on weights at a given position being proportional to the logarithm of the observed base frequencies. These are increased by a small term that prevents the logarithm of zero and minimizes sampling errors. The weight matrix is computed as shown in Eq. 11.1. The term $f_{b,i}$ is the frequency of base b at position i, and e_b represents the expected frequency of base b, c_i a column specific constant, and s, a smoothing percentage.

$$W(b, i) = \log_2(\frac{f_{bi}}{e_{bi}} + \frac{s}{100}) + c_i \tag{11.1}$$

These optimized weight matrices can be used to search for functional signals in nucleotide sequences. Any nucleotide fragment of length L is analyzed and tested for assignment to the proper functional signal. A matching score of $\Sigma_{i=1}^{L} W(b_i, i)$ is assigned to the nucleotide position being examined along the sequence. In the search formulation, b_i is the base at position i along the oligonucleotide sequence, and $W(b_i, i)$ represents the corresponding weight matrix entry for base b_i occurring along the i^{th} position in the motif. For example, the weight matrix reported for the functional pattern commonly known as the `TATAA-Box` is shown in Table 11.1. Matrices such as PAM and

Table 11.1 Weight Matrix for `TATAA` box

T	6	49	1	56	6	22	6	20
C	14	6	0	0	3	0	1	2
A	8	4	58	4	51	38	53	30
G	32	1	1	0	0	0	0	8

BLOSUM matrices are derived in a manner similar to the process described above. The term Position Specific Scoring Matrix, (PSSM), as these are called, is often used to define the individual score profile within the various columns of the pattern. A PSSM can be used to search for a match in a longer sequence by evaluating a score S_j for each starting point j in the sequence from position 1 to $(N - L + 1)$ where L is the length of the PSSM.

Example 11.1

Consider a block of DNA sequences representing an ungapped alignment:

Alignment

A	A	C	T	T	A
A	A	T	T	T	C
A	T	T	T	A	A
C	A	A	A	A	A
A	T	A	A	G	A
A	T	G	A	A	T

The computation of frequency values used in the above score matrices utilize the Laplace rule, such that all frequency values are incremented by `1` to avoid occurrences of zero probability. Thus, frequencies of {A, C, G, T} in the first column are set to {`(5+1), (1+1), (0+1), (0+1)`}.

Score Matrix

A	6	4	3	4	4	5
C	2	1	2	1	1	2
G	1	1	2	1	2	1
T	1	4	3	4	3	2

Thus, $p_A = \frac{26}{60} = \frac{13}{30}$, $p_C = \frac{9}{60} = \frac{3}{20}$, $p_G = \frac{8}{60} = \frac{2}{15}$, $p_T = \frac{17}{60}$.

Next, the weight matrix is constructed by considering the log-odds score of $\frac{f_{b,i}}{e_b}$. For example, assuming that $s = 10$ and $c = 0$, the log-odds score of the nucleotide `A` at column `1` is $\log_2(\frac{f_{A,1}}{e_A} + \frac{s}{100} = \log_2(\frac{6/10}{13/30} + 0.1) = 0.570$. The value thus obtained is multiplied by 100, and the fractional part is dropped, yielding a weight of `57` for nucleotide `A` in column `1` of the matrix shown below. Other values are similarly computed.

Weight Matrix

A	57	3	-34	3	3	33
C	52	-38	52	-38	-38	52
G	-23	-23	68	-23	68	-23
T	-114	60	21	60	21	-31

End of Example

11.3 Position Dependent Markov Models

Markov models have been considered as a means to define the background DNA sequence. Markov models enable us to define the probability of a nucleotide conditioned upon the nucleotides that occur in the preceding position. However, the modeled dependency is position-invariant. A position dependent Markov model may be utilized for the representation of a sequence signal or motif. This model is defined on the sample space Σ^* and assigns to every sequence x on Σ^* a probability given by Eq. 11.2 below:

$$P(x|M) = P_1(x_1) \prod_{i=2,\ldots,n} P_i(x_i|x_{i-1}) \tag{11.2}$$

This model has $|\Sigma| + (n-1) \times |\Sigma|^2$ parameters. The first Σ parameters in this equation are first-order probabilities estimated using the occurrences of the symbol $\alpha \in \Sigma$ at the first position of a pattern. The other $(n-1) \times |\Sigma|^2$ probability values are for the conditional occurrence (i.e. first order Markov process) of symbol α at position i given that symbol β occurred at position $(i-1)$. These $|\Sigma|^2$ parameters need to be estimated for each of the remaining $(n-1)$ positions in the pattern. Thus, the position-specific dependencies on the previous position are determined by allowing a unique set of transition probabilities to be associated with each position along the signal. Generally, this model assumes that an ungapped multiple sequence alignment of the pattern instances is available, and that the number of sequences is sufficient training to induce position specific Markov probabilities.

Example 11.2

An ungapped alignment of a set of sequences and the corresponding parameters are shown below:

Alignment

A	A	A	T	T	A
A	A	T	T	T	A
A	T	T	T	A	A
A	A	A	A	A	A
A	T	A	A	A	A

Frequencies for Position 1

A	C	T	G
6	1	1	1

Convert the frequencies to probability by multiplying it with $(\frac{1}{9})$.

Position specific Markov frequencies for positions 2–N

i	A	A	A	A	C	C	C	C	G	G	G	G	T	T	T	T
i-1	A	C	T	G	A	C	T	G	A	C	T	G	A	C	T	G
i=2	4	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1
i=3	4	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1
i=4	4	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1
i=5	4	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1
i=6	4	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1

The frequency values listed in the above table may be converted to probability values by multiplying it with $(\frac{1}{21})$.

End of Example

11.3.1 Profiles

Profiles was the earliest representation methodology for capturing variability in the sequence data. Profiles, or the profile matrix, are often computed from the multiple sequence alignment of the patterns that we aim to represent. Multiple sequence alignments of the various forms the pattern may take are used to find a profile matrix representing the statistical estimate of the probability of finding a base at a given location. It is possible to perform a comparison of a profile matrix against a sequence or series of sequences. The profile matrix specifies a different score for each letter (column) in each position in the alignment (row). This differs from scoring matrices, which assign a single score for a two-letter match independently of the match's position in the alignment. Pattern search methodologies are devised to detect the highest pattern scores using the weight assigned to each column of the pattern.

Profiles are similarly defined for modeling functional motifs in amino-acid sequences. A profile is a scoring matrix $M(p, a)$ comprised of 21 columns and N rows, where N is the length of the motif. The first 20 scores represent the weight for each individual amino acid, and the 21^{st} column specifies the cost associated with an insertion or deletion at that position. The value of the profile for amino acid **b** defined for position p is given by Eq. 11.3 below:

$$M(a, p) = \sum_{b=1}^{20} W(b, p) \times Y(b, a) \tag{11.3}$$

The term $Y(b, a)$ is Dayhoff's matrix and $W(b, p)$ is the weight for the occurrence of amino acid b at position p. The position specific weight is defined by $\log \frac{f(b,p)}{N}$, or the frequency of occurrence of the amino acid b as a fraction of the total N sequences utilized for construction of the profile, with a frequency of 1 being used for any amino acid that does not appear at position p.

11.3.2 Hidden Markov Models

There are several extensions to the classical Markov chains, and Hidden Markov Models (HMM) are one such extension. The rationale for building HMMs comes from noticing that our observations could arise from a model characterizing a pattern, or from a model characterizing the background. For example, the DNA sequence HMMs are developed to characterize a pattern as an island within the sea of DNA. The Markov chain characterizing both

these types of DNA is embedded within the same model, with the ability to switch from one type to the other. In this manner, a HMM utilizes a set of hidden states with an emission of the symbols associated with each state. From a symbol generation perspective, the state sequence executed by the model is not observed. An N-state HMM is parameterized using the set $\lambda = \{A, B, \pi\}$. Individual elements of this set are defined as follows:

1. **A:** The $N \times N$ matrix $A = \{a_{ij}\}$ represents the state transition probabilities. $a_{ij} = Pr[q_{x+1} = S_j \mid q_x = S_i] \quad 1 \leq i, j \leq N$
2. **B:** The $Q \times |\Sigma|$ emission probabilities corresponding to the emitting states. As we discuss below a subset of the N states have emissions associated with them. The elements of this matrix, $E = \{e_k(b)\}$, are defined as follows: $e_k(b) = Pr[O_x = b \mid q_x = S_k] \quad 1 \leq k \leq Q, 1 \leq b \leq |\Sigma|$
3. π : The initial state distribution probabilities, $\pi = \{\pi_i\}$. $\pi_i = Pr[q_1 = S_i] \quad 1 \leq i \leq N$

Although a general topology of a fully connected HMM allows state transitions from any state to any other, this structure is almost never used. Often, the over-generalized model produces sub-optimal results due to the lack of training data. Consequently, more restrictive HMMs that rely on the problem characteristics to suitably reduce the model's complexity and the number of model parameters that are needed are utilized. One such model is defined to be the profile-HMM, which is induced from a multiple sequence alignment. The initial transition and observation probabilities are set in a reasonable manner with all the state transitions and symbol emission probabilities as being equally likely. The Maximally Likelihood Estimation procedure proposed by Baum-Welch is next utilized for training the HMM.

seq-1	G	C	C	C	A
seq-2	A	G	C		
seq-3	A	A	G	C	
seq-4	A	G	A	A	
seq-5	A	A	A	C	
seq-6	A	G	C		

(a)

seq-1	ϕ	ϕ	G	C	C	C	A
seq-2	ϕ	A	G	C	ϕ	ϕ	ϕ
seq-3	A	A	G	C	ϕ	ϕ	ϕ
seq-4	ϕ	A	G	A	A	ϕ	ϕ
seq-5	ϕ	A	A	A	C	ϕ	ϕ
seq-6	ϕ	A	G	C	ϕ	ϕ	ϕ
	*	M_1	M_2	M_3	*	*	*

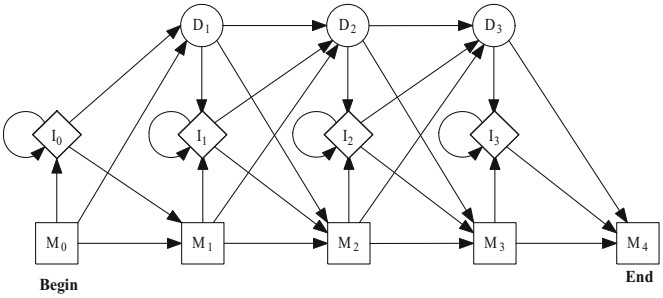
(b)

Fig. 11.1 The set of sequences in (a) are aligned and shown in (b)

Each of the steps outlined above are described in more detail using an example. Let us consider a set of six DNA sequences shown in Fig. 11.1(a). A multiple sequence alignment of these sequences is the first step towards the process of inducing the Hidden Markov Model.

Model Topology: As a first step towards inducing the model, the topology of the HMM is established using the consensus sequence. The aligned

columns of symbols correspond to either emissions from the same match state or to emissions from the same insert state. In this formalism therefore, the columns that correspond to the match state are established to define the match states of the HMM architecture. As shown in Fig. 11.2 there columns are marked as M_1, M_2 and M_3.



(a)

seq-1	M_0	D_1	M_2	M_3	I_3	I_3	I_3	M_4
seq-2	M_0	M_1	M_2	M_3	M_4			
seq-3	M_0	I_0	M_1	M_2	M_3	M_4		
seq-4	M_0	M_1	M_2	M_3	M_4			
seq-5	M_0	M_1	M_2	M_3	I_3	M_4		
seq-6	M_0	M_1	M_2	M_3	M_4			

(b)

Fig. 11.2 The consensus columns are used to define the match states M_1, M_2 and M_3 for the HMM. After having defined the match states, the corresponding insert and delete states are defined to complete the profile-HMM topology.

Transition Probabilities: The value of each transition probability is computed using the frequency of the transitions as each sequence is considered. The model parameters are computed using the state transition sequences defined in Fig. 11.2(b). The frequency of each of the transitions and corresponding probabilities are shown in Fig. 11.3.

Emission Probabilities: Having thus specified the state transition sequence, the emission probabilities for each of the symbol, $\alpha \in |Sigma|$ is computed for each match and insert state, k, in the model. The emission probability is computed using the formula Eq 11.4. Thus an emission probability is associated with each state, and specifies the probability of emitting each of the symbols in $|\Sigma|$ in the state k.

$$e_k(\alpha) = \frac{Freq_k(\alpha)}{\Sigma_\nu(Freq_k(\nu))} \tag{11.4}$$

	State			
	0	1	2	3
M→M	4	5	6	4
M→D	1	0	0	-
M→I	1	0	0	2
I→M	1	0	0	2
I→D	0	0	0	-
I→I	0	0	0	2
D→M	-	1	0	0
D→D	-	0	0	-
D→I	-	0	0	0

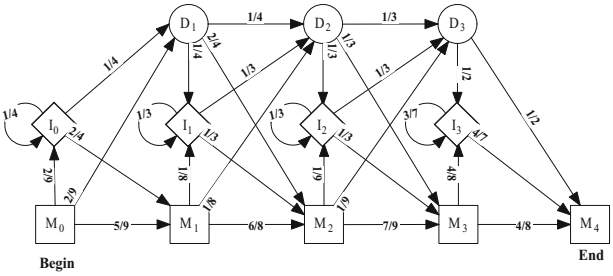


Fig. 11.3 The state transitions inferred from the above topology are used to compute the frequency of transitions for the various states in the model and state transition sequences in Fig. 11.2. These frequency values are subsequently utilized to compute the transition probabilities shown on the model above. Laplace rule is used to avoid zero probabilities.

Using the above formulation, the emission probability for each state is computed as shown in Fig. 11.4.

	State			
	0	1	2	3
Match Emissions				
A	-	5	1	2
C	-	0	0	4
G	-	0	5	0
T	-	0	0	0
Insert Emissions				
A	1	0	0	1
C	0	0	0	2
G	0	0	0	0
T	0	0	0	0

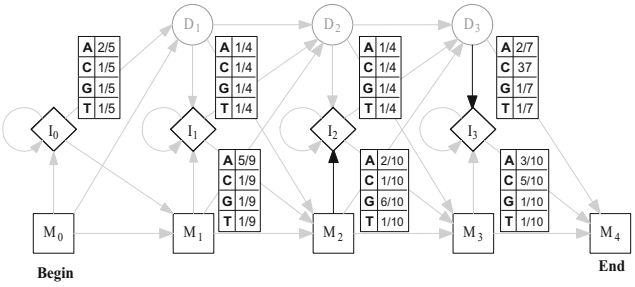


Fig. 11.4 The state specific frequency of observation of a symbol is used for determining the probabilities of emissions. Again, Laplace rule is used to avoid zero probabilities.

From a standpoint of parameters used to characterize an HMM, all states have an emission probability vector associated with them. It may be easy to think of the emission matrix as an $(N \times (|\Sigma + 1|))$ matrix, corresponding to the extension of the base alphabet with the symbol ϕ used for denoting a deletion. The probability of the emission of the symbol ϕ is set to `0` for the insert and the match states. Correspondingly, the probability of emission of symbol ϕ is set to `1` for the delete state with all other probabilities being set to `0`.

Example 11.3

Similar process is followed for inducing the HMM for an aligned set of protein sequences. Consider the following sequence alignment representing a protein motif.

```
HBA_HUMAN        ... VGA--HAGEY ...
HBB_HUMAN        ... V----NVDEV ...
GLB3_CHITP       ... VKG------D ...
LGB2_LUPLU       ... FNA--NIPKH ...
GLB1_GLYDI       ... IAGADNGAGV ...
Match States         ***  *****
```

A HMM with 8 match states may be constructed based on this alignment. The residues AD in GLB1_GLYD1 are treated as insertions, with respect to the consensus. In match state 1, the emission probabilities are (using Laplace's rule): $e_{M1}(V) = \frac{4}{25}$, $e_{M1}(F) = \frac{2}{25}$, $e_{M1}(I) = \frac{2}{25}$, and $e_{M1}(a) = \frac{1}{25}$ for all other residues.
The transition probabilities from match state 1 are as follows: $a_{M1,M2} = \frac{5}{8}$, $a_{M1,D2} = \frac{2}{8}$, $a_{M1,I1} = \frac{1}{8}$, corresponding to the one deletion in HBB_HUMAN, and no insertions. The emission probabilities for the state I_1 will be all equal to (1/20).

End of Example

11.4 Hidden Markov Models with MATLAB

A multiple sequence alignment must be generated as a prerequisite to constructing Hidden Markov models. The MSA is then fed into the HMM generation function which analyzes the alignment in a column wise fashion and comes up with the requisite number of match, insert, and delete states that are maximally probable from the observations of the multiple sequence alignment. So, the process is begun by generating a MSA as shown below.

11.4.1 Multiple Sequence Alignment

In this example, we will construct an alignment and induce an HMM. As discussed earlier, the input to a HMM estimation program is multiple sequence alignment. Let's first look at the tool that performs MSA.

Relevant functions:

multialign	Construct a multiple sequence alignment.
multialignviewer	Viewing a multiple sequence alingment

11.4.2 Multialign

```
SeqMultiAligned = multialign (seqs, 'ScoringMatrix', s,
                                    'GapOpen', g,
                                    'ExtendedGap', e,
                                    'TerminalGapAdjust', t
                                )
```

All parameters, except for *seqs*, are optional. The parameter *seqs* is a vector of structures with the fields 'Sequence' for the residues and 'Header' for labels. The parameter *seqs* may alos be a cell array of strings or a character array.

The following example constructs a cell array of three sequences and constructs a multiple sequence alignment:

```
>> s1= {'AAAC'};
>> s2= {'AAAACT'};
>> s3= {'ACACTAC'};

>> seq = [s1; s2; s3]

seq =
    'AAAC'
    'AAAACT'
    'ACACTAC'

>> ma =  multialign(seq)

ans =
    AAA---C
    AAAAC-T
    ACACTAC
```

The multiple sequence alignment thus created may be graphically visualized by issuing the command *multialignviewer (ma)*.

Relevant functions in this set are:

hmmprofstruct	Construct an HMM.
hmmprofestimate	Induce an HMM from an alignment
showhmmprof	Display an HMM
pfamhmmread	Reads an HMM from a file

Hidden Markov Models will be constructed using function *hmmprofstruct* and the probabilities will be computed with *hmmprofestimate*:

```
>> hmm = hmmprofstruct (6, 'Alphabet', 'NT')

hmm =
        ModelLength: 6
           Alphabet: 'NT'
      MatchEmission: [6x4 double]
     InsertEmission: [6x4 double]
       NullEmission: [0.2500 0.2500 0.2500 0.2500]
             BeginX: [7x1 double]
             MatchX: [5x4 double]
            InsertX: [5x2 double]
            DeleteX: [5x2 double]
    FlankingInsertX: [2x2 double]
              LoopX: [2x2 double]
              NullX: [2x1 double]


>> hmmprofestimate(hmm, ma)

>> showhmmprof(hmm)
```

Try a real example with seven tumor antigen sequences provided with MatLab.

- Align dataset

```
p53 = fastaread('p53samples.txt')
ma = multialign(p53,'verbose',true)
showalignment(ma)
```

- Induce HMM for primate data set. Submit your results.

11.5 Profiles and Model Searches

MATLAB provides a method for aligning a model with HMM: *hmmproalign*. Note that HMM treats HMMs as a special case of profiles which is a somewhat generalized form of PSSM. Please review methods for constructing profiles, *seqprofile*, and for aligning a given sequence with a profile, *profalign*.

11.6 PFAM Database

Pfam or Protein families is a database of know protein domains and patterns. Pfam is a large collection of multiple sequence alignments and hidden Markov models covering many common protein domains and families. Individual patterns have been aligned and stored as a database of HMM models. A HMM downloaded from the PFAM database can be read into MATLAB using the

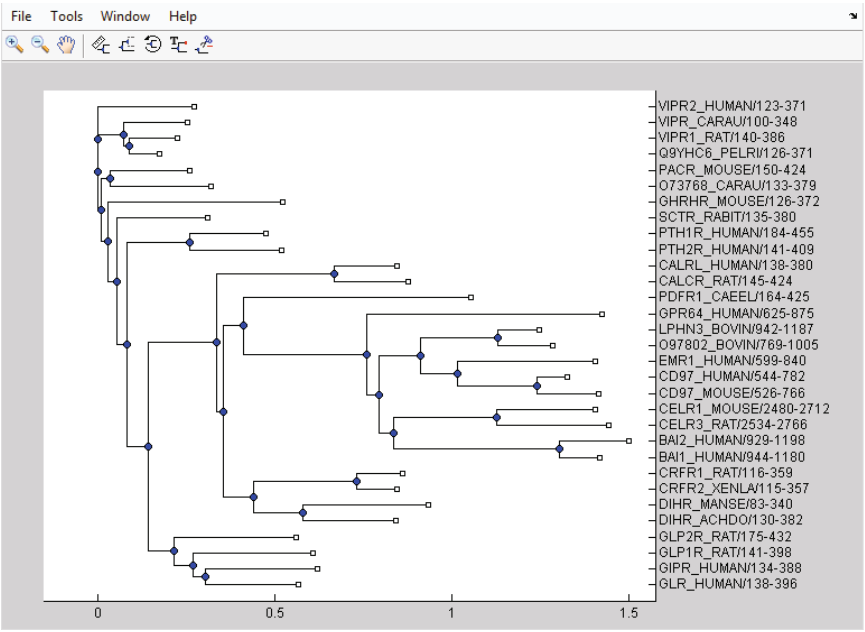


Fig. 11.5 Visualization for a Pfam profile

function: *pfamhmmread(fileName)* where the HMM from Pfam database has been stored in a file.

Pfam, a comprehensive database of over 13,000 conserved protein families that have been characterized using Markov models. This allows transfer of annotation from functionally and sometimes also structurally characterized proteins to proteins of unknown function. This is used for classifying and annotating proteins and for analyzing proteomes and for identifying interesting new targets for structure determination.

Pfam is composed of two types of families, Pfam-A and Pfam-B. Each manually curated Pfam-A family has a seed alignment, which contains a set of representative sequences for that family, from which we generate a profile hidden Markov model (HMM) with each Pfam-A profile HMM is searched against a primary sequence database. Pfam-A families are also annotated with structural and functional information if available and are cross-linked to other relevant databases. Pfam-B families are derived from sequence segments are supplemental to the Pfam-As and give an indication of additional conserved regions. Pfam-B families have an associated alignment but do not have any annotation or literature references. An HMM is generated for each Pfam-B to enable searches to be run against them. Unlike Pfam-A alignments, Pfam-B alignments have not been manually checked for quality by a Pfam curator.

Listing 11.1

```
>> tree=gethmmtree('PF00002', 'type', 'seed');
>> phytreeviewer(tree)
```

end-listing-11.1

The result of the code listing above are illustrated in Fig. 11.5. The function *gethmmtree* downloads the *seed* sequences used for creating the HMM model stored in the Pfam database with the accession number of `PF00002`. The downloaded HMM aligned sequences are portrayed as a tree.

Further Readings

1. Durbin, R.: Biological sequence analysis: probabalistic models of proteins and nucleic acids. Cambridge University Press, Cambridge (1998)
2. Rabiner, L.: A tutorial on hidden Markov models and selected applications in speech recognition. Proc. of the IEEE 77, 257–286 (1989)
3. Kobayashi, H., Mark, B.L., Turin, W.: Probability, random processes, and statistical analysis. Cambridge University Press, Cambridge (2012)
4. Finn, R.D., Bateman, A., Clements, J., Coggill, P., Eberhardt, R.Y., Eddy, S.R., Heger, A., Hetherington, K., Holm, L., Mistry, J., Sonnhammer, E.L., Tate, J., Punta, M.: Pfam: the protein families database. Nucleic Acids Res. 42(Database issue), D222–D230 (2014)
5. Finn, R.D., Miller, B.L., Clements, J., Bateman, A.: ipfam: a database of protein family and domain interactions found in the protein data bank. Nucleic Acids Res. 42(Database issue), D364–D373 (2014)
6. Barber, D.: Bayesian reasoning and machine learning. Cambridge University Press, Cambridge (2011)
7. Zhang, Z., Wood, W.I.: A profile hidden Markov model for signal peptides generated by HMMER. Bioinformatics 19(2), 307–308 (2003)
8. David, K., David, H., Martin, G.R., Frank, H.E.: A generalized hidden markov model for the recognition of human genes in DNA. In: Proc. 4th Conf. on Intelligent Systems in Molecular Biology (1996)

11.7 Exercises

1. Consider the following MSA block:

```
-ATA-CCCT
ACTACGCCT
-ATTCG-GA
```

   Compute its profile. Recall that the first four rows are respectively the normalized frequencies of nucleotides, A, C, G and T, while the fifth row provides the frequency of the gap character.
2. Supposing we are given the following sequence fragments:

```
s1: ATCCGTTGAAGACCCGCCA
s2: ATTATGGACGCCA
s3: TAATTAAATTCGGTGGGGGC
s4: CGGAGGTTATTTACCTT
s5: ATTATCCAAAAAATGGACC
s6: AATAAATGGCCCGC
s7: TAATAAACCTGTCGAGTGCCT
```

   (a) Use MATLAB to compute a multiple sequence alignment and find a consensus sequence. Note that some sequences may need to be complemented for achieving the best MSA score. MATLAB function discussed in class *does not* automatically perform this step.
   (b) Using the best MSA obtained in your answer above to induce a hidden Markov model for the consensus sequence obtained. Evaluate the likelihood the following sequence is generated by the model. Again, you must complement the query sequence and take the better likelihood score of the original and complemented sequence.

```
tacaactcaagaggccct
```

3. Visit the pfam web site, `http://pfam.sanger.ac.uk/`, and answer the following questions:

   (a) Discuss the IT framework upon which the PFAM database and services are based.
   (b) What are clans in PFAM and what advantages do they offer in protein family representation?
   (c) What are sequence logos and how do they help in visualization of protein families?

Chapter 12
Gene Models

Genetics is gaining increasing significance as the discovery of new genes continues to have considerable impact in the medical sciences. One of the main scientific projects in genetics is the Human Genome Project, a multidisciplinary endeavor that aims to learn the identity of every single base stored in the human genome. The genome stores the blueprints for the synthesis of a variety of proteins – the macromolecules that enable an organism to be structurally and functionally viable. The blueprint or the program for the synthesis of a single protein is called a *gene*, a unit in the DNA sequence that is generally between 1×10^3–1×10^6 *bp* in length based upon the complexity of the protein that it *codes* for. A higher level eukaryote contains as many as 30,000-40,000 genes. It has been estimated that gene *coding* region only account for 2–5% of the genome. The gene identification problem is to recognize these regions from an anonymous sequence of DNA.

The earlier phases of genomic research focused on the construction of physical maps. However, the current emphasis has been shifting continually towards intensive sequencing. This has enabled us to study the structure and function of eukaryotic genes that may span tens or hundreds of kilobases. Only a small percentage of the total gene-span actually codes for proteins. This renders the detection of eukaryotic genes using the traditional approaches, i.e. those based on cDNA selection, exon trapping and the random cloning of cDNA, to be quite laborious for sequences that are larger than a few tens of kilobases. Consequently, the Genome Sequencing Centers routinely use computational approaches for exon prediction in addition to other means for detecting genes.

These approaches have been proposed by theoretical biology researchers since the early 1980s. These programs analyzed the DNA sequence and labeled a region to be potentially coding based upon its local codon usage, presence of ancient conserved patterns, or its significant deviations from the composition of a random sequence. As scientists learn about the genetic basis in the etiology of many diseases, the quest for discovering new genes using computational methods continues to be more significant than ever before.

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology*,
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_12

Newer programs that utilize a wide variety of *decision theoretic* and *machine learning* technologies are being developed. This review provides a comprehensive summary of some of these approaches and describes the methods that are being used today to establish a software's performance in gene prediction accuracy.

A compendium of the currently used software systems for the identification of genes in anonymous segments of DNA follows.

12.1 GRAIL

The GRAIL gene identification systems utilizes a neural network for the recognition of genes. The neural networks in GRAIL 1, GRAIL 1a, and GRAIL 2 are trained to combine the results from a number of coding predictors. GRAIL 1 has been in place for more than six years. Using a neural network which recognizes coding potential within a fixed size window of 100 bp, the coding potential is computed without taking into consideration the information about additional features such as splice junctions, etc. GRAIL 1a also uses fixed-length windows to first locate potential coding regions. It then evaluates a number of discrete candidates of different lengths around each potential coding region. The information from the two 60-base regions adjacent to that coding region are analyzed to find the correct boundaries for a coding region. GRAIL 2 uses variable-length windows for each potential open reading frame bounded by a pair of start/donor, acceptor/donor or acceptor/stop sites. Thus, GRAIL 2 uses genomic context information to score the coding regions and is therefore not appropriate for sequences where the regions adjacent to an exon are absent [1]. However, these changes have improved GRAIL 2's overall performance, particularly for short exons. All three systems have been trained to recognize coding regions in human DNA sequences, although they also work well on a number of other organisms, particularly other mammals.

GRAIL is accessible by several methods, including an e-mail server at Oak Ridge National Laboratory (ORNL), which processes DNA sequence(s) contained in e-mail messages. An interactive graphical X-based client-server system called Xgrail is also available from ORNL. Xgrail supports a wide range of analysis tools, including tools for gene modeling and database search.

12.2 MZEF

MZEF is an internal coding exon prediction program. It utilizes Quadratic Discriminant Analysis, or QDA, for the purpose of describing the distributions of exons and pseudo-exons. This method is an extension of the statistical pattern recognition ideas earlier presented in Linear Discriminant Analysis, or LDA, used in HEXON (FGENEH in GeneFinder is an improvement on

HEXON). In fitting a QDA, the surface that separates the distribution of exons and pseudo-exons can be more accurately approximated.

An overview of the algorithm is as follows: Each potential exon that matches the template of $AG \rightarrow ORF \rightarrow GT$ is analyzed. The exons that meet a minimum length criteria are next considered to be putative exons and must be separated from the pseudo-exons. The putative exons are represented using a nine value feature vector, comprised of parameters such as, exon length, branch score, and various differences between the hexamer frequency preferences on the two sides of the donor and acceptor splice sites. This nine dimensional feature vector x is categorized to be an exon or a pseudoexon based on the following log-ratio test derived from QDA:

$$\eta = log\frac{p_1}{p_2} = log\frac{p_1^0}{p_2^0} - \frac{\delta_1 - \delta_2}{2} - \frac{1}{2}log\frac{|\Sigma_1|}{|\Sigma_2|} \tag{12.1}$$

The parameters used in the above equation are as follows: μ_i and Σ_i represent the group mean and covariance matrices obtained from the training sets (the training set was comprised of 1879 true exons and 184,217 pseudoexons). The quantities p_1^0 and p_2^0 denote the prior probability for a putative exon for membership into the group G_1 of true exons, or to the group G_2 of the pseudo-exons. The quantity, $\delta_i = (x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i)$ is the squared Mahalanobis distance between the observed feature vector x and μ_i; $|\Sigma_i|$ is the determinant of the covariance matrix Σ_i.

12.3 GENSCAN

GENSCAN works by building a probabilistic model of the gene structure of human genomic sequences and applying this model to gene prediction. The probabilistic model of a gene includes the specific compositional and functional units of a eukaryotic gene, including exons, introns, splice sites, promoters, and the polyadenylation signals. The occurrence of a partial set of these units and the representation of a partial gene is supported by the implementation of the model search algorithm. Furthermore, the predictions made by the program are not a mere reflection of the *types* of genes that are found in the protein databanks, but instead an independent evaluation that provides information that complements our existing knowledge.

The modeling of a DNA sequence by GENSCAN is based on a Generalized Hidden Markov Model (GHMM), that uses a double stranded DNA sequence and can find the occurrence of multiple genes in a single sequence, on either one or both DNA strands. A simplistic representation of such a HMM is shown in Fig. 12.1, where the arcs represent a set of nucleotides belonging to a class and the nodes represent the DNA transition regions from one class of nucleotides to another. The program's ability to model functional signals and their interrelationships in a natural manner using Maximal Dependence Decomposition is instrumental in providing it the strength needed for a

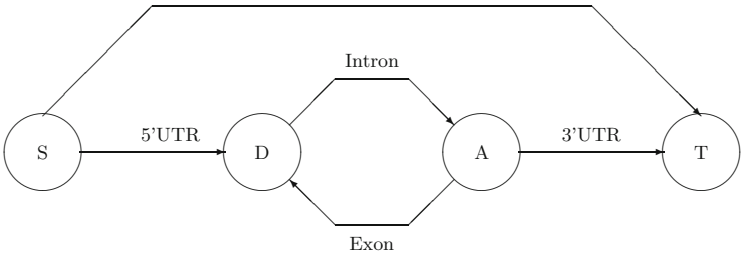


Fig. 12.1 A simple Hidden Markov Model for a multiple exon gene. The arcs represent the DNA sequence belonging to a functional unit of the gene, while the nodes represent the occurrence of specific signals that transition the functional state or the sequence class that the model is scanning from 5'→3'. The nodes S and T represent the start and terminating codons respectively, while the nodes D and A denote the donor and acceptor sites. The single arc from S to T enables the HMM to recognize a single exon gene. Such a model can learn the probabilities of the various *state* transitions using a set of DNA sequences representing genes.

generalized gene detection task. The text output of the program is a list of one or more (or possibly zero) predicted genes and peptide sequences, while the graphical outputs provide a representation of the relative locations of the predicted exons. Versions of the program that are suitable for vertebrate, maize and *Arabidopsis* sequences are currently available[1] [2].

12.4 VEIL and GENIE

VEIL (the Viterbi Exon-Intron Locator) is based on the observation that Hidden Markov Models (HMMs) provide a precise probabilistic method for modeling sequences of discrete data. Consequently, it uses a custom-designed HMM to segment uncharacterized genomic DNA sequences into exons, introns, and intergenic regions. The exon-HMM module is designed to capture the regularities in codon usage and periodicity that appear in exons as well as to rule out in-frame stop codons. A similar module represents the intron-HMM. The HMM models for the probabilistic representation of splice sites resemble a *pipeline*, since these signals are of a well defined length. For example, the donor acceptor site models in VEIL are comprised of 9 and 15 stages respectively. Other HMMs include those for the start codon, the polyadenylation signal AAATAA, and intergenic regions that are upstream of a start codon. These simpler models were put together into the overall gene model with the schematic similar to the one shown in Fig. 12.1. The final HMM is comprised of 241 states and 1003 edges. The representation of a *gene* in GENSCAN and VEIL and Genie are quite similar – with the differences being manifested in the representation of the individual segments of a gene [3].

[1] The vertebrate version works well on *Drosophila* sequences.

After the determination of these regions, the Viterbi algorithm is used for parsing the query sequence into its component exons and introns. Such an algorithm is based on dynamic programming techniques where the most likely set of states that a given sequence is expected to traverse are determined. In addition, the probability that the model will produce a given sequence is computed by the Viterbi algorithm. This represents the probability that a given DNA sequence contains a gene. Glimmer is a corresponding program designed with a similar idea of an interpolated hidden Markov model and is applicable for analysis of microbial genomes [4].

A Generalized Hidden Markov Model (GHMM) is general enough to enable the generation of subsequences from DNA symbols at each state of a hidden Markov chain. The model required to produce a subsequence at each state of an GHMM (as shown in Fig. 12.1) can be quite complex and in turn be another HMM. In Genie, each component is designed and trained independently and combined into a modular system. The length distributions of introns and exons in the training set are used to learn the average length (and variance) for a string generated by a state in GHMM. The donor and acceptor sites are recognized by a neural network that uses a 15 bp window for donor and a 41 bp window for acceptor sites. Thus, the nodes representing the Donor and Acceptor sites in Fig. 12.1 have a neural network embedded in them that returns the posterior probabilities for a given position to be a donor or acceptor site. After the construction of such a model, it is trained to learn the probabilities of transitions between states in the GHMM and for the generation of each nucleotide base given a particular state. Machine learning techniques are applied to optimize these probabilities using a standardized gene data set [5].

12.5 Morgan

MORGAN (Multi-frame Optimal Rule-based Gene Analyzer) distinguishes itself from the other systems for finding genes using a *decision tree* classifier, a technique that is derived from the statistics community. Decision trees are often utilized for a variety of classification tasks, such as cancer diagnosis, speech understanding, image understanding, and optical character recognition. Decision trees are often applied to objects represented in terms of their *features*. For example, the representation of an object in a *d-dimensional* feature space may be denoted as $f_1, f_2 \ldots, f_d$. Subsequently, knowledge about the classification process is embedded into a tree like structure and by performing a series of tests, generally of the form $f_k < T$, the identity of an unknown object is established. Thus, each question node in the decision tree corresponds to a linear discriminant, and helps in partitioning the search space into a set of compartments or leaves which individually represent an entity that we are interested in classifying. Thus, there were two partitions

created by MORGAN in gene classification. These are C for coding and N for non-coding.

The MORGAN system was build using a 19-feature set. These included features that measure the longest ORF, the dicodon usage, two hexamer frequencies, codon usage, position asymmetries for {A,C,T,G} and Fourier coefficients for periods 2–9. The decision tree was trained on 290,628 examples, with each example comprising of 54 bps of human DNA sequence. Using these examples, and knowing their classification *a-priori*, the MORGAN decision tree was trained. It resulted in 20 *leaf* and 19 *test* or intermediate nodes. For a given DNA sequence that needs to be classified as coding or non-coding, the 19-features are first computed for their subsequences. The optimal segmentation is dependent on a separate scoring function that takes a subsequence and assigns to it a score reflecting the probability that the sequence is an exon. Each subsequence is scored by a decision tree and the individual scores are combined to give a probability estimate [6][7].

12.6 GeneFinder (FGENEH)

The GeneFinder is a suite of tools available from Baylor College of Medicine. The tools that are of interest from a standpoint of gene identification are, *fgeneh* for prediction of gene structure, *fexh* for 5', internal and 3' exon prediction, *hspl* for splice site prediction, and *hexon* for prediction of internal exons in human DNA.

The algorithm used in *fgeneh* begins by predicting all possible potential internal exons, and potential 5'- and 3'-exons. Each exon is described using a linear discriminant function that combines the contextual features of these exons. Next, the dynamic programming method is used to search for the optimal combination of these exons to produce a gene model. The algorithm used in *hexon* is also based on the discriminant analysis of open reading frames flanked by GT and AG base pairs. Prediction is performed by a linear discriminant function that combines characteristics of the donor and acceptor splice sites, 5'- and 3'-intron regions and the properties of the coding region of the open reading frame. This program can only predict internal exons with GT and AG conserved base pairs for donor and acceptor splice sites[2] [8].

12.7 GeneParser and GeneLang

The GeneParser program looks for gene specific features in the given sequence and subsequently applies *dynamic programming* (DP) to form their combinations in order to obtain a configuration that maximizes a likelihood function. Specifically, the anonymous DNA sequence is scanned for finding the locations of splice sites as well as the computation of content parameters such as codon

[2] However, this usually includes more than 99% of the all authentic splice sites.

usage, local compositional complexity, 6-tuple frequency, length distribution and periodic asymmetry. The program scores all subintervals in a sequence for content statistics indicative of introns, exons, and their boundaries. The content statistics are fed into a neural network that provides a log-likelihood estimate that the given subinterval exactly represents an intron, first exon, internal exon or the last exon (Weights for the feed-forward neural network are optimized to maximize the number of correct predictions). A dynamic programming algorithm is then applied to this classification of each subinterval so that the overall likelihood of a gene model is maximized. The DP algorithm operates under the constraints that introns and exons must be adjacent and non-overlapping. The highest scoring combination amongst the groups of high scoring combinations represents the maximally likely model of a gene [9].

Similar to GeneParser, GeneLang is a syntactic pattern recognition system, which uses the tools and techniques of computational linguistics to find genes and other higher-order features in biological sequence data. For example, formal language theory uses a set of rules, called *grammar*, to define the valid sets of strings over a given alphabet. The motivation for building a gene grammar is the availability of *parsers* that can recognize if the input string satisfies the rules specified by a *gene* grammar. Thus, in GeneLang, the patterns over the DNA alphabet are described using a set of rules and a general purpose parser, implemented in the logic programming language Prolog [10]. Thus, the system treats components of a gene such as donor and acceptor sites, introns and exons, start and stop codons, etc. as words that are formed on the four character DNA alphabet. The gene is next in the level of this syntactic hierarchy, i.e. the gene is a valid *sentence* formed by these words. Genes in a DNA sequence can be recognized in a manner analogous to the recognition of grammatically correct sentences in the English language.

12.8 AAT: Analysis and Annotation Tool

This tool is used for identifying genes in a DNA sequence by comparing the sequence against protein and cDNA sequence databases. The analysis and annotation tool (AAT) includes two pairs of programs, with each pair comprised of a database search and an alignment program. The first program pair is designed to compare the query sequence to the *protein* database, while the second pair performs a similar comparison against cDNA databases. Furthermore, the alignment programs construct a consensus of all sequence-database alignments into a multiple sequence alignment to enhance the prediction of splice junctions. Finally, sequence alignments that score low are filtered out from the results and the final Protein and cDNA alignments are combined and presented to the users.

The first program pair compares a query DNA sequence against a protein database using two programs called DPS and NAP. The DPS program is

used for computing high scoring chains of segment pairs between the query DNA sequence and a protein database. The global alignment program NAP finds the optimal alignment between a DNA sequence and the corresponding protein sequence. The alignment model for NAP accommodates introns and frameshifts within codons, and is thus able to identify the exact locations of introns using the (GT) and (AG) consensus for splice site identification. The second program pair, comprised of DDS and GAP, is used for comparing the query DNA sequence against a cDNA library. The DDS program is an improvement over the BLASTN program. The GAP program is a global alignment program that is sufficiently powerful for aligning a DNA sequence containing introns to a cDNA sequence.

One of the goals of AAT is to aid in an automatic annotation of DNA sequences. This task has traditionally been done manually, where the alignments between the coding regions of a DNA sequence and the existing proteins is established by BLASTX and linked to the sequence as an annotation in a post-hoc manner. This helps in providing a clue for the functional significance of a given gene as is evident in the function of the related protein sequence. The AAT, on the another hand, performs such an *alignment* and is able to display it as the *basis* for predicting genes. Furthermore, the alignment produced by BLASTX is prone to frameshift errors. This shortcoming is overcome by AAT by the development of a customized program for DNA-protein sequence alignment.

12.9 Comparison of Gene Finding Algorithms

In order to compare the performance of gene finding systems, performance comparison metrics were defined and a dataset was created by Burset and Guigo [11]. This dataset is comprised of sequences from GenBank release 85.0. The process by which this dataset was constructed is as follows.

First, Burset and Guigo collected vertebrate protein coding sequences from GenBank. Next, in a series of quality control steps, they removed all entries with pseudo-genes, with in-frame stop codons, with no introns (essentially the cDNA sequences), and with non-standard slice junctions (i.e. those that did not have the GT and AG at the beginning and end of an intron). They further removed the immunoglobulins and histocompatibility antigens, and were finally left with 570 complete sequences, each with exactly one gene and and at least one intron. There are a total of 2649 exons and 2079 introns in this set.

12.9.1 Performance Parameters

The parameters described below are utilized for the computation of accuracy statistics at the nucleotide level. Each nucleotide in the sequence being analyzed is classified as predicted positive (PP) if it is in a predicted coding

region, or predicted negative (PN) otherwise. The nucleotide's actual positive (AP) or actual negative (AN) value is also known according to the sequence annotation. These assignments are then compared to calculate the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). The sensitivity and specificity of the prediction porgram is given by Eq. 12.2 and 12.3.

$$Sensitivity: \quad Sn = \frac{TP}{AP} \tag{12.2}$$

$$Specificity: \quad Sp = \frac{TP}{PP} \tag{12.3}$$

and Approximate Correlation, AC, is defined by Eq. 12.4.

$$AC = \frac{\frac{TP}{TP+FN} + \frac{TP}{TP+FP} + \frac{TN}{TN+FP} + \frac{TN}{TN+FN}}{2} - 1 \tag{12.4}$$

At the exon level, predicted exons (PE) are compared to annotated exons (AE). True exons (TE) is the number of predicted exons which are exactly identical to an annotated exon (i.e. both endpoints correct). The sensitivity and specificity at the exon level is given by Eq. 12.5 and 12.6 respectively.

$$Sensitivity: \quad S_n = \frac{TE}{AE} \tag{12.5}$$

$$Specificity: \quad S_p = \frac{TE}{PE} \tag{12.6}$$

The average of S_n and S_p is typically used as an overall measure of accuracy at the exon level in lieu of a correlation measure. Two additional accuracy measures are also calculated at the exon level:

- Missing Exons (ME), the fraction of annotated exons not overlapped by any predicted exon;
- and Wrong Exons (WE), the fraction of predicted exons not overlapped by any true exon.

Accuracy measures for a set of sequences are calculated by averaging the values obtained for each sequence separately, the average being taken over all sequences for which the measure is defined.

12.9.2 Performance Results

The performance of the various tools discussed in this review on the above dataset is shown in Table 12.1. The presentation was ordered in the tools performance as evident in the *Sn* parameter for the correctly predicted nucleotides. However, it may sometimes be more realistic to compare a tool's

performance based on $\frac{Sn+Sp}{2}$ for the predicted exons. In such a comparison, MZEF, GENSCAN and AAT are the three clear winners. The prediction of human protein-coding genes in newly sequenced DNA becomes very important in large genome sequencing projects. Many of the systems continue to be developed in this field with the goal of increasing their reliability when identifying genes in DNA . Although the systems developed so far are not completely accurate, the results that they provide are vital to keep pace with the rapid analysis of sequence data in this age of high throughput genomic sequencing. Some of the problems that are faced so far stem from the inability to accurately predict the intron/exon boundaries, which in turn results in an inability to identify the eukaryotic gene structure and a poor performance on most short exons. Furthermore, a large number of false splice site predictions eventually lowers the reliability of predicted exons. Due to the different types of processing performed by the various gene identification systems, it therefore seems plausible to look at their results in a combined manner and look for a region of coding consensus produced by the various gene identification systems evaluated in Table 12.1. Furthermore, understanding the methodology adopted for a given method will be vital to explain the results produced by that method.

Table 12.1 Accuracy of the various gene identification systems on the Burset and Guigo [11] data set comprised of 570 vertebrate sequences

Method	Predicted Nucleotides			Predicted Exons		
	Sn	Sp	AC	Sn	Sp	$\frac{Sn+Sp}{2}$
AAT	0.94	0.97	0.95	0.74	0.78	0.76
GENSCAN	0.93	0.93	0.91	0.78	0.81	0.80
MZEF	0.88	0.95	0.90	0.84	0.92	0.88
VEIL	0.83	0.72	0.73	0.53	0.49	0.51
MORGAN	0.82	0.80	0.78	0.58	0.54	0.56
Genie	0.78	0.84	0.77	0.61	0.64	0.63
GeneFinder	0.77	0.85	0.78	0.61	0.61	0.61
GeneID	0.63	0.81	0.67	0.44	0.45	0.45
GeneParser2	0.66	0.79	0.66	0.35	0.39	0.37
GeneLang	0.72	0.84	0.75	0.50	0.49	0.50
GRAIL-II	0.72	0.84	0.75	0.36	0.41	0.38
Xpound	0.61	0.82	0.68	0.15	0.17	0.16

12.10 MATLAB Functions for Finding Genes

Although at the time of writing, MATLAB does not include a gene detection algorithm per se, the following set of functions in MATLAB are related to the general topic of searching for coding regions.

Function	Description
basecount	Counts the number of nucleotides in a sequence
codoncount	Counts the number of codons in a sequence
ntdensity	Displays the nucleotide density - (A+T) & (G+C)
seqshoworfs	Displays the open reading frames in three forward frames

Further Readings

1. Xu, Y., Uberbacher, E.: Automated gene identification in large-scale genomic sequences. J. Comp. Biol. 4, 325–338 (1997)
2. Burge, C., Karlin, S.: Prediction of complete gene structures in human genomic DNA. J. Mol. Biol. 268, 78–94 (1997)
3. Henderson, J., Salzberg, S., Fasman, K.: Finding genes in human dna with a hidden markov model. J. Comp. Biol. 4, 127–141 (1997)
4. Salzberg, S., Delcher, A., Kasif, S., White, O.: Microbial gene identification using interpolated markov models. Nucleic Acid Res. 26, 544–548 (1998)
5. David, K., David, H., Martin, G.R., Frank, H.E.: A generalized hidden markov model for the recognition of human genes in DNA. In: Proc. 4th Conf. on Intelligent Systems in Molecular Biology (1996)
6. Salzberg, S., et al.: Locating protein coding regions in human DNA using a decision tree algorithm. J. Comp. Biol. 2, 473–485 (1995)
7. Salzberg, S., Delcher, A., Fasman, K., Henderson, J.: A decision tree system for finding genes in DNA (1997)
8. Solovyev, V.V., Salamov, A.A.: Infogene: a database of known gene structures and predicted genes and proteins in sequences of genome sequencing projects. Nucleic Acids Res. 27(1), 248–250 (1999)
9. Snyder, E.E., Stormo, G.D.: Identification of coding regions in genomic DNA. J. Mol. Biol. 248, 1–18 (1995)
10. Dong, S., Searls, D.: Gene structure prediction by linguistic methods. Genomics 23, 540–551 (1994)
11. Burset, M., Guigo, R.: Evalation of gene structure prediction programs. Genomics 34, 353–367 (1996)
12. Snyder, E.E., Stormo, G.D.: Identification of protein coding regions in genomic DNA. J. Mol. Biol. 248(1), 1–18 (1995)
13. Solovyev, V., Salamov, A.: The gene-finder computer tools for analysis of human and model organisms genome sequences. In: Proc. Int. Conf. Intell. Syst. Mol. Biol., vol. 5, pp. 294–302 (1997)
14. Salzberg, S., Delcher, A.L., Fasman, K.H., Henderson, J.: A decision tree system for finding genes in DNA. J. Comput. Biol. 5(4), 667–680 (1998)
15. Zhang, C.-T., Zhang, R.: Evaluation of gene-finding algorithms by a content-balancing accuracy index. J. Biomol. Struct. Dyn. 19(6), 1045–1052 (2002)
16. Zhang, M.Q.: Using mzef to find internal coding exons. Curr. Protoc Bioinformatics, Chapter 4:Unit 4.2 (August 2002)

Part IV
Phylogenetics and Systems Biology

Chapter 13
Introduction to Phylogenetic Reconstruction

Phylogenetics, the study of evolutionary relationships in organisms, is one part of the larger field of systematics, which also includes taxonomy. The term **taxonomy** connotes the process and methodology for the naming and classification of organisms. The context of evolutionary biology is phylogeny, the connections between all groups of organisms as understood by ancestor/descendant relationships. The molecular mechanisms of organisms studied strongly suggests that all organisms on earth have a common ancestor. Thus, the species are related to each other by the virtue of having evolved from the same common (now extinct) ancestor. Such a relationship of species is called *phylogeny* and it's graphical representation is called a *phylogenetic tree*.

Computational methods infer these relationships from currently thriving species and reconstruct what their course of evolution might have been. The phylogenetic tree construction help us go back in time and develop a "hypothesis" of how life evolved from the single common ancestor. This hypothesis (a phylogenetic tree) is represented as a cladogram, a branching diagram. Cladograms bear a lot in common with the notion of family trees. In a family tree we trace back our ancestry. For example, in the family tree on the right, the ancestors of all the rest of the family are the initial black dot and yellow square. These ancestors give rise to three children, one of which mates and has two children. We can all trace our lineage back to one set of ancestors.

All species have ancestors too. So, for example, sometime in the past an ancestral species (father) of Homo sapiens walked the earth. This ancestor went extinct (died), but left descendant species (children). In family trees, we can talk coherently about real ancestors. In biology, the ancestors are often gone sometimes without a trace. All we have left are the children. Also unlike family trees, new species form from the splitting of old species, and the formation of the two descendant species is called a splitting event. The ancestor is usually assumed to "die" after the splitting event.

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_13

13.1 Terminology

The nodes of the tree in a cladogram are marked. The stems of the tree end with the taxa under consideration. Each node marks a splitting event. The node therefore represents the end of the ancestral taxon, while the stems represent the species that split from the ancestor. The two taxa that split from the node are called *sister taxas*. They are called sister taxa because they are like the siblings from the parent or ancestor in a family tree. The sister taxa must each be more closely related to one another than to any other group because they share a close common ancestor.

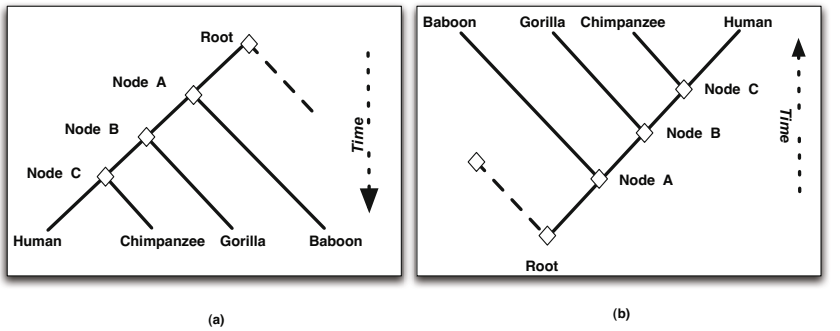


Fig. 13.1 Implicit in the representation of a phylogenetic tree is the passage of evolutionary time shown along the vertical axis. Two commonly utilized representations of the the phylogenetic tree are shown above. In (a) the passage of time begins with the one common ancestor at the top of the tree and continues down to the taxons currently in existence, namely, the species human, chimpanzee, gorilla and baboon. In an equivalent representation shown in (b) the passage of evolutionary time is from the bottom to the top. The time axis is seldom labeled on the tree as it evident from the series of speciation events resulting in the number of species becoming larger with the passage of time.

Two equivalent representations for phylogenetic trees are shown in Figure 13.1. The most closely related species in the cladogram are *humans* and *chimpanzees*. Their common ancestor is represented by *Node C*. At the node, the ancestor goes extinct but leaves two siblings hypothesized to be humans and chimpanzees. The humans and chimpanzees are known as **sister taxas** and are more closely related to each other than to any other taxa on the tree. Working through the tree we come to *Node B* – a node where the ancestor for *human* and *chimpanzees* split from *gorillas*. The *gorilla* is a sister taxa to the *human* and *chimpanzee* ancestor. Similarly, the common ancestor of the now extinct species represented by *Node B* and the *baboon* is another extinct species denoted by *Node A*.

In practice, one does not label the internal ancestor as the phylogenetic tree is usually derived from the known taxas which are labeled on the leaf

nodes. Sometimes, when the common ancestor has been identified through fossil remains, an ancestral node may also be labeled. An ancestor plus all its descendants is called a **clade**. A cladogram thus shows us the hypothesized *clades*.

13.1.1 Tree Representation Formats

Trees are often graphically represented and drawn in a two-dimensional space. For example, such a representation for a 6-taxa tree is shown in Figure 13.2(a). Individual taxons (a, b, c, d, e, f) are shown as the leaves of the tree and their distance from their parents are marked on the edges. For example, the taxon-*a* has diverged 2 units from its common parent that it shares with taxon-*b* which has diverged 2.5 units from the same common parent. The units of evolutionary distance is *years*. However, in molecular phylogenetic tree reconstruction methods, often the divergence of a species is estimated by the divergence of a *gene* family. In these instances, which anyway is our focus, the distance annotated on the edge could be a measure or a metric produced by a sequence comparison program.

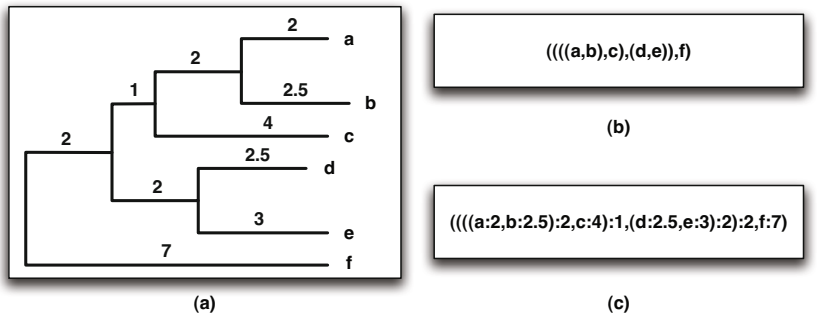


Fig. 13.2 (a) Topology of a tree with edge lengths shown. (b) Textual representation or symbolic expression of the tree without the edge lengths – only topological information is encoded. (c) Symbolic expression for the phylogenetic tree including the encoding of the edge lengths.

As shown in Figure 13.2(b), the tree topology may be represented using a text representation or a **symbolic expression**. The level of nesting of the parenthesis in this representation is equal to the depth of the leaf node. For example, the taxon-*a* and taxon-*b* are at a depth of 4, and they are also at the fourth level of parenthesis nesting in the symbolic representation.

The symbolic representation of tree may also be extended to include the edge length, as shown in Figure 13.2(a).

13.2 Types of Trees

13.2.1 Unrooted and Rooted Trees

Phylogenetic relationships between the genes of organisms are used to form (a) unrooted, or (b) rooted trees. The branching pattern in either case is called the *topology* of the tree for a given number of *taxa*. A *taxa* is defined to be any kind of taxonomic unit such as families, species or DNA sequences. The true biological phylogeny has a "root" – the ultimate ancestor of all the taxas and their ancestors. While some algorithms that construct phylogenetic tree provide information on the what might be the root, others (notably parsimony and probabilistic methods) do not yield any information pertaining to the location of the root.

Figure 13.3 presents both a rooted and an unrooted tree corresponding to a cladogram where ancestors of individual taxas are correspondingly defined. In the rooted tree, the root is generally drawn at the top and all of the taxa is drawn as terminal nodes, or leaves, at the base of the tree. The unrooted tree is a branching structure with no clear top to bottom hierarchy. The internal nodes in an unrooted tree represent the ancestors of the taxa which terminate the branches. The complete definition of a tree includes the edge length along with the topology of the branching patterns.

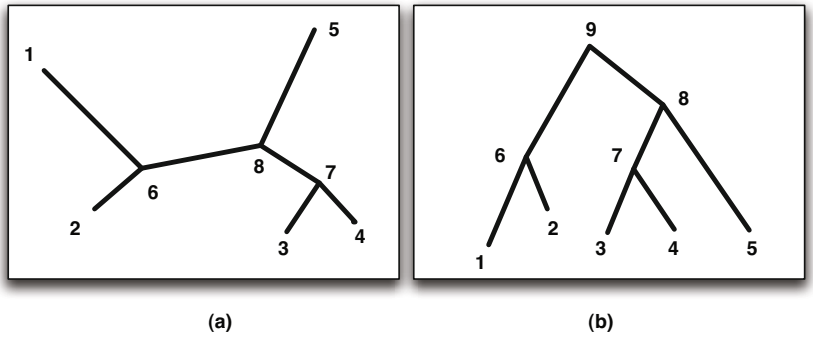


Fig. 13.3 (a) An Unrooted and (b) a Rooted tree. Note that the number of nodes in the unrooted tree is one less than the number of nodes in the rooted tree for the same number of taxas (five in this example shown). This is because the unrooted trees do not designate a root node – the node representing the one common ancestor to all the species in the phylogenetic tree.

13.2.2 Orthologues and Paralogues

Scientists are often interested in studying the phylogenetic tree that represents the evolutionary history of a group of species or populations. In this

type of tree, the time of divergence between two species refers to the duration of time that the two species have been reproductively isolated. However when the phylogenetic tree is constructed based on one gene from each species, the tree obtained does not necessarily agree with the species tree. This is observed because of the presence of polymorphic alleles at a given gene locus, which results in the expected time of divergence of genes sampled from different species to be longer than the time for the divergence of species themselves. Thus we can expect the branching patterns obtained from the phylogenetic tree constructed from the gene, often referred to as the **gene tree**, to be different from the branching pattern in species phylogenetic tree.

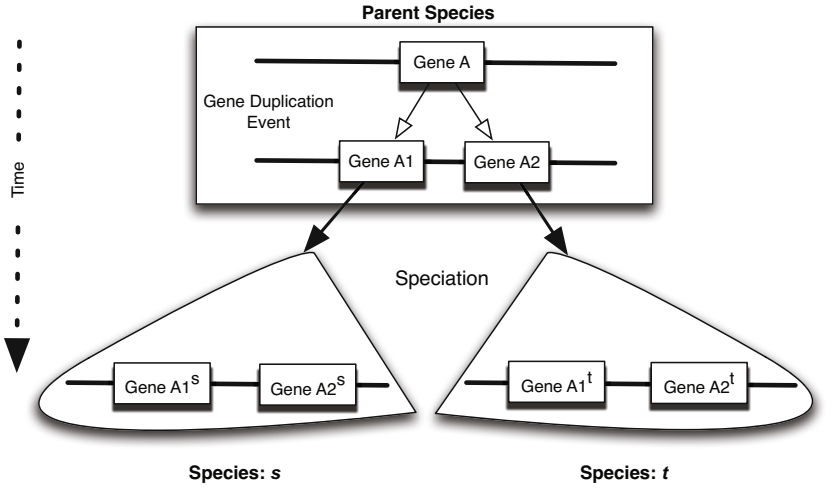


Fig. 13.4 Gene is duplicated in the parent species prior to the speciation. Genes A1 and A2 are paralogs. As the species evolve and two new species s and t are formed, the genes $A1^s$ and $A1^t$ as well as $A2^s$ and $A2^t$ are orthologs. The pairs of genes $A1^s$ and $A2^s$, or $A1^t$ and $A2^t$, or $A1^s$ and $A2^t$, or $A2^s$ and $A1^t$ are also paralogs.

Another problem occurs when the gene studied belongs to a multigene family. For example, consider the situation shown in Figure 13.4. Two related species s and t have evolved from their ancestor where the duplication event occurred before the divergence of the two species. The result of the gene duplication produced genes $A1$ and $A2$. These are known as the **paralogs**. After the speciation event, the genes in the divergent species s and t have been denoted as ($A1^s$, $A2^s$) and ($A1^t$, $A2^t$). In the species s and t, gene pairs ($A1^s$, $A1^t$) and ($A2^s$, $A2^t$) are known as the **orthologues**. The gene pairs ($A1^s$, $A2^s$), ($A1^t$, $A2^t$), ($A1^s$, $A2^t$) and ($A2^s$, $A1^t$) are known as the **paralogues**.

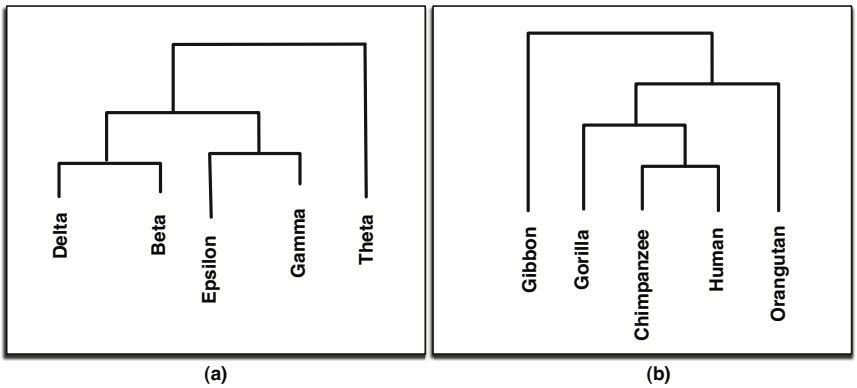


Fig. 13.5 (a) A phylogenetic tree constructed using paralog genes, and (b) A phylogenetic tree construction using ortholog genes. Both these trees are **gene trees**. The gene tree in (a) does not correspond to a species tree but represents the history of duplication events leading to the evolution of the beta-, delta-, epsilon-, gamma- and theta-chains of the human haemoglobin gene family. In contrast, the gene tree in (b) is developed with a kilobase fragment of mitochondrial DNA from five primate species. Here, the diversity of the mitochondrial genes are representative of the evolutionary history of the primate species themselves.

Scientist take great care to use the appropriate type of genes for the construction of appropriate tree. For example, if one is interested in studying the evolutionary history of gene duplication events, as shown in Figure 13.5(a), the paralogues from a single specie(s) will be used in the phylogenetic tree construction process. On the other hand, if the objective is to develop a evolutionary tree of the species, such as the one shown in Figure 13.5(b), one must utilize orthologues for the tree construction algorithm. In practice, the distinction between the orthologous and paralogous genes is not always easy, particularly when there are many copies of the duplicate genes in the genome. Therefore, great care must be taken to infer a species tree from a gene tree.

13.3 Counting Phylogenetic Trees

This section utilizes basic combinatorics to develop a formulation for the total number of possible rooted and unrooted trees that may be constructed given a number of taxas, n. Basic formulations for the total number of vertices and edges is used to inductively develop the formulation for the total number of trees.

Theorem 1. *Every rooted phylogenetic tree with n-taxas is an acyclic graph with (2n-1) vertices and (2n-2) edges.*

Proof: Let us begin by counting the number of vertices in a phylogenetic tree with n taxas (i.e. leaf nodes). Intuitively, we can begin the construction of the tree by linking the two closest sister taxas and denoting them as having been evolved from the same common ancestor. Thus, we add one vertex for the common ancestor in the process of linking the two taxa leaves *(n-2)*. The addition of one parent vertex results in reducing the problem size to the construction of a tree with $(n-2+1) = (n-1)$ vertices. Continuing further, the addition of each parent vertex will result in the reduction of the problem size by 1. Assuming that a rooted phylogeny is being constructed, the final problem size is 1, corresponding to the one root vertex (the single common ancestor) that is left when this process is complete. This requires the addition of $(n-1)$ parent vertices. Consequently, the total number of vertices in a rooted phylogenetic tree is $(n+(n-1))$ or $(2n-1)$.

As for the number of edges in the tree we can reason as follows. Since each of the $(n-1)$ steps in the construction process results in the addition of 2 edges, the total number of edges in a rooted tree is $2(n-1)$ or $(2n-2)$. Alternatively, this result may be arrived at by observing that the number of edges in a tree is one less than the number of vertices. A tree with two vertices has a single edge. A tree with three vertices can only have two edges as a tree may not have any cycles. A rooted phylogenetic tree is a therefore an acyclic graph $G_r = (2n-1, 2n-2)$.

Theorem 2. *Every unrooted phylogenetic tree with n-taxas is an acyclic graph with (2n-2) vertices and (2n-3) edges.*

Proof: The proof for an unrooted tree is very similar to the proof for a rooted tree. In an unrooted tree, the construction process is terminated when there are 2 vertices remaining which are then connected with an edge, yielding the final tree. Since there are $(n-2)$ parent nodes added, the total number of vertices in the final tree, which includes the n taxas is $(n+(n-2))$ or $(2n-2)$. Correspondingly, there are are $(2n-3)$ edges that are utilized in the construction of an unrooted tree. An unrooted tree is therefore an acyclic graph $G_u = (2n-2, 2n-3)$.

With that basic background, the task of counting the total number of possible topologies given the number of taxas to be n is considerably simplified. Once again let us first consider the number of possible topologies of rooted trees. The number of rooted trees possible with $n = 2$ is 1. This is shown in Figure 13.6(a). A given two-taxa tree may be extended to a three-taxa tree by adding the third taxon in one of two ways. Firstly, the third taxon could be linked to any of the existing edges of the two-taxa tree. Secondly, the third taxon could directly evolve from the one common ancestor – the root node. Since the two-taxa tree has two edges, the total number of three-taxa tree topologies possible is 3, as shown in Figure 13.6(b). The third taxon C evolves from a common ancestor of A or B, or it evolves from an earlier common ancestor from which an ancestor of both A and B evolved. Continuing a similar analysis, for the total number of topologies possible for four-taxas

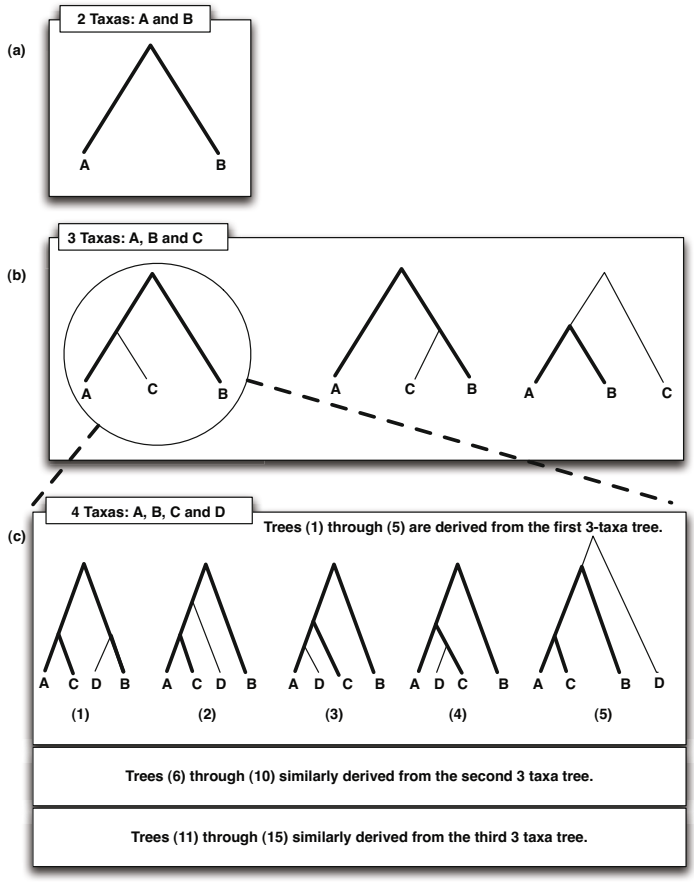


Fig. 13.6 Given two taxons, A and B, the total number of rooted trees possible is "1" as shown in (a). The third taxon in (b) may be added from any of the two branches of the two node tree or may be directly linked to a new root node. This yields three possible rooted trees with three taxons A, B and C. Each of the three-taxon trees further yields 5 trees, each resulting in a total of 3×5 or 15 possible trees with four taxons A, B, C and D as shown in (c) . In general the number of rooted trees possible with n taxons in $(2n\text{-}3)!!$.

we observe from Figure 13.6(c) that each of the three-taxa topologies may be extended by five different ways, yielding the total count of rooted tree topologies for four-taxas to be (3×5) or 15. The process of counting phylogenetic trees can be inductively generalized as follows:

Theorem 3. *The total number of possible topologies for rooted phylogenetic tree with n-taxas is* $(2n-3) \times (2n-5) \times \ldots \times 1 =$ *(2n-3)!!.*

Proof: The correctness of this formula can be shown by induction. Let us denote the count number of possible trees with i taxas to be R_i.

Base case: For three taxas $i = 3$, the number of trees possible is $R_3 = 3$. The formula $3!! = 3$ is correct.
Basis: Assume the formula is correct for a number of taxas = *(n-1)*. Therefore, the number rooted tree topologies for $(n-1)$ taxas is $2(n-1)-3 \times 2(n-1)-5 \ldots 1$
i.e. the number of tree rooted topologies is $R_{n-1} = (2n-5) \times (2n-7) \times \ldots 1$.

Inductive Step: Based on theorem 1, each of the possible *(n-1)*-taxa tree has $(2(n-1)-2)$ or $(2n-4)$ edges. The n^{th} taxon can thus branch off from any of these edges. Alternatively, the n^{th} taxon could be a direct descendant of a *new* common ancestor designated as the new root node. This gives us $(2n-4+1)$ or $(2n-3)$ ways of extending each of the *(n-1)*-taxa trees. Thus, the total number of n-taxa trees is:
$R_n = (2n-3) \times R_{n-1}$
$R_n = (2n-3) \times (2n-5) \times \ldots \times 1 =$ (2n-3)!!.
This concludes the proof. The formula works for the base case. Assuming the formula works for the basis case of $(n-1)$, the formula is shown to work for n in the inductive step. Therefore the formula is correct for $n \geq 3$.

Theorem 4. *The total number of possible topologies unrooted phylogenetic tree with n-taxas is* $(2n-5) \times (2n-7) \times ... \times 1$ *= (2n-5)!!.*

Proof: The proof for this theorem is by inspection. Let us denote the count number of possible trees with i taxas to be U_i.

According to theorem 2, an unrooted tree with n-taxas has $(2n-3)$ edges. Each of these edges may be the site where a root of the rooted tree is inserted. The number of rooted trees is therefore $(2n-3)$ times the number of unrooted trees. This is illustrated in Figure 13.7. This completes the proof.

13.4 Comparing Phylogenetic Trees

Often reconstructed trees need to be compared to measure the extent of topological differences between them. The distance between two unrooted trees is defined using a **topological distance**. This method is based on the comparison of partitions in the two trees. A partition is created by removing an internal edge of an unrooted tree. In general, the method uses the notion of tree partitioning obtained by removing an internal edge from the two trees being compared. Figure 13.8 shows the partitions formed by removing the internal edges of a 6-taxa bifurcating unrooted tree. For a bifurcating tree, which is different from a multifurcating tree, where more than two edges may emanate from a vertex, the number of partitions with n sequences as the leaves is $(n-3)$. This is evident from the observation that there are $(n-2)$ internal nodes in an unrooted tree which in turn must be connected with $(n-3)$ edges. Thus for a bifurcating tree with n-taxa, there are $(n-3)$ possible partitions.

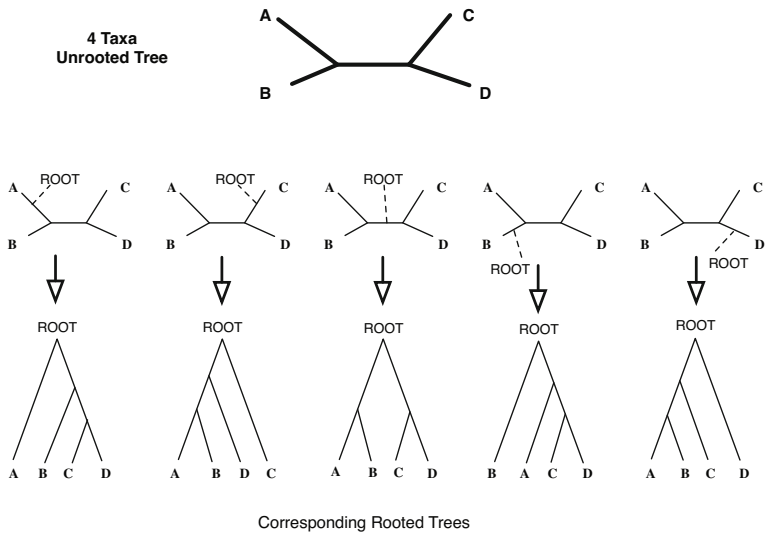
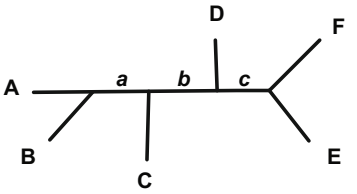


Fig. 13.7 Each n-taxa unrooted tree may be transformed to $(2n-3)$ rooted trees as a root node may be inserted at any one of its edges. In the example shown above 5 rooted trees can be generated from a given 4-taxa unrooted tree corresponding to the inserting of a root at any of the 5 edges of the unrooted tree. Note that since there are three topologies possible for 4-taxa unrooted trees, there are 15 corresponding topologies possible for 4-taxa rooted trees.



Internal Edge Removed	Partitions Created	
a	(A,B)	(C,D,E,F)
b	(A,B,C)	(D,E,F)
c	(A,B,C,D)	(E,F)

Fig. 13.8 A partition of a tree is defined to the two subsets of taxa (leaves) created by removing an internal edge. As there are $(n-2)$ internal vertices of an unrooted n-taxa tree, it will have $(n-3)$ distinct partitions. For example, the 6-taxa tree shown above has 3 partitions.

While comparing two trees, all possible partitions of the two trees are created. Let there be q_1 partitions for the tree T_1 and q_2 partitions for the tree T_2. Although both the trees are defined on the same set of leaves or taxons, it is possible that q_1 and q_2 are not equal as the two trees may not be strictly bifurcating. Furthermore, let there be p partitions ($p \leq \min(q_1, q_2)$) from the trees T_1 and T_2 that are identical. The topological distance $d_T(T_1, T_2)$ between T_1 and T_2 is defined by Eq. 13.2.

$$d_T(T_1, T_2) = 2 \cdot [min(q_1, q_2) - p] + |q_1 - q_2| \tag{13.1}$$

For strictly bifurcating trees, $q_1 = q_2 + q$ and the above equation reduces to Eq 13.2. This yields $0 \leq d_T(T_1, T_2) \leq 2(n-3)$ as the range for topological distance between two n-taxa bifurcating trees.

$$d_T(T_1, T_2) = 2 \cdot [q - p] \tag{13.2}$$

13.5 Evolution

Many groups of organisms are now extinct, and without their fossils we would not have as clear a picture of how modern organisms are interrelated. We express the relationships among groups of organisms through diagrams called cladograms, which are like genealogies of species. Over the last 3.7 billion years or so, living organisms on the Earth have diversified and adapted to almost every environment imaginable. The diversity of life is truly amazing, but all known living organisms do share certain similarities. All living organisms can replicate, and the replicator molecule is DNA. As well, all living organisms contain some means of converting the information stored in DNA into products used to build cellular machinery from fats, proteins, and carbohydrates.

13.6 Phylogenetic Tree Object in Matlab

MATLAB provides the function phytree(B) for creating an ultrametric phylogenetic tree object, where B is a numeric array of size `[NUMBRANCHES × 2]` where every row represents a branch of the tree and it contains two pointers to the branches or leaves nodes which are its children.Leaf nodes are numbered from 1 to `NUMLEAVES` and branch nodes are numbered from `NUMLEAVES` + 1 to `NUMLEAVES` + `NUMBRANCHES`.

For example, the following array specifies a phylogenetic tree with four leaves and three branches:

```
B = [1 2; 3 4; 5 6];
Tree = phytree (B);
phytreeviewer (Tree);
```

Additionally, a seccond parameter to this function may be used:

```
TREE = phytree(B,C)
```

This creates an additive phylogenetic tree object where branch distances defined by C. C is a numeric array of size `[NUMBRANCHES × 1]` with the distances of every branch added to the tree.

```
C = [0.5 ; 2.0 ; 4.0];
TreeWithDist = phytree (B, C);
phytreeviewer (TreeWithDist)
```

Fig. 13.9 illustrates the result of building a phylogenetic object in MATLAB.

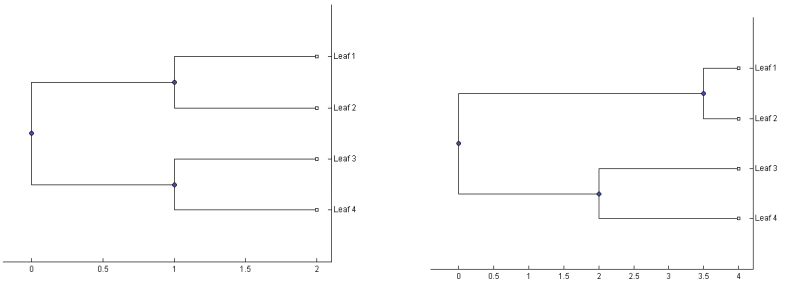


Fig. 13.9 Result of constructing a phylogenetic tree object in MATLAB. The tree on the left is constructed where distances between the nodes have not been specified. The tree on the right includes a specification of distances. Both trees are ultrameric trees where the evolutionary distance between all the leaves and the root node is the same.

MATLAB also provides functions *phytreeread* and *phytreewrite* which are used for reading and writing a tree to a file respectively.

13.6.1 Phylogenetic Trees in BioPerl

PHYLIP is a computational phylogenetics package containing many precompiled programs that are used to build phylogenetic trees. In order to estimate an evolutionary tree from sequence data, data must be passed through various PHYLIP programs, the output of which is then used as input for the next program in line. BioPerl enables a user to automate this pipeline through wrapper classes that interact with the various programs (these programs are not included in BioPerl and must be obtained separately). BioPerl takes each PHYLIP program's output and either stores it in memory or writes it to a supported file type such that BioPerl can then feed this output as input into the next applicable program. The input for such a pipeline consists of some number of supposedly related biological sequences and the output is a tree estimating the sequences' evolution.

An example pipeline can be used to generate a phylogenetic tree is one consisting of, in order, ClustalW, SeqBoot, ProtPars, Consense, and lastly DrawGram. The first step in this pipeline is to use BioPerl to generate a multiple-sequence alignment of the input sequences. This is accomplished by way of BioPerl's wrapper class for ClustalW.

Listing 13.1

```
use Bio::Tools::Run::Alignment::Clustalw;
@params = ('ktuple' => 2,
    'matrix' => 'BLOSUM');
$factory = Bio::Tools::Run::Alignment::Clustalw->new(@params);
$seq_file = "sequence_file.fasta";
$aln = $factory->align($seq_array_ref);
```

end-listing-13.1

Next, each of the PHYLIP programs is used in turn—piping the output data from the previous program as input data into the next. Each of these programs possesses it's own wrapper class in BioPerl of the name Bio::Tools::Run::Phylo::Phylip::*, where * is the name of the PHYLIP program. Each program is in turn initialized and run in the following form, where $var_0 is the previous output.

Listing 13.2

```
$factory = ClassName->new(%params);
$var_1 = $factory->run($var_0);
```

end-listing-13.2

So the first PHYLIP program that is run is SeqBoot, which is a bootstrapping program. Next, we pass the bootstrapped alignment to ProtPars, a parsimony program that estimates phylogenies. The results from ProtPars are then passed to Consense, which computes the consensus trees. Finally, the Consense output is drawn as a rooted tree by DrawGram.

Listing 13.3

```
# SeqBoot
my $seqboot_factory =
Bio::Tools::Run::Phylo::Phylip::SeqBoot->new();
my $aln_ref= $seqboot_factory->run($aln);
# ProtPars
$tree_factory = Bio::Tools::Run::Phylo::Phylip::ProtPars-> new();
$tree = $tree_factory->run($aln_ref);
# Consense
my $con_factory = Bio::Tools::Run::Phylo::Phylip::Consense->new();
my $tree = $con_factory->run($tree);
# Drawgram
my $draw_factory = Bio::Tools::Run::Phylo::Phylip::DrawTree->new();
my $image_filename = $draw_factory->draw_tree($tree);
```

end-listing-13.3

If this sample is run on the following Fast input file, the produced tree will look like Figure 13.10.

Listing 13.4

```
>sequence_1
ACGTACGTACGT
>sequence_2
ACGTACGTACGTACGT
>sequence_3
ACGTACGTACGTTGCA
>sequence_4
ATCGATCGATCG
>sequence_5
ATCGATCGATCGATCG
```

end-listing-13.4

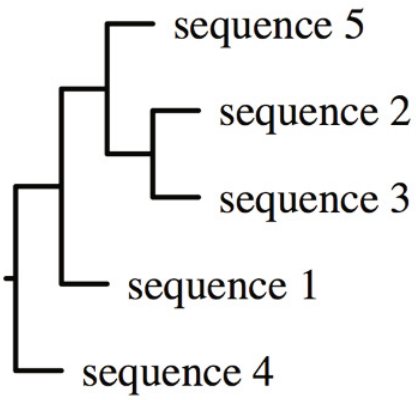


Fig. 13.10 The resulting phylogenetic tree produced by the PHYLIP pipeline

13.7 Significance of Trees Constructed

The accuracy and validity of comparative genomic conclusions drawn are directly related to the accuracy and validity of the phylogenetic tree reconstructed inferred using a phylogeny reconstruction method. It is therefore desirable to be able to assign a quantitative "quality" parameter to the tree as the phylogenetic reconstruction methods are often prone to making errors resulting from inferring wrong branches, complex biological processes such as recombinant and horizontal gene transfers which can not be modeled as a single tree, as well as due to issues related with inaccuracy and incompleteness of data. Consequently, various methods have been introduced for quantitatively estimating *confidence* in the branch of an reconstructed phylogenetic tree.

Of the two common ones, namely the bootstrap method and Bayesian inference techniques, the former is discussed in somewhat greater detail below.

13.7.1 Bootstrapping

The bootstrap method is usually used for trees generated by maximum parsimony (MP) or maximum likelihood (ML) methods. The confidence in a brach is computed by estimating many trees over subsamples of the dataset and using the the percent of trees containing that branch as a measure of its support.

Original sequence data is sub-sampled to to produce **new input data** of the same length. The result of the sampling process may create duplicates of the original sites (columns in the multiple sequence alignment) or may completely eliminate certain sites. Notwithstanding, the sub-sampling process maintains statistical similarity between the new dataset and the original input data. A phylogenetic tree is subsequently constructed with each new data set using the particular method of interest.

The support or confidence in a labeled tree is constructed by taking the majority consensus of the set of trees created during the bootstrapping iterations. Essentially then the support for a branch is its likeness to the a majority rule consensus tree created after the bootstrapping analysis.

The bootstrapping method may be applied for evaluating the significance of trees generated by distance based methods as well. An extra step is need to first compute the distance matrix from each of the replicate data sets produced as a result of sub-sampling the original data set.

Further Readings

1. Schuh, R.T., Brower, A.V.Z.: Biological systematics: principles and applications, 2nd edn. Comstock Pub. Associates/Cornell University Press, Ithaca (2009)
2. Baum, D.A., Smith, S.D.: Tree thinking: an introduction to phylogenetic biology. Roberts, Greenwood Village (2012)
3. Forster, P., Renfrew, C.: Phylogenetic methods and the prehistory of languages. McDonald Institute monographs. McDonald Institute for Archaeological Research, University of Cambridge, Cambridge (2006)
4. Huerta-Cepas, J., Capella-Gutierrez, S., Pryszcz, L.P., Denisov, I., Kormes, D., Marcet-Houben, M., Gabaldón, T.: Phylomedb v3.0: an expanding repository of genome-wide collections of trees, alignments and phylogeny-based orthology and paralogy predictions. Nucleic Acids Res. 39(Database issue), D556–D560 (2011)
5. Planet, P.J.: Tree disagreement: measuring and testing incongruence in phylogenies. J. Biomed. Inform. 39(1), 86–102 (2006)
6. Felsenstein, J.: Evolutionary trees from dna sequences: a maximum likelihood approach. J. Mol. Evol. 17(6), 368–376 (1981)

13.8 Exercises

1. Phylogenetics is...

   A) The grouping of organisms by their physical characteristics
   B) The study of evolutionary relationships in organisms
   C) The study of gene expression in organisms
   D) The extraction of phylo from genetic sequences

2. What is the difference between a rooted and unrooted tree?

   A) Rooted trees are unable to reveal information about evolutionary relationships
   B) Rooted trees are considered more reliable for establishing evolutionary relationships
   C) Rooted trees attempt to establish the relatedness of organisms to a common ancestor
   D) Rooted trees are significantly easier to construct than unrooted trees

3. What is the term for an individual species in a phylogenetic tree?

   A) Leaf
   B) Operational Tanonomic Unit (OTU)
   C) External Node
   D) All of the Above

4. At the Tree of Life website, how are organisms connected?

   A) A single species will link directly to all related species
   B) Organisms are linked by the root page, will contains direct links to all species and groupings
   C) Each branch connects to related groupings via other branch pages, terminating in individual species
   D) The tree is divided into several root pages that link to a specific type of living creature, which possess links to sub-groupings that terminate in individual species

5. What is the difference between using an optimality criterion and a clustering algorithm for reconstructing phylogenetic trees?

   A) Optimality criterions function by determining the number of steps necessary to transform one tree into another, while clustering algorithms function by creating hereditary groupings
   B) In the steps necessary to reconstruct a phylogenetic tree, the optimality criterion must be applied first in order to provide valid input data for the clustering algorithm
   C) In the steps necessary to reconstructing a phylogenetic tree, the clustering algorithm must be applied first in order to provide valid input data for the optimality criterion method

D) Clustering algorithms function by determining the number of steps necessary to transform one tree into another, while optimality criterions function by determining the number of steps necessary to transform one tree into another

6. How does a branch and bound optimization render exhaustive searches more efficient?

   A) It generates all possible trees
   B) It discovers and prunes unproductive paths
   C) It guarantees the discovery of an optimal tree
   D) It makes use of breadth-first search

7. What is the correct order of the steps the ClustalW algorithm uses for Multiple Sequence Alignment:

   1. ClustalW constructs a distance matrix of N(N-1)/2 pairs of sequences by pairwise alignment of the sequences
   2. ClustalW builds a guide tree from the distance matrix using the clustering method (neighbor-joining) by Saitou and Nei
   3. ClustalW will convert the similarity scores to evolutionary distances based on the model by Kimura

   Correct order of processing steps:

   A) 1,2,3
   B) 2,3,1
   C) 1,3,2
   D) 2,1,3

8. In topological distance, the more related the trees the larger the value resulting from the partition metric will be. True or False. Justify.
9. Consider the two unrooted phylogenetic trees shown in Fig. 2b. Recall that the topological distance between two trees, dT is defined as $dT = 2 \times [min(q_1, q_2) - p] + |q_1 - q_2|$ where q_1 and q_2 is the number of partitions in each of the two trees and p is the number of common partitions. Find the topological distance between the following two unrooted trees. Note that edges a, b and c are internal edges.
10. Consider the following set of sequences. Perform the following analysis using the functions provided in MATLAB Bioinformatics toolbox:

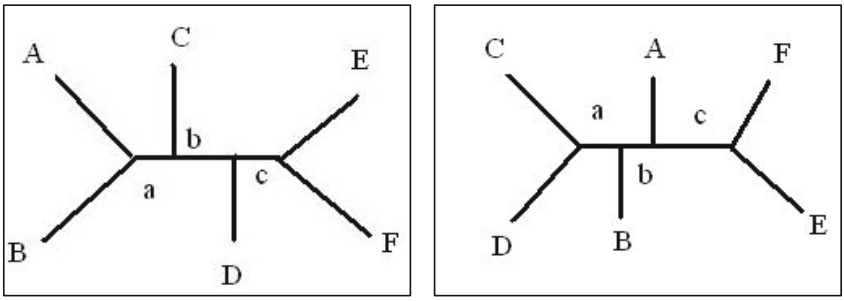


Fig. 13.11 Comparison of Phylogenetic Trees

```
>sequence_1
ACGTACGTACGT
>sequence_2
ACGTACGTACGTACGT
>sequence_3
ACGTACGTACGTTGCA
>sequence_4
ATCGATCGATCG
>sequence_5
ATCGATCGATCGATCG
```

(a) Compute the pairwise distance between each sequence using MATLAB's global alignment function *nwalign*.
(b) Select the pair of sequences with the smallest distance and construct the alignment. Construct a sequence structure with closest neighbors and compute the consensus sequence *cons*. The following example assumes that *seqA* and *seqB* are to be merged into an internal node. Note down the score.

```
[score aln] = nwalign (seqA, seqB, 'Alphabet', 'NT');
seqStruct(1).Sequence = aln(1,:);
seqStruct(2).Sequence = aln(3,:);
cons = seqconsensus (seqStruct);
```

(c) Redo the distance computations with the merged sequences replaced with their consensus sequence. Continue till all sequences have been merged into a single node noting down the score at each step.
(d) Use the order in which the sequences were merged and the distances of the nodes merged to construct a phylogenetic tree using MATLAB function *phytree*.
(e) Display and print the tree.
(f) Save the tree to a disk file, read the saved tree and display it.

Chapter 14
Distance Based Methods

The fundamental prerequisite necessary to deal with all these topics is, of course, an efficient and biologically plausible way to assess sequence similarity: given an alphabet and initially just two sequences, how are we to measure in a biologically meaningful way the similarity between s and t? The answer depends, of course, on what mutational operations one perceives as biologically relevant can be performed to produce one sequence from another one. Two sequences s and t are considered to be similar if a few such operations can change s into t, and dissimilar otherwise. The problem which remains to be discussed in this context is, of course, how to assess the biological likelihood of the various operations which can be performed to change one sequence into another one and how to quantify numerically the (dis)similarity of any two given sequences.

The most simple operation is, of course, point mutation: at a certain position "i" of sequence, base `X` is replaced by another base `Y`. This operation does not change the length m of sequence s; so, given two sequences and length m and n, respectively, one can change s into t by repeated applications of this operation if and only if m equals n, in which case the minimal number of point mutations necessary to change s into t coincides with the Hamming distance between s and t.

14.1 Sequence Similarity

As discussed in forthcoming chapters on phylogenetic reconstruction using character based and probabilistic methods, the assumption made when constructing phylogenetic trees is that we have an ungapped multiple sequence alignment of the sequences we intend to use to reconstruct the tree. The prerequisite to using the distance based methods is the computation of intersequence distance or similarity. Under our general assumption, Hamming distance is a primitive but useful estimation of genetic distance.

In distance based methods, the actual sequence data is not required beyond the computation of the similarity or distance scores between each

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_14

sequence pair. Therefore, with the the availability of string edit distance the requirement of having an ungapped multiple sequence alignment is considerably relaxed. However, from the standpoint of genetic kinship, the problems arising from admitting insertions and deletions - or indels for short - is, of course, whether the new metric introduced is a biologically justifiable. There are some complications that generally have been observed in distance based methods.

For example, taking genetic shuffling into account, transforming the sequence `AAAACCCC` into a sequence `CCCCAAAA` can be accomplished in just one evolutionary step. Or other events such as reversals, which using complement property of DNA nucleotides, could in a single evolutionary step transform the sequence `CATCCGCCA` into the sequence `TGGCGGATG` by reading the original sequence backwards replacing a nucleotide with its complement. Single point mutations may be assigned different weights depending upon their context and whether they are in a coding region, if they represent a transitional or transversional change, or if they are silent.

Notwithstanding these limitations, distance based methods are the simplest to implement. Trees constructed using these methods are often called **phenograms** as the original use of these methods was to represent phenotypic similarity for a group of species in a numerical taxonomy. They work well under the assumption that the rate of gene substitution is more or less constant. As the inter-species distance is represented by a similarity matrix is the only input for these methods, the gene sequence distances may be augmented by phenotypic markers to develop a composite vector which may be utilized for developing a *hybrid* tree drawing upon the benefits of the new trends in genetic tree construction and traditional approaches for building species trees.

14.2 Linkage Analysis

Let us assume that a family of taxa, a measure of similarity has been established associating for a family of sequences, $s_1, s_2 \ldots s_n$. Furthermore, for any pair of sequences s_i and s_j in this family of sequences, a real number d_{ij} has been established satisfying the properties:

$$d_{ij} = d_{ji} > 0 \quad for\ i \neq j$$

and

$$d_{ii} = 0$$

The question we want to address now is how this system of numbers can be used to create a tree representing the true phylogenetic history of sequences. Distance based methods assume the existence of one common ancestral sequence from which all sequences have evolved. If d_{ij} is directly proportional to the time span t_{ij} that elapsed since the last common ancestor

of the sequences s_i and s_j existed, the construction of the family tree by distance based methods is correct under the *additive* assumption. Trees are said to satisfy the "additivity" property if the edge lengths in the tree are additive. Specifically, a phylogenetic tree is additive if the distance, or alternatively the time span, between two nodes is equal to the sum of edges connecting them.

The construction of a distance based phylogeny is based upon the availability of the pairwise distance matrix. Thus, if the phylogeny tree is required from a set of n sequences, the distances between individual sequence pairs s_i and s_j is used as the term D_{ij} of the symmetric distance matrix. Furthermore, a phylogeny reconstructed using distance based methods assumes that the tree satisfies the *ultrameric* property. An ultrameric tree is defined to be the one where all the sequences have evolved from the same common ancestor and the distance of all sequence from the common ancestor is the same. Consequently, a uniform evolution rate is the underlying presumption of ultrameric tree which a distance based phylogenetic reconstruction method yields.

14.3 UPGMA

As discussed above, the distance based methods rely on the reconstructing a tree based on pairwise inter-sequence distances. Similarity score could possibly be used as an inverse of distance measures. However, the additive property and ultrameric properties of the trees constructed are defined with respect to using a true distance measure to capture inter-sequence divergence. The procedure for distance based phylogeny is essentially san agglomerative clustering procedure which begins by n clusters and when the number of clusters is reduced to 1. Initially, each of the n clusters is comprised of a single sequence, and when the process terminates all of the n sequences are merged into a single cluster. It is various stages of the merging steps that create the phylogenetic tree.

Unweighted Pair Group Method using (arithmetic) Averages or UPGMA is the simplest hierarchical agglomerative clustering method that begins with n clusters, $C_1, C2, \ldots C_n$, that are individually comprised of a single sequence each. Correspondingly, the inter-cluster distance is initially the same as the inter-sequence distance. Each stage of the tree building process selects two closest clusters and merges them into a single cluster. The distances between the new, i.e. merged, cluster and all the remaining clusters is subsequently calculated. This iterative process continues till we are left with only a single cluster.

Assume for example that clusters C_i and C_j are the nearest neighbors at a specific stage of the tree building process. Let us further denote the number of sequences in these clusters as $|C_i|$ and $|C_j|$ respectively. After these two clusters are merged into a single cluster C_k, the number of sequences in C_k will be $|C_i| + |C_j|$. The distance D_{kl} of cluster C_k from a cluster C_l can be

computed as a weighted average of the distance of cluster C_l from original clusters C_i and C_j as follows:

$$D_{kl} = \frac{|C_i| \cdot D_{il} + |C_j| \cdot D_{jl}}{|C_i| + |C_j|} \tag{14.1}$$

14.4 Phylogenetic Analysis in MATLAB

MATLAB function *seqpdist* computes distances between sequences using one of the computes pairwise distance between sequences. It returns a vector containing biological distances between each pair of sequences passed. The pairwise distances may be computed using one of the several methods passed as a parameter to this function.

Given a pairwise distance matrix, the MATLAB function *seqlinkage* performs a linkage analysis to construct the tree. This linkage analysis is functionally equivalent to performing UPGMA as it is the standard procedure for agglomerative clustering. Parameters to *seqlinkage* further control the type of clustering performed.

As an example, MATLAB default data directory contains a set of FASTA formatted sequences which are read in. The pairwaise distances between these sequences is next computed and stored in a distance matrix, *dist*. This matrix is contains the pair wise distances of each sequence computed using a method specified. Next, the function *seqlinkage* creates a phylogenetic tree which is saved a tree object `treeDist`. The resulting tree may be viewed using the function *phytreeviewer*.

```
seqs = fastaread('pf00002.fa');
dist = seqpdist(seqs,'method','jukes-cantor','indels','score',...
                    'scoringmatrix',@pam250,'pairwisealignment',true);
distTree = seqlinkage(dist);
phytreeviewer(distTree);
```

14.4.1 Neighbor Joining Algorithm

Neighbor joining is a greedy algorithm for optimizing a tree according to the "balanced minimum evolution" [1] (BME) criterion which defines the tree length – sum of all branch weights in the tree – as a weighted sum of the distances in the distance matrix. The weights are dependent on the topology with the BME optimal topology being the one that minimizes this tree length. It is a polynomial time algorithm with each step of the algorithm aimed at greedily joining a pair of taxa that yields that results in the largest reduction in the estimated tree length. This procedure is not guaranteed to find the optimal the BME optimal topology although it produces a close result.

MATLAB function *seqneighjoin(DIST)* computes a phylogenetic tree object from the pairwise distances DIST between the species or products applying the neighbor-joining method. Additionally, the command

```
treeNN = seqneighjoin (DIST, Method)
```

enables the specifications of parameters for selecting a method for neighbor joining.

The default method assumes equal variance and independence of evolutionary distance estimates (a = 1/2) [2, 3]. The "firstorder" method as a parameter assumes a first order model of the variances and covariances of evolutionary distance estimates [4].

In continuing with the distance matrix computed in the example above, the neighbor joining method is utilized to compute a phylogenetic tree:

```
nnTree = seqneighjoin (dist);
phytreeviewer(nnTree);
```

As illustrated in Fig. 14.1, the tree produced by the UPGMA algorithms is an ultrameric tree since the initial distances between the nodes is preserved. Neighbor Joining method on the other hand, produces a typically a phylogeny with a "star" topology where the resulting tree need not be an ultrameric tree.

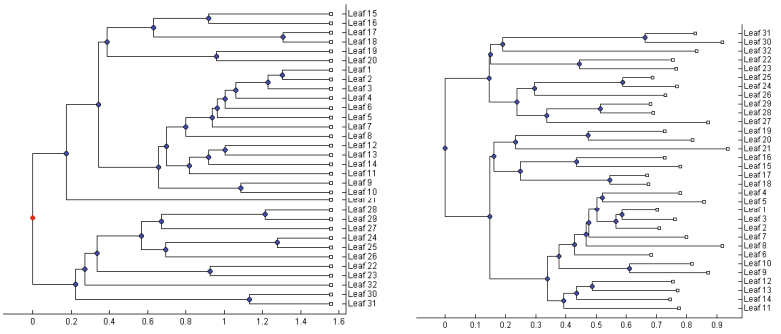


Fig. 14.1 Trees produced by the distance based phylogenetic tree construction. The tree on the left is produced by the UPGMA algorithm, while the tree to the right is produced by Neighbor Joining algorithm.

Further Readings

1. Gascuel, O., Steel, M.: Neighbor-joining revealed. Mol. Biol. Evol. 23(11), 1997–2000 (2006)
2. Studier, J.A., Keppler, K.J.: A note on the neighbor-joining algorithm of saitou and nei. Mol. Biol. Evol. 5(6), 729–731 (1988)

3. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. 4(4), 406–425 (1987)
4. Gascuel, O.: Bionj: an improved version of the nj algorithm based on a simple model of sequence data. Mol. Biol. Evol. 14(7), 685–695 (1997)
5. De Bruyn, A., Martin, D.P., Lefeuvre, P.: Phylogenetic reconstruction methods: an overview. Methods Mol. Biol. 1115, 257–277 (2014)
6. Hall, B.G.: Building phylogenetic trees from molecular data with mega. Mol. Biol. Evol. 30(5), 1229–1235 (2013)
7. Bruno, W.J., Socci, N.D., Halpern, A.L.: Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction. Mol. Biol. Evol. 17(1), 189–197 (2000)
8. Li, W.H.: Simple method for constructing phylogenetic trees from distance matrices. Proc. Natl. Acad. Sci. U. S. A. 78(2), 1085–1089 (1981)
9. Prager, E.M., Wilson, A.C.: Construction of phylogenetic trees for proteins and nucleic acids: empirical evaluation of alternative matrix methods. J. Mol. Evol. 11(2), 129–142 (1978)
10. Fitch, W.M., Margoliash, E.: Construction of phylogenetic trees. Science 155(3760), 279–284 (1967)

14.5 Exercises

1. Consider the following piece of MATLAB code:

```
data = {'German_Neanderthal' 'AF011222';
 'Russian_Neanderthal' 'AF254446';
 'European_Human' 'X90314' ;
 'Mountain_Gorilla_Rwanda' 'AF089820';
 'Chimp_Troglodytes' 'AF176766';
};

for ind = 1:5
  seqs(ind).Header = data{ind,1};
  seqs(ind).Sequence = getgenbank(data{ind,2},...
    'sequenceonly', true);
end

distances = seqpdist(seqs,'Method','Jukes-Cantor','Alphabet','DNA');
tree = seqlinkage(distances,'UPGMA',seqs)
```

   Answer the following questions:

   (a) Comment on what is stored in the `data` array. Also, comment on the purpose of the array `seqs`.
   (b) Run the program and examine the output. Briefly comment on the purpose of the program.
   (c) What can you say about the phylogenetic relationships between the species studied in the data.

2. Consider once again the structure **seqs** that was created earlier in Question 1. The function *seqpdist*, a specialized function corresponding to function *pdist*, must be informed that we are working with the DNA alphabet as by default it treats the sequence data to be comprised of protein sequences. The *seqlinkage* specialization of the function *linkage* provides us with the option of choosing the algorithm for clustering (UPGMA in this case) and allows us to label the nodes with sequence headers.
   Answer the following questions:

   (a) Depending upon whether the input sequences are protein or nucleotide sequences, the function *seqpdist* allows the inter-sequence distances to be computed by a number of methods. Name the three methods that are supported for both the nucleotide and protein sequences. Which of these methods is the default method used for distance computation by the function *seqpdist*. (Hint: Type `help seqpdist` at the MATLAB command prompt).
   (b) Is a different UPGMA tree produced for the five sequences under study when other two non-default methods for computing inter-sequence distances is used. Show the commands you issue and resulting tree produced by each of the other non-default methods.

(c) MATLAB pre-installs a number of data sets. For example, a set of 32 protein sequences are installed as the `pf00002.fa` dataset. The following command will load this dataset into seqs structure

```
seqs = fastaread('pf00002.fa');
```

Similar to Question 2 above, compare the three trees produced by *seqpdist* for this new protein sequence dataset. Show the commands you used to generate the trees and resulting trees generated.

(d) It is tedious to use visual inspection to compare large trees such the one with *32-nodes* you just build. MATLAB bioinformatics toolbox also contains a function for comparing phylogenetic trees. Research and find this function. List the command you use to compare the *32-node* trees in part2.

(e) In addition to UPGMA, what other linkage algorithms is supported by the function seqlinkage. Provide a list with a short description.

Chapter 15
Character Based Methods: Parsimony

Maximum Parsimony (MP) methods were originally developed for comparing morphologies. However, they are increasingly being used on molecular data for inferring species trees from gene trees. In MP methods, four or more aligned nucleotide or amino acid sequences are considered. The parsimony analysis for each site in the aligned sequence is performed for each possible tree, where the tree that produces the alignment with the minimal number of transformations is considered to be the one with maximum parsimony. The theoretical basis for the method is Occam's razor which states that the explanation of any phenomenon should make as few assumptions as possible, eliminating, or "shaving off," those that make no difference in the observable predictions of the explanatory hypothesis or theory. Thus, the simplest explanation is the one that is most likely correct.

MP methods offer some advantages over the other tree construction methods. Firstly, since they are based on counting the number of character transformations, they are free from the amino acid and protein substitution assumptions required for distance and maximum likelihood methods. Since it is not clear whether the assumptions made by the other models are robust, the tree construction from MP methods with no such assumptions may be more reliable. It has been demonstrated experimentally that when the sequences used for tree inference are similar, the rate of nucleotide substitution is almost constant, and the number of nucleotides examined is large, MP methods are more reliable for obtaining the topology of a phylogenetic tree.

MP methods are broadly divided into **unweighted** and **weighted** MP methods. The unweighted methods treat all possible substitutions at a given location as being equally likely. In reality, this is seldom the case. Correspondingly, the weighted methods take the transitional and transversional changes into account while computing the number of changes for evaluating a given tree topology.

15.1 Finding the Maximum Parsimony Tree

The process of finding a Maximum Parsimony (MP) tree entails a search in the space of all tree topologies. Assuming that we are interested

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_15

in reconstructing a MP tree with n sequences, our search space is comprised of the unrooted phylogenetic trees with n leaves. We then search this space to find the tree(s) that explains the data set with the minimum number of changes or substitutions. There may be more than one such tree that explains the data with maximum parsimony. The tree(s) are then the nodes in the search space that is reported. Since the pruning of the search space is often possible using branch and bound methods, these techniques are often utilized for improving the search times for constructing MP trees. We next look at the methodology for counting the number of substitutions that a given tree purports for explaining a sequence data set.

15.1.1 Counting Substitutions for a Tree

The procedure for counting the number of substitutions needed to explain the data set under a given tree topology begins by starting at the leaves and propagating the intersections set. This process is explained in Figure 15.1 which illustrates the results of the substitution counting process for a *single location* in the multiple sequence alignment of sequences from six species shown. A candidate tree topology being considered is shown in Figure 15.1 (a).

The process of computing the MP phylogeny involves the consideration of all possible tree topologies. For a given topology, we begin the calculation of the substitution count by moving up the tree and labeling the vertices as we go along and asking whether there is an intersection in the label used for the children. If the intersection of the children's label is non-empty, then the intersection is used as the label for the parent. And if the intersection is an empty set, the parent's label is instead the union set of the labels of its children.

The various panels in Figure 15.1 exemplify the individual steps of the counting process:

(a) The candidate tree topology with 6-taxa. The nucleotides on the leaf node are those observed at a particular location in the multiple sequence alignment of the sequences from the six species being compared. The number of substitutions needed to begin with is 0.
(b) For the leaves originating from the parent node a, the intersection of the nucleotides $\{A\} \cap \{A\} = \{A\}$. Since the intersection set is non-empty, the number of substitutions is still 0.
(c) The next set of leaves to be merged are the children of the parent b. The intersection of nucleotides is $\{T\} \cap \{C\} = \{\}$. Thus the number of substitutions is incremented by 1 and the node is assigned the label of $\{T\} \cup \{C\} = \{T, C\}$.
(d) The node b and the other child of internal node c are merged next. The intersection of the labels of node c's children is $\{T, C\} \cap \{T\} = \{T\}$. Since this non-empty, the label assigned to node c is T, and the substitution count is held at 1.

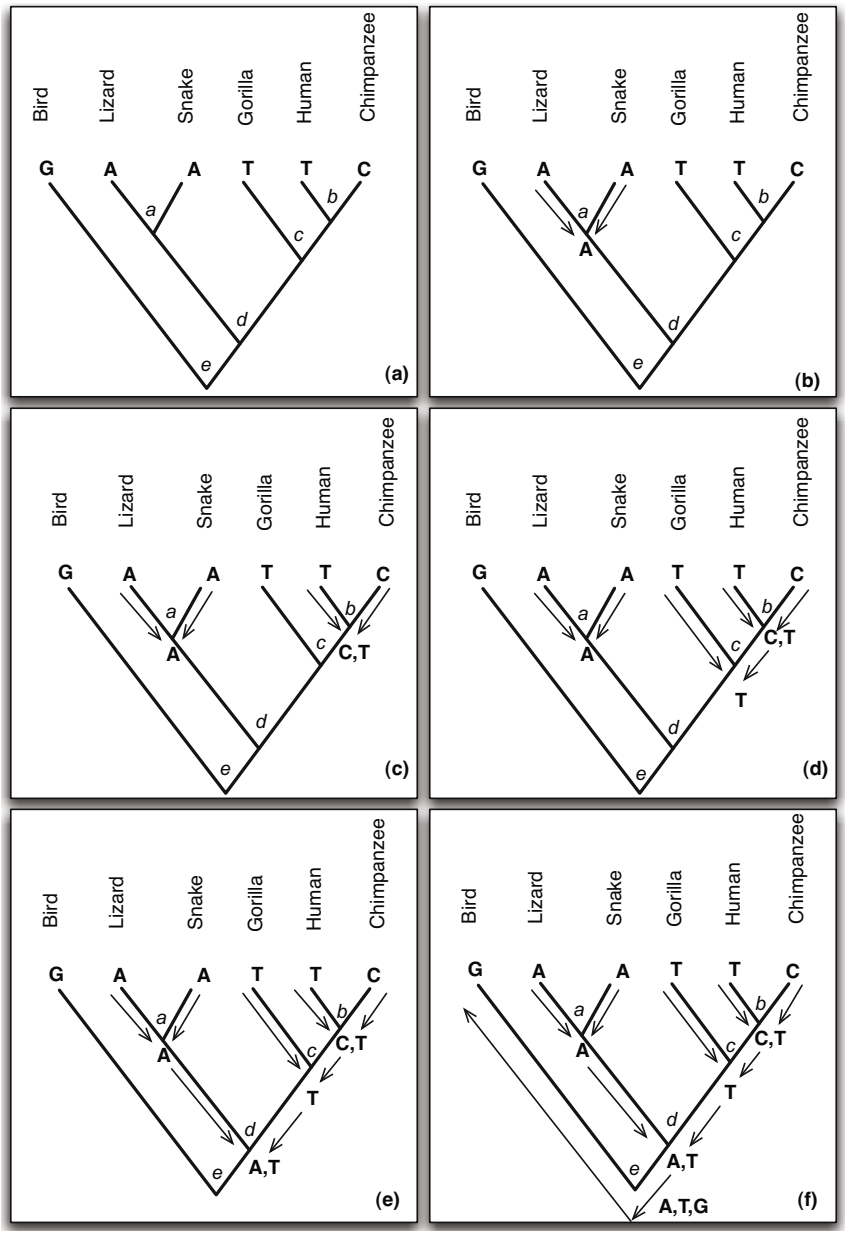


Fig. 15.1 Counting the number of substitutions for each location in the multiple sequence alignment sequences from six species

(e) The two children for node d are labeled A and T. The intersection of these is empty ($\{A\} \cap \{T\} = \{\}$). This results in incrementing the substitution count to 2 and labeling the node d as $\{A, T\}$, the union of $\{A\} \cup \{T\}$.
(f) Finally, the label for node d and the last terminal node. Since the label placed on node d and the last terminal node with the label of G have no labels in common, the label for node e is set as $\{A, T, G\}$ which is the union of the label for node d and the last terminal node. Correspondingly, the substitution count is increased to 3.

15.1.2 Computing Tree Length for an Alignment

The MP tree determination algorithm aims at finding the tree topology with minimum **tree length**. The tree length, or L_T, for a given tree is defined to be the total number of substitutions needed for all the sites in the alignment. The substitution count for a single site in the alignment is obtained by the method described in the previous section. By considering the tree lengths, we therefore consider the overall optimality of the tree for the entire length of the multiple sequence alignment. The algorithm yields the tree with minimum substitutions over all sites and over all topologies.

The complexity of the algorithm is $m \cdot (2n - 5)!!$, where m is the length of alignment and n is the number of taxas. Recall from section 13.3 that the number of unrooted trees possible for n-taxas is $(2n - 5)(2n - 7) \ldots 1$ or $(2n - 5)!!$. This MP algorithm is generally computationally intractable for large values of n. For reducing the computational inefficiencies for small and computationally tractable values of n (generally $n \leq 10$), some observations help reduce the computational time attributed to m.

Nucleotide or amino acid sites that are identical in all of the taxa sequences are called **invariable sites**. These sites do not contribute to the substitution counts and thus are eliminated from further consideration. For MP tree determination, only the **variable sites** are used. However, not all variable sites are useful for finding the most parsimonious topology. Any sites that have singletons can also be explained by the same number of substitutions in all topologies. They may thus also be removed from the computation of tree lengths. For a site to be **informative** there must be at least two different kinds of nucleotides, each represented at least two times.

For example, consider the 5-nt multiple sequence alignment below. Only site 4 is an informative site. Columns 2 and 5 are invariable sites while columns 1 and 3 singletons. Thus, the tree topology with minimum substitution count for site 4 will also be the tree with minimum L_T and the maximally parsimonious tree.

```
COLUMN   1 2 3 4 5


SEQ1     C C C C G
SEQ2     C C T T G
SEQ3     T C G T G
SEQ4     G C A C G
```

A four sequence taxa has three possible unrooted trees as shown in Figure 15.2(a). These are the canonical trees with sequence labels shown on the leaves. There are two internal nodes labeled a and b.

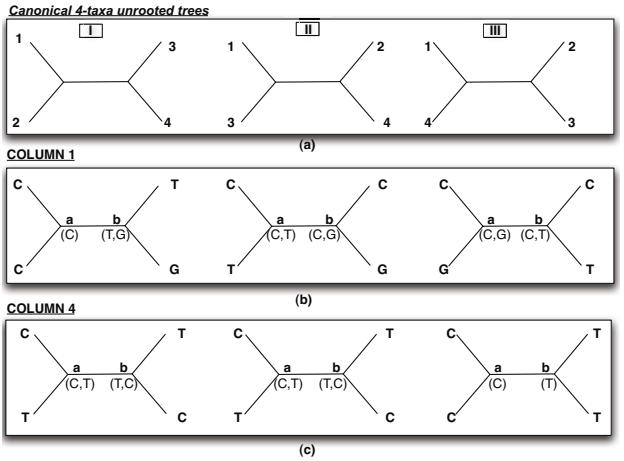


Fig. 15.2 The canonical unrooted tree with 4-taxas is shown in (a). The first column in the alignment contains two singletons and correspondingly all the trees for column 1 have a substitution count of 2 as shown in (b). The only informative column in the alignment shown is column 4. The substitution counts for the three topologies for this column shown in (c) are 2, 2 and 1. The third tree topology is therefore the optimal and maximally parsimonious.

The three trees corresponding to column 1 are shown in Figure 15.2(b). Each of these trees has a substitution count of 2. In general, any singleton column can be explained by the number of singleton characters which in this case is 2. The singleton characters in column 1 are T and G.

The only informative column for this alignment is column 4. The three trees corresponding to column 4 are shown in Figure 15.2(c). The substitution counts for the three trees are 2, 2 and 1. The maximally parsimonious tree topology corresponding to this column is the third tree with the substitution count of 1. Since other columns are either invariable or singleton, the third canonical tree topology shown is also the tree with smallest length or the maximally parsimonious tree.

15.1.3 Computing Branch Lengths

The common usage of MP methods is the construction of tree topology without the assignment of branch lengths. This is primarily because parsimony tree construction primarily estimates the *configuration* of the taxa and parent nodes that yields the overall topology with the smallest number of character substitutions. However, methods have been developed that estimate branch lengths under a set of assumptions. As a general rule, the branch length is the average number of substitutions made at each branch considering all the MP trees. Thus, if there are three MP trees possible for a given number of taxas, and a substitution is necessary along a certain branch in two out of the three MP topologies, then the branch is assigned a length of $\frac{2}{3}$. The branch length thus corresponds to the average number of substitutions for a branch (exterior or interior) where the average is computed over all parsimonious trees.

15.1.4 Branch and Bound Optimization

When the number of taxa is small, a computer can exhaustively generate the list of possible trees and compute the tree length for each to yield the topologies with maximum parsimony (and minimum substitution costs). The process of generating all possible tree is thus an **exhaustive search** of the space of all possible n-taxa trees. With each node we ask if the tree is optimal and if so, we retain it. As n becomes large, this approach becomes intractable and we look towards approaches that can effectively prune the search space so that finding a solution may becomes more feasible.

One such approach for pruning the search space is to utilize the **branch-and-bound** approach. In this method, the trees that have a value of L_T longer than the ones previously examined are all ignored, and the MP is determined by evaluating the tree lengths for a group of trees that potentially have a shorter length. Thus, the search space is set up as a tree with each node on the tree expanding out in a manner corresponding to the possibility that a single taxon may be added at that node. Thus, the path through the tree corresponds to the various stages of the additions of the taxa. When a node is encountered that has a higher tree-length than the minimum tree length seen thus far in a systematic traversal of the search space, all subsequent expansions for that tree are pruned away. Intuitively, this corresponds to taking a subset of m taxa, where $m < n$, and computing the tree length. If this tree length is larger than the tree length we have seen for *any* n taxa configuration, the m taxa tree topology and all its children are removed from further consideration.

Heuristic Algorithms

Heuristic approximations to branch and bound methods have been developed. One method that works well in practice is based on the construction of core trees. The search for an MP tree begins by constructing an initial core tree of three taxas. A single unrooted core tree with 3 taxa utilizes those sequences that yield the largest tree length. Thus, every possible selection of three sequences is analyzed, and the combination of three sequences that yields the largest tree length is retained as the core 3 taxa tree. The rationale is that we want to construct an MP tree and would like the algorithm to converge to the optimal tree length L_T as quickly as possible.

After the three-taxa core tree has been constructed, each remaining taxon is in turn placed on each of the three branches of the three-taxa tree. For each taxon we record the tree length such that the minimum number of substitutions are needed. This is because we are constructing an MP tree and so must place this taxon at its optimal location. After obtaining the MP tree lengths for four taxas, we select the taxon which maximizes the tree length. Again, the rationale is that we want to construct the MP tree and would like the algorithm to converge to the the optimal tree length L_T as quickly as possible.

This algorithm is therefore often referred to as the **min-max** algorithm because we must find the minimum number of substitutions for adding a given taxon and then select the taxon which maximizes the overall tree length.

15.2 Weighted Parsimony Algorithms

As a general rule, the MP methods are expected to produce reliable trees when the extent of homoplasy (backward or parallel substitution) is small. Consequently, the result of computing an MP tree generated from the sites in an alignment that evolve slowly will be more accurate than the tree generated from the sites in the alignment that are evolving rapidly. Herein lies the basis for the weighted parsimony tree construction. If we could assign a higher weight to the sites that are evolving slowly, our overall MP tree will be more accurate.

We next look towards the basis for determining when a site is slowly evolving versus when it is not. It has been well established that the nucleotides at the first, second and third position in a codon evolve at different rates. The nucleotide at the third position of a codon evolves at the fastest rate, while those at the second position are slowest to evolve or change. The estimated rate of nucleotide changes per 1000 bp for genes in mitochondrial DNA is 15.5 for the fist nucleotide, 8.5 for the second and 36.8 for the third nucleotide. The rate of change for nucleotides in a codon thus approximately follow the ratio of {2:1:5}. While counting the number of substitutions, one could assign weights in inverse proportion to the nucleotide's propensity to change.

For example, one could assign weights of $w_1 = \frac{1}{2}$, $w_2 = 1$, and $w_3 = \frac{1}{5}$ for the corresponding substitutions observed at three nucleotide positions in a codon. These weights could further vary from one gene to another and also within a given gene.

Table 15.1 Substitution matrices utilized for the computation of weighted parsimony scores. (a) In general, transversional substitutions are given a higher substitution cost of w in comparison to the transitional substitution cost of 1. (b) illustrates a typical case where the transversional scores are twice as much as the transitional scores. Sometimes, as in (c), we may wish to treat the transitional substitutions as synonymous and not count them towards the substitution cost of the tree.

	A	**C**	**G**	**T**
A		w	1	w
C	w		w	1
G	1	w		w
T	w	1	w	

(a)

	A	**C**	**G**	**T**
A		2	1	2
C	2		2	1
G	1	2		2
T	2	1	2	

(b)

	A	**C**	**G**	**T**
A	0	1	0	1
C	1	0	1	0
G	0	1	0	1
T	1	0	1	0

(c)

If a parsimony tree is generated using nucleotide data, where the relative codon position is not known, the transitional versus transversional characterization of the nucleotide substitutions may also be used to assign weights. The transitional substitutions are defined as those where purine substitutes another purine, or a pyrimidine substitutes another pyrimidine. Transversional substitutions involve the substitution of a purine by a pyrimidine or vice versa. Recall that two of the bases in nucleic acids, adenine (A) and guanine (G), are purines. In DNA, these bases form hydrogen bonds with their complementary pyrimidines thymine (T) and cytosine (C). The relative weights of the transitional and transversional substitutions are often represented as a ratio of *1:w*, as shown in Table 15.1.

When the parsimony tree utilizes the substitution matrix shown in Table 15.1(c), all the transitional substitutions are essentially ignored. Since the substitution cost is computed only by considering the transversional changes, the resulting parsimony is called **transversional parsimony**. Such possibilities demonstrate a collateral drawback of MP methods. For example, it is often not known a-priori the appropriate weight matrix to utilize for the computation of a weighted parsimony tree. Therefore, a technique known as **dynamically weighted parsimony** is often utilized for the construction of MP trees. An initial tree is constructed using the substitution matrix that appears to be most appropriate for the given data set. This tree is then utilized for computation of a new substitution matrix which is subsequently utilized for the computation of a new MP tree. This process continues until the tree topology becomes stable and the substitution cost is minimized.

15.3 Protein Alignments

Using protein sequences for the inference of phylogenetic trees is often preferred over the use of DNA sequences because the evolutionary patterns of DNA are often very complex. The substitution pattern is not the same for all positions within a codon with the higher propensity for position three to undergo change. This in turn requires an a-priori knowledge of the coding regions in order to assign appropriate weighting factors to the nucleotides to effectively compute the weighted parsimony scores, as discussed in section 15.2. Alternative techniques have been developed to directly utilize protein sequences (where known) to construct MP trees.

A simplistic approach for constructing MP trees may be utilized by treating each of the 20 amino acids as a unique character and counting the substitutions, as we did for DNA sequences. However, the genetic code clearly shows records that some amino acid substitutions may require two or three nucleotide changes, while others only require changing a single nucleotide. Thus, some amino acids are biochemically similar to one another. Consequently, their mutual substitutions should not be weighed on par with their substitutions by those that they do not share significant biochemical similarity with. Maximum Parsimony algorithms for proteins essentially proceed in a similar manner as the algorithms developed for DNA sequences. The essential difference is in the manner by which the substitution costs are computed. Instead of assigning an equal cost for replacing one character by another, amino acid costs are computed by counting the number of characters that are different in their underlying codon. As the genetic code is degenerate, with more than one and often two or more codons coding for an amino acid, simplifying assumptions need to be made. One such assumption utilized in practice is to take the *minimum* substitution score between the two codon sets and use that as the amino acid substitution cost.

15.4 Matlab Functions for Codon Substitution Rates

It is evident from the genetic code that some amino acids have multiple codons and have higher degeneracy. They are thus better able to handle point mutations – Argenine (Arg), Leucine (Leu), and Serine (Ser) are all coded by six different codons, whereas Tryptophan (Trp) is only coded by one. Codons that code for the same amino acid are called synonymous codons. Contrarily, if the changes to a codon result in changing the coded amino acid, the change is considered to be nonsynonymous.

MATLAB function *dnds(SEQ1,SEQ2)* estimates the synonymous and nonsynonymous substitutions rates per site between two homologous sequences by comparing codons and returns the nonsynonymous substitution rate (DN), the synonymous substitution rate (DS), as well as their respective variances. In computation of these rates, codon that includes gaps or ambiguous

nucleotide characters is excluded from calculation. This analysis is performed considering the number of codons in the shortest sequence and assumes sequences do not have frame shifts.

Consider the following MATLAB code where two sequences are downloaded from GenBank. The coding regions from the sequences are extracted and converted into amino acid sequences that are then globally aligned using the function *nwalign*:

```
s01 = getgenbank('AF509094');
s02 = getgenbank('DQ094287');
    s01_cds = featuresparse(s01,'feature','CDS','Sequence',true);
s02_cds = featuresparse(s02,'feature','CDS','Sequence',true);

aa01 = nt2aa(s01_cds);
aa02 = nt2aa(s02_cds);

[score, aln] = nwalign (aa01, aa02);
```

The alignment information is used to insert gaps into the coding segments. Function *seqinsertgaps (SEQ, GAPPEDSEQUENCE, N)* is used for this purpose. This function is called with a first sequence `SEQ` into which the gaps need to be inserted and a second gapped sequence `GAPPEDSEQ`. The parameter $N = 1$ is used when both sequences have the same alphabet, and $N = 3$ when `SEQ` contains nucleotides representing codons and `GAPPEDSEQ` contains amino acids. Default value for N is 3. After gaps have been inserted into the two aligned coding sequences, the MATLAB function *dnds* is called to computed the rate of synonymous and nonsynonymous substitution rates.

```
s01_aln = seqinsertgaps(s01_cds, aln(1,:), 3);
s02_aln = seqinsertgaps(s02_cds, aln(3,:), 3);
[dn, ds] = dnds(s01_aln, s02_aln)
```

Further Readings

1. Liò, P., Bishop, M.: Modeling sequence evolution. Methods Mol. Biol. 452, 255–285 (2008)
2. Gambin, A., Slonimski, P.P.: Hierarchical clustering based upon contextual alignment of proteins: a different way to approach phylogeny. C. R. Biol. 328(1), 11–22 (2005)
3. Simmons, M.P., Ochoterena, H., Carr, T.G.: Incorporation, relative homoplasy, and effect of gap characters in sequence-based phylogenetic analyses. Syst. Biol. 50(3), 454–462 (2001)
4. Simmons, M.P., Ochoterena, H.: Gaps as characters in sequence-based phylogenetic analyses. Syst. Biol. 49(2), 369–381 (2000)

15.5 Exercises

1. Answer the following questions considering the 5-nt multiple sequence alignment below: Only site 4 is an informative site. Columns 2 and 5 are invariable sites while columns 1 and 3 singletons. Thus, the tree topology with minimum substitution count for site 4 will also be the tree with minimum L_T and the maximally parsimonious tree.

```
COLUMN   1 2 3 4 5

SEQ1     C G C A G
SEQ2     C G T T G
SEQ3     C A G T G
SEQ4     C A A C G
```

   (a) Identify the informative site(s).
   (b) Identify the invariable site(s).
   (c) Identify the singleton site(s).
   (d) A tree topology with the minimum number of substitutions for which site will be the maximally parsimonious tree? Justify.
   (e) Picking suitable labels for internal nodes, draw all possible canonical trees unrooted trees with four sequence taxa.
   (f) Identify the most parsimonious site based on the informative site. How many substitutions are needed for parsimonious tree.

2. Consider the two DNA sequences:

```
seq1 = 'ATGCCCGACTAG'
seq2 = 'ATTCCCGAGTAA'
```

   Perform the following analysis using MATLAB:

   (a) Convert each sequence to the corresponding amino-acid sequence using the function *nt2aa*.
   (b) Identify if the two translated sequences have any common amino acids for each location.
   (c) Compute the non-synonymous and synonymous substitution rates using MATLAB function *dnds*:

```
[dn ds] = dnds (seq1, seq2);
```

   (d) Does the fact both *seq1* and *seq2* end with a stop codon affect the synonymous substitution rate?

3. Issue the following commands to download and compare two GenBank sequences. Comment on the output. Experiment with the other parameters supported by the *dnds* command.

```
sq1 = getgenbank ('L11768');
sq2 = getgenbank ('L11770');
[dn dn dnv dsv] = dnds (sq1, sq2, 'verbose', true);
```

Chapter 16
Probabilistic Methods: Maximum Likelihood

Probabilistic methods for phylogeny aim at ranking trees according to the likelihood of observing the data (i.e. the multiple sequence alignment) given the topology of the tree. In order to compute the probability, the probabilistic tree construction methods estimate $P(x \mid T, t)$. Here the data is the set of n sequences (taxa), T is the tree and t denotes the specific edge lengths of the branches of the tree. To quantitatively define this probability, an underlying model of evolution is assumed.

16.1 Preliminary Example

The procedure for evaluating maximum likelihood is essentially similar to the one utilized for computing maximum parsimony. In theory, all possible tree topologies are considered and the overall likelihood for the entire alignment is computed in a manner similar to the computation of the tree length utilized in maximum parsimony. The number of possible tree topologies being exponentially large, efficient search procedures are utilized to yield the maximally likely tree, i.e. the tree that maximizes the probability of $P(data \mid tree)$. The probability of each column is computed under a given tree topologies. The overall probability is simply the product of the column probabilities under the assumption that the observations of individual columns are independent of each other.

Consider the simple alignment of four sequences in Fig. 16.1 (a). With n sequences there are $k = (2n-5)!!$ trees possible, labeled as $\{T_1, T_2 \ldots T_k\}$. Let us next illustrate the process of computing the likelihood of observing an alignment shown for one such tree T_i in Fig. 16.1 (b). The likelihoodd of observing each column of the alignment is next computed as shown in Fig. 16.1 (c), where the overall probability of data alignment given the tree is computed as a product of the individual column likelihoods. The likelihood for each n taxa tree configuration is similarly computed. The tree(s) yielding the higher likelihood value is chosen to be the **maximally likely tree**. Let

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_16

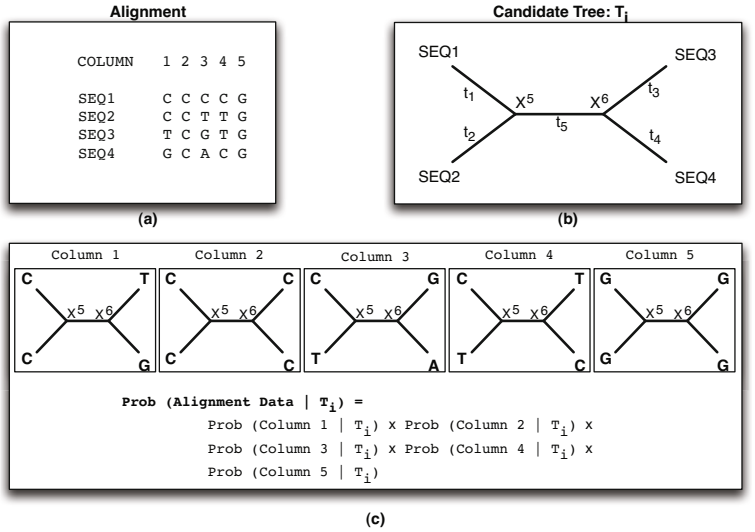


Fig. 16.1 An illustration of the Maximum Likelihood Method. For a given alignment with four sequences, all possible tree topologies will need to be explored to determine the topology that maximizes the likelihood of observing the data. The process of computing the likelihood is essentially to compute the probability of observing each column in the ungapped alignment computing the overall alignment probability by taking the product of individual likelihoods under the assumption that the column alignments are independent of each other.

us next look at the procedure for calculating the likelihood of a single column in the alignment.

16.2 Probabilistic Models of Evolution

A model of evolution helps us define the probability $P(x \mid y, t)$, that is, the probability of a sequence x arising from an ancestral sequence y over an edge of length t. During the course of evolution, residues are inserted and deleted as well as substituted for one another. In Maximum Likelihood (ML) methods for phylogenetic reconstruction, we make the simplifying assumptions that there are no insertions and deletions and that the substitution at a given site occur independent of the substitution at another site. Our sequences therefore must form an ungapped alignment with independent evolution at each site.

If we let $P(x_i \mid y_i, t)$ be the probability that a site with residue y_i in the parent sequence is substituted with a residue x_i over a branch of length t units of time, then the probability that the sequence y of length k will be substituted by a sequence x of the same length over t units of time is:

$$P(x \mid y, t) = \prod_{i=1}^{k} P(x_i \mid y_i, t) \tag{16.1}$$

Several matrices for nucleotide substitutions have been proposed. One such substitution model is proposed by Jukes and Cantor. This matrix provides the *rates* of substitution, and assumes that the nucleotide substitution rate is the same for all nucleotides. In Kimura's model, the substitution rates for transitional substitutions are different from transversional substitutions. The substitution matrices are shown in Table 16.1 below. For example, in the simplistic Jukes-Cantor model, all nucleotide substitution rates are constant and set to α. Thus, the rate that a nucleotide is preserved is (-3α). Nucleotide mutation rates and preservation rates under the Kimura model are determined based on transition and transversion.

Table 16.1 Jukes-Cantor (a) and Kimura (b) nucleotide substitution rate matrices. In Jukes-Cantor model, the rate at which a given nucleotide is substituted by another is equal irrespective of the nucleotide that the original nucleotide is substituted with. In Kimura model the rates of transitional substitutions (purine by purine and pyrimidine by pyrimidine) is different from transversional substitutions (pyrimidine by purine or vice versa).

	A	**C**	**G**	**T**
A	-3α	α	α	α
C	α	-3α	α	α
G	α	α	-3α	α
T	α	α	α	-3α

(a)

	A	**C**	**G**	**T**
A	$-(2\beta+\alpha)$	β	α	β
C	β	$-(2\beta+\alpha)$	β	α
G	α	β	$-(2\beta+\alpha)$	β
T	β	α	α	$-(2\beta+\alpha)$

(b)

If should be noted that these matrices provide the *rates* of substitution. The expected number of changes are computed as $(v \cdot t)$ where v is the rate of substitution. When t is small, the expected number of changes may be treated as a probability of change. For example, if we let $\alpha = 1$ change/million years, then the probability that a nucleotide will change in one year is 10^{-6}. Thus, the probability of a nucleotide changing in time Δt is $\alpha \cdot \Delta t$.

Let us next derive the fraction of nucleotides that will be different between two sequences when they diverge from the same common ancestor. Let r_t be the probability that the nucleotide in sequence X at a given position i does not change as the sequence evolves to another sequence Y over a period of t time units. Let us first denote a nucleotide A_k at position i in a sequence X as $X^i_{A_k}$. Thus, r_t measures the probability that the nucleotide in the evolved sequence Y is the same as the parent after t time units of evolutionary change has occurred. Also, s_t is the probability that the nucleotide at a given position has changed from A_k in the parent (sequence X) to another nucleotide A_j

in the child (sequence Y) during the t time units elapsed after the speciation event.

$$r_t = Prob(Y^i_{A_k} \mid X^i_{A_k}, t) \tag{16.2}$$

$$s_t = Prob(Y^i_{A_j} \mid X^i_{A_k}, t) \tag{16.3}$$

Considering the 4-character DNA alphabet, a nucleotide may be substituted by any of the other three nucleotides. Under the Jukes-Cantor model all three of these mutations are equally probable. Thus, the probability that a given nucleotide will mutate to one of the other is $\frac{1}{3}$ of the probability that a nucleotide will mutate which is $(1 - r_t)$.

$$s_t = \frac{1}{3}(1 - r_t) \tag{16.4}$$

We next develop the differential equations to compute $P(Y^i_{A_k} \mid X^i_{A_k}, t+\Delta t)$ – the probability that the child and the parent nucleotide stay the same after $t + \Delta t$ time units have elapsed since the speciation event. There are two possible chains of mutually exclusive events that must be considered to compute the overall probability $P(Y^i_{A_k} \mid X^i_{A_k}, t + \Delta t)$.

(a) In the first chain of events, the nucleotide at position i is the same as its parent after time t has elapsed – an event that has a probability of r_t; furthermore, the nucleotide does not undergo any subsequent change from time t to $t + \Delta t$ – an event that has a probability of $(1 - 3\alpha \cdot \Delta t)$. The joint probability of these two events is thus, $r_t \cdot (1 - 3\alpha \cdot \Delta t)$.

(b) In the second chain of events, the nucleotide at position i is different from its parent at time t – an event that has a probability of $3 \cdot s_t$; furthermore, the nucleotide undergoes a mutation from t to $t + \Delta t$ and becomes the same as the parent's nucleotide at position i – an event that has a probability of $\alpha \cdot \Delta t$. The joint probability of these two events is thus, $3 \cdot s_t \cdot \alpha \Delta t$ as there are three possible ways that the nucleotide could be different.

Since the two chain of events are independent of each other, the probability $r_{t+\Delta t}$ is equal to the sum of the two probability events listed above as shown in Eq. 16.5.

$$r_{t+\Delta t} = r_t \cdot (1 - 3\alpha \cdot \Delta t) + 3 \cdot s_t \cdot \alpha \Delta t \tag{16.5}$$

From this we can derive in Eq. 16.6 the *rate of change* of r_t as follows:

$$\begin{aligned} r_{t+\Delta t} &= r_t \cdot (1 - 3\alpha \Delta t) + 3 s_t \alpha \Delta t \\ &= r_t - 3\alpha r_t \Delta t + 3 s_t \alpha \Delta t \end{aligned}$$

$$\text{Substituting} \quad s_t = \frac{1}{3}(1 - r_t) \quad \text{we get}$$

$$r_{t+\Delta t} = r_t - 3\alpha r_t \Delta t + 3\frac{(1-r_t)}{3}\alpha\Delta t$$

Thus

$$r_{t+\Delta t} - r_t = -4\alpha\Delta t + \alpha\Delta t$$
$$\frac{\Delta r}{\Delta t} = -4\alpha r_t + \alpha$$

In the limit $\Delta t \to 0$,

$$\frac{dr}{dt} = -4\alpha r_t + \alpha \tag{16.6}$$

The above differential equation has solution of the form

$$r_t = C_0 e^{-4\alpha t} + C_1$$

The two boundary conditions are

$$at\ t = 0,\ r_t = 1$$
$$at\ t = \infty,\ r_t = \tfrac{1}{4}$$

The rationale for the boundary conditions is as follows. Initially, at time $t = 0$ the sequence X has not diverged from the parent and thus each position has 0 probability of being different from the parent. At time $t = \infty$, the probability that a nucleotide is the same as its parent is only 0.25 – what is expected by chance between any two random sequences.

The solution for r_t and s_t may thus be obtained as:

$$r_t = \tfrac{1}{4} + \tfrac{3}{4}e^{-4\alpha t} = \frac{1}{4}(1 + 3e^{-4\alpha t}) \tag{16.7}$$

$$s_t \quad = \tfrac{(1-r_t)}{3} \quad = \frac{1}{4}(1 - e^{-4\alpha t}) \tag{16.8}$$

It can readily be seen that as $t \to \infty$ the probability that a parent and the child nucleotide will match is $\frac{1}{4}$.

16.3 Likelihood - Two Sequences

This section considers the case where two sequences have diverged and we are given their gapless alignment. We can use the MLE method to estimate the time duration that separates them. Let us for example look at the sequence in Fig. 16.2 where the two sequences have diverged from the same common ancestor. Let us further assume that t_1 time units have elapsed since the divergence of *seq-1* and t_2 time units have elapsed since the divergence of *seq-2*. Given the two sequences and their alignment, our objective is to find the Maximally Likely Estimate (MLE) for the values of t_1 and t_2. In general, the computation of the MLE will require the estimation of the tree topology

as well as the branch lengths. The tree topology being fixed given that there are only two taxa in this case, the expression for MLE need only be optimized with respect to branch lengths.

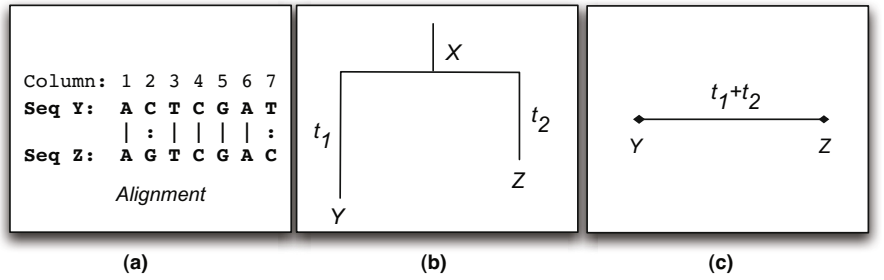


Fig. 16.2 The two sequences are aligned as shown in (a). The aligned sequences have diverged from the same common ancestor, X. The time duration elapsed from their divergence from the common ancestor X for sequences Y and Z are t_1 and t_2 respectively.

The MLE estimate for the entire alignment is simply the product of the MLE for each column in the alignment under our independent column assumption. Given the alignment, there are two cases possible for a column in the alignment. These are (1) the two nucleotides in the column agree as in columns 1 and 3 through 6, and (2) the two nucleotides in the column do not agree as in columns 2 and 7. Under the Jukes-Cantor model, the probability of observing all events of the type (1) will be the same; as will be the probability of observing all events of type (2). The likelihood of transitional mutations under the Kimura model will be higher than transversional mutations. However, under the Jukes-Cantor model these probabilities are the same as all mutations are equally likely and set equal to α in the rate matrix shown in Table 16.1.

Case I: The nucleotides in the alignment are identical

Let us consider the probability of observing (i.e. the likelihood) the data in column 1 of the alignment shown in Fig. 16.2 (a) under the tree shown in Fig. 16.2 (b). The nucleotide at column 1 in the parent sequence X could be a $\{A, C, T, G\}$ with a probability of $\frac{1}{4}$ each. The observations in sequences Y and Z after t_1 and t_2 are independent events and thus have a probability given by Eq. 16.9 below.

$$\begin{aligned} P(Y_A^1, Z_A^1 \mid T, t_1, t2) = & \tfrac{1}{4} \cdot P(Y_A^1 \mid X_A^1, t_1) \cdot P(Z_A^1 \mid X_A^1, t_2) + \\ & \tfrac{1}{4} \cdot P(Y_A^1 \mid X_C^1, t_1) \cdot P(Z_A^1 \mid X_C^1, t_2) + \\ & \tfrac{1}{4} \cdot P(Y_A^1 \mid X_G^1, t_1) \cdot P(Z_A^1 \mid X_G^1, t_2) + \\ & \tfrac{1}{4} \cdot P(Y_A^1 \mid X_G^1, t_1) \cdot P(Z_A^1 \mid X_T^1, t_2) \end{aligned} \quad (16.9)$$

The first term in the above equation corresponds to the event that column 1 in the parent sequence X is also the nucleotide A. The remaining terms correspond to it being nucleotides $\{C, G, T\}$. We can rewrite Eq. 16.9 using Eqs. 16.2 and 16.3 defined in the previous section.

$$P(Y_A^1, Z_A^1 \mid T, t_1, t2) = \frac{1}{4}(r_{t_1} \cdot r_{t_2} + 3 \cdot s_{t_1} \cdot s_{t_2}) \tag{16.10}$$

Further substituting the results obtained from the Jukes-Cantor model shown in Eqs. 16.7 and 16.8 into Eq. 16.10 above we get:

$$\begin{aligned} P(Y_A^1, Z_A^1 \mid T, t_1, t2) &= \frac{1}{4^3} \cdot (1 + 3e^{-4\alpha t_1}) \cdot (1 + 3e^{-4\alpha t_2}) + \\ &\quad 3 \cdot \frac{1}{4^3} \cdot (1 - e^{-4\alpha t_1}) \cdot (1 - 3e^{-4\alpha t_2}) \\ &= \frac{1}{16}(1 + 3e^{-4\alpha(t_1+t_2)}) \end{aligned} \tag{16.11}$$

Case II: The nucleotides in the alignment are different

For the analysis of this case, let us consider the probability of observing the data in column 2 of the alignment shown in Fig. 16.2 (a). The nucleotide at column 2 in the parent sequence X could again be A, C, T, orG, with the probability of $\frac{1}{4}$ each. The observations in sequences Y and Z after t_1 and t_2 are independent events and thus have a probability given by Eq. 16.12 below.

$$\begin{aligned} P(Y_C^2, Z_G^2 \mid T, t_1, t2) = &\tfrac{1}{4} \cdot P(Y_C^2 \mid X_A^2, t_1) \cdot P(Z_G^2 \mid X_A^2, t_2) + \\ &\tfrac{1}{4} \cdot P(Y_C^2 \mid X_C^2, t_1) \cdot P(Z_G^2 \mid X_C^2, t_2) + \\ &\tfrac{1}{4} \cdot P(Y_C^2 \mid X_G^2, t_1) \cdot P(Z_G^2 \mid X_G^2, t_2) + \\ &\tfrac{1}{4} \cdot P(Y_C^2 \mid X_G^2, t_1) \cdot P(Z_G^2 \mid X_T^2, t_2) \end{aligned} \tag{16.12}$$

The four terms in the above equation corresponds to the events that the column 2 in the parent sequence X is A, C, G, or T respectively. We can again rewrite Eq. 16.12 using the Eqs. 16.2 and 16.3.

$$P(Y_C^2, Z_G^2 \mid T, t_1, t2) = \frac{1}{4}(2 \cdot s_{t_1} \cdot s_{t_2} + \cdot r_{t_1} \cdot s_{t_2} + s_{t_1} \cdot r_{t_2}) \tag{16.13}$$

Further substituting the results obtained for the Jukes-Cantor model shown in Eqs. 16.7 and 16.8 into the Eq. 16.13 above we get:

$$\begin{aligned} P(Y_C^2, Z_G^2 \mid T, t_1, t2) = &\frac{2}{4^3} \cdot (1 - e^{-4\alpha t_1}) \cdot (1 - e^{-4\alpha t_2}) + \\ &\frac{1}{4^3} \cdot (1 + 3e^{-4\alpha t_1}) \cdot (1 - e^{-4\alpha t_2}) + \end{aligned}$$

$$\frac{1}{4^3} \cdot (1 - e^{-4\alpha t_1}) \cdot (1 + 3e^{-4\alpha t_2})$$
$$= \frac{1}{16}(1 - e^{-4\alpha(t_1+t_2)}) \quad (16.14)$$

As the likelihood of consensus alignment is the same irrespective of the actual nucleotides, and as the likelihood of nucleotide disagreement is the same irrespective of the nucleotides under the Jukes-Cantor model, the likelihood of observing an alignment where m nucleotides match and n nucleotide do not match in a two sequence alignment is given by Eq. 16.17.

$$P(Y, Z \mid X, t_1, t_2) = \frac{1}{16^{m+n}} \cdot (1 + 3e^{-4\alpha(t_1+t_2)})^m \cdot (1 - e^{-4\alpha(t_1+t_2)})^n \quad (16.15)$$

Thus, the likelihood for the sequence shown in Fig. 16.2 under the tree shown with $m = 5$ and $n = 2$ is:

$$P(Y, Z \mid X, t_1, t_2) = \frac{1}{16^7} \cdot (1 + 3e^{-4\alpha(t_1+t_2)})^5 \cdot (1 - e^{-4\alpha(t_1+t_2)})^2 \quad (16.16)$$

Note that the likelihood is independent of the individual time spans t_1 and t_2 and is a function only of $(t_1 + t_2)$. This goes to show that generating a rooted tree is not possible using the maximization of likelihood method. This is because we can only maximize the likelihood to yield an optimal value for $(t_1 + t_2)$. We can therefore replace our rooted tree shown in Fig. 16.2 (b) with an unrooted tree shown in Fig. 16.2 (c), with the divergence between taxon Y and Z set to $\tau = (t_1 + t_2)$. The parent X is therefore some ancestor along the link joining taxa Y and Z.

16.3.1 Maximizing the Likelihood

By replacing $\tau = (t_1 + t_2)$ in Eq. 16.17, it can be rewritten as follows:

$$\mathcal{L} = P(Y, Z \mid X, t_1, t_2) = \frac{1}{16^{m+n}} \cdot (1 + 3e^{-4\alpha\tau})^m \cdot (1 - e^{-4\alpha\tau})^n \quad (16.17)$$

Further, we can take the logarithm of the likelihood function which yields Eq. 16.18:

$$\log \mathcal{L} = k + m \log(1 + 3e^{-4\alpha\tau}) + n \log(1 - e^{-4\alpha\tau}) \quad (16.18)$$

We can next maximize the likelihood by setting $\frac{\partial \mathcal{L}}{\partial \tau} = 0$ as follows:

$$\begin{aligned} \frac{\partial \log \mathcal{L}}{\partial \tau} = & \quad \frac{-12\alpha m}{1+3e^{-4\alpha\tau}} + \frac{4\alpha n}{1-e^{-4\alpha\tau}} = \quad 0 \\ \Leftrightarrow & \quad \frac{3m}{1+3e^{-4\alpha\tau}} = \frac{n}{1-e^{-4\alpha\tau}} \\ \Leftrightarrow & \; 3(m+n)e^{-4\alpha\tau} = 3m - n \end{aligned}$$

$$\tau = -\frac{1}{4\alpha} \ln \frac{3m-n}{3(m+n)} \tag{16.19}$$

Thus the divergence between two sequences is obtained by the number of matching and non-matching nucleotides. Again, for our alignment in Fig. 16.2 with five matching and two mismatching nucleotides, the maximally likely value of the phylogenetic distance between sequences Y and Z is obtained by substituting $m = 5$ and $n = 2$ in Eq. 16.19:

$$Dist_{\text{MLE}}(Y, Z) = -\frac{1}{4\alpha} \ln \frac{13}{21}$$

It is instructive to note here that for this simple two sequence case the maximization of likelihood was possible analytically. This will not be the case in multiple sequence alignments with a larger number of sequences. Numerical methods, such as Newton's, secant or fixed-point, are often utilized to solve the partial differential equations used for maximizing the likelihood sequence data with respect to branch length. It may also be noted that all possible topologies for the given number of taxa must be explored. Search space pruning using branch and bound methods as applied in the maximum parsimony method is once again utilized in MLE based phylogenetic reconstruction methods. Thus, one would find the optimal branch lengths for each topology in the search space and ultimately pick the topology and the branch lengths that maximize the likelihood of observing the sequence data.

16.4 Likelihood for Ungapped Alignments

The previous section developed the methodology for formulating the likelihood of nucleotides in an alignment of two sequences. That approach is now extended to develop a maximum likelihood algorithm for n taxa. For example, one possible phylogenetic tree representation for a set of seven (7) sequences is shown in Fig. 16.3 (a). Recall that the maximum likelihood will be found by evaluating the probability of observed sequences calculated as the product of all column probabilities with the observed nucleotides assigned to leaves of the tree. For clarity, the leaves of the phylogenetic tree have been shown as circles and the internal nodes are shown as diamonds.

Fig. 16.3 (b) represents a node T_k and its left and right subtrees. These subtrees are rooted at nodes T_i and T_j. The branch lengths for the edges connecting T_k to T_i and T_j are t_i and t_j respectively. Let $P(T_k)$ be the probability of entire clade below T_k. Furthermore, let $P(T_k \mid z)$ be the probability of the entire subtree rooted at T_k given that the nucleotide at node T_k is z. Our objective is to represent $P(T_k \mid z)$ in terms of $P(T_i \mid x)$ and $P(T_j \mid y)$. Two cases need to be considered.

i $\underline{T_k \text{ is a leaf node}}$
If T_k is a leaf node, the probability $P(T_k \mid z)$ is 1 only when the observed

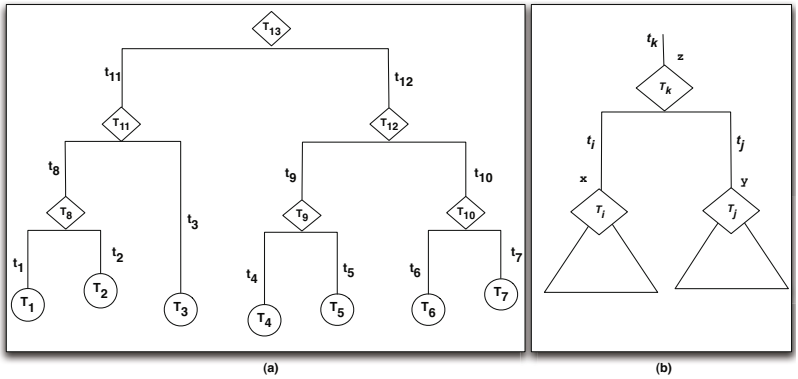


Fig. 16.3 Structure of a phylogenetic tree where the nodes and branches are annotated. (a) The leaf nodes are annotated by a circle and the internal parent nodes are represented by diamonds. With the exception of the root node, the branch adjacent to a node T_j is labeled as t_j. Each leaf node represents a column in the multiple sequence alignment of the taxa sequences used to infer the phylogenetic tree. (b) A generalized representation of a node T_k and its left and right subtrees that are rooted at nodes T_i and T_j respectively. The probability of finding a nucleotide a at node T_k can be written in terms of the probability of finding nucleotides b and c at nodes T_i and T_j respectively.

nucleotide from the sequence mapped to node T_k is identical to z. The probability $P(T_k \mid z)$ is zero for all other values of z.

ii T_k is an internal node
If T_k is a non-leaf internal node, the probability $P(T_k \mid z)$ is computed by considering the probabilities of $P(T_i \mid x)$ and $P(T_j \mid y)$ as shown in Eq. 16.20 below:

$$P(T_k \mid z) = \sum_{x,y} P(T_i \mid x) \cdot P(x \mid z, t_i) P(T_j \mid y) \cdot P(y \mid z, t_j) \qquad (16.20)$$

The basis for Eq. 16.20 is as follows. The residue z at node T_k could mutate to a residue x over a branch length of t_i with the probability $P(x \mid z, t_i)$. Similarly, residue z at node T_k could mutate to a residue y over a branch length of t_j with the probability $P(y \mid z, t_j)$. These probabilities are summed over all the 16 values that nucleotide x and y could take at nodes T_i and T_j respectively.

In section 16.2 above the conservation and mutation probabilities are defined under the Jukes-Cantor model based on the branch length t as parameters. The above two cases are thus represented using the conservation probability r_t and mutation probability s_t in Fig. 16.4. To illustrate the computation process, two situations are depicted in this figure. In Fig. 16.4 (a) the two left and right subtrees are themselves leaves. For any leaf, the

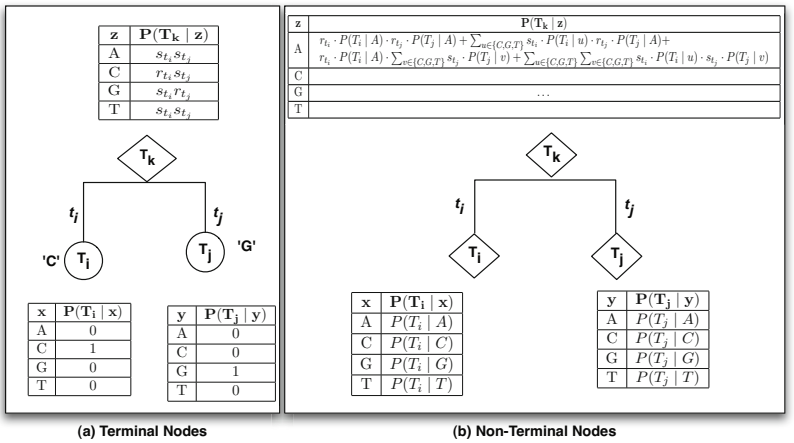


Fig. 16.4 The conditional probabilities are defined as $P(T_k \mid z)$, representing the conditional probability of the entire subtree under T_k given that the nucleotide at node T_k is z. The conditional probabilities for the root nodes are defined in (a). The node probabilities for an internal node such as is shown in (b) is computed by considering the conditional probability table of its children and the branch lengths interconnecting the parent to its left and right subtrees.

probability that the node is zero unless the nucleotide matches with the data associated with that node. Thus, the $P(T_i \mid x)$ is zero for nucleotides $\{A, G$ *and* $T\}$ as the data associated with the node T_i is a C. Similar probabilities are associated with the node T_j. These probabilities are utilized for the computation of $P(T_k \mid z)$. If we let $z = A$ for example, then 'A' must mutate to a 'C' over the branch length t_i with a probability of s_{t_i} and mutate to a 'G' over the branch length t_j with a probability of s_{t_j}. This yields the probability of $P(T_k \mid A) = s_{t_i} \cdot s_{t_j}$ as shown. Other probabilities in Fig. 16.4 (a) are similarly derived.

We next turn our attention to Fig. 16.4 (b) where the left and right subtrees are both internal nodes themselves. Thus, none of the subtree probabilities are zero. In general for DNA sequences there will be 16 terms used to define $P(T_k \mid z)$ corresponding to the summation over all the possible nucleotide-pairs that could exist at nodes T_i and T_j . The values for $P(T_k \mid A)$ is shown in Eq. 16.21.

$$P(T_k \mid A) = \begin{array}{l} r_{t_i} \cdot P(T_i \mid A) \cdot r_{t_j} \cdot P(T_j \mid A) + \\ \sum_{u \in \{C,G,T\}} s_{t_i} \cdot P(T_i \mid u) \cdot r_{t_j} \cdot P(T_j \mid A) + \\ \sum_{v \in \{C,G,T\}} r_{t_i} \cdot P(T_i \mid A) \cdot s_{t_j} \cdot P(T_j \mid v) + \\ \sum_{u \in \{C,G,T\}} \sum_{v \in \{C,G,T\}} s_{t_i} \cdot P(T_i \mid u) \cdot s_{t_j} \cdot P(T_j \mid v) \end{array} \tag{16.21}$$

One term out of the sixteen in Eq. 16.21 corresponds to the conservation of nucleotides in both subtrees. That is, an A is associated with both nodes T_i and T_j. There are six terms corresponding to the mutation in one branch and conservation in the other. Finally, there are nine terms corresponding to the mutation of the nucleotide at T_k over the two branches t_i and t_j. The conservation and mutation probabilities over a branch of length t_i are r_{t_i} and s_{t_i} respectively.

16.4.1 A Three Sequence Example

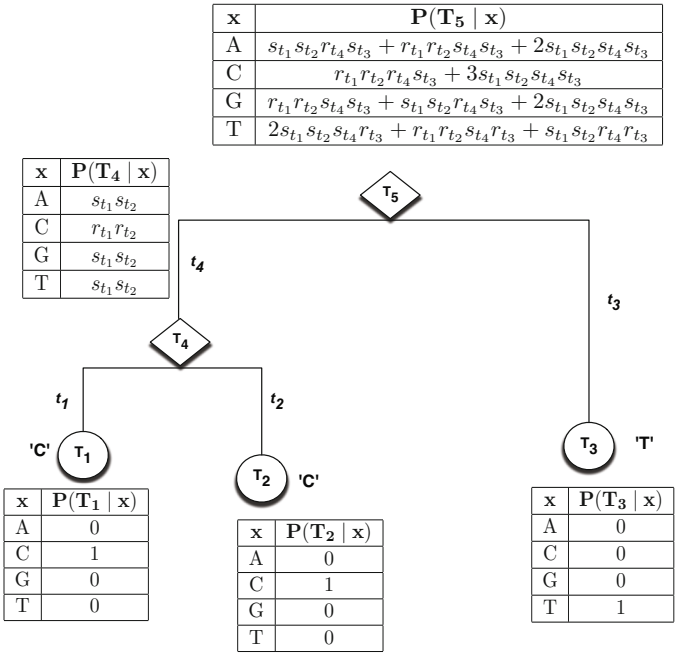


Fig. 16.5 A sample tree showing the likelihood of the first column of the three sequence alignment for the example shown

Consider the following alignment of three sequences:

```
COLUMN   1 2 3 4 5

SEQ1     C C C C G
SEQ2     C C T T G
SEQ3     T C G T G
```

The probabilities for the various nodes in column 1 of the above alignment are shown in Fig. 16.5. Similar expressions will be derived for the remaining four columns. The overall likelihood of observing the alignment under the

given phylogenetic tree would then be computed as a product of the column likelihoods assuming the observation in each column is independent of the others. The likelihood value thus obtained will next be maximized to yield the values of $t_1 \ldots t_4$. The probability of the root node $P(T_5)$ is given as follows:

$$P(T_5) = q_A P(T_5 \mid A) + q_C P(T_5 \mid A) + q_G P(T_5 \mid A) + q_T P(T_5 \mid A) \quad (16.22)$$

The above equation uses the *a-pirori* probabilities for the four nucleotides to compute the final probability of the tree. Generally, these probabilities are set equal to the relative frequency of the nucleotides in the aligned sequences. It should be noted that the likelihood of the tree is a function of the branch lengths. Therefore this function must be maximized by choosing the optimal values of branch length. Numerical methods are utilized for this purpose. Consideration of the logarithm of the likelihood function further reduces the complexity by converting the overall alignment likelihood function written as product of column likelihoods into a sum prior to the application of numerical analysis methods. The value of branch lengths and the optimal likelihood of the tree are saved. The optimal likelihood and corresponding branch lengths yield the maximum likelihood amongst all possible topologies. The topology yielding the maximum likelihood (and the associated branch lengths) is chosen as the ML phylogenetic tree.

Further Readings

1. Felsenstein, J.: Evolutionary trees from dna sequences: a maximum likelihood approach. J. Mol. Evol. 17(6), 368–376 (1981)
2. Olsen, G.J., Matsuda, H., Hagstrom, R., Overbeek, R.: fastdnaml: a tool for construction of phylogenetic trees of dna sequences using maximum likelihood. Comput. Appl. Biosci. 10(1), 41–48 (1994)
3. Yang, Z.: Paml: a program package for phylogenetic analysis by maximum likelihood. Comput. Appl. Biosci. 13(5), 555–556 (1997)
4. Xu, B., Yang, Z.: Pamlx: a graphical user interface for paml. Mol. Biol. Evol. 30(12), 2723–2724 (2013)
5. Yang, Z.: Maximum-likelihood estimation of phylogeny from dna sequences when substitution rates differ over sites. Mol. Biol. Evol. 10(6), 1396–1401 (1993)
6. Kosiol, C., Bofkin, L., Whelan, S.: Phylogenetics by likelihood: evolutionary modeling as a tool for understanding the genome. J. Biomed. Inform. 39(1), 51–61 (2006)

16.5 Exercises

1. Consider the following alignment of three sequences:

```
COLUMN   1 2 3 4 5

SEQ1     T C A T C
SEQ2     C C T T G
SEQ3     C C C T G
```

   (a) Compute the probabilities for the five columns in the above alignment.
   (b) Compute the overall likelihood of observing the alignment under each possible phylogenetic tree.
   (c) Identify the yielding the Maximum Likelihood (ML) and compute the branch lengths associated with the ML phylogenetic tree.

2. Download and compare two DNA sequences using MATLAB's function for computing synonymous and nonsynonymous substitution rates using maximal likelihood estimation.

```
s1 = getgenbank('L11768')
s2 = getgenbank('L11770')
[dn ds like] = dndsml(s1, s2)
```

   (a) Compare the substitution rates obtained with this method to those obtained with the character substitution methods discussed in the previous chapter.
   (b) What is the value of the return parameter *like*.
   (c) Comment on the significance of the return parameter *like*.

Chapter 17
Microarrays

A DNA Microarray or a Gene Chip comprises of a collection of microscopic spots where each spot is a DNA sequence probe representing a single gene. The DNA sequences representing at a specific location are arrayed on a solid surface and attached to a chemically suitable matrix through covalent bonds. DNA microarrays are often used for expression profiling and rely on the strength of DNA-DNA hybridization and DNA-RNA hybridization to detect and measure gene expression levels.

17.1 Introduction

Microarrays are extensively utilized to study the regulation of gene expression associated with a wide variety of basic biological functions such as development, hormonal signaling, and circadian rhythms. A rapidly growing area of application is cancer research, in which microarrays have helped researchers discover new tumor classes, assign patient samples to known tumor classes, reveal cancer-related alterations in molecular pathways, predict clinical outcomes, and identify new drug targets. Affymetrix expression arrays are widely used in biomedical research. The chips consist of sets of DNA probes carefully chosen to record expression of species specific genes.

Micro-arrays for Arabidopsis, Drosophila, Homo sapiens, Mus musculus, Pseudomonas, Rattus norvegicus, Saccharomyces cerevisiae are available from Affymetrix. Fig. 17.1 shows the phylogenetic tree schematic that specifies the GeneChip expression arrays that are available today from Affymetrix. However, the analysis of the microarray data continues to be a bottleneck compared to the microarray data generation rates; and researchers are exploring the expression of genes in an organism where the microarray probe set do not yet exist or need to be custom made. The situation is likely to become more serious in the future as DNA microarray technology becomes less expensive and projects increase both in number and in size. The fundamental question is whether one can assess the transcriptome of an organism using the microarray build using the transcriptome of another closely related organism.

G.B. Singh, *Fundamentals of Bioinformatics and Computational Biology,*
Modeling and Optimization in Science and Technologies 6, DOI: 10.1007/978-3-319-11403-3_17

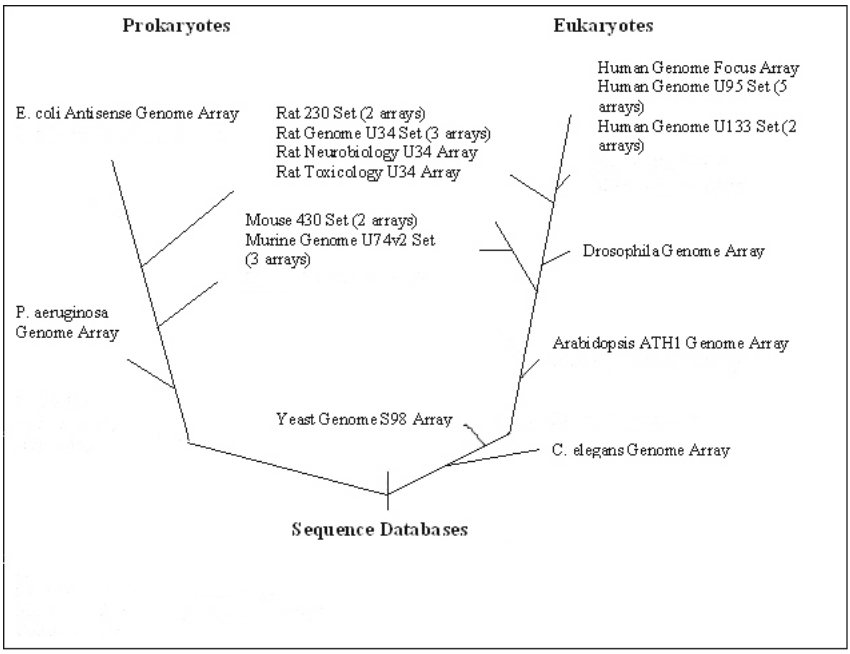


Fig. 17.1 Affymetrix micro-arrays are available for the species shown above. The numbers to the side of each species indicates the number of chips that are available for that particular species.

Affymetrix Gene Chips allow changes in gene expression to be measured for a very large number of genes. As illustrated in Fig. 17.2, each of the genes on the microarray is represented as a set of 16 probe pairs where each spot or cell is a 25-mer oligonucleotide and the intensity on each cell or spot is proportional to the amount of hybridized labeled cDNA that gets bound to it. The relative expression of a gene is represented with a signal value calculated using the intensities of the 16 probe pairs. Each probe pair comprises of a perfect match or PM spot, and a mismatch or MM spot. The PM spot contains an oligo probe that is 25 nt sequence that perfectly matches a 25 nt sequence from the gene, and the MM spot is a 25 nt oligo that has mismatch at the central nucleotide.

17.2 Affymetrix Microarrays

17.2.1 Terminology

Each gene or portion of a gene is represented by 11 to 20 oligonucleotides of 25 base-pairs.

Probe: an oligonucleotide of 25 base-pairs, i.e., a 25-mer.
Perfect match (PM): A 25-mer complementary to a reference sequence of interest (e.g., part of a gene).
Mismatch (MM): same as PM but with a single homomeric base change for the middle (13th) base (transversion purine¡-¿ pyrimidine, G $\rightleftharpoons$ C, A $\rightleftharpoons$ T).
Probe-pair: a (PM,MM) pair.
Probe-pair set: a collection of probe-pairs (11 to 20) related to a common gene or fraction of a gene.
AffyID: an identifier for a probe-pair set.

The purpose of the MM probe design is to measure non-specific binding and background noise.

17.2.2 Example Data

A wide rangle of demo data is available from the site:

```
http://www.affymetrix.com/support/technical/sample_data/
demo_data.affx
```

Chip Definition files for Affymetrix chips are avaiable from the following **library** site:

```
http://www.affymetrix.com/support/technical/
libraryfilesmain.affx
```

You can download and unzip this data and look for the file`XXX.CEL`, where XXX specifies some experimental conditions. This contains the actual cell intensities for the array. Affymetrix uses a large number of arrays. Each array comes with its unique definition. This file is named as **YYY.CDF**, where CDF is the cell definition of a given array type named YYY. You need both the cell intensites and the cell definitions to interpret the results of an experiment.

The expression level for any gene is computed using an average difference between average intensity bound to the perfectly matched oligonucleotides to the average intensity of mismatched oligonucleotides. The results of this difference are displayed as grid where each square represents a particular gene and the relative level of expression as a color intensity or a gray scale intensity.

17.3 Gene Data Matrix

MATLAB utilizes a specialized data structure, DataMatrix, to encapsulate data and metadata (row and column names) from a microarray experiment.

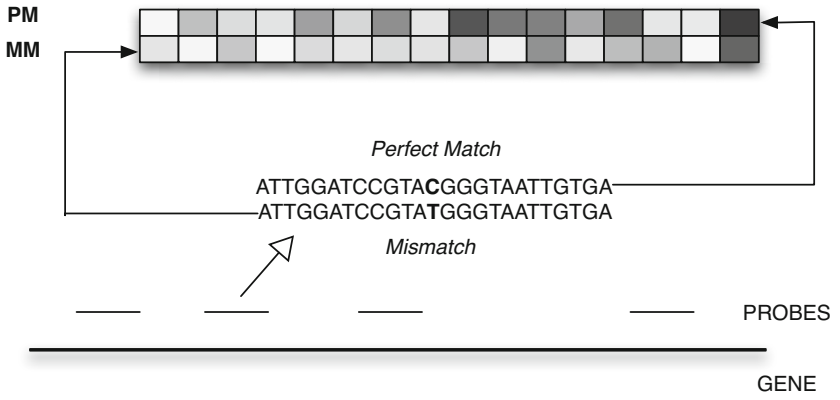


Fig. 17.2 Probes are selected from a set of predefined locations on the gene. Each probe set comprises of sixteen probes where each probe has two spots one corresponding to a perfect match and another to a mismatch. The mismatch probe contains a single mismatch located in the middle of the 25-base probe sequence, serving as a control since it is as likely to bind to nonspecific sequences as the perfect match probe. Thus, cross hybridization events events and other false signals are eliminated.

A DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers, or time values when the expression levels are measured. A DataMatrix object is created by specifying the expression data values, gene names or probe identifiers as the row names, and sample identifiers as the column names.

A sample MAT file `filteredyeastdata` contains three matrices. The first, *yeastdata* is a 614-by-7 matrix of gene expression data, the second is *genes* names as a cell array of 614 `GenBank` accession numbers, and *times*, a 1-by-7 vector of time values for labeling the columns.

The MATLAB code snippet below creates variables to contain a subset of the data, specifically the first five rows and first four columns of the *yeastvalues* matrix, the corresponding genes from *genes* cell array, and the time instants from *times* vector when the first four samples were captured. A DataMatrix object is next created using these matrices.

```
load filteredyeastdata
expressionVals = yeastvalues(1:5,1:4);
geneNames = genes(1:5,:);
sampleTimes = times(1:4);

import bioma.data.*;
```

```
dmo = DataMatrix(expressionVals, geneNames, sampleTimes);

>> dmo
dmo =

                    0        9.5      11.5       13.5
    SS DNA      -0.131     1.699     -0.026      0.365
    YAL003W      0.305     0.146     -0.129     -0.444
    YAL012W      0.157     0.175      0.467     -0.379
    YAL026C      0.246     0.796      0.384      0.981
    YAL034C     -0.235     0.487     -0.184     -0.669
```

The metadata for a DataMatrix object can be manipulated using a *get* and *set* method provided by the object. As an example, the *get* method can display the current properties.

```
get(dmo)
            Name: ''
        RowNames: {5x1 cell}
        ColNames: {'   0'  ' 9.5'  '11.5'  '13.5'}
           NRows: 5
           NCols: 4
           NDims: 2
    ElementClass: 'double'
```

A *set* similarly returns a modified object when a command such as the following is issued to change the `Name` property of the DataMatrix object:

```
dmo = set(dmo,'Name','YeastSubset');
```

It should be noted however that when a subset of a DataMatrix object is extracted for example by issuing a command belowe, the extracted subset DataMatrix is also a DataMatrix that includes the corresponding row and column labels.

```
>> dmo2 = dmo(3:5,1:3)
dmo2 =
                    0        9.5      11.5
    YAL012W      0.157     0.175      0.467
    YAL026C      0.246     0.796      0.384
    YAL034C     -0.235     0.487     -0.184
```

17.3.1 Microarray Analysis

In a microarray analysis experiment, the rows correspond to genes or probes, while the columns correspond to the values obtained for specific samples

where the samples are obtained from different tissues for example, of at different times to investigate the time dependence of gene expression after the administration of a specific drug for example. The `DataMatrix` data structure allows for a useful representation of a microarray experiment by enabling the storage of the gene or probe names as well as the sample labels. Furthermore, extraction of subsets matrices from the `DataMatrix` is accompanied by a transparent extraction of the gene and sample labels.

MATLAB provides several functions for analysis of microarray data. Many of these functions directly work with the `DataMatrix` object. In the examples below, we construct one data-matrix object containing the entire expression data, and two smaller data-matrix objects containing a subset of data.

The data set contains gene expressions at seven different time points. The first data-matrix object, `dmoEarly` is used for capturing the gene expression at the first three time points, and the second object `dmoLater` captures the gene expression data at the next three time points. The following code snippet constructs these objects.

Loading the predefined MAT file *filteredyeastdata* results in populating the workspace with three matrices: `yeastvalues` – a 614×7 matrix of real numbers representing the gene expression of 614 genes at 7 time points; `genes` – a 614×1 cell array where each cell array has the name of the 614 genes; and `times` – a 1×7 array with time points where the data was recorded.

```
load filteredyeastdata
>> dmoFull = DataMatrix(yeastvalues, genes, times);
>> dmoEarly = DataMatrix(yeastvalues(:,1:3), genes, times(1:3));
>> dmoLater = DataMatrix(yeastvalues(:,4:6), genes, times(4:6));
```

Microarray Box Plot: The purpose of this function is to display the variances of expression data around its mean in the various samples. As illustrated in Fig. 17.3, the maximum variance is observed towards the later timed samples. The following code snippet is used to generate a box-plot.

```
>> maboxplot (dmoFull);
```

Microarray Clustergram: Clustergram is probably the most widely utilized data analysis tool in microarray analysis. A clustergram essentially computes a hierarchical clustering of microarray data both along the rows and along the columns. Thereby two independent trees emerge – both being represented along the gene axis and along the sample axis.

In order to construct the gene- or probe-level clustergram, the pair-wise distance between each row in the microarray data is used to construct a clustering tree. Similarly, the pairwise distances between each column of the microarray data is used to build a sample-based clustering tree. A color coded representation is utilized to display the tree as illustrated in Fig. 17.4.

The following code snippet builds a smaller clustergram where the gene and sample labels are visible. A larger clustergram comprising of the entire gene expression values may also be built as illustrated.

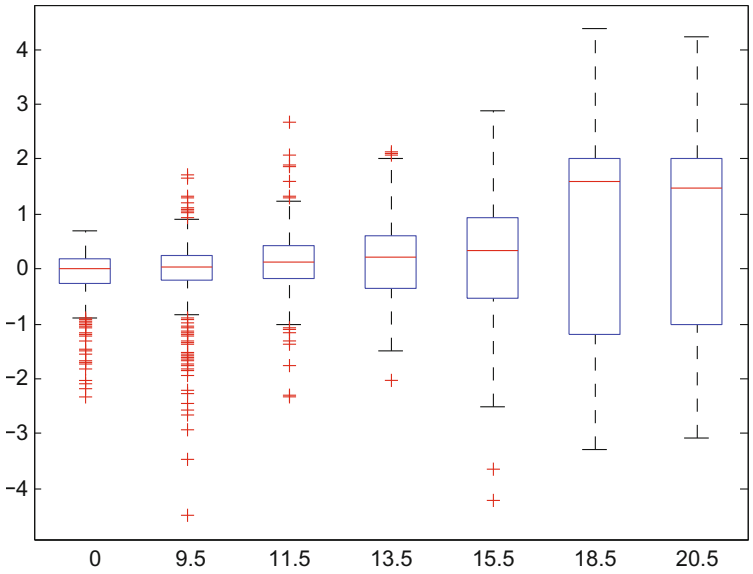


Fig. 17.3 A box plot of representing the variation in gene expression values for the seven samples taken at different times

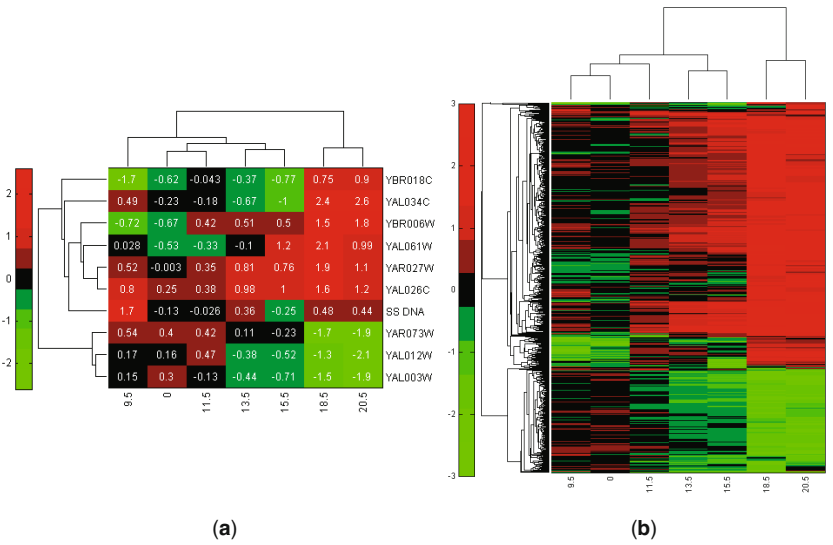


Fig. 17.4 Microarray clustergram representing hierarchical clustering along the gene (row) axis and the sample (column) axis. Smaller clustergram in (a) shows similarity of expression patterns in ten genes and seven samples. Gene names and distance measures are annotated. A clustergram for the entire microarray is illustrated in (b) which performs an analysis for the all 614 genes.

```
dmoSmall = DataMatrix(yeastvalues(1:10,:), genes(1:10), times);
clustergram(dmoSmall);
clustergram(dmoFull);
```

Principal Component Analysis: Principal component analysis may also be performed of the entire microarray experimental data. A representation of the sample points across the two principal components can help visualize any co-expression patterns. A principal component analysis of the entire gene-expression values is performed and illustrated in Fig. 17.5. Genes selected in one principal component axis are correspondingly labeled in the other axis so that similarities in clustering across multiple principal components becomes evident. Similarly, the gene names of the selected genes is displayed. The following code snippet is used to display principal component plot.

```
>> mapcaplot(dmaFull);
```

Comparing Expression Profiles: Distributions of expression patterns may be compared using the *t-statistic* using MATLAB function *mattest*. As illustrated in the Fig. 17.6 the expression profiles of the two data matrix objects captured in `dmoEarly` and `dmoLater` us compared by the following code snippet. The resulting plot and histogram determine if the expression profiles in the first three time samples in `dmoEarly` is different from the expression distribution in `dmoLater` as determined by the *t-statistic*.

```
>> mattest(dmoEarly, dmoLater, 'showplot', true. 'showhist', true);
```

17.4 Expression Data Sets

The Human Genome U133 (HG-U133) Set, consisting of two gene-chip arrays, contains approximately 45,000 probe sets representing more than 39,000 transcripts derived from approximately 33,000 well-substantiated human genes. The higher number of transcripts compared to the number of genes is representative of alternative splicing.

Each probe set is comprised of 16 probes where each probe is a 25-mer sequence from a human gene. Each 25-mer probe sequence is sampled from the sequences selected from GenBank, dbEST, and RefSeq (Baxevanis 2003). The sequence clusters were created from the UniGene database and refined by analysis and comparison with a number of other publicly available databases including the Washington University EST trace repository and the University of California, Santa Cruz Golden Path human genome database. A 25-mer probe pair is utilized at each location and the difference between the perfect match and the mismatch is used to determine the signal intensity.

Affymetrix software provides the aggregate intensity obtained from all the 16 probe pairs in a probe set. Analysis utilizes this aggregate intensity value for each of the probe sets indexed via the left column in Fig. 17.7. There are

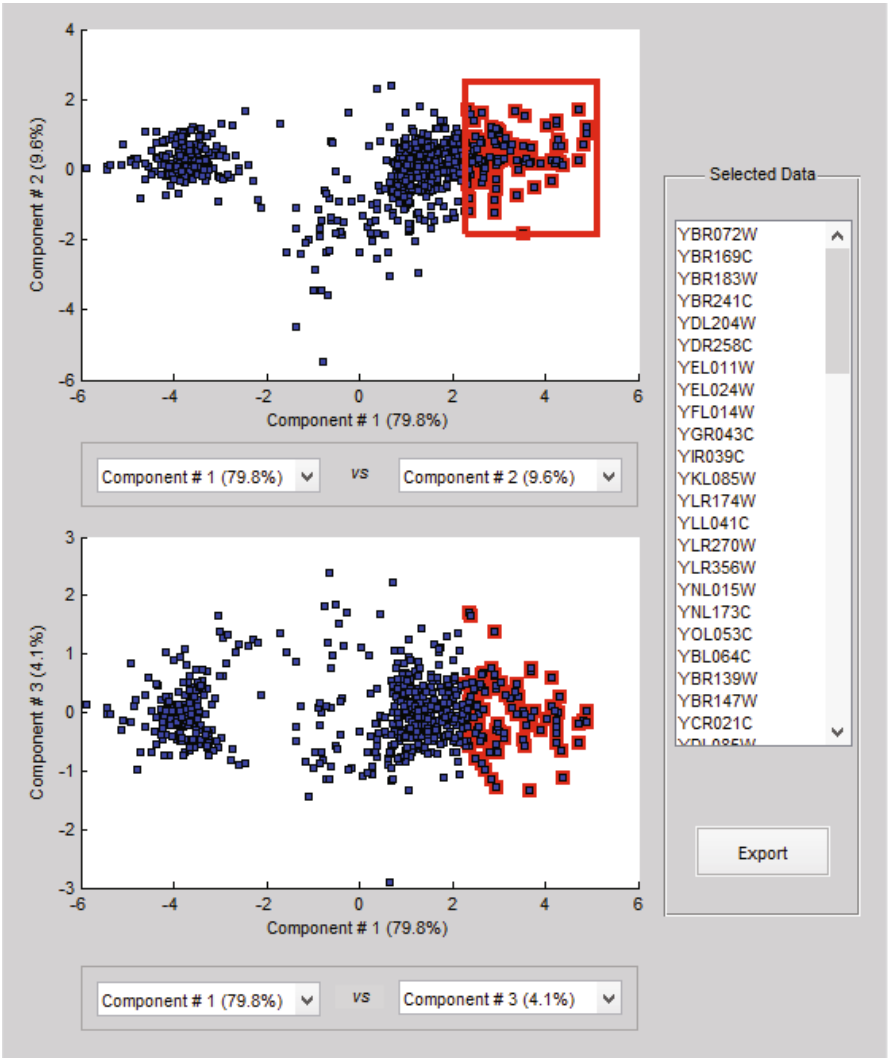


Fig. 17.5 The principal component analysis of a microarray expression matrix can demonstrate the clustering patterns in gene expression whereby the set of genes that belong to a cluster along a principal component may be identified

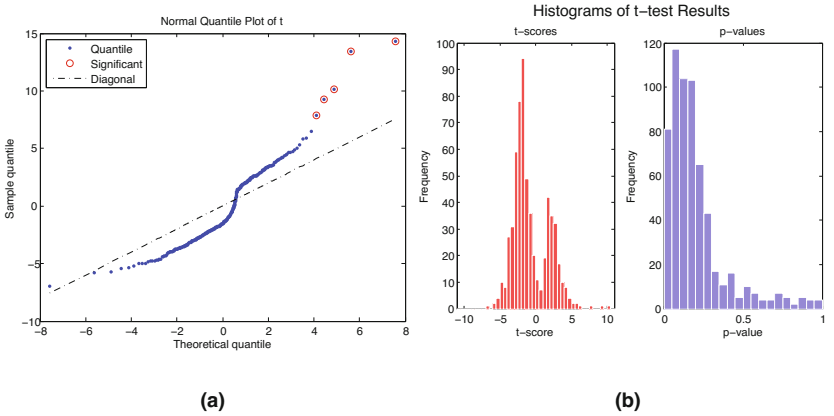


Fig. 17.6 Computation of *t-statistic* to determine if the expression distribution for each row in the first three time samples is significantly different from the expression distribution for the next three time samples. A *t-statistic* plot is shown in (a) and a histogram of *p-values* and *t-values* is shown in (b).

ProbeSet	H67Signal	H94Signal	H76Signal	Pan1Singal	Pan2Signal	Ggor1Signal	Ggor2 Signal	M27Signal	M40Signal
211326_x_at	66.9	32.9	79.7	17.9	24.9	15.8	20.5	44.4	100.4
211327_x_at	91.1	60.5	87.8	89	78.6	66.3	82.5	130	95
211328_x_at	111.5	124.6	120.9	123.7	117.8	160.7	117.3	164.3	199.8
211329_x_at	29.6	5.9	6.6	62.9	46.2	63.3	58.7	21.4	68.6
211330_s_at	25.1	25.3	47.3	32.7	20.7	42.1	35.8	33.1	32.8
211331_x_at	27.1	15.5	16.9	22.6	94.8	60.5	73.9	42.8	91.3
211332_x_at	102.9	71.4	123.7	133.2	96	121.7	101.2	269.7	218.7
211333_s_at	97.2	53.7	124.8	69.9	54.7	64.7	66.8	111.3	122.4
211334_at	17.3	15	7.5	25.7	12	35	29.3	23.4	28.7
211336_x_at	56.2	39.6	84.6	118.3	28.4	177.4	102.8	35.4	37.6
211337_s_at	259.7	298.9	397.7	694.8	432.1	123.6	152	557.8	707.1
211338_at	1.1	1	1.6	3.7	3.6	5.1	0.9	6.6	2.1
211339_s_at	94.7	50.5	49.6	76.3	32	57.8	76.2	108	123.1
211340_s_at	296.9	180.9	242.7	176.6	112.9	524.4	545.6	268.4	241.3
211341_at	2.6	0.8	2.8	4.1	3.1	3.9	4.3	3.4	2.4
211342_x_at	218.6	194.6	176.3	217.1	317	320.4	319.3	501.8	399.6
211343_s_at	1.1	19.1	39.7	27.1	47	12.9	54.2	18.2	48.3
211345_x_at	3789.3	3643.8	3396.7	3473.2	3929.8	3904.9	3950.4	3980	3943.9
211347_at	27	51.9	85.8	71.7	39.4	76.8	53.7	20.4	55.2
211348_s_at	95.4	88.1	111.5	3.4	42.1	47.5	79.8	7.1	7.9
211349_at	51.8	31.4	65.6	28.5	46.1	50.5	39.1	62.4	55.2
211350_s_at	79	62	10.9	58.6	40.9	14.5	44.6	108.3	100.4
211351_at	1.1	1.7	1.1	2.1	0.5	1.3	2.1	1.5	5.3
211352_s_at	6.2	33.9	9.2	3.9	5.2	17.4	23.2	14.1	38.1
211353_at	10.4	6.3	4	9.8	3.1	7.6	2.3	11.2	3.5
211354_s_at	72	56	81.9	21.6	33.5	73.6	45.3	56.9	55.1
211355_x_at	67.1	42.7	65.6	89.9	48.4	87.2	77.8	86.4	69.3
211356_x_at	111.6	92	86.6	90.1	60.7	84.4	92.6	115.5	67
211357_s_at	6.7	2.4	8.5	9.6	5.2	5.7	3.9	4.8	5.9
211358_s_at	121.3	135.5	123.8	190.7	148.2	201.2	141	149.2	171.5
211359_s_at	70.3	65	85.5	67	68.6	96.9	67.4	112	126.7
211360_s_at	22.4	22.2	25.1	40.8	57.7	14.6	70.7	80.4	69.7
211361_s_at	9.5	8.7	8.8	17.1	8.2	21.2	11.7	17.7	33.1
211362_s_at	79.7	38.5	68.8	109.9	59.3	82.6	89.5	99.4	138.3
211363_s_at	22.9	4.7	2.3	21.6	21.5	2.5	3.6	7.5	47.6

Fig. 17.7 A snapshot of hybridization results using Human Genome chip HG U133A. Samples for Humans: H67, H94 and H76; Chimpanzee: Pan1, Pan2; Gorilla: Ggor1, Ggor2; and Macaques: M27, and M40. Each of the two chips U133A and U133B contains roughly 20,000 probe sets annotated in the first column. Each probe set comprise of 11 probe-pairs per gene, where each probe is a 25-mer oligonucleotide. The intensities shown are an aggregate of all 11-probes in the set. The actual DNA sequences for each probe in a probe set are available from Affymetrix.

a total of 44,728 records corresponding to the total number of probe-sets and the nine (9) columns correspond to the samples shown in Fig.17.7.

17.5 MATLAB Support for Affymetrix Microarrays

17.5.1 Terminology

Each gene or portion of a gene is represented by 11 to 20 oligonucleotides of 25 base-pairs.
Probe: an oligonucleotide of 25 base-pairs, i.e., a 25-mer.
Perfect match (PM): A 25-mer complementary to a reference sequence of interest (e.g., part of a gene).
Mismatch (MM): same as PM but with a single homomeric base change for the middle (13th) base (transversion purine¡-¿ pyrimidine, G $\rightleftharpoons$ C, A $\rightleftharpoons$ T).
Probe-pair: a (PM,MM) pair.
Probe-pair set: a collection of probe-pairs (11 to 20) related to a common gene or fraction of a gene.
AffyID: an identifier for a probe-pair set.

The purpose of the MM probe design is to measure non-specific binding and background noise.

17.5.2 Example Data

A wide range of demo data is available from the site:
`http://www.affymetrix.com/support/technical/sample_data/demo_data.affx`

You can download and unzip this data and look for the file `XXX.CEL`, where XXX specifies some experimental conditions. This contains the actual cell intensities for the array. Affymetrix uses a large number of arrays. Each array comes with its unique definition. This file is named as **YYY.CDF**, where CDF is the cell definition of a given array type named YYY. Both the cell intensities and the cell definitions are needed to interpret the results of an experiment. MATLAB provides a function for viewing the CEL file image. The function

```
maimage(celStruct)
```

results in displaying the contents of the expression values displayed as an image as illustrated in Fig. 17.8:

In order to analyze the data in microarrays, the hybridization intensities need to be tagged with the the identify of the probe located at each grid point. Therefore, the CEL file and the CDF file should be rationalized. As illustrated in the steps below, after reading the CEL and CDF file, the function *probelibraryinfo* accomplishes this task.

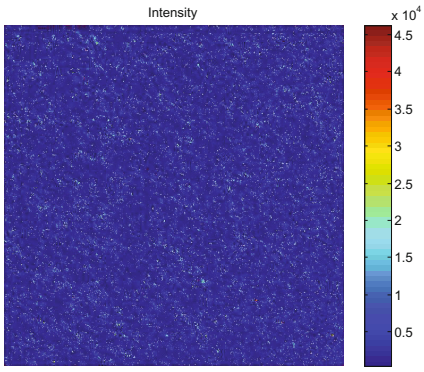


Fig. 17.8 Default display of *intensity* using the *maimage* function. The function may be used to display any of eight values stored values in the cell-structure specifying the corresponding column name. As for example, the command `maimage(celStruct, 'Pixels')` displays the pixel values as an image.

Read Cell Data Files

The first step in the process is to read the contents of a CEL file into a MATLAB structure. The CEL file is created by extracting the intensity of hybridization for each probe on the chip. The image data is next analyzed using an image analysis software after preprocessing and normalization using intensity of control probes. The intensity information for a given probe is an average of 16 or more individual sub-probes, or DNA strands located at each coordinate location. This average value is stored into CEL file.

```
celStruct = affyread('Mouse430A.CEL');
```

Members of the `CEL` file are shown below:

```
>> celStruct
celStruct =
               Name: 'Mouse430A.CEL'
           DataPath: 'Z:\singh\Data\Microarrays\M.Musculus'
            LibPath: 'Z:\singh\Data\Microarrays\M.Musculus'
       FullPathName: 'Z:\singh\Data\Microarrays\M.Musculus\Mouse430A.CEL'
           ChipType: 'MOE430A'
               Date: '10-Mar-2003 09:48:02'
        FileVersion: 3
          Algorithm: 'Percentile'
          AlgParams: 'Percentile:75;CellMargin:2;OutlierHigh:1.500;OutlierLow:1.004'
       NumAlgParams: 4
         CellMargin: 2
               Rows: 712
               Cols: 712
          NumMasked: 0
        NumOutliers: 873
          NumProbes: 506944
         UpperLeftX: 229
         UpperLeftY: 233
        UpperRightX: 4502
        UpperRightY: 278
```

```
         LowerLeftX: 192
         LowerLeftY: 4504
       LowerRightX: 4465
       LowerRightY: 4548
  ProbeColumnNames: {8x1 cell}
            Probes: [506944x8 single]
```

Note that the last element of this structure provides the probe data. There are 506,944 measured probe intensities in this array. As is evident from the cell array structure, this cell array belongs to a chip type of `MOE430A` and there are eight values captured for each of the 506,944 probes. As for the specific values captured for each probe, the column names are stored in the member *ProbeColumnNames* as shown below. The hybridization strength is measured by the `Intensity` value captured in the third column.

```
>> celStruct.ProbeColumnNames
ans =
    'PosX'
    'PosY'
    'Intensity'
    'StdDev'
    'Pixels'
    'Outlier'
    'Masked'
    'ProbeType'


>> celStruct.Probes(1,:)
ans =

  Columns 1 through 8
       0    0  183.300  24.1000   16.0000    0    0   1.0000
```

This demonstrates that the first probe came from location (0,0) on the cell array. Its intensity was 183.3 ± 24.1 averaged over 16 intensities corresponding to strengths of hybridization with 16 DNA strands affixed at coordinate $(0, 0)$ on the raw image.

Read Chip Definition File

The next step is to the contents of a corresponding Chip Definition, or a CDF file, into a MATLAB structure.

```
cdfStruct = affyread('MOE430A.CDF');
```

The *affyread* function also allows for the second argument to point to the folder where the definition file is stored if it is not stored in the current directory. Members of the `CDF` structure are shown below:

```
>> cdfStruct
cdfStruct =
                    Name: 'MOE430A.CDF'
                ChipType: 'MOE430A'
                 LibPath: 'z:\singh\Data\Microarrays\M.Musculus'
            FullPathName: [1x67 char]
                    Date: '29-Sep-2005 08:56:12'
                    Rows: 712
                    Cols: 712
            NumProbeSets: 22690
          NumQCProbeSets: 10
    ProbeSetColumnNames: {6x1 cell}
              ProbeSets: [22700x1 struct]
```

The definition file provides the information for the a chip with the same number of rows and columns and the chip type which matches with the corresponding information in the CEL structure. Most of the information in the file is about the probe sets. It should be noted that for this particular microarray, *Mouse 430A*, the number of probe sets (generally, there are multiple probes for each gene) is 22,690. That is, the entire set of 506,944 intensity measurements in the `CEL` file are hybridization strengths for 22,690 real probe pairs and 10 quality control probe pairs, or a total of 22,700 probe pairs. As an illustration, the contents of the 100^{th} probe-set is shown below:

```
>> cdfStruct.ProbeSets(100)
ans =

                Name: '1460646_at'
       ProbeSetType: 'Expression'
     CompDataExists: 0
           NumPairs: 11
        NumQCProbes: 0
             QCType: 0
         GroupNames: {'1460646_at'}
         ProbePairs: [11x6 int32]
```

This indicates that there are 11 probe-pairs with distinct pairs of locations on the CEL array where the probes corresponding to the probe with the name of `1460646_at` are gridded. The exact coordinates of these probes are found in the member `ProbePairs`. This is a matrix with one row for each probe pair and six columns. The information in the columns corresponds to the *ProbeSetColumnNames* of the CDF structure. There are a total of 11 probe pairs corresponding to the probe-set `1460646_at` – locations for each of the match and mismatch probe for each is shown below:

```
>> cdfStruct.ProbeSetColumnNames

ans =
    'GroupNumber'
```

```
    'Direction'
    'PMPosX'
    'PMPosY'
    'MMPosX'
    'MMPosY'

>> cdfStruct.ProbeSets(100).ProbePairs(:,:)
ans =

1           2         393         221         393         222
1           2         567         217         567         218
1           2         644         565         644         566
1           2         443         219         443         220
1           2         504         323         504         324
1           2         326           1         326           2
1           2         104         487         104         488
1           2         542         449         542         450
1           2         570         201         570         202
1           2         579         161         579         162
1           2          20         579          20         580
```

The third and fourth columns give the X and Y coordinates of the PM or Perfect Match probe, and the fifth and sixth column provide the coordinates of the MM or Mismatch probe on the chip for each of the probe-pairs (a probe pair is the pair of match and mismatch probes). These probe coordinates may be used to look up the values for a probe from the `celStruct` – the CEL array structure we read with the intensity values.

The code segment shown below looks up the index of the match and mismatch probes corresponding to the first probe-pair in the CDF file for probe-set 100.

```
>> PMX = cdfStruct.ProbeSets(100).ProbePairs(1,3);
>> PMY = cdfStruct.ProbeSets(100).ProbePairs(1,4);
>> matchProbe = find((celStruct.Probes(:,1) == PMX) & ...
                       (celStruct.Probes(:,2) == PMY))

>> MMX = cdfStruct.ProbeSets(100).ProbePairs(1,5);
>> MMY = cdfStruct.ProbeSets(100).ProbePairs(1,6);
>> mismatchProbe = find((celStruct.Probes(:,1) == MMX) & ...
                       (celStruct.Probes(:,2) == MMY))

matchProbe =
      157746

mismatchProbe =
      158458
```

Next, all the information about these two probes, the match probe located at index 157746, and the mismatch probe located at index 158458 is looked

up from the CEL structure follows. Note that the coodinate locations match with the information provided in the CDF file.

```
>> celStruct.Probes(matchProbe,:)
ans =
  393.0000  221.0000  364.3000   36.1000   16.0000    0
0    1.0000

>> celStruct.Probes(mismatchProbe,:)
ans =
  393.0000  222.0000  163.3000   21.3000   16.0000    0
0    1.0000
```

This demostrates that for the match probe intentsity is 364.3 ± 36.1 and the mismatch probe intensity is 163.3 ± 21.3. Correspondingly, as far as the first probe in this probe set indicates strength of hybridization with the match probe is over $2\times$ more than the mismatch probe strongly suggesting the presence of the corresponding probe set, and possibly the gene, in the sample.

Correlate Probe Information from Two Files

The process of reconcilation information contained in these two structures, as illustrated in the example above, must be accomplished for the entire array. The command *probelibraryinfo* accomplishes this task. It creates a table of information linking the probe data from CELStruct, a structure created from reading an Affymetrix CEL file, with probe set information from CDFStruct, a structure created from reading an Affymetrix CDF file.

```
plinfo = probelibraryinfo(celStruct, cdfStruct);
```

This command returns a three values for each of the probe locations. The first value is the index of the probe set to which the probe belongs. The second value contains the probe pair number since there are more than one instance of each probe pair scattered all throughout the microarray, and the third value indicates whether the probe is a perfect match (1) or mismatch (-1) probe.

For the example discussed above, the *probelibraryinfo* function will generate a $506,944 \times 3$ vector where each row corresponds to a specific probe on the CEL file. The first column of this matrix lists the identifier corresponding to the probe-pair for refering to the CDF file where details of that probe-pair, and the gene it represents, are stored. Thus, for example the $10,000^{th}$ reading of the CEL array corresponds to the probe pair

```
>> cdfStruct.ProbeSets(plinfo(10000,1)).Name

ans =
   1450595_at
```

We can also verify that the match and mismatch probes form our earlier example refer to the same probe:

```
>> cdfStruct.ProbeSets(plinfo(matchProbe)).Name
ans =
 1460646_at

>> cdfStruct.ProbeSets(plinfo(mismatchProbe)).Name
ans =
 1460646_at
```

You can search through the structure for a particular probe set. Alternatively, you can use the function `probesetlookup` to find out information the gene name for a probe set. The member `GINIndex` refers to the index of the gene from the `GIN` file discussed next.

```
>> info = probesetlookup(cdfStruct, '1460646_at')
info =
      Identifier: 'gb:NM_009974.1 '
    ProbeSetName: '1460646_at'
        CDFIndex: 100
        GINIndex: 22591
     Description: [1x282 char]
          Source: ''
       SourceURL: ''

>> info.Description
ans =
gb:NM_009974.1 /DB_XREF=gi:6753539 /GEN=Csnk2a2 /FEA=FLmRNA /CNT=104 /TID=Mm.28881.1
/TIER=FL+Stack /STK=60 /UG=Mm.28881 /LL=13000 /DEF=Mus musculus casein kinase II,
alpha 2, polypeptide (Csnk2a2), mRNA. /PROD=casein kinase II, alpha 2, polypeptide
/FL=gb:NM_009974.1 gb:AF012251.1
```

The function probesetvalues does the reverse of this lookup and creates a matrix of information from the CEL and CDF structures containing all the information about a given probe set. This matrix has 18 columns corresponding to ProbeSetNumber, ProbePairNumber, UseProbePair, Background, PMPosX, PMPosY, PMIntensity, PMStdDev, PMPixels, PMOutlier, PMMasked, MMPosX, MMPosY, MMIntensity, MMStdDev, MMPixels, MMOutlier, and MMMasked. Try this command:

```
psvals = probesetvalues(celStruct,cdfStruct,'1460646_at')
```

Gene Names and Probe Set IDs

The Affymetrix probe set IDs are not particularly descriptive. The mapping between the IDs and the gene names is stored in the GIN file. This is a text file so you can open it in an editor and browse through the file, or you can use affyread to read the information into a structure.

```
>> ginStruct = affyread ('MOE430A.GIN')
ginStruct =
            Name: 'MOE430A'
         Version: 2
    ProbeSetName: {22690x1 cell}
              ID: {22690x1 cell}
     Description: {22690x1 cell}
     SourceNames: ''
       SourceURL: ''
        SourceID: 0
```

Though it does not have to be the same, the GINIndex is also 100 in this case. Whatever is the GINIndex, we can look up the information on the gene from the structure. For the example above, the information for the probeset for `1460646_at` indicates that GINIndex to be 22591. This can be used to lookup the information on the identified gene:

```
>> ginStruct.ProbeSetName{22591}
ans =
    '1460646_at'

>>ginStruct.Description{22591}
ans =
gb:NM_009974.1 /DB_XREF=gi:6753539 /GEN=Csnk2a2 /FEA=FLmRNA /CNT=104 /TID=Mm.28881.1
/TIER=FL+Stack /STK=60 /UG=Mm.28881 /LL=13000 /DEF=Mus musculus casein kinase II,
alpha 2, polypeptide (Csnk2a2), mRNA. /PROD=casein kinase II, alpha 2, polypeptide
/FL=gb:NM_009974.1 gb:AF012251.1
```

17.5.3 Utility Functions

- **PSStruct = probesetlookup(AffyStruct, ID)**: returns a structure containing information for a probe set specified by ID. Example:
  `probesetlookup(cdfStruct,'1460646_at')`
- **PSValues = probesetvalues(CELStruct, CDFStruct, PS)** creates a table of intensity values for PS, a probe set, from the probe-level data in CELStruct, a structure created by the affyread function from an Affymetrix CEL file. Example:
  `probesetvalues (celStruct, cdfStruct,'1460646_at')`
- **probesetplot(CELStruct, CDFStruct, PS)**: plots the PM (perfect match) and MM (mismatch) intensity values for a specified probe set. Example:

  ```
  {probesetplot(celStruct, cdfStruct, '1460646_at','showstats', true)
  ```

  The result of this command is shown in Fig. 17.9 where the intensity plot depicts the observed intensity for each of the 11 probe pairs associated with the probe set `1460646_at`.

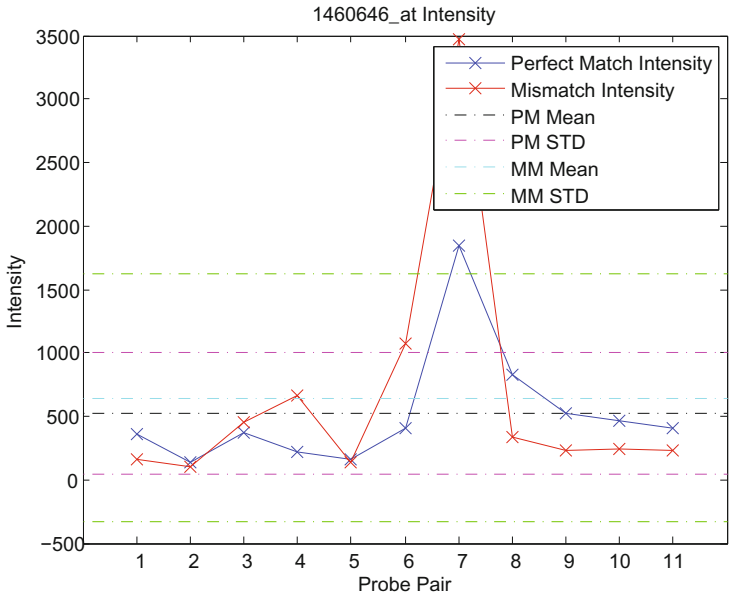


Fig. 17.9 Illustrates the hybridization strengths for each of the instances of the probe pairs gridded on the microarray

- **probesetlink(AffyStruct, PS)** opens a Web Browser window displaying information on the NetAffx Web site about a probe set specified by PS, a probe set index or the probe set ID/name, and AffyStruct, a structure created from an Affymetrix CHP file or Affymetrix CDF library file.
  The following example assumes that the `MOE430A.CHP` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `MOE430A.CDF`, is stored at `D:\Affymetrix\LibFiles\Mouse`.
  Read the contents of a CHP file into a MATLAB structure.
  ```
  chpStruct = affyread('MOE430A.CHP', 'D:\Affymetrix\LibFiles\Mouse');
  ```
  Display information from the NetAffx Web site for the `1460646_at` probe set.
  ```
  probesetlink(chpStruct,'1460646_at')
  ```

17.6 Gene Expression Omnibus (GEO)

Gene Expression Omnibus (GEO), accessible at the NCBI at `http://www.ncbi.nlm.nih.gov/geo/` is an international public repository of high throughput microarray data containing, at the time of writing, over a million samples from over a hundred organisms. GEO organizes gene chip data into

Table 17.1 MATLAB Affymetrix Parameters and their Explanations

1	ProbeSetNumber	Number identifying the probe set to which the probe pair belongs
2	ProbePairNumber	Index of the probe pair within the probe set
3	UseProbePair	This field is for backward compatibility only and is not currently used
4	Background	Background-adjusted probe intensity values of the probe pair
5	PMPosX	x-coordinate of the perfect match probe
6	PMPosY	y-coordinate of the perfect match probe
7	PMIntensity	Intensity value of the perfect match probe
8	PMStdDev	Standard deviation of intensity value of the perfect match probe
9	PMPixels	Number of pixels in the cell containing the perfect match probe
10	PMOutlier	True/false flag indicating if the perfect match probe was marked as an outlier
11	PMMasked	True/false flag indicating if the perfect match probe was masked
12	MMPosX	x-coordinate of the mismatch probe
13	MMPosY	y-coordinate of the mismatch probe
14	MMIntensity	Intensity value of the mismatch probe
15	MMStdDev	Standard deviation of intensity value of the mismatch probe
16	MMPixels	Number of pixels in the cell containing the mismatch probe
17	MMOutlier	True/false flag indicating if the mismatch probe was marked as an outlier
18	MMMasked	True/false flag indicating if the mismatch probe was masked
19	GroupNumber	Number identifying the group to which the probe pair belongs
20	Direction	Number identifying the direction of the probe pair

`Platforms`, `Samples`, and `Series`. A `Platform` contains information about microarray chip, such as an *Affymetrix* chip, that was used for a specific functional genomic study. The GEO data may contain many `Samples` belonging to a particular `Platform`. A `Sample` is the fundamental data point for an experiment. It's where individual hydridization values are recorded. From an experimental standpoint, each `Sample` in turn will be typically be part of many samples collected for an experiment. Collectively, all related samples belong to a `Series` of samples collected for a functional genomic study. A `Sample` may be a part of multiple studies. Thus, As illustrated in the schema in Fig. 17.10, there is a One-Many or 1:N relationship between with `Platform` and `Samples`, and a Many-Many or M:N relationship between `Samples` and `Series`.

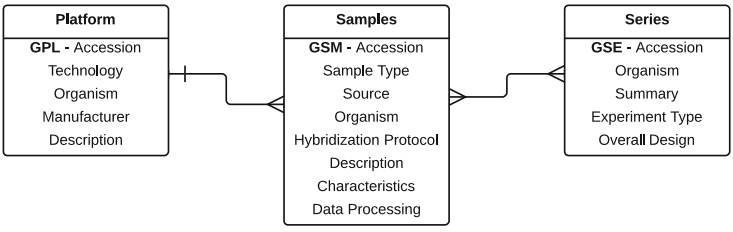


Fig. 17.10 Organization of microarray data in the Gene Expression Omnibus (GEO). A *Platform* contains information about the microarray, *Sample* contains experimental data, and *Series* about the collective group of samples produced by a specific functional genomic experiment.

17.6.1 Platform

A Platform record contains summary description of the array or sequencer. The record also stores a data table defining the array template is stored for an array-based Platform. Each Platform record is assigned a unique and stable GEO accession number. All platform records have an accession number beginning with the three letter code of **GPL** and have the format of (GPLxxx), where xxx represents a number. A Platform could reference many Samples. Each sample could have been been submitted by different submitter.

MATLAB provides functions for downloading a sample, series and platform data using the function `getgeodata` with a corresponding accession number. As illustrated, the type of data is implicit in the accession number and therefore the same function is used to access the different type of data-sets.

```
>> geoPlatform = getgeodata('GPL74')
geoPlatform =
                  Scope: 'PLATFORM'
              Accession: 'GPL74'
                 Header: [1x1 struct]
    ColumnDescriptions: {16x1 cell}
           ColumnNames: {16x1 cell}
                  Data: {2059x16 cell}
```

17.6.2 Samples

A Sample contains information about the actual experiment including the conditions under which an individual Sample was handled, the different types of processing and manipulations that were applied to it, and the set of measurements of the abundance of each biological product observed during the experiment. Each Sample record is assigned a unique and stable GEO accession number. All sample records have an accession number beginning with the three letter code of **GSM** and have the format of (GSMxxx), where xxx represents a number. While a Sample can be included in multiples Series, it must reference only one Platform.

Again MATLAB function *getgeodata* can retrieve Sample data from the GEO database.

```
>>  geoSample = getgeodata('GSM1768')
geoSample =
                  Scope: 'SAMPLE'
              Accession: 'GSM1768'
                 Header: [1x1 struct]
    ColumnDescriptions: {3x1 cell}
           ColumnNames: {3x1 cell}
                  Data: {1656x3 cell}
```

17.6.3 Series

A Series record links together a group of related Samples. A Series then is the thread that runs through an entire group of samples providing a basis for evaluating and understanding the significance of a specific biomarker. It serves as a focal point of the whole functional genomic study. Series records may also contain tables describing extracted data, summary conclusions, or analyses. All Series records have an accession number beginning with the three letter code of **GSE** and have the format of (GSExxx), where xxx represents a number.

As illustrated below, MATLAB function *getgeodata* retrieve Series data from the GEO database.

```
>> geoSeries = getgeodata('GSE1441')
geoSeries =
    Header: [1x1 struct]
      Data: [9128x35 bioma.data.DataMatrix]
```

GEO Data-Sets

GEO also contains some curated series focused on specific studies of biological interest. The curated data is encapsulated as a `DataSet` object. A `DataSet` contains collection of biologically and statistically comparable GEO `Samples`. Information related to the experimental factors is also included within GEO `DataSet`. GEO web interface allows users to search for `DataSets` studies relevant to their interests.

A `DataSet` is used by GEO's suite of data display and analysis tools. GEO `DataSets` records have an accession number of the form (GDSxxx). Samples within a DataSet refer to the same Platform. The advantage of using a `DataSet` object is that the value measurements for each Sample within a DataSet are assumed to be calculated in an equivalent manner. That is, they are normalized using considerations such as background processing to make the values consistent across the DataSet.

GEO `DataSets` form the basis of GEO's advanced data display and analysis tools. Some of the advanced analyses applicable on `DataSets` includes gene expression profile charts and clustering. Illustrated below is an example of retrieving GEO `DataSets` using MATLAB function *getgeodata*. In this example, the data-set is saved to a text file.

```
>> geoDataSet = getgeodata('GDS2602','tofile','gds2602.txt')

geoDataSet =
                  Scope: 'DATASET'
              Accession: 'GDS2602'
                 Header: [1x1 struct]
    ColumnDescriptions: {30x1 cell}
```

```
ColumnNames: {30x1 cell}
      IDRef: {16896x1 cell}
 Identifier: {16896x1 cell}
       Data: [16896x30 double]
```

Profiles

Researchers are often interested in longitudinal studies where the expression of an individual gene across a samples is sought. Since a `DataSet` contains information on a series of related samples, a the `Profile` object provides expression measurements for an individual gene across all samples in a `DataSet`. GEO web interface allows searching for Profiles. Some of the tools available for profiles include the ability to find profile neighbors and homologs based on profiles.

Further Readings

1. Liu, D., Sartor, M.A., Nader, G.A., Pistilli, E.E., Tanton, L., Lilly, C., Gutmann, L., IglayReger, H.B., Visich, P.S., Hoffman, E.P., Gordon, P.M.: Microarray analysis reveals novel features of the muscle aging process in men and women. J. Gerontol. A. Biol. Sci. Med. Sci. 68(9), 1035–1044 (2013)
2. Faith, J.J., Driscoll, M.E., Fusaro, V.A., Cosgrove, E.J., Hayete, B., Juhn, F.S., Schneider, S.J., Gardner, T.S.: Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. Nucleic Acids Res. 36(Database issue), D866–D870 (2008)
3. Bellazzi, R., Zupan, B.: Towards knowledge-based gene expression data mining. J. Biomed. Inform. 40(6), 787–802 (2007)
4. Barrett, T., Edgar, R.: Mining microarray data at ncbi's gene expression omnibus (geo)*. Methods Mol. Biol. 338, 175–190 (2006)
5. Zhong, S., Li, C., Wong, W.H.: Chipinfo: Software for extracting gene annotation and gene ontology information for microarray analysis. Nucleic Acids Res. 31(13), 3483–3486 (2003)
6. Tamames, J., Clark, D., Herrero, J., Dopazo, J., Blaschke, C., Fernández, J.M., Oliveros, J.C., Valencia, A.: Bioinformatics methods for the analysis of expression arrays: data clustering and information extraction. J. Biotechnol. 98(2-3), 269–283 (2002)
7. Edgar, R., Domrachev, M., Lash, A.E.: Gene expression omnibus: Ncbi gene expression and hybridization array data repository. Nucleic Acids Res. 30(1), 207–210 (2002)

17.7 Exercises

1. Collect and write information about the U133. With mRNA probes coming only from the humans, do you think the expression levels obtained for our close cousins will be accurate? Justify.
2. The site `http://www.affymetrix.com/products_services/arrays/` provides additional information on Affymetrix microarrays. Conduct research on this and other sites and complete the following table with regards to the current level of integration of probes in microarrays:

Table 17.2 Current level of Array Integration

Array Format	
Feature Size	
Total Number of Distinct Probes	
Oligonucleotide Probe Length	
Gene-level Probe Sets with Ensembl Support	
Gene-level Probe Sets with Putative Full-length Transcript Support (GenBank and RefSeq)	

   As an example, there were 28,000 gene level probes integrated into a microarray.
3. A cladogram depicts individual genes along its rows and some property of these genes along its column. In this particular case, the property depicted is the expression level of a gene at a certain time after an event. An example of such an event would be the time when a therapeutic, such as a medication, is administered. The rows in a cladogram are arranged to cluster genes whose time dependent expression profiles are in agreement. Now, answer the following questions:
   (a) What is the significance of the first column (labeled 0-hour) being colored black. That is, what does it tell you about the relative level of expressions of the genes being studied?
   (b) With respect the gene expression a few hours after the administration of the therapeutic, are the genes colored in red expressed more or less than the genes colored in green. What is the approximate ratio of their expression level.
   (c) Looking at the smaller cladogram where the genes have been labeled, name the genes, if any, that whose expression profiles are similar.

4. You are provided Affymetrix gene-chip data on on *A. Thalinia* and *H. Sapiens*. Using the functions provided in Matlab, complete the following

table. For this exercise you need to only use the information U133A gene chip, and ignore U133B chip data. (Note that for completing the last row of the table, you will need to conduct some research to determine the number of genes in each of the two organisms)

Size of Arrays (rows × columns)	*A. Thalinia*	*H. Sapiens*
Number of Functional Probes		
Number of Control Probes		
Number of Functional Probe Sets		
Number of Control Probe Sets		
Number of genes probed by the chip		
Total number of genes (*estimated*)		

5. The following genes have been implicated in breast cancer:
`BRCA1, BRCA2, P53, CHEK2, PALB2, FGFR2, TNRC9, MAP3K1, LSP1, TOX3, AKAP9, ATM, STK11, CDH1, PALB2, BR1P1`
Using the information of the genes annotation file for U133A, determine which of the genes listed above are present on the U133A gene chip.

6. (Research Question) You are given a CEL definition file for U133B gene chip. Try and locate the CDF and GIN files for this chip. Using these files, determine the number of genes probed using U133B. Are there any genes that are probed by both the U133A and U133B gene chips. What significance, if any, is associated with U133B chip.

Appendix A
Matlab

MATLAB or "Matrix Laboratory" is an interpreted environment that allows mathematical calculations and graphics. MATLAB program are written in the M programming language. The availability of a number of toolboxes, such as the Bioinformatics toolbox, makes MATLAB a really powerful environment. We will begin by looking at the core datatypes, inbuilt functions and graphing capabilities in this chapter. Bioinformatics specific features are presented in subsequent chapters to accompany the presentation of underlying theoretical foundations.

When you start MATLAB you are greeted with a window that looks similar to Fig. A.1. You can then type commands at the prompt (>>) within the command window. To get started, type one of these commands: *helpwin, helpdesk*, or *demo*. The *demo* command offers you the ability to watch movies that can will help you come up to speed quickly.

To see how it can be used on the simplest level, define variable x to be 3, variable y to be 4, and variable z to be the sum of x and y. The result will look something like this:

```
>> x = 10

x =
     10

>> y = 15

y =
     15

>> z = x + y

z =
     25
```

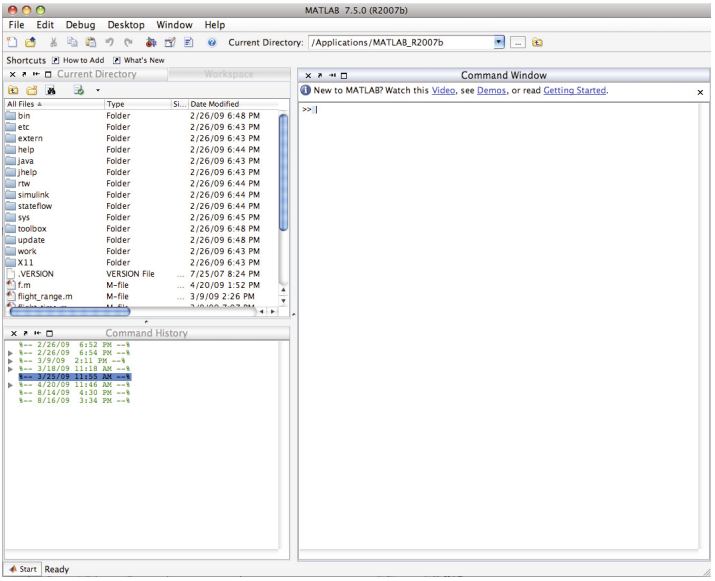


Fig. A.1 MATLAB startup screens comprising of command window, the history window and file browser

You might notice that MATLAB prints out the value of each variable automatically when you type each line. You can prevent it from printing this value if you place a semicolon after the line:

```
>> z = x + y;        Does not print the value of z
```

MATLAB supports a number of mathematical functions in addition to the normally used symbols for arithmetic such as + (add), - (subtract), * (multiply), and / (divide). A few examples of common inbuilt functions include, *sin, cos, tan, asin, acos, atan, log, log10, exp, power, sqrt,* etc. More information on these and other MATLAB functions is available by typing the function name in the search window in helpdesk.

Example 1.1

Let's perform the following set of computations using MATLAB. Simply type the commands as shown below. The set of commands shown determine if the three numbers satisfy the triangular inequality, i.e. $z^2 \leq x^2 + y^2$. The first three statements assign the values of x, y and z.

```
>> x = 3;
>> y = 9;
>> z = 10;
```

```
>> a = x * x;
>> b = y ^ 2;
>> c = power (z, 2);
>> result = (c <= (a + b))

result =

     0

>>
```

The above example shows three different ways in which a number can be squared. The variable result is a boolean variable which takes on the value of true (1) or false (0).

End of Example

M-Files: As mentioned above, the programming MATLAB programs are written in the M-programming language. The examples shown above demonstrate the use of MATLAB in its *calculator* mode. One can store the M-program on an M-files and run files instead. The M-files are ASCII text files that contain sequences of MATLAB commands and stored on your disk as a file with a **.m** file extension.

There are many types of M-files. These are the script files, which automate long sequences of commands; or function files that extend MATLAB by developing new commands. You can create M-files using the inbuilt MATLAB editor which starts when you type **edit** at the command prompt. As M-files are really text files, they also be created using any text editor.

The matlab Path is the list of directories searched when you type a function name or a script name on the command prompt. If you type *path* at the command prompt, MATLAB lists the current set of directories it will search. Let us say that you are developing all your M-files in your home folder and would like MATLAB to find these when you type in the function or script name in the command prompt. You can add this directory to the MATLAB path with the help of *pathtool* which will let you browse and add your directory to the path as well as allow you to change the order in which the directories are searched. If you know the path to the directory, you can quickly add or remove it using the command *addpath* and *rmpath* respectively. A path added by the *addpath* command is always at the top of the search list.

A.1 MATLAB Data Types and Operators

The default numeric data-type in MATLAB is a double precision floating point number. You can create single precision data types, as well as long, integers, and unsigned short, and byte data. The latter data types are particularly useful for storing grayscale image and text data.

A.1.1 Boolean Operators

Boolean operators return a value of TRUE (1) or FALSE (0) based on whether the condition tested has a logic value of truth or falsity. MATLAB provides the following operators:

Table A.1 Boolean Operators

Operator	Meaning
==	equal to
~	not
~=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
&	and
|	or
isfinite()	returns true if finite
any()	returns true if any element is nonzero
all()	returns true if all elements are nonzero
isinf()	returns true if infinite
isnan()	returns true if NaN (not a number)

Run the following code snippet to understand the usage of boolean operators.

```
>> a = 6;
>> a > 3
>> a ~= 8
>> b = [ -4 8 Inf 500 NaN 0]
>> b <  3
>> isfinite(b)
>> all(b)
>> isnan(b)
```

A.1.2 Element by Element Operations

These operations are carried out on each of the elements of an array or a matrix. Addition and subtraction are by definition already element by element operations since when two arrays are added or subtracted, the operation is executed with the elements that are in the same position in the arrays. Element-by-element operations must be done with arrays of the same size.

Element-by-ement multiplication, division and exponentiation operations of two vectors or matrices are entered in MATLAB by typing a period in front of the arithmetic operator. If two vectors a and b are defined as $a = [a_1 a_2 \ldots a_n]$ and $b = [b_1 b_2 \ldots b_n]$, the results of their element-by-element multiplication and division are respectively:

$a. * b = [a_1 b_1 \quad a_2 b_2 \quad \ldots \quad a_n b_n]$, and

$a./b = [a_1/b_1 \quad a_2/b_2 \quad \ldots \quad a_n/b_n]$.

The power of element-by-element operations is realized by calculating the value of a function at many values of its argument. This is done by first defining a vector independent values and then defining function values for each of the elements through element by element operations. Furthermore, element-by-element operations are defined for each of the inbuilt functions. As an example, the computation result of the $sin(a)$, with a defined as the vector above is:

$sin(a) = [sin(a_1) \quad sin(a_2) \quad \ldots \quad sin(a_n)]$

A.2 Matrices

Created using square brackets. A comma or space indicates a separate entry in the same row and a semicolon indicates the end of a row. Further, you can use colons to define numeric sequences which are linearly space monotonically increasing or decreasing numbers. Numerical sequences are defined using the syntax:

`sequence = min:step:max`, or
`sequence = min:max`

The latter definition assuming a step of 1. Parentheses are used to reference individual elements. Complete indexing within a matrix comprises of (row, column). Let's look at the following example.

```
>> x=[1 5; 10 15]
x =
        1       5
        10      15

>> y  = [ 4,5,6; 7,8,9]
y =
       4      5      6
       7      8      9

>>  z= [ .4   cos(pi/3)  6/7^2]
```

```
z =
        0.400      0.500       0.1224

>> z(2,3) =.9
z =
       0.400      0.500       0.1224
       0             0        0.9000
```

Let's next look at a code sample that uses sequencing.

```
>> t= 0:2:10
t =
       0    2    4    6    8    10

>>v =20:25
v =
       20  21  22  23  24  25
```

MATLAB can create **multidimensional arrays** or arrays with more than two subscripts. Let's look at the following code sample where we are 3D physical data as a sequence of matrices; or samples of a time-dependent 2D or 3D data.

```
>> a = [2 4 6; 7 8 9; 1 2 3]
>> a(:,:,2)= [10 11 12; 0 1 2; 4 5 6]
```

When you add elements and expand the size of the multidimensional array, the requisite number of interleaved elements are are set to zero. For example, consider the following command:

```
>> a(:,:,4) = [ 1 1 1; 2 2 2; 3 3 3]
```

The command above will set the values for the fourth plane, where each plane is a 3 by 3 matrix. The values in the first three planes will be set to zero.

A.2.1 String Arrays

String arrays are created using single quotes. Internally, a string is stored as row vectors and takes 2 bytes per character. Strings are concatenated using [,] operator. The operator [;] is used to vertically concatenate strings, which must be of equal length. MATLAB function *strvcat* is can vertically concatenate strings arrays of non-equal lengths. Consider the following examples:

```
str1='Hi there.   '
str2='How are you?'
```

```
str3= 'Bye'
c1= [str1,' ',str2]  % join two strings
c2 = [str1;str2]     % vertical concatenation-same length strings
c3=strvcat(str1,str2,str3)   % vertically concatenate (matrix)
```

A.2.2 Cell Arrays

A cell array is a general purpose matrix where each of its elements can contain data of a different type, size and dimension. Storage for cell arrays is allocated dynamically. Cell arrays are created using the cell command or by using curly braces:

```
>> cell_name{row,col} = data;
```

In the following example, the function *cellplot* creates a graphical depiction of a cell array.

Example 1.2

```
>>
A= {rand(2,2,2), ' February', 10.28}
A =
[2x2x2 double] 'February', [10.2800]

B{1,1}=1:8;
B{1,2}=strvcat('Monday','Tuesday','Wednesday','Thursday');
B{2,2}=A;
B{1,1}
A{1,1}(2,:,1)
cellplot(B)
```

End of Example

A.2.3 Structures

Structures are multidimensional arrays where the elements of a structure are accessed by named fields. Fields can contain any type of data and as in the case of cell arrays, storage needed for a structure is allocated dynamically. Syntax for setting fields of a structure is shown below. record$\tilde{\#}$ defaults to 1.

```
>> struct_name(record #).field_name=data
```

The code segment below demonstrates the multiple ways of manipulating structures.

```
students.name = 'Sally';
students.grades= [97 93 ];
students(2).name = 'John';
students(2).grades= [94 96 ];

or

students= struct('name',{'Sally','John'},'grades',{[97 93], [94 96]});

students(1).name
students.grades
```

A.3 Programming Constructs

MATLAB programs are written using the standard constructs for conditional statements and looping. Logic Control Constructs Iterative Loops for while

A.3.1 Logic Control

Conditional statements are written using the *if/then* or *if/then/else* or *if/then/elseif/else* construct based on the application need. Here is an example illustrating the use of conditional statements:

```
x=34;
y=26;
if x>y
    z=1
elseif x==y
    z=0
else
   z=-1
end
```

It is often easier to write a case statement when a given variable takes on a multitude of values as illustrated in the following example:

```
month ='april';
switch month
case{ 'january' 'march' 'may' 'july' 'august' 'october' 'december'}
     days=31
case { 'april' 'june' 'september' 'november' }
    days =30
case {'february' }
     days=28
otherwise
     days=-1
end
```

A.3.2 Loops

MATLAB provides both the *for* loops that are are executed a specific number of time and *while* loops, the execution of which depends upon the truth or falsity of a sentinel condition. Again, the semantics of loops in MATLAB is quite similar to other programming languages.

An example of the *for* loop which repeats the statements enclosed within the loop construct a number of times based on index value is shown below.

```
a=100
for i=1:a
    for j=1:a
       x(i,j)=i^2+3*j +1;
    end
end
```

Shown below is an example of a *while* loop that repeats the statements with the loop construct until logical condition returns false. Although single loop constructs are shown in these examples, these loop constructs can be nested.

```
done=false;
max=1300;
i=1;
while(~done)
    y(i)= i^3*2+1
    if y(i) >max
      done=true;
    end
    i=i+1
end
```

A.3.3 Vectorization Looping

MATLAB provides element by element operators defined above in section A.1.2. These operators provide the ability to perform implicit looping over the elements of an array or a vector. Moreover, if loops are written using such a looping construct, the resulting code will be simpler and will run faster. Let's take an example where the two vectors m and v contain the mass and volume data. The first example shown computes the corresponding density value using the traditional looping operators as shown below.

Example 1.3

```
m=rand(5,1000);
v=rand(5,1000);
```

```
[rows,cols] =size(m);
for i= 1:rows
    for j=1:cols
        density(i,j)=m(i,j) / v(i,j);
    end
end
```

Vectorization offers performance Increase since MATLAB is designed to work with matrices. The above computation can be performed using code segment utilizing implicit vectorization.

```
m=rand(5,1000);
v=rand(5,1000);
density=m ./ v;
```

End of Example

A.4 File Operations

MATLAB provides load and save commands for storing the entire workspace or values of specific variables in your workspace. The data is stores in a in a platform-independent binary format, and the default filename is *matlab.mat*. All files are saved with the extension of *.mat* and stored in the current directory. Unless specified otherwise in the load or save command, the read/write binary data files by default. Here are the examples of two mirrored *load* and *save* commands. In the last *save* command shown, the data will be saved in ascii format.

```
save
save filename
save filename x y z
save filename -ascii
```

The following load commands mirror the load commands above.

```
load
load filename
load filename x y z
load filename
```

Try the following code snippet to get familiarized with these commands:

```
>>clear
a= rand(3,2);
b=ones(3,2);
c = a.^2+b;
```

```
save mydata
clear
load mydata
save -ascii mydata_ascii
clear
load mydata_ascii
```

A.4.1 Importing Data Into Matlab

The MATLAB command *importdata* command uses file extension if possible to determine file type, and uses file type determines data type. If no recognizable file extension, it assumes delimited data. Further, *uiimport* command is a GUI interface to import data. The command *textscan* imports to cell arrays.

To get help with use for non-standard data formats and large files, try the following commands:

```
help iofun
help fileformats
uiimport
mydata=importdata('Sample_Data_File.txt')
```

Here is an example of a code segment illustrating the use of *textscan*. The function *fopen* opens a file and returns a file handle. And as you might have guessed, the statements beginning with s % sign are comments.

Example 1.4

```
% textscan_example.m - example of using textscan command
% to import data
fid=fopen('Sample_Data_File.txt');
numCols=10;
numHeaders=1;
format1=repmat('%s ',1,numCols);
format2=repmat('%d ',1,numCols-1);
format2=['%s ' format2];
myHeader=textscan(fid,format1,1);
myData=textscan(fid,format2);
fclose(fid);
disp('here are the first 5 elements from column 1)';
myData{1,1}(1:5)
disp('here are the 10 elements from column 9');
myData{1,9}(1:10)
disp('here are the first 8 columns in the header row');
myHeader{1,1:8}
```

End of Example

A.5 Functions

MATLAB has a large repository of core built-in functions. You should always perform a search with appropriate set of keywords and look for in-built functions provided. These include, the *sin, abs, exp*, ..., etc. Additionally, several of the functions available in MATLAB have been written in the M-programming language. Some examples of these functions include, *mean, std, erf, ...*

Similarly, a user a create M-file functions, and as long as the proper syntax is followed and the path to the function directory is included in the path variable, the function will be similarly usable by any subsequent function or script you develop. Consider an example of a function:

```
function y = mymean(x)
%MYMEAN Average or mean value.
%   For vectors, MYMEAN(X) is the mean value of  X. For
%   matrices, MYMEAN(X)is a row vector with mean value of
%   each column. For N-D arrays, MYMEAN(X) is the means
%   elements along the first non-singleton dimension of X.

[m,n]=size(x);
if m==1
     m=n;
end
y=sum(x)/m;
```

Here are some requirements for developing a function. It has to have the required keyword of *function*. The name of the function must be the same as the file name where the function is saved. So, the above function will have to be saved in a file named `mymean.m`. The results generated by the function must be stored in variable(s) with the same name as the output arguments specified in the function statement, which is y in the example shown above.

Optionally, input and output arguments may be specified which define the internal variable names. Further, online help- comment lines following the function definition line may be specified. MATLAB's *lookfor* command uses the first comment line in its search. You can execute the function above

```
>> a=rand(5);
>> b=mymean(a)
```

Note that multiple inputs and outputs for a function may be specified as in the example below:

```
function [avg,stdev,r] = ourstats(x,tol)
%OURSTATS finds the average, std. dev. and rank of a matrix
[m,n]=size(x)
```

```
if m==1;
    m=n;
avg=sum(x)/mean;
stdev=sqrt(sum(x.^2)/m - avg.^2);
s=svd(x);
r=sum(s > tol);
```

Here is an example of a function that takes in three lengths for its inputs and outputs a string that says if a triangle can or cannot be formed from the lengths provided.

Example 1.5

```
function str_tri_test = istriangle(a,b,c)
% istriangle determine if a,b,c form a triangle
v =[ a b c];
vsort=sort(v);
if vsort(1) + vsort(2) > vsort(3)
    s1=['The sides ',num2str(v),' form a triangle.'];
    % start extra credit
    if (a==b) & (b==c) % all sides equal
        s2=' The triangle is an equialateral triangle.';
    elseif (a==b) |(b==c) |(a==c)
        s2= ' The triangle is an isosceles triangle.';
    else
        s2= ' The triangle is a scalene triangle.';
    end
    %end extra credit
    str_tri_test=strcat(s1,s2);
else
    str_tri_test= ['The sides ',num2str(v),' do not form a triangle.'];
end

try:
>> istriangle(3,4,5)
```

End of Example

A.6 2-D Plotting

MATLAB provides graphing capabilities, for plotting both 2D and 3D plots. The basic plot command and some of its options is shown below:

Table A.2 Plotting Command and Options

Command	Functionality
plot	make 2-D plots
grid on/off	turn grid on and off
hold on/off	holds the current plots
title	adds a title to the plot
xlabel	adds an x axis label
ylabel	adds a y axis label
legend	add a legend to the plot
text	add text to a specified position on the plot
gtext	interactively place text on the plot
ginput	pick off coordinates from a plot

The following code segment illustrates the use of graphing commands in MATLAB.

Example 1.6

```
echo on
x = 0:0.1:2*pi;
y=sin(x);
plot(x,y)
grid on
hold on
plot(x,exp(-x),'r:*');
axis ([ 0 2*pi 0 1])
title('2-D Plots');
xlabel('Time');
ylabel('Sin(t)');
text(pi/3,sin(pi/3),'<--sin(\pi/3)')
legend('Sine Wave','Decaying Exponential');
echo off
```

End of Example

A.6.1 Graphics Objects

MATLAB provides a variety of commands for working with its graphics objects.

The following example illustrates the use of graphics objects.

```
x=0:.1:10;
y=sin(x);
plot(x,y);
axes_props=get(gca);
set(gca,'TickLength',[.04 .04]);
a=get(gca,'YLim');
```

Table A.3 Working With Graphics Objects

Command	Functionality
gca	return the handle of the current axes
gcf	return the handle of the current figure
gco	return the handle of the current object
get	query the values of an object's properties
set	set the values of an object's properties
findall	find all graphics objects
findobj	find the handles of objects having specified property values

```
set (gca,'YLim', [-1 2]);
b=findobj(gcf,'Color','blue');
get(b(1))
```

A.7 Matlab Bioinformatics Toolbox

The Matlab Bioinformatics Toolbox makes use of the wide range of Matlab's extensive collection of statistics and graphing functions. Matlab Bioinformatics Toolbox uses software libraries that enable it to make use of various bioinformatics file types, databases, programs, and algorithms. Using these libraries, one can efficiently make use of biological data without being required to know how it is stored or how each transformative or comparative process works; instead, one can simply know what the data represents or what the process does with the given data.

For example, using either BioPerl or the Matlab Bioinformatics Toolbox one can

- Read in a variety of genomic, proteomic, and gene expression data files
- Access biological databases over the Internet
- Perform pairwise and multiple sequence alignments
- Analyze sequences for composition and occurrence of patterns
- Construct and analyze phylogenetic trees
- Perform Gene Ontology analysis
- Process and visualize microarray data analysis
- Process and analyze mass spectrometry data
- Develop algorithms using statistical learning functionality

In this manner, the Bioinformatics Toolbox extends MATLAB to provide an integrated software environment for genome and proteome analysis. One can use the basic bioinformatic functions provided with this toolbox to create more complex algorithms and applications in drug discovery, genetic engineering, and biological research. Examples of the MATLAB toolbox are included with the relevant sections of the text.

A.8 Exercises

1. Show the value of the variable `seqs` upon executing the following MATLAB commands:

```
>> seq1 = 'ATTA';
>> seq2 = 'ATTTA';
>> seq3 = 'ATTTAA';
>> seqs = char(seq1, seq2, seq3);
```

2. Consider the following MATLAB cell array initialization for the variable `cellseq`. What command(s) would you issue to extract and assign the value of the second cell (i.e. string `'TTTGGG'`) to a character string variable named `seq2`?

```
>> cellseq = {'TTGGTT', 'TTTGGG', 'TGGTTGGT', 'GGGTTT'};
```

3. What is the output of the following commands:

```
>> s.id = 1;
>> s.name = 'John';
>> t.id = 2;
>> t.name = 'Jill';
>> sa(1) = s;
>> sa(2) = t;
>> sa(2)
```

4. What output is generated upon the execution of following MATLAB code:

```
>> seq = 'ATTATT';
>> fwd = seq
>> rev = seqrcomplement(seq)
```

5. What is the output of excuting the following MATLAB code:

```
>> cellseq = {'TTGGTT', 'TTTGGG', 'TGGTTGGT', 'GGGTTT'};
>> comp = regexpi (cellseq, 'TTT');
>> ind = find(~cellfun('isempty',comp));
>> cellseq (ind)
```

6. Consider the following DNA sequence:

   ACCCA TAGGG AGACA TAGTA GATCC ATTAG

   (a) Perform a 6 frame translation of a given DNA strand.
   (b) Compute the length of Open Reading Frames (ORF) in each of 6 frames. Based on your analysis, which strand and which frame within that strand has the highest likelihood for coding.

Appendix B
BioPerl

BioPerl is a perl library build using the Perl programming languages. BioPerl is an open source library which is quite mature and free. As an extension of the Perl programming language, BioPerl makes good use of Perl's powerful string processing functions such as regular expressions. BioPerl is specifically designed to process strings. As considerable portion of bioinformatics data comprises of strings, knowledge of a language that supports rapid middleware development for processing data for information exchange between native applications and MATLAB is a useful tool to have in your repertoire. Middleware or "enabling technology" is a category of technology that facilitates interaction between various software technologies across one or more systems. For example, a database management system (DBMS) can be viewed as middleware for a web application as the DBMS resides between the file system and the web application and allows for a black-box approach to retrieving data.

Therefore, a working knowledge of Perl in general, and BioPerl in particular, will be very useful in quickly processing and reformatting biological data so that it is easily exchangeable between the various platforms, websites, application, and MATLAB.

B.1 BioPerl

Since BioPerl is a perl library, it requires Perl interpreter be installed before the middleware functions provided by BioPerl are invoked. If using a Linux, Unix, or Mac OS X system; Perl very well may have come pre-installed. On a Microsoft Windows system, Perl it will probably be necessary to install Perl if it hasn't been done already – it can be obtained freely via the Internet (www.perl.org). If using Windows, the easiest way to install Perl is to use ActivePerl from ActiveState. This distribution is free and uses the standard Windows installer.

If using a variant of Linux, Unix, or Mac OS X it may be unnecessary to install Perl. Type "perl -v" at the command line to determine whether Perl

is installed and, if it is, what version. If the system does not have Perl or the Perl installation is old (BioPerl currently requires Perl 5.6 or greater), the easies way might be to obtain the source (www.cpan.org), uncompress it, and use the command-line tool make to install Perl. An ActivePerl installer also exists for Mac OS X.

The next step is to install BioPerl itself. On Windows, the simplest way to install BioPerl is to use the Perl Package Manager GUI, which can be run from the start menu. Go to Edit¿¿Preferences and after clicking on the Repositories tab, add the following repositories depending on which version of Perl is on your system. If using Perl 5.8, add:

- BioPerl-Regular Releases—http://bioperl.org/DIST
- Kobes—http://theoryx5.uwinnipeg.ca/ppms
- Bribes—http://www.Bribes.org/perl/ppm

If using Perl 5.10, add:

- BioPerl-Regular Releases—http://bioperl.org/DIST
- Kobes—http://cpan.uwinnipeg.ca/PPMPackages/10xx
- Bribes—http://www.Bribes.org/perl/ppm

If problems occur, please see www.bioperl.org/wiki/Installing_Bioperl_on_Windows for further instructions.

On other platforms, the easiest way to install Perl might be to obtain the appropriate compressed source distribution from http://bioperl.org/DIST/. Once downloaded, using the command line navigate the terminal to the download directory then issue the commands:

```
>gunzip bioperl-1.5.2_102.tar.gz
>tar xvf bioperl-1.5.2_102.tar
>cd bioperl-1.5.2_102
>perl Build.PL
>./Build test
```

This will run a testing suite to determine how adequately your system can handle BioPerl. If a few tests fail, you can probably still use BioPerl normally. The last step may require administrator privileges, simply run:

```
>./Build install
```

B.2 Using Perl

Perl is a non-compiled scripting language that is similar to C and bash scripting. As of Perl 5, Perl supports object-oriented programming. Other advantages of Perl include many built-in string subroutines including natively handling of regular expressions.

Syntactically, Perl code looks like C code. As in C, line endings in Perl are denoted by the semi-colon (;) and all white space is ignored. Opening and closing braces (and) are required for use around control structures such as *if*, *while*, and *for* statements. Functions, called subroutines in Perl, from another class or library are invoked in the form ClassName-¿SubroutineName(). Perl also supports many built-in subroutines that can be called simply by typing the subroutine's name. In Perl, parameters passed as an array are not type-checked and any number can be passed to any subroutine. As a result, parameter checking must be implemented by each subroutine and it is common practice for subroutines to accept parameters as associative arrays. Everything following the pound sign (#) in a line is considered to be a comment.

There are 5 data types in Perl. In most cases, one can determine the type of a variable by its name. Perl possesses a naming convention that requires variables of specific types to be named in a specific manner. The first data type is a scalar, which may store a number, a string, or a reference. Perl does not distinguish between integers and other types of numbers. A scalar is indicated by a variable name that begins with the dollar sign ($). The next data type is the array, which is single dimensional in Perl, with each element storing a scalar. Variable names for arrays begin with the ampersand (@). Associative arrays, or hashes, provide for a matching of keys in the form of strings to scalar values. A hashes variable name begins with the percent sign (%). When referring to element stored by an array or hash, a dollar sign is used.

```
# The pound symbol denotes comments.
# Assign the number 24 to the variable $scalar.

$scalar = 24;

# Assign the string "test" to the same variable $scalar.
$scalar = "test";

# Set the first three elements of array @array.
@array = (1, "two", $scalar);

# This can also be accomplished by
$array[0] = 1;
$array[1] = "two";
$array[2] = $scalar;

# Assign three key/value pairs to the hash %hash.
%hash = {"key1"=>1, 2 => "value 2", "key3" = $scalar};

# This can also be accomplished by
$hash{"key1"} = 1;
$hash{2} = "value 2";
$hash{"key3"} = $scalar;
```

Subroutines, as discussed, are another data type which requires no specific naming convention. Lastly, file handles are implemented in Perl and allow reading from or writing to files and are denoted by a variable name consisting of all capital letters.

As mentioned, regular expressions are a powerful native capability of Perl. Using the =~ operator, Perl can process a regular expression and either simply find a given pattern or find the pattern and replace it with another string. The string to the left of the =~ operator is the target string, which is searched. The pattern for which is search is on the right-hand side of the operator within forward slashes (e.g. /'test'/). For example, code looks to match the string "opqrs" within the alphabet and prints an appropriate message if found.

```
$alphabet = "abcdefghijklmnopqrstuvwxyz";
if ($alphabet =~ m/'opqrs'/) {
    print "The alphabet contains 'opqrs'\n";
}

# Omitting the m before the pattern produces the same result
if ($alphabet =~ /'opqrs'/) {
    print "The alphabet still contains 'opqrs'\n";
}
```

The above code will output:

```
The alphabet contains 'opqrs'
The alphabet still contains 'opqrs'
```

The following script populates an array with strings and for each element replaces the term "policeman", if one exists, with the term "police officer". All the elements of the array are then printed.

```
@array = ('The policeman wore a blue uniform',
    'What did one policeman say to the other policeman?',
    'The term is not in this string',
foreach $string (@array) {
    $string =~ s/'police man'/'police officer'/g;
    print $string . "\n";
}
```

This code will output:

```
The police officer wore a blue uniform
What did one police officer say to the other police officer?
```

B.3 Entrez Sample in BioPerl

BioPerl can be used to easily handle automated queries of the Entrez database online. Using the class *Bio::DB::Query::GenBank* one can create a query using the same parameters present in the web form. For example, the query string "Human[ORGN] AND 200:1000[SLEN] and Smith[Author]" will return both protein and nucleotide sequences that are from humans, are between 200 and 1000 compounds in length, and were authored by someone named Smith. Once such a query object has created, passing it to the an instance of the *Bio::DB::GenBank* class will actually send the query via the Internet to the Entrez servers at NCBI, which will then return the results that BioPerl interprets as an array of sequence objects of the class *Bio::Seq*.

The following code, queries GenBank for human protein sequences that are between 200 and 1000 compounds in length and were authored by someone named Smith. The ID, species, and length of each sequence is then printed to the console.

Listing 2.1

```
use Bio::DB::Genbank;
use Bio::DB::Query::GenBank;
#define the query
$query = "Human[ORGN] AND 200:1000[SLEN] AND Smith[AUTHOR]";

# Create wrapper object for Entrez.
#This object will search the protein database.

$query_obj = Bio::DB::Query::GenBank
    ->new(-db => protein,  -query => $query );

# Create a handle to the GenBank database

$gb = Bio::DB::GenBank;

# Get a stream of results from GenBank

$stream_obj = $gb->get_Stream_by_query($query_obj);

# Step through the results -  Print sequence data

while ($seq_obj = $stream_obj->next_seq) {
    print $seq_obj->display_id .
    $seq_obj->species->common_name .
    $seq_obj->length, "\n";
}
```

end-listing-2.1

B.4 Exercises

1. Consider the following DNA sequence:

   ACCCA TAGGG AGACA TAGTA GATCC ATTAG

   (a) Write a bioperl program to compute the reverse complement of the sequence.
   (b) Write a bioperl program to perform a 6 frame translation of a given DNA strand.
   (c) Compute the length of Open Reading Frames (ORF) in each of 6 frames. Based on your analysis, which strand and which frame within that strand has the highest likelihood for coding.

Index