NOTES ON LESSON
BIOINFORMATICS I (BT2301)
V SEMESTER

B.TECH DEGREE PROGRAMME

Department of Biotechnology

RAJALAKSHMI ENGINEERING COLLEGE
THANDALAM
CHENNAI

Prepared by

N. CHAKRAVARTHY
Lecturer

Department of Biotechnology

UNIT I INTRODUCTION

An Introduction to Bioinformatics

Bioinformatics is the application of statistics and computer science to the field of molecular biology.

The term *bioinformatics* was coined by **Paulien Hogeweg** and **Ben Hesper** in 1978 for the study of informatic processes in biotic systems. Its primary use since at least the late 1980s has been in genomics and genetics, particularly in those areas of genomics involving large-scale DNA sequencing.

Bioinformatics now entails the creation and advancement of databases, algorithms, computational and statistical techniques and theory to solve formal and practical problems arising from the management and analysis of biological data.

Over the past few decades rapid developments in genomic and other molecular research technologies and developments in information technologies have combined to produce a tremendous amount of information related to molecular biology. It is the name given to these mathematical and computing approaches used to glean understanding of biological processes.

Common activities in bioinformatics include mapping and analyzing DNA and protein sequences, aligning different DNA and protein sequences to compare them and creating and viewing 3-D models of protein structures.

The primary goal of bioinformatics is to increase the understanding of biological processes. What sets it apart from other approaches, however, is its focus on developing and applying computationally intensive techniques (e.g., pattern recognition, data mining, machine learning algorithms, and visualization) to achieve this goal. Major research efforts in the field include sequence alignment, gene finding, genome assembly, drug design, drug discovery, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, genome-wide association studies and the modeling of evolution.

Introduction

Bioinformatics was applied in the creation and maintenance of a database to store biological information at the beginning of the "genomic revolution", such as nucleotide and amino acid sequences. Development of this type of database involved not only design

issues but the development of complex interfaces whereby researchers could both access existing data as well as submit new or revised data.

In order to study how normal cellular activities are altered in different disease states, the biological data must be combined to form a comprehensive picture of these activities. Therefore, the field of bioinformatics has evolved such that the most pressing task now involves the analysis and interpretation of various types of data, including nucleotide and amino acid sequences, protein domains, and protein structures. The actual process of analyzing and interpreting data is referred to as computational biology. Important sub-disciplines within bioinformatics and computational biology include:

- the development and implementation of tools that enable efficient access to, and use and management of, various types of information.
- the development of new algorithms (mathematical formulas) and statistics with which to assess relationships among members of large data sets, such as methods to locate a gene within a sequence, predict protein structure and/or function, and cluster protein sequences into families of related sequences.

There are two fundamental ways of modelling a Biological system (e.g. living cell) both coming under Bioinformatic approaches.

- Static
    - Sequences - Proteins, Nucleic acids and Peptides
    - Structures - Proteins, Nucleic acids, Ligands (including metabolites and drugs) and Peptides
    - Interaction data among the above entities including microarray data and Networks of proteins, metabolites
- Dynamic
    - Systems Biology comes under this category including reaction fluxes and variable concentrations of metabolites
    - Multi-Agent Based modelling approaches capturing cellular events such as signalling, transcription and reaction dynamics

A broad sub-category under bioinformatics is structural bioinformatics.

Major research areas in Bioinformatics

Sequence analysis

Since the Phage Φ-X174 was sequenced in 1977, the DNA sequences of thousands of organisms have been decoded and stored in databases. This sequence information is analyzed to determine genes that encode polypeptides (proteins), RNA genes, regulatory sequences, structural motifs, and repetitive sequences. A comparison of genes within a species or between different species can show similarities between protein functions, or relations between species (the use of molecular systematics to construct phylogenetic trees). With the growing amount of data, it long ago became impractical to analyze DNA sequences manually. Today, computer programs such as BLAST are used daily to search sequences from more than 260 000 organisms, containing over 190 billion nucleotides. These programs can compensate for mutations (exchanged, deleted or inserted bases) in the DNA sequence, in order to identify sequences that are related, but not identical. A variant of this sequence alignment is used in the sequencing process itself. The so-called shotgun sequencing technique (which was used, for example, by The Institute for Genomic Research to sequence the first bacterial genome, *Haemophilus influenzae,* does not produce entire chromosomes, but instead generates the sequences of many thousands of small DNA fragments (ranging from 35 to 900 nucleotides long, depending on the sequencing technology). The ends of these fragments overlap and, when aligned properly by a genome assembly program, can be used to reconstruct the complete genome. Shotgun sequencing yields sequence data quickly, but the task of assembling the fragments can be quite complicated for larger genomes. For a genome as large as the human genome, it may take many days of CPU time on large-memory, multiprocessor computers to assemble the fragments, and the resulting assembly will usually contain numerous gaps that have to be filled in later. Shotgun sequencing is the method of choice for virtually all genomes sequenced today, and genome assembly algorithms are a critical area of bioinformatics research.

Another aspect of bioinformatics in sequence analysis is annotation, which involves computational gene finding to search for protein-coding genes, RNA genes, and other functional sequences within a genome. Not all of the nucleotides within a genome are part of genes. Within the genome of higher organisms, large parts of the DNA do not

serve any obvious purpose. This so-called junk DNA may, however, contain unrecognized functional elements. Bioinformatics helps to bridge the gap between genome and proteome projects — for example, in the use of DNA sequences for protein identification.

Genome annotation

In the context of genomics, **annotation** is the process of marking the genes and other biological features in a DNA sequence. The first genome annotation software system was designed in 1995 by Dr. Owen White, who was part of the team at The Institute for Genomic Research that sequenced and analyzed the first genome of a free-living organism to be decoded, the bacterium *Haemophilus influenzae*. Dr. White built a software system to find the genes (places in the DNA sequence that encode a protein), the transfer RNA, and other features, and to make initial assignments of function to those genes. Most current genome annotation systems work similarly, but the programs available for analysis of genomic DNA are constantly changing and improving.

Computational evolutionary biology

Evolutionary biology is the study of the origin and descent of species, as well as their change over time. Informatics has assisted evolutionary biologists in several key ways; it has enabled researchers to:

- trace the evolution of a large number of organisms by measuring changes in their DNA, rather than through physical taxonomy or physiological observations alone,
- more recently, compare entire genomes, which permits the study of more complex evolutionary events, such as gene duplication, horizontal gene transfer, and the prediction of factors important in bacterial speciation,
- build complex computational models of populations to predict the outcome of the system over time
- track and share information on an increasingly large number of species and organisms

  Future work endeavors to reconstruct the now more complex tree of life.

The area of research within computer science that uses genetic algorithms is sometimes confused with computational evolutionary biology, but the two areas are not necessarily related.

Analysis of gene expression

The expression of many genes can be determined by measuring mRNA levels with multiple techniques including microarrays, expressed cDNA sequence tag (EST) sequencing, serial analysis of gene expression (SAGE) tag sequencing, massively parallel signature sequencing (MPSS), or various applications of multiplexed in-situ hybridization. All of these techniques are extremely noise-prone and/or subject to bias in the biological measurement, and a major research area in computational biology involves developing statistical tools to separate signal from noise in high-throughput gene expression studies. Such studies are often used to determine the genes implicated in a disorder: one might compare microarray data from cancerous epithelial cells to data from non-cancerous cells to determine the transcripts that are up-regulated and down-regulated in a particular population of cancer cells.

Analysis of regulation

Regulation is the complex orchestration of events starting with an extracellular signal such as a hormone and leading to an increase or decrease in the activity of one or more proteins. Bioinformatics techniques have been applied to explore various steps in this process. For example, promoter analysis involves the identification and study of sequence motifs in the DNA surrounding the coding region of a gene. These motifs influence the extent to which that region is transcribed into mRNA. Expression data can be used to infer gene regulation: one might compare microarray data from a wide variety of states of an organism to form hypotheses about the genes involved in each state. In a single-cell organism, one might compare stages of the cell cycle, along with various stress conditions (heat shock, starvation, etc.). One can then apply clustering algorithms to that expression data to determine which genes are co-expressed. For example, the upstream regions (promoters) of co-expressed genes can be searched for over-represented regulatory elements.

Analysis of protein expression

Protein microarrays and high throughput (HT) mass spectrometry (MS) can provide a snapshot of the proteins present in a biological sample. Bioinformatics is very much involved in making sense of protein microarray and HT MS data; the former approach faces similar problems as with microarrays targeted at mRNA, the latter

involves the problem of matching large amounts of mass data against predicted masses from protein sequence databases, and the complicated statistical analysis of samples where multiple, but incomplete peptides from each protein are detected.

Analysis of mutations in cancer

In cancer, the genomes of affected cells are rearranged in complex or even unpredictable ways. Massive sequencing efforts are used to identify previously unknown point mutations in a variety of genes in cancer. Bioinformaticians continue to produce specialized automated systems to manage the sheer volume of sequence data produced, and they create new algorithms and software to compare the sequencing results to the growing collection of human genome sequences and germline polymorphisms. New physical detection technologies are employed, such as oligonucleotide microarrays to identify chromosomal gains and losses (called comparative genomic hybridization), and single-nucleotide polymorphism arrays to detect known *point mutations*. These detection methods simultaneously measure several hundred thousand sites throughout the genome, and when used in high-throughput to measure thousands of samples, generate terabytes of data per experiment. Again the massive amounts and new types of data generate new opportunities for bioinformaticians. The data is often found to contain considerable variability, or noise, and thus Hidden Markov model and change-point analysis methods are being developed to infer real copy number changes.

Another type of data that requires novel informatics development is the analysis of lesions found to be recurrent among many tumors .

Comparative genomics

The core of comparative genome analysis is the establishment of the correspondence between genes (orthology analysis) or other genomic features in different organisms. It is these intergenomic maps that make it possible to trace the evolutionary processes responsible for the divergence of two genomes. A multitude of evolutionary events acting at various organizational levels shape genome evolution. At the lowest level, point mutations affect individual nucleotides. At a higher level, large chromosomal segments undergo duplication, lateral transfer, inversion, transposition, deletion and insertion. Ultimately, whole genomes are involved in processes of hybridization, polyploidization and endosymbiosis, often leading to rapid speciation. The complexity of

genome evolution poses many exciting challenges to developers of mathematical models and algorithms, who have recourse to a spectra of algorithmic, statistical and mathematical techniques, ranging from exact, heuristics, fixed parameter and approximation algorithms for problems based on parsimony models to Markov Chain Monte Carlo algorithms for Bayesian analysis of problems based on probabilistic models. Many of these studies are based on the homology detection and protein families computation.

Modeling biological systems

Systems biology involves the use of computer simulations of cellular subsystems (such as the networks of metabolites and enzymes which comprise metabolism, signal transduction pathways and gene regulatory networks) to both analyze and visualize the complex connections of these cellular processes. Artificial life or virtual evolution attempts to understand evolutionary processes via the computer simulation of simple (artificial) life forms.

High-throughput image analysis

Computational technologies are used to accelerate or fully automate the processing, quantification and analysis of large amounts of high-information-content biomedical imagery. Modern image analysis systems augment an observer's ability to make measurements from a large or complex set of images, by improving accuracy, objectivity, or speed. A fully developed analysis system may completely replace the observer. Although these systems are not unique to biomedical imagery, biomedical imaging is becoming more important for both diagnostics and research. Some examples are:

- high-throughput and high-fidelity quantification and sub-cellular localization (high-content screening, cytohistopathology, Bioimage informatics)
- morphometrics
- clinical image analysis and visualization
- determining the real-time air-flow patterns in breathing lungs of living animals
- quantifying occlusion size in real-time imagery from the development of and recovery during arterial injury

- making behavioral observations from extended video recordings of laboratory animals
- infrared measurements for metabolic activity determination
- inferring clone overlaps in DNA mapping, e.g. the Sulston score

Structural Bioinformatic Approaches

Prediction of protein structure

Protein structure prediction is another important application of bioinformatics. The amino acid sequence of a protein, the so-called primary structure, can be easily determined from the sequence on the gene that codes for it. In the vast majority of cases, this primary structure uniquely determines a structure in its native environment. (Of course, there are exceptions, such as the bovine spongiform encephalopathy - aka Mad Cow Disease - prion.) Knowledge of this structure is vital in understanding the function of the protein. For lack of better terms, structural information is usually classified as one of *secondary*, *tertiary* and *quaternary* structure. A viable general solution to such predictions remains an open problem. As of now, most efforts have been directed towards heuristics that work most of the time.

One of the key ideas in bioinformatics is the notion of homology. In the genomic branch of bioinformatics, homology is used to predict the function of a gene: if the sequence of gene *A*, whose function is known, is homologous to the sequence of gene *B,* whose function is unknown, one could infer that B may share A's function. In the structural branch of bioinformatics, homology is used to determine which parts of a protein are important in structure formation and interaction with other proteins. In a technique called homology modeling, this information is used to predict the structure of a protein once the structure of a homologous protein is known. This currently remains the only way to predict protein structures reliably.

One example of this is the similar protein homology between hemoglobin in humans and the hemoglobin in legumes (leghemoglobin). Both serve the same purpose of transporting oxygen in the organism. Though both of these proteins have completely different amino acid sequences, their protein structures are virtually identical, which reflects their near identical purposes.

Other techniques for predicting protein structure include protein threading and *de novo* (from scratch) physics-based modeling.

Molecular Interaction

Efficient software is available today for studying interactions among proteins, ligands and peptides. Types of interactions most often encountered in the field include - Protein-ligand (including drug), protein-protein and protein-peptide.

Molecular dynamic simulation of movement of atoms about rotatable bonds is the fundamental principle behind computational algorithms, termed **docking algorithms** for studying molecular interactions.

Docking algorithms

In the last two decades, tens of thousands of protein three-dimensional structures have been determined by X-ray crystallography and Protein nuclear magnetic resonance spectroscopy (protein NMR). One central question for the biological scientist is whether it is practical to predict possible protein-protein interactions only based on these 3D shapes, without doing protein-protein interaction experiments. A variety of methods have been developed to tackle the Protein-protein docking problem, though it seems that there is still much work to be done in this field.

Software and tools

Software tools for bioinformatics range from simple command-line tools, to more complex graphical programs and standalone web-services available from various bioinformatics companies or public institutions.

Web services in bioinformatics

SOAP and REST-based interfaces have been developed for a wide variety of bioinformatics applications allowing an application running on one computer in one part of the world to use algorithms, data and computing resources on servers in other parts of the world. The main advantages derive from the fact that end users do not have to deal with software and database maintenance overheads.

Basic bioinformatics services are classified by the EBI into three categories: SSS (Sequence Search Services), MSA (Multiple Sequence Alignment) and BSA (Biological Sequence Analysis). The availability of these service-oriented bioinformatics resources demonstrate the applicability of web based bioinformatics solutions, and range from a

collection of standalone tools with a common data format under a single, standalone or web-based interface, to integrative, distributed and extensible bioinformatics workflow management systems.

Bioinformatics companies

1. Accelrys (Accelrys is a software company headquartered in the US, with representation in Europe and Japan. It provides software for chemical research, especially in the areas of drug discovery and materials science).
2. NCBI (National Centre for biotechnological information)
3. EMBL (The **European Molecular Biology Laboratory)**
4. **DDBJ** *(***DNA Data Bank of Japan**)
5. AstraZeneca
6. BIOBASE provides biological databases, bioinformatics solutions for expression data, promoter and pathway analysis and KPO services
7. Biomatters Ltd. is the New Zealand-based company that creates the Geneious software suite.
8. Biomax Informatics AG bioinformatics services and solutions
9. CLC Bio free Bioinformatics workbenches.
10. DNASTAR provides DNA sequence assembly and analysis, including Sanger and next generation sequence assembly and gene expression analysis.
11. Gene Codes Corporation
12. Genedata provides software products and services for data analysis and storage in Transcriptomics, Toxicogenomics, High-throughput screening (HTS), Genomics and related disciplines.
13. Geneious combines many DNA and protein sequence analysis tools.
14. Genomatix offering biology driven analysis pipelines for microarray analysis, ChIP on Chip and Solexa/454 data. Multiple tools and databases for analysis of gene regulation. Comparative genomics and most complete and quality checked genomic annotation for 17 species.
15. Genostar provides streamlined bioinformatics solutions: sequence assembly, mapping, annotation transfer and identification of protein domains, comparative

genomics, structural searches, metabolic pathway analysis, modeling and simulation of biological networks

16. Ingenuity Systems is a provider of information solutions and custom services for life science researchers, computational biologists and bioinformaticists, and life science industry suppliers
17. Inte:Ligand
18. Korea Computer Centre Sinhung Company
19. MacVector, providing MacVector and Assembler. MacVector is a Macintosh application that provides sequence editing, primer design, internet database searching, protein analysis, sequence confirmation, multiple sequence alignment, phylogenetic reconstruction, coding region analysis, and a wide variety of other functions.
20. PREMIER Biosoft International is a bioinformatics company with an expertise in software development, marketing and bioinformatics consultancy for in-silico experiment design. PREMIER Biosoft has authored software for pcr primer & probe design, microarray design, glycan structure identificatin, plasmid map drawing and tissue microarray data analysis.
21. Qlucore Qlucore Omics Explorer is a bioinformatics software program.
22. Rosetta Biosoftware
23. SimBioSys develops the eHITS software for molecular docking (flexible ligand docking & fast pre-docking), pharmacophore modelling, de novo design and retrosynthetic analysis software tools
24. Strand Life Sciences offers solutions for micro array gene expression analysis, computational chemistry, data analysis and visualizations.
25. VADLO Search engine for bioinformatics software, databases and online tools

BASIC UNIX COMMANDS

What is UNIX?

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.

Types of UNIX

There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.

Here in the college, we use Solaris on our servers and workstations, and Fedora Linux on the servers and desktop PCs.

The UNIX operating system

The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

The kernel

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.

As an illustration of the way that the shell and the kernel work together, suppose a user types **rm myfile** (which has the effect of removing the file **myfile**). The shell searches the filestore for the file containing the program **rm**, and then requests the kernel, through system calls, to execute the program **rm** on **myfile**. When the process **rm myfile** has finished running, the shell then returns the UNIX prompt % to the user, indicating that it is waiting for further commands.

The shell

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt (% on our systems).

The adept user can customise his/her own shell, and users can use different shells on the same machine. Staff and students in the school have the **tcsh shell** by default.

The tcsh shell has certain features to help the user inputting commands.

Filename Completion - By typing part of the name of a command, filename or directory and pressing the [**Tab**] key, the tcsh shell will complete the rest of the name automatically. If the shell finds more than one name beginning with those letters you have typed, it will beep, prompting you to type a few more letters before pressing the tab key again.

History - The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list or type history for a list of previous commands.

Files and processes

Everything in UNIX is either a file or a process.

A process is an executing program identified by a unique PID (process identifier).

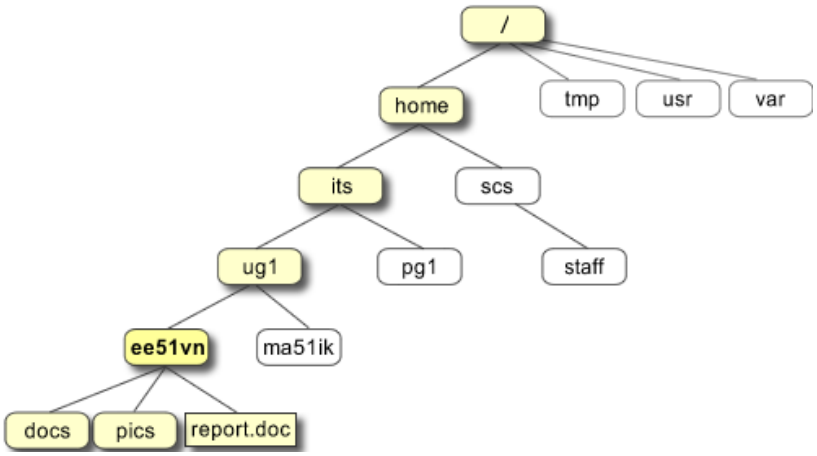
A file is a collection of data. They are created by users using text editors, running compilers etc.

Examples of files:

- a document (report, essay etc.)
- the text of a program written in some high-level programming language
- instructions comprehensible directly to the machine and incomprehensible to a casual user, for example, a collection of binary digits (an executable or binary file);
- a directory, containing information about its contents, which may be a mixture of other directories (subdirectories) and ordinary files.

The Directory Structure

All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called **root** (written as a slash /)

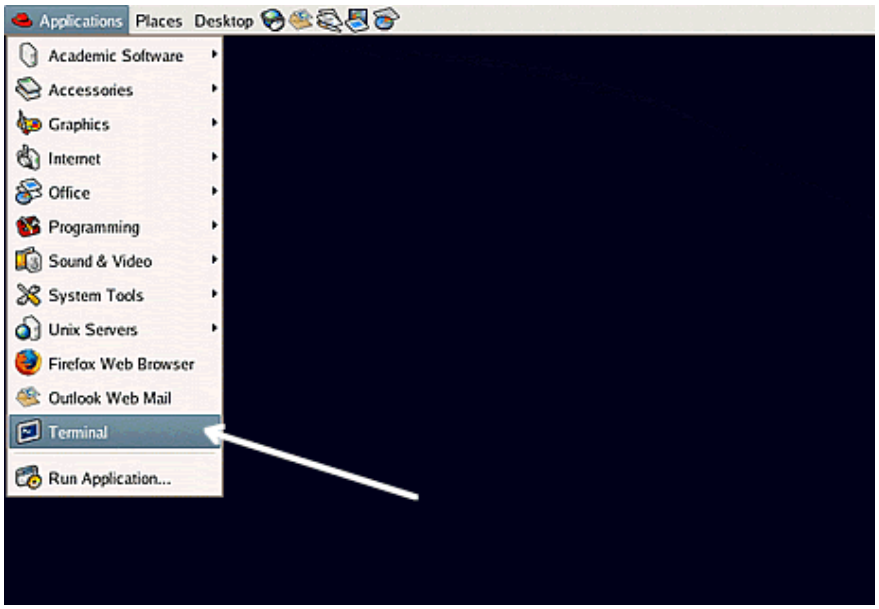


In the diagram above, we see that the home directory of the undergraduate student **"ee51vn"** contains two sub-directories (**docs** and **pics**) and a file called **report.doc**.

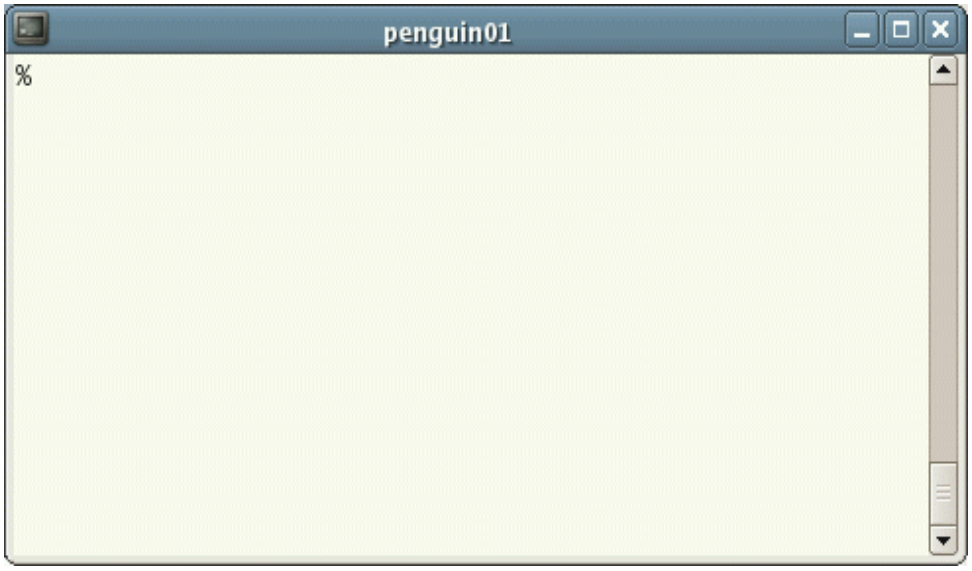
The full path to the file **report.doc** is **"/home/its/ug1/ee51vn/report.doc"**

Starting an UNIX terminal

To open an UNIX terminal window, click on the "Terminal" icon from Applications/Accessories menus.



An UNIX Terminal window will then appear with a % prompt, waiting for you to start entering commands.



UNIX Tutorial One

1.1 Listing files and directories

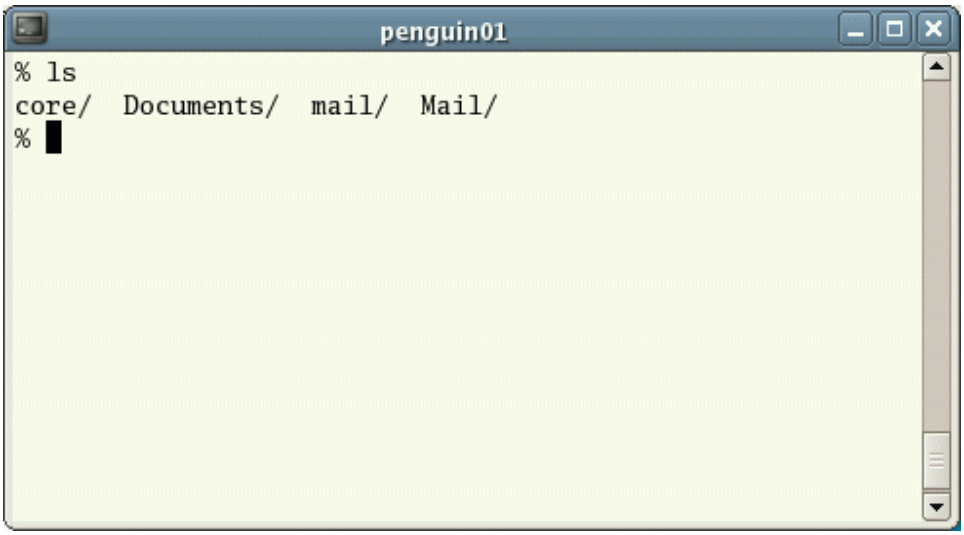
ls (list)

When you first login, your current working directory is your home directory. Your home directory has the same name as your user-name, for example, **ee91ab**, and it is where your personal files and subdirectories are saved.

To find out what is in your home directory, type

% ls

The **ls** command (lowercase L and lowercase S) lists the contents of your current working directory.



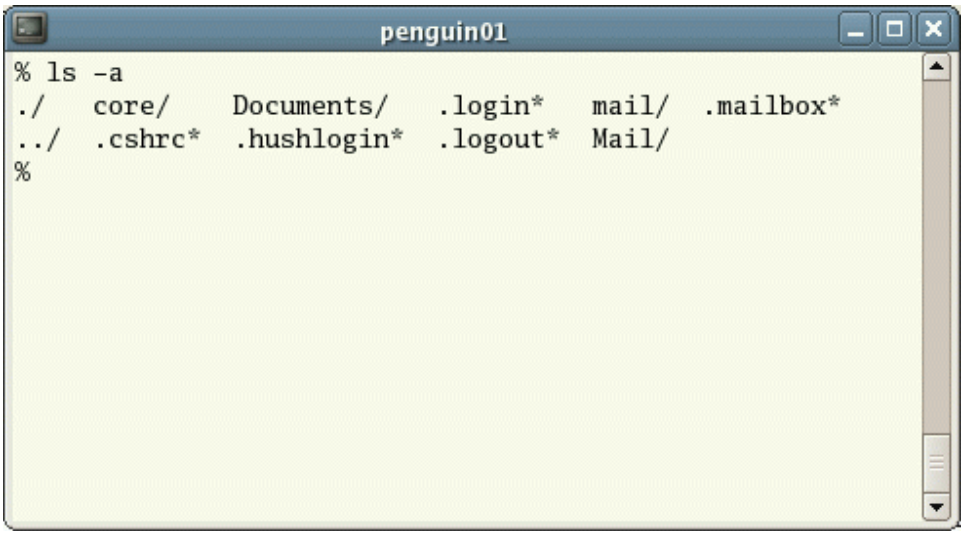
There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created.

ls does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.) Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with UNIX!!!

To list all files in your home directory including those whose names begin with a dot, type

% ls -a

As you can see, **ls -a** lists files that are normally hidden.



ls is an example of a command which can take options: **-a** is an example of an option. The options change the behaviour of the command. There are online manual pages that tell you which options a particular command can take, and how each option modifies the behaviour of the command. (See later in this tutorial)

1.2 Making Directories

mkdir (make directory)

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called unix stuff in your current working directory type

% mkdir unixstuff

To see the directory you have just created, type

% ls

1.3 Changing to a different directory

cd (change directory)

The command **cd directory** means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.

To change to the directory you have just made, type

% cd unixstuff

Type **ls** to see the contents (which should be empty)

Exercise 1a

Make another directory inside the **unixstuff** directory called **backups**

1.4 The directories . and ..

Still in the **unixstuff** directory, type

% ls -a

As you can see, in the **unixstuff** directory (and in all other directories), there are two special directories called (**.**) and (**..**)

The current directory (.)

In UNIX, (**.**) means the current directory, so typing

% cd .

NOTE: there is a space between cd and the dot

means stay where you are (the **unixstuff** directory).

This may not seem very useful at first, but using (**.**) as the name of the current directory will save a lot of typing, as we shall see later in the tutorial.

The parent directory (..)

(**..**) means the parent of the current directory, so typing

% cd ..

will take you one directory up the hierarchy (back to your home directory). Try it now.

Note: typing **cd** with no argument always returns you to your home directory. This is very useful if you are lost in the file system.

1.5 Pathnames

pwd (print working directory)

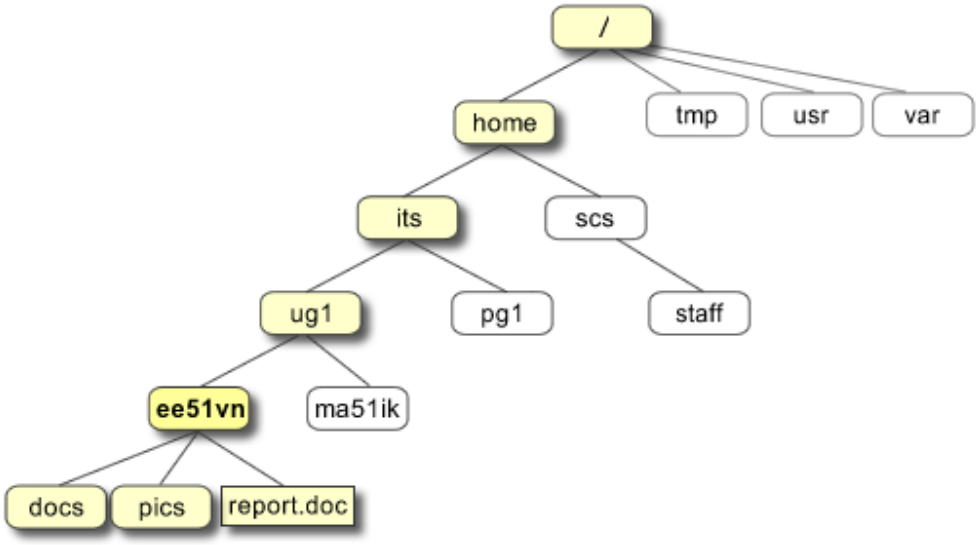
Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the absolute pathname of your home-directory, type **cd** to get back to your home-directory and then type

% pwd

The full pathname will look something like this -

/home/its/ug1/ee51vn

which means that **ee51vn** (your home directory) is in the sub-directory **ug1** (the group directory),which in turn is located in the **its** sub-directory, which is in the **home** sub-directory, which is in the top-level root directory called " / " .



Exercise 1b

Use the commands **cd, ls** and **pwd** to explore the file system.

(Remember, if you get lost, type **cd** by itself to return to your home-directory)

1.6 More about home directories and pathnames

Understanding pathnames

First type cd to get back to your home-directory, then type

% ls unixstuff

to list the conents of your unixstuff directory.

Now type

% ls backups

You will get a message like this -

backups: No such file or directory

The reason is, **backups** is not in your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either **cd** to the correct directory, or specify its full pathname. To list the contents of your backups directory, you must type

% ls unixstuff/backups

~ (your home directory)

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory. So typing

% ls ~/unixstuff

will list the contents of your unixstuff directory, no matter where you currently are in the file system.

What do you think

% ls ~

would list?

What do you think

% ls ~/..

would list?

Summary

Command	**Meaning**
ls	list files and directories
ls -a	list all files and directories
mkdir	make a directory
cd directory	change to named directory
cd	change to home-directory
cd ~	change to home-directory
cd ..	change to parent directory
pwd	display the path of the current directory

UNIX Tutorial Two

2.1 Copying Files

cp (copy)

cp file1 file2 is the command which makes a copy of **file1** in the current working directory and calls it **file2**

What we are going to do now, is to take a file stored in an open access area of the file system, and use the **cp** command to copy it to your unixstuff directory.

First, **cd** to your **unixstuff** directory.

% cd ~/unixstuff

Then at the UNIX prompt, type,

% cp /vol/examples/tutorial/science.txt .

Note: Don't forget the dot **.** at the end. Remember, in UNIX, the dot means the current directory.

The above command means copy the file **science.txt** to the current directory, keeping the name the same.

(Note: The directory **/vol/examples/tutorial/** is an area to which everyone in the school has read and copy access. If you are from outside the University, you can grab a copy of the file here. Use 'File/Save As..' from the menu bar to save it into your **unixstuff** directory.)

Exercise 2a

Create a backup of your **science.txt** file by copying it to a file called **science.bak**

2.2 Moving files

mv (move)

mv file1 file2 moves (or renames) **file1** to **file2**

To move a file from one place to another, use the mv command. This has the effect of moving rather than copying the file, so you end up with only one file rather than two.

It can also be used to rename a file, by moving the file to the same directory, but giving it a different name.

We are now going to move the file science.bak to your backup directory.

First, change directories to your unixstuff directory (can you remember how?). Then, inside the **unixstuff** directory, type

% mv science.bak backups/.

Type ls and ls backups to see if it has worked.

2.3 Removing files and directories

rm (remove), rmdir (remove directory)

To delete (remove) a file, use the **rm** command. As an example, we are going to create a copy of the **science.txt** file then delete it.

Inside your **unixstuff** directory, type

% cp science.txt tempfile.txt

% ls

% rm tempfile.txt

% ls

You can use the **rmdir** command to remove a directory (make sure it is empty first). Try to remove the **backups** directory. You will not be able to since UNIX will not let you remove a non-empty directory.

Exercise 2b

Create a directory called **tempstuff** using **mkdir** , then remove it using the **rmdir** command.

2.4 Displaying the contents of a file on the screen

clear (clear screen)

Before you start the next section, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood.

At the prompt, type

% clear

This will clear all text and leave you with the % prompt at the top of the window.

cat (concatenate)

The command cat can be used to display the contents of a file on the screen. Type:

% cat science.txt

As you can see, the file is longer than than the size of the window, so it scrolls past making it unreadable.

less

The command less writes the contents of a file onto the screen a page at a time. Type

% less science.txt

Press the [**space-bar**] if you want to see another page, and type [**q**] if you want to quit reading. As you can see, **less** is used in preference to **cat** for long files.

head

The **head** command writes the first ten lines of a file to the screen.

First clear the screen then type

% head science.txt

Then type

% head -5 science.txt

What difference did the -5 do to the head command?

tail

The **tail** command writes the last ten lines of a file to the screen.

Clear the screen and type

% tail science.txt

Q. How can you view the last 15 lines of the file?

2.5 Searching the contents of a file

Simple searching using less

Using **less**, you can search though a text file for a keyword (pattern). For example, to search through **science.txt** for the word **'science'**, type

% less science.txt

then, still in **less**, type a forward slash [/] followed by the word to search

/science

As you can see, **less** finds and highlights the keyword. Type [**n**] to search for the next occurrence of the word.

grep (don't ask why it is called grep)

grep is one of many standard UNIX utilities. It searches files for specified words or patterns. First clear the screen, then type

% grep science science.txt

As you can see, **grep** has printed out each line containg the word **science**.

Or has it ????

Try typing

% grep Science science.txt

The **grep** command is case sensitive; it distinguishes between Science and science.

To ignore upper/lower case distinctions, use the -i option, i.e. type

% grep -i science science.txt

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type

% grep -i 'spinning top' science.txt

Some of the other options of grep are:

-v display those lines that do NOT match

-n precede each matching line with the line number

-c print only the total count of matched lines

Try some of them and see the different results. Don't forget, you can use more than one option at a time. For example, the number of lines without the words science or Science is

% grep -ivc science science.txt

wc (word count)

A handy little utility is the **wc** command, short for word count. To do a word count on **science.txt**, type

% wc -w science.txt

To find out how many lines the file has, type

% wc -l science.txt

Summary

Command	Meaning
cp file1 file2	copy file1 and call it file2
mv file1 file2	move or rename file1 to file2
rm file	remove a file
rmdir directory	remove a directory
cat file	display a file
less file	display a file a page at a time
head file	display the first few lines of a file
tail file	display the last few lines of a file
grep 'keyword' file	search a file for keywords
wc file	count number of lines/words/characters in file

UNIX Tutorial Three

3.1 Redirection

Most processes initiated by UNIX commands write to the standard output (that is, they write to the terminal screen), and many take their input from the standard input (that is, they read it from the keyboard). There is also the standard error, where processes write their error messages, by default, to the terminal screen.

We have already seen one use of the **cat** command to write the contents of a file to the screen.

Now type **cat** without specifing a file to read

% cat

Then type a few words on the keyboard and press the [**Return**] key.

Finally hold the [**Ctrl**] key down and press [**d**] (written as **^D** for short) to end the input.

What has happened?

If you run the **cat** command without specifing a file to read, it reads the standard input (the keyboard), and on receiving the 'end of file' (**^D**), copies it to the standard output (the screen).

In UNIX, we can redirect both the input and the output of commands.

3.2 Redirecting the Output

We use the > symbol to redirect the output of a command. For example, to create a file called **list1** containing a list of fruit, type

% cat > list1

Then type in the names of some fruit. Press [**Return**] after each one.

pear

banana

apple

^D {this means press [Ctrl] and [d] to stop}

What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called **list1**

To read the contents of the file, type

% cat list1

Exercise 3a

Using the above method, create another file called **list2** containing the following fruit: orange, plum, mango, grapefruit. Read the contents of **list2**

3.2.1 Appending to a file

The form >> appends standard output to a file. So to add more items to the file **list1**, type

% cat >> list1

Then type in the names of more fruit

peach

grape

orange

^D (Control D to stop)

To read the contents of the file, type

% cat list1

You should now have two files. One contains six fruit, the other contains four fruit.

We will now use the cat command to join (concatenate) **list1** and **list2** into a new file called **biglist**. Type

% cat list1 list2 > biglist

What this is doing is reading the contents of **list1** and **list2** in turn, then outputing the text to the file **biglist**

To read the contents of the new file, type

% cat biglist

3.3 Redirecting the Input

We use the < symbol to redirect the input of a command.

The command sort alphabetically or numerically sorts a list. Type

% sort

Then type in the names of some animals. Press [Return] after each one.

dog

cat

bird

ape

^D (control d to stop)

The output will be

ape

bird

cat

dog

Using < you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, type

% sort < biglist

and the sorted list will be output to the screen.

To output the sorted list to a file, type,

% sort < biglist > slist

Use cat to read the contents of the file **slist**

3.4 Pipes

To see who is on the system with you, type

% who

One method to get a sorted list of names is to type,

% who > names.txt

% sort < names.txt

This is a bit slow and you have to remember to remove the temporary file called names when you have finished. What you really want to do is connect the output of the who command directly to the input of the sort command. This is exactly what pipes do. The symbol for a pipe is the vertical bar |

For example, typing

% who | sort

will give the same result as above, but quicker and cleaner.

To find out how many users are logged on, type

% who | wc -l

Exercise 3b

Using pipes, display all lines of **list1** and **list2** containing the letter 'p', and sort the result.

Answer available here

Summary

Command	**Meaning**
command > file	redirect standard output to a file
command >> file	append standard output to a file
command < file	redirect standard input from a file
command1 | command2	pipe the output of command1 to the input of command2
cat file1 file2 > file0	concatenate file1 and file2 to file0
sort	sort data
who	list users currently logged in

UNIX Tutorial Four

4.1 Wildcards

The * wildcard

The character ***** is called a wildcard, and will match against none or more character(s) in a file (or directory) name. For example, in your **unixstuff** directory, type

% ls list*

This will list all files in the current directory starting with **list....**

Try typing

% ls *list

This will list all files in the current directory ending with **....list**

The ? wildcard

The character **?** will match exactly one character.

So **?ouse** will match files like **house** and **mouse**, but not **grouse**.

Try typing

% ls ?list

4.2 Filename conventions

We should note here that a directory is merely a special type of file. So the rules and conventions for naming files apply also to directories.

In naming files, characters with special meanings such as **/ * & %** , should be avoided. Also, avoid using spaces within names. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and . (dot).

Good filenames	Bad filenames
project.txt	project
my_big_program.c	my big program.c
fred_dave.doc	fred & dave.doc

File names conventionally start with a lower-case letter, and may end with a dot followed by a group of letters indicating the contents of the file. For example, all files consisting of C code may be named with the ending **.c**, for example, **prog1.c** . Then in

order to list all files containing C code in your home directory, you need only type **ls *.c** in that directory.

4.3 Getting Help

On-line Manuals

There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behaviour of the command. Type **man command** to read the manual page for a particular command.

For example, to find out more about the **wc** (word count) command, type

% man wc

Alternatively

% whatis wc

gives a one-line description of the command, but omits any information about options etc.

Apropos

When you are not sure of the exact name of a command,

% apropos keyword

will give you the commands with keyword in their manual page header. For example, try typing

% apropos copy

Summary

Command	Meaning
*	match any number of characters
?	match one character
man command	read the online manual page for a command
whatis command	brief description of a command
apropos keyword	match commands with keyword in their man pages

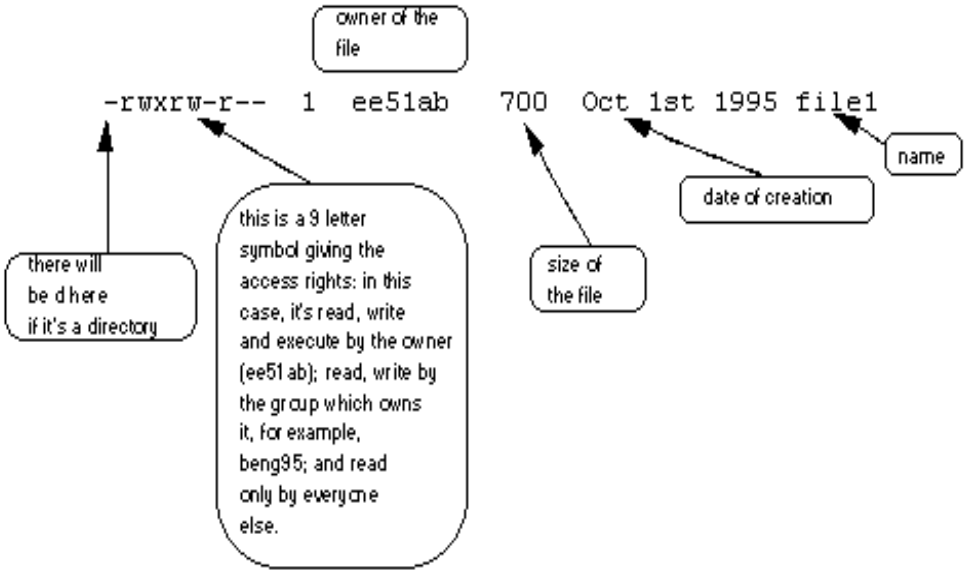
UNIX Tutorial Five

5.1 File system security (access rights)

In your **unixstuff** directory, type

% ls -l (l for long listing!)

You will see that you now get lots of details about the contents of your directory, similar to the example below.



Each file (and directory) has associated access rights, which may be found by typing **ls -l**. Also, **ls -lg** gives additional information as to which group owns the file (beng95 in the following example):

-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the left hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string.

The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

- The left group of 3 gives the file permissions for the user that owns the file (or directory) (ee51ab in the above example);
- the middle group gives the permissions for the group of people to whom the file (or directory) belongs (eebeng95 in the above example);

- the rightmost group gives the permissions for all others.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access rights on files.

- r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file
- w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file
- x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate

Access rights on directories.

- r allows users to list files in the directory;
- w means that users may delete files from the directory or move files into it;
- x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.

So, in order to read a file, you must have execute permission on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.

Some examples

-rwxrwxrwx	a file that everyone can read, write and execute (and delete).
-rw-------	a file that only the owner can read and write - no-one else can read or write and no-one has execution rights (e.g. your mailbox file).

5.2 Changing access rights

chmod (changing a file mode)

Only the owner of a file can use chmod to change the permissions of a file. The options of chmod are as follows

Symbol	**Meaning**
u	user
g	group

o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

For example, to remove read write and execute permissions on the file **biglist** for the group and others, type

% chmod go-rwx biglist

This will leave the other permissions unaffected.

To give read and write permissions on the file **biglist** to all,

% chmod a+rw biglist

Exercise 5a

Try changing access permissions on the file **science.txt** and on the directory **backups**

Use **ls -l** to check that the permissions have changed.

5.3 Processes and Jobs

A process is an executing program identified by a unique PID (process identifier). To see information about your processes, with their associated PID and status, type

% ps

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the UNIX prompt until the current process has finished executing. Some processes take a long time to run and hold up the terminal. Backgrounding a long process has the effect that the UNIX prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

Running background processes

To background a process, type an **&** at the end of the command line. For example, the command **sleep** waits a given number of seconds before continuing. Type

% sleep 10

This will wait 10 seconds before returning the command prompt %. Until the command prompt is returned, you can do nothing except wait.

To run sleep in the background, type

% sleep 10 &

[1] 6259

The **&** runs the job in the background and returns the prompt straight away, allowing you do run other programs while waiting for that one to finish.

The first line in the above example is typed in by the user; the next line, indicating job number and PID, is returned by the machine. The user is be notified of a job number (numbered from 1) enclosed in square brackets, together with a PID and is notified when a background process is finished. Backgrounding is useful for jobs which will take a long time to complete.

Backgrounding a current foreground process

At the prompt, type

% sleep 1000

You can suspend the process running in the foreground by typing **^Z**, i.e.hold down the [**Ctrl**] key and type [**z**]. Then to put it in the background, type

% bg

Note: do not background programs that require user interaction e.g. vi

5.4 Listing suspended and background processes

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type

% jobs

An example of a job list could be

[1] Suspended sleep 1000

[2] Running netscape

[3] Running matlab

To restart (foreground) a suspended processes, type

% fg %jobnumber

For example, to restart sleep 1000, type

% fg %1

Typing **fg** with no job number foregrounds the last suspended process.

5.5 Killing a process

kill (terminate or signal a process)

It is sometimes necessary to kill a process (for example, when an executing program is in an infinite loop)

To kill a job running in the foreground, type **^C** (control c). For example, run

% sleep 100

^C

To kill a suspended or background process, type

% kill %jobnumber

For example, run

% sleep 100 &

% jobs

If it is job number 4, type

% kill %4

To check whether this has worked, examine the job list again to see if the process has been removed.

ps (process status)

Alternatively, processes can be killed by finding their process numbers (PIDs) and using kill PID_number

% sleep 1000 &

% ps

PID TT S TIME COMMAND

20077 pts/5 S 0:05 sleep 1000

21563 pts/5 T 0:00 netscape

21873 pts/5 S 0:25 nedit

To kill off the process **sleep 1000**, type

% kill 20077

and then type **ps** again to see if it has been removed from the list.

If a process refuses to be killed, uses the **-9** option, i.e. type

% kill -9 20077

Note: It is not possible to kill off other users' processes !!!

Summary

Command	Meaning
ls -lag	list access rights for all files
chmod [options] file	change access rights for named file
command &	run command in background
^C	kill the job running in the foreground
^Z	suspend the job running in the foreground
bg	background the suspended job
jobs	list current jobs
fg %1	foreground job number 1
kill %1	kill job number 1
ps	list current processes
kill 26152	kill process number 26152

UNIX Tutorial Six

Other useful UNIX commands

quota

All students are allocated a certain amount of disk space on the file system for their personal files, usually about 100Mb. If you go over your quota, you are given 7 days to remove excess files.

To check your current quota and how much of it you have used, type

% quota -v

df

The **df** command reports on the space left on the file system. For example, to find out how much space is left on the fileserver, type

% df .

du

The **du** command outputs the number of kilobyes used by each subdirectory. Useful if you have gone over quota and you want to find out which directory has the most files. In your home-directory, type

% du -s *

The **-s** flag will display only a summary (total size) and the * means all files and directories.

gzip

This reduces the size of a file, thus freeing valuable disk space. For example, type

% ls -l science.txt

and note the size of the file using **ls -l** . Then to compress science.txt, type

% gzip science.txt

This will compress the file and place it in a file called **science.txt.gz**

To see the change in size, type **ls -l** again.

To expand the file, use the **gunzip** command.

% gunzip science.txt.gz

zcat

zcat will read gzipped files without needing to uncompress them first.

% zcat science.txt.gz

If the text scrolls too fast for you, pipe the output though **less** .

% zcat science.txt.gz | less

file

file classifies the named files according to the type of data they contain, for example ascii (text), pictures, compressed data, etc.. To report on all files in your home directory, type

% file *

diff

This command compares the contents of two files and displays the differences. Suppose you have a file called **file1** and you edit some part of it and save it as **file2**. To see the differences type

% diff file1 file2

Lines beginning with a **<** denotes file1, while lines beginning with a **>** denotes file2.

find

This searches through the directories for files and directories with a given name, date, size, or any other attribute you care to specify. It is a simple command but with many options - you can read the manual by typing **man find**.

To search for all fies with the extention **.txt**, starting at the current directory (.) and working through all sub-directories, then printing the name of the file to the screen, type

% find . -name "*.txt" -print

To find files over 1Mb in size, and display the result as a long listing, type

% find . -size +1M -ls

history

The C shell keeps an ordered list of all the commands that you have entered. Each command is given a number according to the order it was entered.

% history (show command history list)

If you are using the C shell, you can use the exclamation character (**!**) to recall commands easily.

% !! (recall last command)

% !-3 (recall third most recent command)

% !5 (recall 5th command in list)

% !grep (recall last command starting with grep)

You can increase the size of the history buffer by typing

% set history=100

UNIX Tutorial Seven

7.1 Compiling UNIX software packages

We have many public domain and commercial software packages installed on our systems, which are available to all users. However, students are allowed to download and install small software packages in their own home directory, software usually only useful to them personally.

There are a number of steps needed to install the software.

- Locate and download the source code (which is usually compressed)
- Unpack the source code
- Compile the code
- Install the resulting executable
- Set paths to the installation directory

Of the above steps, probably the most difficult is the compilation stage.

Compiling Source Code

All high-level language code must be converted into a form the computer understands. For example, C language source code is converted into a lower-level language called assembly language. The assembly language code made by the previous stage is then converted into object code which are fragments of code which the computer understands directly. The final stage in compiling a program involves linking the object code to code libraries which contain certain built-in functions. This final stage produces an executable program.

To do all these steps by hand is complicated and beyond the capability of the ordinary user. A number of utilities and tools have been developed for programmers and end-users to simplify these steps.

make and the Makefile

The **make** command allows programmers to manage large programs or groups of programs. It aids in developing large programs by keeping track of which portions of the entire program have been changed, compiling only those parts of the program which have changed since the last compile.

The **make** program gets its set of compile rules from a text file called **Makefile** which resides in the same directory as the source files. It contains information on how to

compile the software, e.g. the optimisation level, whether to include debugging info in the executable. It also contains information on where to install the finished compiled binaries (executables), manual pages, data files, dependent library files, configuration files, etc.

Some packages require you to edit the Makefile by hand to set the final installation directory and any other parameters. However, many packages are now being distributed with the GNU configure utility.

configure

As the number of UNIX variants increased, it became harder to write programs which could run on all variants. Developers frequently did not have access to every system, and the characteristics of some systems changed from version to version. The GNU configure and build system simplifies the building of programs distributed as source code. All programs are built using a simple, standardised, two step process. The program builder need not install any special tools in order to build the program.

The **configure** shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a **Makefile** in each directory of the package.

The simplest way to compile a package is:

1. **cd** to the directory containing the package's source code.
2. Type **./configure** to configure the package for your system.
3. Type **make** to compile the package.
4. Optionally, type **make check** to run any self-tests that come with the package.
5. Type **make install** to install the programs and any data files and documentation.
6. Optionally, type **make clean** to remove the program binaries and object files from the source code directory

The configure utility supports a wide variety of options. You can usually use the **--help** option to get a list of interesting options for a particular configure script.

The only generic options you are likely to use are the **--prefix** and **--exec-prefix** options. These options are used to specify the installation directories.

The directory named by the **--prefix** option will hold machine independent files such as documentation, data and configuration files.

The directory named by the **--exec-prefix** option, (which is normally a subdirectory of the **--prefix** directory), will hold machine dependent files such as executables.

7.2 Downloading source code

For this example, we will download a piece of free software that converts between different units of measurements.

First create a download directory

% mkdir download

Download the software here and save it to your new download directory.

7.3 Extracting the source code

Go into your **download** directory and list the contents.

% cd download

% ls -l

As you can see, the filename ends in tar.gz. The tar command turns several files and directories into one single tar file. This is then compressed using the gzip command (to create a tar.gz file).

First unzip the file using the gunzip command. This will create a .tar file.

% gunzip units-1.74.tar.gz

Then extract the contents of the tar file.

% tar -xvf units-1.74.tar

Again, list the contents of the **download** directory, then go to the **units-1.74** sub-directory.

% cd units-1.74

7.4 Configuring and creating the Makefile

The first thing to do is carefully read the **README** and **INSTALL** text files (use the less command). These contain important information on how to compile and run the software.

The units package uses the GNU configure system to compile the source code. We will need to specify the installation directory, since the default will be the main system area which you will not have write permissions for. We need to create an install directory in your home directory.

% mkdir ~/units174

Then run the configure utility setting the installation path to this.

% ./configure --prefix=$HOME/units174

NOTE: The **$HOME** variable is an example of an environment variable. The value of **$HOME** is the path to your home directory. Just type

% echo $HOME

to show the contents of this variable. We will learn more about environment variables in a later chapter.

If **configure** has run correctly, it will have created a Makefile with all necessary options. You can view the Makefile if you wish (use the **less** command), but do not edit the contents of this.

7.5 Building the package

Now you can go ahead and build the package by running the make command.

% make

After a minute or two (depending on the speed of the computer), the executables will be created. You can check to see everything compiled successfully by typing

% make check

If everything is okay, you can now install the package.

% make install

This will install the files into the **~/units174** directory you created earlier.

7.6 Running the software

You are now ready to run the software (assuming everything worked).

% cd ~/units174

If you list the contents of the units directory, you will see a number of subdirectories.

bin	The binary executables
info	GNU info formatted documentation
man	Man pages
share	Shared data files

To run the program, change to the **bin** directory and type

% ./units

As an example, convert 6 feet to metres.

You have: 6 feet

You want: metres

*** 1.8288**

If you get the answer 1.8288, congratulations, it worked.

To view what units it can convert between, view the data file in the share directory (the list is quite comprehensive).

To read the full documentation, change into the **info** directory and type

% info --file=units.info

7.7 Stripping unnecessary code

When a piece of software is being developed, it is useful for the programmer to include debugging information into the resulting executable. This way, if there are problems encountered when running the executable, the programmer can load the executable into a debugging software package and track down any software bugs.

This is useful for the programmer, but unnecessary for the user. We can assume that the package, once finished and available for download has already been tested and debugged. However, when we compiled the software above, debugging information was still compiled into the final executable. Since it is unlikey that we are going to need this debugging information, we can strip it out of the final executable. One of the advantages of this is a much smaller executable, which should run slightly faster.

What we are going to do is look at the before and after size of the binary file. First change into the **bin** directory of the units installation directory.

% cd ~/units174/bin

% ls -l

As you can see, the file is over 100 kbytes in size. You can get more information on the type of file by using the file command.

% file units

units: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), not stripped

To strip all the debug and line numbering information out of the binary file, use the strip command

% strip units

% ls -l

As you can see, the file is now 36 kbytes - a third of its original size. Two thirds of the binary file was debug code!!!

Check the file information again.

% file units

units: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), stripped

Sometimes you can use the **make** command to install pre-stripped copies of all the binary files when you install the package. Instead of typing **make install**, simply type **make install-strip**

UNIX Tutorial Eight

8.1 UNIX Variables

Variables are a way of passing information from the shell to programs when you run them. Programs look "in the environment" for particular variables and if they are found will use the values stored. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

Standard UNIX variables are split into two categories, environment variables and shell variables. In broad terms, shell variables apply only to the current instance of the shell and are used to set short-term working conditions; environment variables have a farther reaching significance, and those set at login are valid for the duration of the session. By convention, environment variables have UPPER CASE and shell variables have lower case names.

8.2 Environment Variables

An example of an environment variable is the OSTYPE variable. The value of this is the current operating system you are using. Type

% echo $OSTYPE

More examples of environment variables are

- USER (your login name)
- HOME (the path name of your home directory)
- HOST (the name of the computer you are using)
- ARCH (the architecture of the computers processor)
- DISPLAY (the name of the computer screen to display X windows)
- PRINTER (the default printer to send print jobs)
- PATH (the directories the shell should search to find a command)

Finding out the current values of these variables.

ENVIRONMENT variables are set using the **setenv** command, displayed using the **printenv** or **env** commands, and unset using the **unsetenv** command.

To show all values of these variables, type

% printenv | less

8.3 Shell Variables

An example of a shell variable is the history variable. The value of this is how many shell commands to save, allow the user to scroll back through all the commands they have previously entered. Type

% echo $history

More examples of shell variables are

- cwd (your current working directory)
- home (the path name of your home directory)
- path (the directories the shell should search to find a command)
- prompt (the text string used to prompt for interactive commands shell your login shell)

Finding out the current values of these variables.

SHELL variables are both set and displayed using the **set** command. They can be unset by using the unset command.

To show all values of these variables, type

% set | less

So what is the difference between PATH and path ?

In general, environment and shell variables that have the same name (apart from the case) are distinct and independent, except for possibly having the same initial values. There are, however, exceptions.

Each time the shell variables home, user and term are changed, the corresponding environment variables HOME, USER and TERM receive the same values. However, altering the environment variables has no effect on the corresponding shell variables.

PATH and path specify directories to search for commands and programs. Both variables always represent the same directory list, and altering either automatically causes the other to be changed.

8.4 Using and setting variables

Each time you login to a UNIX host, the system looks in your home directory for initialisation files. Information in these files is used to set up your working environment. The C and TC shells uses two files called .login and .cshrc (note that both file names begin with a dot).

At login the C shell first reads **.cshrc** followed by **.login**

.login is to set conditions which will apply to the whole session and to perform actions that are relevant only at login.

.cshrc is used to set conditions and perform actions specific to the shell and to each invocation of it.

The guidelines are to set ENVIRONMENT variables in the **.login** file and SHELL variables in the **.cshrc** file.

WARNING: NEVER put commands that run graphical displays (e.g. a web browser) in your .cshrc or .login file.

8.5 Setting shell variables in the .cshrc file

For example, to change the number of shell commands saved in the history list, you need to set the shell variable history. It is set to 100 by default, but you can increase this if you wish.

% set history = 200

Check this has worked by typing

% echo $history

However, this has only set the variable for the lifetime of the current shell. If you open a new xterm window, it will only have the default history value set. To PERMANENTLY set the value of history, you will need to add the set command to the .cshrc file.

First open the **.cshrc** file in a text editor. An easy, user-friendly editor to use is nedit.

% nedit ~/.cshrc

Add the following line AFTER the list of other commands.

set history = 200

Save the file and force the shell to reread its .cshrc file buy using the shell source command.

% source .cshrc

Check this has worked by typing

% echo $history

8.6 Setting the path

When you type a command, your path (or PATH) variable defines in which directories the shell will look to find the command you typed. If the system returns a message

saying "command: Command not found", this indicates that either the command doesn't exist at all on the system or it is simply not in your path.

For example, to run units, you either need to directly specify the units path (**~/units174/bin/units**), or you need to have the directory **~/units174/bin** in your path.

You can add it to the end of your existing path (the **$path** represents this) by issuing the command:

% set path = ($path ~/units174/bin)

Test that this worked by trying to run units in any directory other that where units is actually located.

% cd

% units

To add this path PERMANENTLY, add the following line to your .cshrc AFTER the list of other commands.

set path = ($path ~/units174/bin)

Reference

http://info.ee.surrey.ac.uk/Teaching/Unix/index.html

What Is UNIX?

- UNIX is a computer operating system.
- An operating system is the program that controls all the other parts of a computer system, both the hardware and the software. It allocates the computer's resources and schedules tasks. It allows you to make use of the facilities provided by the system. Every computer requires an operating system.
- UNIX is a multi-user, multi-tasking operating system. Multiple users may have multiple tasks running simultaneously. This is very different than PC operating systems.
- UNIX is a machine independent operating system. Not specific to just one type of computer hardware. Designed from the beginning to be independent of the computer hardware.

- UNIX is a software development environment. Was born in and designed to function within this type of environment.
- The "UNIX" trademark, previously owned by AT&T and then deeded to UNIX Systems Laboratories (USL), an AT&T subsidiary, passed to Novell when it acquired USL. After a brief period of negotiations with rival Unix vendors Sun Microsystems, Santa Cruz Operation, International Business Machines, and Hewlett-Packard, Novell granted exclusive licensing rights to the UNIX trademark to X/Open Co. Ltd., an Open Systems industry standards branding agent based in the United Kingdom.

History of UNIX

- 1969: Developed at AT&T Bell Labs in Murray Hill, New Jersey, one of the largest research facilities in the world. Created in an environment when most computer jobs were fed into a batch system.
  Developed by researchers who needed a set of computing tools to help them with their projects and their collaborators. Allowed a group of people working together on a project to share selected data and programs.
- 1975: AT&T makes UNIX widely available - offered to educational institutions at minimal cost. Becomes popular with university computer science programs.
  AT&T distributes standard versions in source form: Version 6 (1975), Version 7 (1978), System III (1981).
- 1984 to date: University of California, Berkeley adds major enhancements, creates Berkeley Standard Distribution (BSD)
- 1984 to date: Many Berkeley features incorporated into new AT&T version: System V
- UNIX has become the operating system of choice for engineering and scientific workstations.
- Two variations maintain popularity today, AT&T System V based and the Berkeley Standard Distribution.
- Current versions (1/95)are System V release 4.2 .and 4.4 BSD
- Work is in progress to develop a Portable Operating System specification based on UNIX (IEEE POSIX committee).

UNIX Philosophy

- Make each program do one thing well. Reusable software tools: 1 tool = 1 function
- Expect the output of every program to become the input of another, yet unknown, program to combine simple tools to perform complex tasks
- Prototyping: get something small working as soon as possible and modify it incrementally until it is finished
- Use terse commands and messages: reduces typing and screen output

Why UNIX?

- Hardware independence
    - operating system code is written in C language rather than a specific assembly language
    - operating system software can be easily moved from one hardware system to another
    - UNIX applications can be easily moved to other UNIX machines. Porting is usually as simple as transfer of the source and a recompile
- Productive environment for software development
    - rich set of tools
    - versatile command language
- UNIX is available at virtually all HPC centers, allowing researchers relative ease in utilizing the facilities at each center.
- Distributed processing and multi-tasking

UNIX Components

- Kernel
    - The core of the UNIX system. Loaded at system start up (boot). Memory-resident control program.
    - Manages the entire resources of the system, presenting them to you and every other user as a coherent system. Provides service to user applications such as device management, process scheduling, etc.
    - Example functions performed by the kernel are:
        - managing the machine's memory and allocating it to each process.

        - scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible.
        - accomplishing the transfer of data from one part of the machine to another
        - interpreting and executing instructions from the shell
        - enforcing file access permissions
    - You do not need to know anything about the kernel in order to use a UNIX system. These details are provided for your information only.
- Shell
    - Whenever you login to a Unix system you are placed in a shell program. The shell's prompt is usually visible at the cursor's position on your screen. To get your work done, you enter commands at this prompt.
    - The shell is a command interpreter; it takes each command and passes it to the operating system kernel to be acted upon. It then displays the results of this operation on your screen.
    - Several shells are usually available on any UNIX system, each with its own strengths and weaknesses.
    - Different users may use different shells. Initially, your system adminstrator will supply a default shell, which can be overridden or changed. The most commonly available shells are:
        - Bourne shell (sh)
        - C shell (csh)
        - Korn shell (ksh)
        - TC Shell (tcsh)
        - Bourne Again Shell (bash)
    - Each shell also includes its own programming language. Command files, called "shell scripts" are used to accomplish a series of tasks.
- Utilities
    - UNIX provides several hundred utility programs, often referred to as commands.
    - Accomplish universal functions

        - editing
        - file maintenance
        - printing
        - sorting
        - programming support
        - online info
        - etc.
    - Modular: single functions can be grouped to perform more complex tasks

TELNET

Telnet is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communications facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

Telnet is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communications facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

Telnet was developed in 1969 beginning with RFC 15,extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards.

Historically, Telnet provided access to a command-line interface (usually, of an operating system) on a remote host. Most network equipment and operating systems with a TCP/IP stack support a Telnet service for remote configuration (including systems based on Windows NT). Because of security issues with Telnet, its use for this purpose has waned in favor of SSH.

The term *telnet* may also refer to the software that implements the client part of the protocol. Telnet client applications are available for virtually all computer platforms. *Telnet* is also used as a verb. *To telnet* means to establish a connection with the Telnet protocol, either with command line client or with a programmatic interface. For example, a common directive might be: "*To change your password, telnet to the server, login and*

run the passwd command." Most often, a user will be *telnetting* to a Unix-like server system or a network device (such as a router) and obtain a login prompt to a command line text interface or a character-based full-screen manager

What is Telnet?

Telnet is a user command and an underlying TCP/IP protocol for accessing remote computers. Through Telnet, an administrator or another user can access someone else's computer remotely. On the Web, HTTP and FTP protocols allow you to request specific files from remote computers, but not to actually be logged on as a user of that computer. With Telnet, you log on as a regular user with whatever privileges you may have been granted to the specific application and data on that computer.

A Telnet command request looks like this (the computer name is made-up):

File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is a standard network protocol used to copy a file from one host to another over a TCP-based network, such as the Internet. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server. FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.

The first FTP client applications were interactive command-line tools, implementing standard commands and syntax. Graphical user interface clients have since been developed for many of the popular desktop operating systems in use today.

Short for ***F**ile **T**ransfer **P**rotocol,* the protocol for exchanging files over the Internet. FTP works in the same way as HTTP for transferring Web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet in that, like these technologies, FTP uses the Internet's TCP/IP protocols to enable data transfer.

FTP is most commonly used to download a file from a server using the Internet or to upload a file to a server (e.g., uploading a Web page file to a server).

Hardware – topology

A personal computer is made up of multiple physical components of **computer hardware**, upon which can be installed a system software called operating system and a multitude of software applications to perform the operator's desired functions.

Though a PC comes in many different forms, a typical personal computer consists of a case or chassis in a tower shape (desktop), containing components such as a motherboard

Hardware of a modern Personal Computer.

1. Monitor
2. Motherboard
3. CPU
4. RAM
5. Expansion cards
6. Power supply
7. Optical disc drive
8. Hard disk drive
9. Keyboard
10. Mouse

A personal computer is made up of multiple physical components of computer hardware, upon which can be installed a system software called operating system and a multitude of software applications to perform the operator's desired functions.

Though a PC comes in many different forms, a typical personal computer consists of a case or chassis in a tower shape (desktop), containing components such as a motherboard.

Motherboard

The motherboard is the main component inside the case. It is a large rectangular board with integrated circuitry that connects the rest of the parts of the computer including the CPU, the RAM, the disk drives (CD, DVD, hard disk, or any others) as well as any peripherals connected via the ports or the expansion slots.

Components directly attached to the motherboard include:

- The **central processing unit (**CPU) performs most of the calculations which enable a computer to function, and is sometimes referred to as the "brain" of the computer. It is usually cooled by a heat sink and fan.
- The **chip set** mediates communication between the CPU and the other components of the system, including main memory.

- **RAM** (Random Access Memory) stores resident part of the current running OS (OS core and so on) and all running processes (applications parts, using CPU or input/output (I/O) channels or waiting for CPU or I/O channels).
- The **BIOS** includes boot firmware and power management. The **B**asic **I**nput **O**utput **S**ystem tasks are handled by operating system drivers.
- **Internal Buses** connect the CPU to various internal components and to expansion cards for graphics and sound.
    - **Current**
        - The north bridge memory controller, for RAM and PCI Express
        - PCI Express, for expansion cards such as graphics and physics processors, and high-end network interfaces
        - PCI, for other expansion cards
        - SATA, for disk drives
    - **Obsolete**
        - ATA (superseded by SATA)
        - AGP (superseded by PCI Express)
        - VLB VESA Local Bus (superseded by AGP)
        - ISA (expansion card slot format obsolete in PCs, but still used in industrial computers)
- **External Bus Controllers** support ports for external peripherals. These ports may be controlled directly by the south bridge I/O controller or based on expansion cards attached to the motherboard through the PCI bus.
    - USB
    - FireWire
    - eSATA
    - SCSI

Power supply

Inside a custom-built computer: the power supply at the bottom has its own cooling fan.

A power supply unit (PSU) converts alternating current (AC) electric power to low-voltage DC power for the internal components of the computer. Some power

supplies have a switch to change between 230 V and 115 V. Other models have automatic sensors that switch input voltage automatically, or are able to accept any voltage between those limits. Power supply units used in computers are nearly always switch mode power supplies (SMPS). The SMPS provides regulated direct current power at the several voltages required by the motherboard and accessories such as disk drives and cooling fans.

Removable media devices

- CD (compact disc) - the most common type of removable media, suitable for music and data.
  - CD-ROM Drive - a device used for reading data from a CD.
  - CD Writer - a device used for both reading and writing data to and from a CD.
- DVD (digital versatile disc) - a popular type of removable media that is the same dimensions as a CD but stores up to 12 times as much information. It is the most common way of transferring digital video, and is popular for data storage.
  - DVD-ROM Drive - a device used for reading data from a DVD.
  - DVD Writer - a device used for both reading and writing data to and from a DVD.
  - DVD-RAM Drive - a device used for rapid writing and reading of data from a special type of DVD.
- Blu-ray Disc - a high-density optical disc format for data and high-definition video. Can store 70 times as much information as a CD.
  - BD-ROM Drive - a device used for reading data from a Blu-ray disc.
  - BD Writer - a device used for both reading and writing data to and from a Blu-ray disc.
- HD DVD - a discontinued competitor to the Blu-ray format.
- Floppy disk - an outdated storage device consisting of a thin disk of a flexible magnetic storage medium. Used today mainly for loading RAID drivers.
- Iomega Zip drive - an outdated medium-capacity removable disk storage system, first introduced by Iomega in 1994.
- USB flash drive - a flash memory data storage device integrated with a USB interface, typically small, lightweight, removable, and rewritable. Capacities vary,

from hundreds of megabytes (in the same ballpark as CDs) to tens of gigabytes (surpassing, at great expense, Blu-ray discs).

- Tape drive - a device that reads and writes data on a magnetic tape, used for long term storage and backups.

Secondary storage

Hardware that keeps data inside the computer for later use and remains persistent even when the computer has no power.

- Hard disk - for medium-term storage of data.
- Solid-state drive - a device similar to hard disk, but containing no moving parts and stores data in a digital format.
- RAID array controller - a device to manage several internal or external hard disks and optionally some peripherals in order to achieve performance or reliability improvement in what is called a RAID array.

Sound card

Enables the computer to output sound to audio devices, as well as accept input from a microphone. Most modern computers have sound cards built-in to the motherboard, though it is common for a user to install a separate sound card as an upgrade. Most sound cards, either built-in or added, have surround sound capabilities.

Input and output peripherals

Input and output devices are typically housed externally to the main computer chassis. The following are either standard or very common to many computer systems.

Wheel Mouse

Input

- Text input devices
    - Keyboard - a device to input text and characters by depressing buttons (referred to as keys or buttons).
- Pointing devices
    - Mouse - a pointing device that detects two dimensional motion relative to its supporting surface.
        - Optical Mouse - uses light to determine mouse motion.

    - Trackball - a pointing device consisting of an exposed protruding ball housed in a socket that detects rotation about two axes.
    - Touchscreen - senses the user pressing directly on the display
- Gaming devices
    - Joystick - a control device that consists of a handheld stick that pivots around one end, to detect angles in two or three dimensions.
    - Game pad - a hand held game controller that relies on the digits (especially thumbs) to provide input.
    - Game controller - a specific type of controller specialized for certain gaming purposes.
- Image, Video input devices
    - Image scanner - a device that provides input by analyzing images, printed text, handwriting, or an object.
    - Web cam - a video camera used to provide visual input that can be easily transferred over the internet.
- Audio input devices
    - Microphone - an acoustic sensor that provides input by converting sound into electrical signals.

Output

- Printer - a device that produces a permanent human-readable text of graphic document.
- Speakers - typically a pair of devices (2 channels) which convert electrical signals into audio.
    - Headphones - for a single user hearing the audio.
- Monitor - an electronic visual display with textual and graphical information from the computer.
    - CRT - (Cathode Ray Tube) display
    - LCD - (Liquid Crystal Display) as of 2010, it is the primary visual display for personal computers.
    - LED - (light-emitting diode) display
    - OLED - Organic Light-Emitting Diode

SEARCH ENGINES

The three most widely used web search engines and their approximate share as of late 2010.A web search engine is designed to search for information on the World Wide Web and FTP servers. The search results are generally presented in a list of results and are often called *hits*. The information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input. The three most widely used web search engines and their approximate share as of late 2010

A web search engine is designed to search for information on the World Wide Web and FTP servers. The search results are generally presented in a list of results and are often called *hits*. The information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

History

Timeline (full list)		
Year	**Engine**	**Current status**
1993	W3Catalog	Closed
	Aliweb	Active
	JumpStation	Active
1994	WebCrawler	Active
	Go.com	Active
	Lycos	Active
1995	AltaVista	Active
	Daum	Active
	Open Text Web Index	Active [1]
	Magellan	Active

	Excite	Active
	SAPO	Active
	Yahoo!	Active, Launched as a directory
1996	Dogpile	Active
	Inktomi	Active
	HotBot	Active
	Ask Jeeves	Active
1997	Northern Light	Active
	Yandex	Active
1998	Google	Active
	MSN Search	Active as Bing
1999	AlltheWeb	Active
	GenieKnows	Active
	Naver	Active
	Teoma	Active
	Vivisimo	Active
2000	Baidu	Active
	Exalead	Active
2002	Inktomi	Acquired by Yahoo!
2003	Info.com	Active
2004	Yahoo! Search	Active, Launched own web search (see Yahoo! Directory, 1995)
	A9.com	Closed
	Sogou	Active
2005	Ask.com	Active
	GoodSearch	Active
	SearchMe	Active
2006	wikiseek	Active
	Quaero	Active
	Ask.com	Active

	Live Search	Active as Bing, Launched as rebranded MSN Search
	ChaCha	Active
	Guruji.com	Active
2007	wikiseek	Closed
	Sproose	Closed
	Wikia Search	Active
	Blackle.com	Active
2008	Powerset	Acquired by Microsoft
	Picollator	Closed
	Viewzi	Closed
	Boogami	Active
	LeapFish	Active
	Forestle	Active
	VADLO	Active
	Duck Duck Go	Active
2009	Bing	Active, Launched as rebranded Live Search
	Yebol	Active
	Mugurdy	Closed due to a lack of funding
	Goby	Active
2010	Yandex	Active, Launched global (English) search
	Cuil	Closed
	Blekko	Active
	Viewzi	Closed due to a lack of funding
	Yummly	Active
2011	Exalead	Acquired by Dassault Systèmes
	Yebol	Domain name expired

During the early development of the web, there was a list of webservers edited by Tim Berners-Lee and hosted on the CERN webserver. One historical snapshot from 1992 remains. As more webservers went online the central list could not keep up. On the NCSA site new servers were announced under the title "What's New!"

The very first tool used for searching on the Internet was Archie. The name stands for "archive" without the "v". It was created in 1990 by Alan Emtage, Bill Heelan and J. Peter Deutsch, computer science students at McGill University in Montreal. The program downloaded the directory listings of all the files located on public anonymous FTP (File Transfer Protocol) sites, creating a searchable database of file names; however, Archie did not index the contents of these sites since the amount of data was so limited it could be readily searched manually.

The rise of Gopher (created in 1991 by Mark McCahill at the University of Minnesota) led to two new search programs, Veronica and Jughead. Like Archie, they searched the file names and titles stored in Gopher index systems. Veronica (*V*ery *E*asy *R*odent-*O*riented *N*et-wide *I*ndex to *C*omputerized *A*rchives) provided a keyword search of most Gopher menu titles in the entire Gopher listings. Jughead (*J*onzy's *U*niversal *G*opher *H*ierarchy *E*xcavation *A*nd *D*isplay) was a tool for obtaining menu information from specific Gopher servers. While the name of the search engine "Archie" was not a reference to the Archie comic book series, "Veronica" and "Jughead" are characters in the series, thus referencing their predecessor.

In the summer of 1993, no search engine existed yet for the web, though numerous specialized catalogues were maintained by hand. Oscar Nierstrasz at the University of Geneva wrote a series of Perl scripts that would periodically mirror these pages and rewrite them into a standard format which formed the basis for W3Catalog, the web's first primitive search engine, released on September 2, 1993.

In June 1993, Matthew Gray, then at MIT, produced what was probably the first web robot, the Perl-based World Wide Web Wanderer, and used it to generate an index called 'Wandex'. The purpose of the Wanderer was to measure the size of the World Wide Web, which it did until late 1995. The web's second search engine Aliweb appeared in November 1993. Aliweb did not use a web robot, but instead depended on being notified by website administrators of the existence at each site of an index file in a particular format.

JumpStation (released in December 1993) used a web robot to find web pages and to build its index, and used a web form as the interface to its query program. It was thus the first WWW resource-discovery tool to combine the three essential features of a web

search engine (crawling, indexing, and searching) as described below. Because of the limited resources available on the platform on which it ran, its indexing and hence searching were limited to the titles and headings found in the web pages the crawler encountered.

One of the first "full text" crawler-based search engines was WebCrawler, which came out in 1994. Unlike its predecessors, it let users search for any word in any webpage, which has become the standard for all major search engines since. It was also the first one to be widely known by the public. Also in 1994, Lycos (which started at Carnegie Mellon University) was launched and became a major commercial endeavor.

Soon after, many search engines appeared and vied for popularity. These included Magellan (search engine), Excite, Infoseek, Inktomi, Northern Light, and AltaVista. Yahoo! was among the most popular ways for people to find web pages of interest, but its search function operated on its web directory, rather than full-text copies of web pages. Information seekers could also browse the directory instead of doing a keyword-based search.

In 1996, Netscape was looking to give a single search engine an exclusive deal to be the featured search engine on Netscape's web browser. There was so much interest that instead a deal was struck with Netscape by five of the major search engines, where for $5Million per year each search engine would be in a rotation on the Netscape search engine page. The five engines were Yahoo!, Magellan, Lycos, Infoseek, and Excite.

Search engines were also known as some of the brightest stars in the Internet investing frenzy that occurred in the late 1990s. Several companies entered the market spectacularly, receiving record gains during their. Some have taken down their public search engine, and are marketing enterprise-only editions, such as Northern Light. Many search engine companies were caught up in the, a speculation-driven market boom that peaked in 1999 and ended in 2001.

Around 2000, rose to prominence.[1] The company achieved better results for many searches with an innovation called. This ranks web pages based on the number and PageRank of other web sites and pages that link there, on the premise that good or desirable pages are linked to more than others. Google also maintained a minimalist

interface to its search engine. In contrast, many of its competitors embedded a search engine.

By 2000, Yahoo was providing search services based on Inktomi's search engine. Yahoo! acquired Inktomi in 2002, and (which owned and AltaVista) in 2003. Yahoo! switched to Google's search engine until 2004, when it launched its own search engine based on the combined technologies of its acquisitions.

Microsoft first launched MSN Search in the fall of 1998 using search results from Inktomi. In early 1999 the site began to display listings from blended with results from Inktomi except for a short time in 1999 when results from AltaVista were used instead. In 2004, began a transition to its own search technology, powered by its own (called).

Microsoft's rebranded search engine, , was launched on June 1, 2009. On July 29, 2009, Yahoo! and Microsoft finalized a deal in which would be powered by Microsoft Bing technology.

How web search engines work

High-level architecture of a standard Web crawler

A search engine operates, in the following order

Web crawling

Index

Web search query

Web search engines work by storing information about many web pages, which they retrieve from the html itself. These pages are retrieved by a (sometimes also known as a spider) — an automated Web browser which follows every link on the site. Exclusions can be made by the use of. The contents of each page are then analyzed to determine how it should be (for example, words are extracted from the titles, headings, or special fields called). Data about web pages are stored in an index database for use in later queries. A query can be a single word. The purpose of an index is to allow information to be found as quickly as possible. Some search engines, such as, store all or part of the source page (referred to as a) as well as information about the web pages, whereas others, such as, store every word of every page they find. This cached page always holds the actual search text since it is the one that was actually indexed, so it can be very useful when the content of the current page has been updated and the search

terms are no longer in it. This problem might be considered to be a mild form of, and Google's handling of it increases by satisfying that the search terms will be on the returned webpage. This satisfies the since the user normally expects the search terms to be on the returned pages. Increased search relevance makes these cached pages very useful, even beyond the fact that they may contain data that may no longer be available elsewhere.

When a user enters a into a search engine (typically by using), the engine examines its and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text. The index is built from the information stored with the data and the method by which the information is indexed. Unfortunately, there are currently no known public search engines that allow documents to be searched by date. Most search engines support the use of the AND, OR and NOT to further specify the. Boolean operators are for literal searches that allow the user to refine and extend the terms of the search. The engine looks for the words or phrases exactly as entered. Some search engines provide an advanced feature called which allows users to define the distance between keywords. There is also concept-based searching where the research involves using statistical analysis on pages containing the words or phrases you search for. As well, natural language queries allow the user to type a question in the same form one would ask it to a human. A site like this would be ask.com.

The usefulness of a search engine depends on the of the **result set** it gives back. While there may be millions of web pages that include a particular word or phrase, some pages may be more relevant, popular, or authoritative than others. Most search engines employ methods to the results to provide the "best" results first. How a search engine decides which pages are the best matches, and what order the results should be shown in, varies widely from one engine to another. The methods also change over time as Internet usage changes and new techniques evolve. There are two main types of search engine that have evolved: one is a system of predefined and hierarchically ordered keywords that humans have programmed extensively. The other is a system that generates an by analyzing texts it locates. This second form relies much more heavily on the computer itself to do the bulk of the work.

Most Web search engines are commercial ventures supported by revenue and, as a result, some employ the practice of allowing advertisers to higher in search results. Those search engines which do not accept money for their search engine results make money by alongside the regular search engine results. The search engines make money every time someone clicks on one of these ads.

Search engine bias

Although search engines are programmed to rank websites based on their popularity and relevancy, empirical studies indicate various political, economic, and social biases in the information they provide. These biases could be a direct result of economic and commercial processes (e.g., companies that advertise with a search engine can become also more popular in its results), and political processes (e.g., the removal of search results in order to comply with local laws), is one example of an attempt to manipulate search results for political, social or commercial reasons.

SEARCH ALGORITHM

In, a **search algorithm**, broadly speaking, is for finding an item with specified properties among of items. The items may be stored individually as in a; or may be elements of a defined by a mathematical formula or procedure, such as the of an with; or a combination of the two,

For virtual search spaces

Algorithms for searching virtual spaces are used in, where the goal is to find a set of value assignments to certain variables that will satisfy specific mathematical and. They are also used when the goal is to find a variable assignment that will a certain function of those variables. Algorithms for these problems include the basic (also called "naïve" or "uninformed" search), and a variety of that try to exploit partial knowledge about structure of the space.

An important subclass are the local search methods, that view the elements of the search space as the vertices of a graph, with edges defined by a set of heuristics applicable to the case; and scan the space by moving from item to item along the edges, for example according to the steepest descent or best-first criterion, or in a stochastic search. This category includes a great variety of general metaheuristic methods, such as

simulated annealing, tabu search, A-teams, and genetic programming that combine arbitrary heuristics in specific ways.

This class also includes various tree search algorithms that view the elements as vertices of a tree, and traverse that tree in some special order. Examples of the latter include the exhaustive methods such as depth-first search and breadth-first search, as well as various heuristic-based search tree pruning methods such as backtracking and branch and bound. Unlike general metaheuristics, which at best work only in a probabilistic sense, many of these tree-search methods are guaranteed to find the exact or optimal solution, if given enough time.

Another important sub-class consists of algorithms for exploring the game tree of multiple-player games, such as chess or backgammon, whose nodes consist of all possible game situations that could result from the current situation. The goal in these problems is to find the move that provides the best chance of a win, taking into account all possible moves of the opponent(s). Similar problems occur when humans or machines have to make successive decisions whose outcomes are not entirely under one's control, such as in robot guidance or in marketing, financial or military strategy planning. This kind of problems has been extensively studied in the context of artificial intelligence. Examples of algorithms for this class are the minimax algorithm, alpha-beta pruning, and the A* algorithm.

For sub-structures of a given structure

The name combinatorial search is generally used for algorithms that look for a specific sub-structure of a given discrete structure, such as a graph, a string, a finite group, and so on. The term combinatorial optimization is typically used when the goal is to find a sub-structure with a maximum (or minimum) value of some parameter. (Since the sub-structure is usually represented in the computer by a set of integer variables with constraints, these problems can be viewed as special cases of constraint satisfaction or discrete optimization; but they are usually formulated and solved in a more abstract setting where the internal representation is not explicitly mentioned.)

An important and extensively studied subclass are the graph algorithms, in particular graph traversal algorithms, for finding specific sub-structures in a given graph

— such as subgraphs, paths, circuits, and so on. Examples include Dijkstra's algorithm, Kruskal's algorithm, the nearest neighbour algorithm, and Prim's algorithm.

Another important subclass of this category are the string searching algorithms, that search for patterns within strings. Two famous examples are the Boyer–Moore and Knuth–Morris–Pratt algorithms, and several algorithms based on the suffix tree data structure.

For quantum computers

There are also search methods designed for (currently non-existent) quantum computers, like Grover's algorithm, that are theoretically faster than linear or brute-force search even without the help of data structures or heuristics.

PERL PROGRAMMING

Description

This document is intended to give you a quick overview of the Perl programming language, along with pointers to further documentation. It is intended as a "bootstrap" guide for those who are new to the language, and provides just enough information for you to be able to read other peoples' Perl and understand roughly what it's doing, or write your own simple scripts.

This introductory document does not aim to be complete. It does not even aim to be entirely accurate. In some cases perfection has been sacrificed in the goal of getting the general idea across. You are *strongly* advised to follow this introduction with more information from the full Perl manual, the table of contents to which can be found in perltoc.

Throughout this document you'll see references to other parts of the Perl documentation. You can read that documentation using the perldoc command or whatever method you're using to read this document.

What is Perl?

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). Its major features are that it's easy to use, supports both procedural and object-oriented (OO) programming, has powerful built-in support for text processing, and has one of the world's most impressive collections of third-party modules.

Different definitions of Perl are given in perl, perlfaq1 and no doubt other places. From this we can determine that Perl is different things to different people, but that lots of people think it's at least worth writing about.

Running Perl programs,,,,,,,,,

To run a Perl program from the UNIX command line:

1. perl progname.pl

Alternatively, put this as the first line of your script:

1. #!/usr/bin/env perl

... and run the script as /path/to/script.pl. Of course, it'll need to be executable first, so chmod 755 script.pl (under Unix).

(This start line assumes you have the **env** program. You can also put directly the path to your perl executable, like in #!/usr/bin/perl).

For more information, including instructions for other platforms such as Windows and Mac OS, read perlrun.

Safety net

Perl by default is very forgiving. In order to make it more robust it is recommended to start every program with the following lines:

1. #!/usr/bin/perl
2. use strict;
3. use warnings;

The two additional lines request from perl to catch various common problems in your code. They check different things so you need both. A potential problem caught by use strict; will cause your code to stop immediately when it is encountered, while use warnings; will merely give a warning (like the command-line switch **-w**) and let your code run. To read more about them check their respective manual pages at strict and warnings.

Basic syntax overview

A Perl script or program consists of one or more statements. These statements are simply written in the script in a straightforward fashion. There is no need to have a `main()` function or anything of that kind.

Perl statements end in a semi-colon:

```
print "Hello, world";
```

Comments start with a hash symbol and run to the end of the line

```
# This is a comment
```

Whitespace is irrelevant:

```
print
    "Hello, world"
    ;
```

... except inside quoted strings:

```
# this would print with a linebreak in the middle
print "Hello
    world";
```

Double quotes or single quotes may be used around literal strings:

```
print "Hello, world";
print 'Hello, world';
```

However, only double quotes "interpolate" variables and special characters such as newlines (\n):

```
print "Hello, $name\n";     # works fine
print 'Hello, $name\n';     # prints $name\n literally
```

Numbers don't need quotes around them:

```
print 42;
```

You can use parentheses for functions' arguments or omit them according to your personal taste. They are only required occasionally to clarify issues of precedence.

```
print("Hello, world\n");
print "Hello, world\n";
```

More detailed information about Perl syntax can be found in perlsyn.

Perl variable types

Perl has three main variable types: scalars, arrays, and hashes.

- **Scalars**

A scalar represents a single value:

```
1.  my $animal = "camel";
2.  my $answer = 42;
```

Scalar values can be strings, integers or floating point numbers, and Perl will automatically convert between them as required. There is no need to pre-declare your variable types, but you have to declare them using the keyword the first time you use them. (This is one of the requirements of use strict ;.)

Scalar values can be used in various ways:

```
3.  print $animal;
4.  print "The animal is $animal\n";
5.  print "The square of $answer is ", $answer * $answer, "\n";
```

There are a number of "magic" scalars with names that look like punctuation or line noise. These special variables are used for all kinds of purposes, and are documented in perlvar. The only one you need to know about for now is $_ which is the "default variable". It's used as the default argument to a number of functions in Perl, and its set implicitly by certain looping constructs.

```
6.  print;       # prints contents of $_ by default
```

- **Arrays**

An array represents a list of values:

```
1.  my @animals = ("camel", "llama", "owl");
2.  my @numbers = (23, 42, 69);
3.  my @mixed   = ("camel", 42, 1.23);
```

Arrays are zero-indexed. Here's how you get at elements in an array:

```
4.  print $animals[0];          # prints "camel"
5.  print $animals[1];          # prints "llama"
```

The special variable $#array tells you the index of the last element of an array:

```
6.  print $mixed[$#mixed];      # last element, prints 1.23
```

You might be tempted to use $#array + 1 to tell you how many items there are in an array. Don't bother. As it happens, using @array where Perl expects to find a scalar value ("in scalar context") will give you the number of elements in the array:

```
7.  if (@animals < 5) { ... }
```

The elements we're getting from the array start with a $ because we're getting just a single value out of the array; you ask for a scalar, you get a scalar.

To get multiple values from an array:

```
8.  @animals[0,1];              # gives ("camel", "llama");
9.  @animals[0..2];             # gives ("camel", "llama", "owl");
10. @animals[1..$#animals];     # gives all except the first element
```

This is called an "array slice".

You can do various useful things to lists:

```
11. my @sorted    = sort @animals;
12. my @backwards = reverse @numbers;
```

There are a couple of special arrays too, such as @ARGV (the command line arguments to your script) and @_ (the arguments passed to a subroutine). These are documented in perlvar.

- **Hashes**

A hash represents a set of key/value pairs:

```
1.  my %fruit_color = ("apple", "red", "banana", "yellow");
```

You can use whitespace and the => operator to lay them out more nicely:

```
2.  my %fruit_color = (
3.      apple  => "red",
4.      banana => "yellow",
5.  );
```

To get at hash elements:

```
6.  $fruit_color{"apple"};      # gives "red"
```

You can get at lists of keys and values with keys () and values().

```
7.  my @fruits = keys %fruit_colors;
8.  my @colors = values %fruit_colors;
```

Hashes have no particular internal order, though you can sort the keys and loop through them.

Just like special scalars and arrays, there are also special hashes. The most well known of these is %ENV which contains environment variables. Read all about it (and other special variables) in perlvar.

Scalars, arrays and hashes are documented more fully in perldata.

More complex data types can be constructed using references, which allow you to build lists and hashes within lists and hashes.

A reference is a scalar value and can refer to any other Perl data type. So by storing a reference as the value of an array or hash element, you can easily create lists and hashes within lists and hashes. The following example shows a 2 level hash of hash structure using anonymous hash references.

```
1.  my $variables = {
2.      scalar  =>  {
3.                  description => "single item",
4.                  sigil => '$',
5.                  },
6.      array   =>  {
7.                  description => "ordered list of items",
8.                  sigil => '@',
9.                  },
10.     hash    =>  {
11.                 description => "key/value pairs",
12.                 sigil => '%',
13.                 },
14. };
15.
16. print "Scalars begin with a $variables->{'scalar'}->{'sigil'}\n";
```

Exhaustive information on the topic of references can be found in perlreftut, perllol, perlref and perldsc.

Variable scoping

Throughout the previous section all the examples have used the syntax:

```
1.  my $var = "value";
```

This is actually not required; you could just use:

```
1.  $var = "value";
```

However, the above usage will create global variables throughout your program, which is bad programming practice. my creates lexically scoped variables instead. The variables are scoped to the block (i.e. a bunch of statements surrounded by curly-braces) in which they are defined.

```
1.  my $x = "foo";
2.  my $some_condition = 1;
3.  if ($some_condition) {
4.      my $y = "bar";
5.      print $x;       # prints "foo"
6.      print $y;       # prints "bar"
7.  }
8.  print $x;           # prints "foo"
9.  print $y;           # prints nothing; $y has fallen out of scope
```

Using my in combination with a use strict; at the top of your Perl scripts means that the interpreter will pick up certain common programming errors. For instance, in the example above, the final print $y would cause a compile-time error and prevent you from running the program. Using strict is highly recommended.

Conditional and looping constructs

Perl has most of the usual conditional and looping constructs. As of Perl 5.10, it even has a case/switch statement (spelled given /when). See Switch statements in perlsyn for more details.

The conditions can be any Perl expression. See the list of operators in the next section for information on comparison and boolean logic operators, which are commonly used in conditional statements.

- **if**

```
1.  if ( condition ) {
```

```
    ...
} elsif ( other condition ) {
    ...
} else {
    ...
}
```

There's also a negated version of it:

```
unless ( condition ) {
    ...
}
```

This is provided as a more readable version of if (!*condition*).

Note that the braces are required in Perl, even if you've only got one line in the block. However, there is a clever way of making your one-line conditional blocks more English like:

```
# the traditional way
if ($zippy) {
    print "Yow!";
}

# the Perlish post-condition way
print "Yow!" if $zippy;
print "We have no bananas" unless $bananas;
```

- **while**

```
while ( condition ) {
    ...
}
```

There's also a negated version, for the same reason we have unless :

```
until ( condition ) {
    ...
}
```

You can also use while in a post-condition:

```
7.  print "LA LA LA\n" while 1;      # loops forever
```

- **for**

  Exactly like C:

```
1.  for ($i = 0; $i <= $max; $i++) {
2.      ...
3.  }
```

  The C style for loop is rarely needed in Perl since Perl provides the more friendly list scanning foreach loop.

- **foreach**

```
1.  foreach (@array) {
2.      print "This element is $_\n";
3.  }
4.
5.  print $list[$_] foreach 0 .. $max;
6.
7.  # you don't have to use the default $_ either...
8.  foreach my $key (keys %hash) {
9.      print "The value of $key is $hash{$key}\n";
10. }
```

For more detail on looping constructs (and some that weren't mentioned in this overview) see perlsyn.

Builtin operators and functions

Perl comes with a wide selection of builtin functions. Some of the ones we've already seen include print, sort and reverse. A list of them is given at the start of perlfunc and you can easily read about any given function by using perldoc -f *functionname*.

Perl operators are documented in full in perlop, but here are a few of the most common ones:

- **Arithmetic**

```
1.  +   addition
2.  -   subtraction
3.  *   multiplication
```

4. / division

- **Numeric comparison**
  1. == equality
  2. != inequality
  3. < less than
  4. > greater than
  5. <= less than or equal
  6. >= greater than or equal
- **String comparison**
  1. eq equality
  2. ne inequality
  3. lt less than
  4. gt greater than
  5. le less than or equal
  6. ge greater than or equal

  (Why do we have separate numeric and string comparisons? Because we don't have special variable types, and Perl needs to know whether to sort numerically (where 99 is less than 100) or alphabetically (where 100 comes before 99).
- **Boolean logic**
  1. && and
  2. || or
  3. ! not

  (and, or and not aren't just in the above table as descriptions of the operators. They're also supported as operators in their own right. They're more readable than the C-style operators, but have different precedence to && and friends. Check perlop for more detail.)
- **Miscellaneous**
  1. = assignment
  2. . string concatenation
  3. x string multiplication
  4. .. range operator (creates a list of numbers)

Many operators can be combined with a = as follows:

```
1.  $a += 1;       # same as $a = $a + 1
2.  $a -= 1;       # same as $a = $a - 1
3.  $a .= "\n";    # same as $a = $a . "\n";
```

Files and I/O

You can open a file for input or output using the open() function. It's documented in extravagant detail in perlfunc and perlopentut, but in short:

```
1.  open(my $in,  "<",  "input.txt")  or die "Can't open input.txt: $!";
2.  open(my $out, ">",  "output.txt") or die "Can't open output.txt: $!";
3.  open(my $log, ">>", "my.log")     or die "Can't open my.log: $!";
```

You can read from an open filehandle using the <> operator. In scalar context it reads a single line from the filehandle, and in list context it reads the whole file in, assigning each line to an element of the list:

```
1.  my $line  = <$in>;
2.  my @lines = <$in>;
```

Reading in the whole file at one time is called slurping. It can be useful but it may be a memory hog. Most text file processing can be done a line at a time with Perl's looping constructs.

The <> operator is most often seen in a while loop:

```
1.  while (<$in>) {     # assigns each line in turn to $_
2.      print "Just read in this line: $_";
3.  }
```

We've already seen how to print to standard output using print(). However, print() can also take an optional first argument specifying which filehandle to print to:

```
1.  print STDERR "This is your final warning.\n";
2.  print $out $record;
3.  print $log $logmessage;
```

When you're done with your filehandles, you should close() them (though to be honest, Perl will clean up after you if you forget):

```
1.  close $in or die "$in: $!";
```

Regular expressions

Perl's regular expression support is both broad and deep, and is the subject of lengthy documentation in perlrequick, perlretut, and elsewhere. However, in short:

- **Simple matching**

  1. if (/foo/) { ... } # true if $_ contains "foo"
  2. if ($a =~ /foo/) { ... } # true if $a contains "foo"

  The // matching operator is documented in perlop. It operates on $_ by default, or can be bound to another variable using the =~ binding operator (also documented in perlop).

- **Simple substitution**

  1. s/foo/bar/; # replaces foo with bar in $_
  2. $a =~ s/foo/bar/; # replaces foo with bar in $a
  3. $a =~ s/foo/bar/g; # replaces ALL INSTANCES of foo with bar in $a

  The s/// substitution operator is documented in perlop.

- **More complex regular expressions**

  You don't just have to match on fixed strings. In fact, you can match on just about anything you could dream of by using more complex regular expressions. These are documented at great length in perlre, but for the meantime, here's a quick cheat sheet:

  1. . a single character
  2. \s a whitespace character (space, tab, newline, ...)
  3. \S non-whitespace character
  4. \d a digit (0-9)
  5. \D a non-digit
  6. \w a word character (a-z, A-Z, 0-9, _)
  7. \W a non-word character
  8. [aeiou] matches a single character in the given set
  9. [^aeiou] matches a single character outside the given set
  10. (foo|bar|baz) matches any of the alternatives specified
  11.

12. ^ start of string
13. $ end of string

Quantifiers can be used to specify how many of the previous thing you want to match on, where "thing" means either a literal character, one of the metacharacters listed above, or a group of characters or metacharacters in parentheses.

14. * zero or more of the previous thing
15. + one or more of the previous thing
16. ? zero or one of the previous thing
17. {3} matches exactly 3 of the previous thing
18. {3,6} matches between 3 and 6 of the previous thing
19. {3,} matches 3 or more of the previous thing

Some brief examples:

20. /^\d+/ string starts with one or more digits
21. /^$/ nothing in the string (start and end are adjacent)
22. /(\d\s){3}/ a three digits, each followed by a whitespace
23. character (eg "3 4 5 ")
24. /(a.)+/ matches a string in which every odd-numbered letter
25. is a (eg "abacadaf")
26.
27. # This loop reads from STDIN, and prints non-blank lines:
28. while (<>) {
29. next if /^$/;
30. print;
31. }

- **Parentheses for capturing**

As well as grouping, parentheses serve a second purpose. They can be used to capture the results of parts of the regexp match for later use. The results end up in $1 , $2 and so on.

1. # a cheap and nasty way to break an email address up into parts
2.

```
3.      if ($email =~ /([^@]+)@(.+)/) {
4.          print "Username is $1\n";
5.          print "Hostname is $2\n";
6.      }
```

- **Other regexp features**

  Perl regexps also support backreferences, lookaheads, and all kinds of other complex details. Read all about them in perlrequick, perlretut, and perlre.

Writing subroutines

Writing subroutines is easy:

```
1.  sub logger {
2.      my $logmessage = shift;
3.      open my $logfile, ">>", "my.log" or die "Could not open my.log: $!";
4.      print $logfile $logmessage;
5.  }
```

Now we can use the subroutine just as any other built-in function:

```
1.  logger("We have a logger subroutine!");
```

What's that shift? Well, the arguments to a subroutine are available to us as a special array called @_ (see perlvar for more on that). The default argument to the shift function just happens to be @_ . So my $logmessage = shift; shifts the first item off the list of arguments and assigns it to $logmessage .

We can manipulate @_ in other ways too:

```
1.  my ($logmessage, $priority) = @_;       # common
2.  my $logmessage = $_[0];                 # uncommon, and ugly
```

Subroutines can also return values:

```
1.  sub square {
2.      my $num = shift;
3.      my $result = $num * $num;
4.      return $result;
5.  }
```

Then use it like:

```
1.  $sq = square(8);
```

OO Perl

OO Perl is relatively simple and is implemented using references which know what sort of object they are based on Perl's concept of packages. However, OO Perl is largely beyond the scope of this document. Read perlboot, perltoot, perltooc and perlobj.

As a beginning Perl programmer, your most common use of OO Perl will be in using third-party modules, which are documented below.

Using Perl modules

Perl modules provide a range of features to help you avoid reinventing the wheel, and can be downloaded from CPAN (http://www.cpan.org/). A number of popular modules are included with the Perl distribution itself.

Categories of modules range from text manipulation to network protocols to database integration to graphics. A categorized list of modules is also available from CPAN.

To learn how to install modules you download from CPAN, read perlmodinstall.

To learn how to use a particular module, use perldoc *Module::Name*. Typically you will want to use *Module::Name*, which will then give you access to exported functions or an OO interface to the module.

perlfaq contains questions and answers related to many common tasks, and often provides suggestions for good CPAN modules to use.

perlmod describes Perl modules in general. perlmodlib lists the modules which came with your Perl installation.

UNIT II DATABASES

Data management

Data Management is a broad field of study, but essentially is the process of managing data as a resource that is valuable to an organization or business. One of the largest organizations that deal with data management, DAMA (Data Management Association), states that data management is the process of developing data architectures, practices and procedures dealing with data and then executing these aspects on a regular basis.

There are many topics within data management, some of the more popular topics include data modeling, data warehousing, data movement, database administration and data mining.

Data Modeling

Data modeling is first creating a structure for the data that you collect and use and then organizing this data in a way that is easily accessible and efficient to store and pull the data for reports and analysis. In order to create a structure for data, it must be named appropriately and show a relationship with other data. It also must fit appropriately in a class. For instance, if you have a database of media, you might have a hierarchal structure of objects that include photos, videos, and audio files. Within each category, you can classify objects accordingly.

Data Warehousing

Data warehousing is storing data effectively so that it can be accessed and used efficiently. Different organizations collect different types of data, but many organizations use their data the same way, in order to create reports and analyze their data to make quality business decisions. Data warehousing is usually an organizational wide repository of data, however for very large corporations in can encompass just one office or one department.

Data Movement

Data movement is the ability to move data from one place to another. For instance, data needs to be moved from where it is collected to a database and then to an end user, but this process takes quite a bit of logistic insight. Not only do all hardware, applications and data collected need to be compatible with one another, they must also be

able to be classified, stored and accessed with ease within an organization. Moving data can be very expensive and can require lots of resources to make sure that data is moved efficiently, that data is secure in transit and that once it reaches the end user it can be used effectively either to be printed out as a report, saved on a computer or sent as an email attachment.

Database Administration

Database administration is extremely important in managing data. Every organization or enterprise needs database administrators that are responsible for the database environment. Database administrators are usually given the authority to do the following tasks that include recoverability, integrity, security, availability, performance and development & testing support.

Recoverability is usually defined as a way to store data as a back up and then test the back ups to make sure that they are valid. The task of integrity means that data that is pulled for certain records or files are in fact valid and have high data integrity. Data integrity is extremely important especially when creating reports or when data is used for analysis. If you have data that is deemed invalid, your results will be worthless.

Database security is an essential task for database administrators. For instance, database administrators are usually in charge of giving clearance and access to certain databases or trees in an organization. Another important task is availability. Availability is defined as making sure a database is up and running. The more up time, usually the higher level of productivity. Performance is related to availability, it is considered getting the most out of the hardware, applications and data as possible. Performance is usually in relation to an organizations budget, physical equipment and resources.

Finally, a database administrator is usually involved in database development and testing support. Database administrators are always trying to push te envelope, trying to get more use out of the data and add better performing and more powerful applications, hardware and resources to the database structure. A database that is administered correctly is not only a sign of competent database administrator, but it also means that all end users have a huge resource in the data that is available. This makes it easy to create reports, conduct analysis and make high quality decisions based on data that is collected and used within the organization.

Data Mining

Another important topic regarding data management is data mining. Data mining is a process in which large amounts of data are sifted through to show trends, relationships, and patterns. Data mining is a crucial component to data management because it exposes interesting information about the data being collected. It is important to note that data is primarily collected so it can be used to find these patterns, relationships and trends that can help a business grow or create profit.

While there are many topics within data management, they all work together from the beginning where data is collected to the end of the process where it is sifted through; analyzed and formatted where specialists can then make quality decisions based upon it.

DATA LIFE CYCLE

Electronic information management is now a primary business and legal concern. Sarbanes Oxley, information security, expanded electronic discovery demands, and new penalties for spoliation of evidence have made "document retention" an issue of urgency for general counsel.

The term "document retention," however, is a profound misnomer. "Document" refers to physical media, like paper. And "retention" implies there is either an "original," existing as unique artifact, or at least physical copies which can be stored in a known place.

But business no longer deals in documents. It deals in "data." And data is given life by ever-flowing electronic records - shared by users on multi-layered networks. Such records are accessed, edited, and transmitted with applications that defy understanding. Electronic data is then saved, not in boxes or files, but in a myriad of "media" that challenge accountability. Data resides in our servers, laptop computers, handheld devices, old back-up tapes, CD-ROM disks, thumb drives, and, now, even cellular phones and pagers. As a result of such dynamics, a new *complexity* has emerged.

The sudden evolution of such an "information ecosystem" poses fundamental questions: Can we still control information, so as to comply with law and business strategies? How do we know who is changing which records? What is authentic? How is a record "private?"

Therefore, how do we impose "internal control" over our share of the information ecosystem? Clearly, a new approach to information must evolve. Business should remember that:

- Electronic information exhibits complex behavior that was absent in the age of documents. When there is an audit, or a dispute, the evidence comprises snapshots of our electronic ecosystem.
- Data management is not just the domain of the IS department. It is the combined concern of the CEO, General Counsel, CFO, outside counsel and auditors with expertise in the field. Strategy requires teamwork.
- There are severe penalties for destruction of information, even if inadvertent.
- With appropriate data management, electronic discovery in litigation is facilitated, and reduced in cost.
- With appropriate data management, proof of the authenticity of one's own records is possible, whereas before, it was problematic.
- There are rapidly emerging privacy rules exemplified by HIPAA, California's SB-1386, and directives from Europe that deal not with *retention* of data, but with *access* and *destruction*. Such rules co-exist with retention, thus necessitating a "life cycle" for data.

Missteps in handling information can cause business failures. These are now both more likely and more severe than before-- given the new, more dynamic behavior of information. Such issues are legal concerns, because the obligations to control information constitute legal obligations, and the risks of lack of control are legal risks. Finally, because the Enterprise exists only in so far as its ability to control information, there is no choice but to adapt.

The Problem

We are awash in electronic information. It is smeared across our technology systems. Managing this morass of information is one of the most serious problems facing business today. Companies, large and small, often don't appreciate the ramifications until it is too late and they are already at risk of serious legal liability. At that point, trying to fix the problem can be dearly expensive, and is often unsuccessful.

During the seemingly ancient days before computers and networks became the predominant business paradigm, most data was kept in the form of laboriously typed paper documents that were neatly filed and, at the end of their life cycle, either thrown in the trash or filed away in document storage facilities. We have personally searched for old documents in places such as an abandoned Colorado mine shaft (hazardous waste suits required), a forgotten storage building in the middle of the Puerto Rican jungle (mosquito netting required), and the garages of long retired engineers (great patience required).

Because paper document preparation and storage were both burdensome and expensive, there was a natural limitation on the volume of documents produced and subsequently preserved. But computers and electronic data storage have changed all of that. Electronic document generation and storage is simple and cheap. The digital equivalent of a warehouse full of paper documents can now be stored in a small server no bigger then the average computer. An encyclopedia can exist on a thin piece of plastic. As a result, rather than cull old files before storage, it is much quicker, easier, and cheaper just to save everything.

The Need for "Data Life Cycle Management"

So, this is a good thing, right? Not really. Data Life Cycle Management requires thoughtful procedures governing what data to keep; what data to discard (we never say "destroy"); and critically, how to control what's left.

First, there is a growing collection of federal and state laws, as well as international rules, that require companies to preserve specific data for prescribed periods of time. For example, Sarbanes Oxley, HIPAA, and the Internal Revenue Code establish data retention and reporting requirements. The European Union Directives establish stringent policies governing the collection, storage, and use of personal information that will apply to any foray into international commerce. The FTC is following suit. Failure to abide by these regulations can subject companies to fines, penalties, and the forfeiture of the privilege of doing business in the respective jurisdictions. And the federal courts are now imposing strict penalties for spoliation of evidence.

So the answer is simple: just save everything. Wrong. If a business saves all of its electronic data, whether required to or not, litigation opening that data up to review can

result in the company literally drowning in its digital waste. Electronic discovery can be debilitatingly expensive and may turn up difficult-to-explain documents or emails that the company had no obligation to save prior to the litigation. And once litigation is foreseeable, disposing of potentially relevant data can result in court ordered sanctions or even a default judgment.

We have seen many examples where the failure to properly save or properly dispose of electronic data either saved the ship, or sank it. Arthur Anderson and Enron went down, at least in part, because they illegally attempted to destroy documents after litigation was on the horizon. Likewise, we can't forget the stock broker e-mails in which brokers trashed the very stocks they were trying to sell to the public.

The Beginnings of A New Policy

With thought and planning, businesses can create electronic data life cycle policies that will eliminate potential liability associated with either saving too much or too little of the company's data; or more important, of losing control of what is there. An electronic data life cycle policy can be built around several basic principles.

1. **Identify The Objectives of the Enterprise.**

First, it is critical that each company self-consciously devise "Objectives" in maintaining its electronic records. This function has become more important now than ever, as it is easier to treat all records the same--just save everything, and let everything go unprotected on the "network." But depending on the business, different types of records have vastly different degrees of importance. Universities treat their academic records as sacrosanct, whereas payments for lawn care might be much lower on the scale. Companies hired to keep track of individuals crossing national borders might have strict record keeping priorities and objectives for certain databases, but not others, such as their own payroll. Health and insurance records are treated one way, and payment for staples and copy services another. Information categories are not the same. Importance and function vary by orders of magnitude in any enterprise.

Accordingly, the first job is to prioritize. What is critical for the business? Devise information life cycle management with such critical records in mind.

2. **Minimalism.**

Next, data should be discarded unless there is a good business and/or legal reason to retain it. Implementation of this principle requires that a company, once again, take a hard look both at the types of data it collects and the regulatory constraints relating to that data. Data should also be preserved if it is potentially relevant to any ongoing or foreseeable litigation, now known as the *Zubulake* standard. The overall goal is to comply with law and to achieve business objectives, but not to save data that is not required by law or for business purposes. And given the fact the digital files can be copied *ad infinitum* onto different media, unless one controls access to data during the time it is stored by a business, one loses control over the ability to discard information. Maintaining such control is no easy task. Achieving it puts one on the cutting edge of business process.

3**. Training and Simplicity of Procedure.**

Of course, for everyone, a data life cycle policy must be simple and easy to implement. As with all things corporate, there is a strong tendency for policy initiatives to become increasingly intricate to the point of dysfunction (only interpretable by those with graduate degrees in operations research). Once the policy becomes too complex, it is virtually guaranteed that employees will simply ignore it.

For example, during the beginning of the Internet boom, the National Security Agency created complex internal rules for the transfer of sensitive data from one NSA employee to another. Rather than comply with the rules governing NSA's secured systems, employees discovered it was easier to simply send data to each other, around the NSA systems and through Internet as email, thus defeating the policy. Therefore, any policy should strive for simplicity by establishing a limited number of broad subject matter categories and functions. While simplicity might result in some over-inclusion of data retained, it nevertheless increases the chances employees will actually comply with the policy.

4. **Adequacy of Infrastructure.**

There must be adequate hardware and software to accomplish the task, once objectives and policy are identified. It is, indeed, often astounding how "out-of-scale" a company's infrastructure is to accomplish appropriate data management. Here, as

elsewhere, teamwork between higher management and IS workers is critical. Hardware is seldom the problem. The problem is the software and human systems infrastructure relating to information security; access control; authenticity; retrievability and auditability.

5. **Information Security.**

Perhaps no practice can enhance Data Life Cycle Management better than appropriate "information security" procedures. A primer on information security is beyond the scope of this article. But how else can one ensure that shared records are not improperly accessed or edited? How else a company keep its valuable information from being stolen, for example, and sold to spammers? Given that such a theft incident may now trigger notice obligations, and perhaps liability, information security reigns supreme. It is fundamental to protecting the assets of the enterprise. It is fundamental to Data Life Cycle Management.

6. **Authenticity of One's Own Records.**

Give thought to how one might prove the authenticity of one's own records if they are ever challenged in court, an administrative proceeding, or an audit. This implicates the need for proactive procedures. Authenticity, which has been stretched to the breaking point by the new information paradigm, should no longer be taken for granted.

7. **Retrievability.**

One of the major problems with electronic record keeping is that when a request for information does come--for example in discovery in litigation--it can be a six to seven figure chore just finding the data that formerly could easily be retrieved from a set of file cabinets. Hence one of the hallmarks of the metamorphosis from a document/ record/ file keeping culture to a culture of data multiplying on a shared networked, edited by many and stored on scalable media.

Accordingly, far more advance planning is now required to ensure future retrievability of data. Law firms, strange to say, are in the vanguard of businesses in this respect. Their handling of huge numbers of different types of electronic files for many different customers has led to databases that facilitate filing by subject matter, with automatic indexing, and easy retrievability. This "subject matter centered" database control of data has yet to make it into the mainstream of businesses' data storage.

Businesses, therefore, must attack a mounting data retrievability issue. Just complying with one discovery request, when one has data stored hodgepodge on various media in various types of systems, could pay for an entirely new infrastructure. Don't be penny wise and pound foolish.

8. **Distribution Controls.**

The interactive ease of networks, including that network of networks, the Internet, mean that once access to specific data is acquired, the data can be transmitted to countless destinations around the globe in a matter of seconds. Once the digital genie is out of the electronic bottle, no amount of wishing can contain it. Every day there are new examples of this phenomenon. It can involve the public release of valuable intellectual property, such as the case of the Swedish man who released highly confidential DVD source code on to the internet. Or it can involve the unconsented dissemination of personal and private matters, such as the apparently unauthorized released of Paris Hilton's "homemade movie." (Remember, there is no such thing as bad publicity).

At least when it comes to business data, unauthorized access can largely be eliminated by employing network security controls. A different problem, however, arises regarding the distribution of data by persons who have legitimate access to the data. Whether the situation involves complex project files shared by a team of engineers or a simple email communication, uncontrolled electronic replication can be a disaster. One solution is to employ one of the available software solutions that encrypt the data and allow the sender to specify the degree of republication rights granted to the recipient. Sophisticated companies are beginning to utilize these types of solutions as part of their overall data management strategy.

9. **Auditability.**

The idea behind the PCAOB's new Auditing Standard No. 2 is "internal control" over information. Public companies, obviously, will need to pass an audit of their financial statements based on Auditing Standard No. 2. Their management of information will need to pass the various types of tests auditors will devise*, in the future*, to gage financial data as represented in electronic records, from the transaction level on up to the Income Statement and Balance Sheet. Unless the Data Life Cycle Management system can pass an audit, a company is put in an unfortunate situation indeed. Accordingly, a

sound Data Life Cycle Management Plan is at the same time a Sarbanes Oxley compliance program. Along these lines, all companies should seriously consider the use of "hashing," "digital signatures," and logging of network events to provide a framework of "testability" for the information flowing in their ecosystem at any point in time.

10. **Consistency.**

Data retention practices must be consistent. Inconsistent document retention actions will create a taint of intentional spoliation and wrongdoing. It's hard to explain why you discarded data pursuant to your 3-year-old policy for the first time just three days before being served with that antitrust complaint. Therefore, if you have a data retention policy, make sure it is implemented consistently. If there are dates or milestones for data review and disposal, they should be adhered to.

11. **Enforcement.**

And enforcement must be simple and consistent. The policy should use both automated systems to dispose of unnecessary data and procedures to motivate employees to appropriately deal with the rest of the data that cannot be picked up through automated systems. So, for example, unnecessary email accumulation can be limited either by strictly controlling the size of employee mailboxes, thus forcing employees to delete old emails in order to receive new ones, or by automated systems that automatically dispose of old emails after a set period of time (*i.e.,* 30 days).

For example, backup tapes can present an glitch in a data retention policy. Backup tapes are intended primarily for one purpose: the emergency restoration of a computer system following a crash. Unfortunately, many businesses make the mistake of saving multiple sets of backup tapes as data archives. This action, alone, can create a nightmare of enormous litigation discovery costs if it is ever necessary to search those tapes. Therefore, a company should limit itself to no more then two sets of backup tapes, which are consistently recycled at particular intervals to capture the existing computer system. This procedure will make it unlikely that the backup tapes will ever be successfully demanded as a source of old and otherwise disposed of data.

Conclusion

Information management is a dynamic concept that has, and will continue, to change in co-evolution with the gigantic "morph" of information from artifact to

ecosystem. Therefore, establishing data life cycle management policies is not a one time process. The advent of electronic data storage and digital communications has provided business, consumers, and the public with untold benefits, including access to vast amounts of information and incredible speed in analysis and distribution. Implementing and maintaining a data life cycle management system is a small, but necessary, price to pay for continuing to be a player in the marketplace.

DATABASE TECHNOLOGY

What is Database Technology?
Why is it Important?
Which Application Areas?
What are the Relevant Technologies?
The ERCIM Competence
The Way Forward

1. What is Database Technology?

The essential feature of database technology is that it provides an INTERNAL Representation (model) of the EXTERNAL world of interest. Examples are the representation of a particular date/time/flight/aircraft in airline reservation or of item code/item description/quantity on hand/reorder level/reorder quantity in a stock control system.

The TECHNOLOGY involved is concerned primarily with maintaining the internal representation consistent with external reality; this involves the results of extensive R&D over the past 30 years in areas such as user requirements analysis, data modelling, process modelling, data integrity, concurrency, transactions, file organisation, indexing, rollback and recovery, persistent programming, object-orientation, logic programming, deductive database systems, active database systems... and in all these (and other) areas there remains much to be done.

2. Why is it Important?

Business in much of world depends on database technology. For example:

Finance: the UK clearing banks have calculated that if their database systems were removed it would take every person in UK working 24 hours per day 7 days per week to process all the financial transactions manually. The London stock exchange relies on computer systems for recording buying and selling of stock which happens very quickly and in large quantities. The amount of money involved in these transactions is enormous.

Transport: The airlines all use online seat reservation systems and have systems for scheduling aircraft, for building and maintaining timetables, for handling the in-flight catering and for mechanical servicing of the planes. Similar systems exist for rail, sea and road transport. They all use database technology extensively.

Utilities: the major utilities (water, electricity, gas) all have generation / distribution systems based on database technology.

Resources: The mineral exploration / extraction companies, and governments who regulate them (especially for oil exploration / extraction) have extensive databases which have complex data structures (usually including GIS (geographical information system) components.

Production engineering: from scheduling workflow through the production lines of machines to stock control and order processing, database technology underpins all activity in this area.

Environment: protection and control of the environment by government agencies depends heavily on database systems with GIS facilities, together with databases of toxic substances and clean-up recommendations.

Tourism: hotel systems and local tourist attractions information and booking facilities rely on database systems, and the major package tour operators have extensive databases for holiday planning and booking, together with financial systems for payment and invoicing.

Leisure: the entertainment industry uses database systems extensively for theatre, concert and cinema ticket bookings.

Culture: museums, art galleries, history exhibitions -all utilise database technology (and especially multimedia database technology) for cataloguing their collections and recording access to them.

Education: courses, materials, and assessment all rely heavily on database technology in all sectors of education. Increasingly the linking of database technology with hypermedia delivery systems allows courseware to be maintained up-to-date and delivered to the consumer.

Healthcare: primary healthcare has long relied on database technology to schedule hospital beds or appointments at clinics. The patient health record has been the subject of intensive study (and R&D resources) over many years because of its complexity of structure, content and media and also because of the security and privacy issues.

Epidemiology utilises database technology to hold and organise key information from many patients in order to allow statistical processing to detect trends and to alert medical practitioners to possible epidemics. More recently, data mining techniques have been applied to this area - relying again on database technology.

Government administration would be paralysed without database technology; the collection of taxes and the payment of social security benefits depends totally on database technology.

Retail: the major retail stores utilise database technology in stock control and PoS (Point of Sale) systems. Modern retailers use advanced data mining techniques to determine trends in sales and consumer preference to optimise stock control, retail performance, customer convenience and profit.

The essential point is that database technology is a CORE TECHNOLOGY with links to:

information management / processing

data analysis / statistics

data visualisation / presentation

multimedia and hypermedia

office and document systems

business processes, workflow, CSCW (computer-supported cooperative work)

But modern DB systems depend on an infrastructure of:

networks both LAN (local area network) and WAN (wide area network)

client-server computing architecture

skilled data analysis and DB design

skilled systems development method(s)
for them to be effective and therefore used in any sector of activity.

3. Which Application Areas?

From the programme of the workshop there is clearly interest in:
culture&scientific information
tourism
telemedicine
natural resources management
production engineering
all of which have been mentioned above as typical DB application areas. There is not enough time in the presentation to describe in any detail how DB technology is used in these application areas.

4. What Relevant Technologies?

Relational DBMS is the modern base technology for many business applications. It offers flexibility and easy-to-use tools at the expense of ultimate performance. More recently relational systems have started to extend their facilities in the directions of information retrieval, object-orientation and deductive/active systems leading to the so-called 'Extended Relational Systems'.

Information Retrieval Systems started with handling library catalogues and extended to full free-text utilising inverted index technology with a lexicon or thesaurus. Modern systems utilise some KBS (knowledge-based systems) techniques to improve retrieval.

Object-Oriented DBMS started for engineering applications where objects are complex, have versions and need to be treated as a complete entity. OODBMSs share many of the OOPL features such as identity, inheritance, late binding, overloading and overriding. OODBMSs have found favour in engineering and office systems but have not yet been successful in traditional application areas.

Deductive / Active DBMS have emerged over the last 20 years and combine logic programming technology with database technology. This allows the database itself to react to external events an to maintain dynamically its integrity with respect to the real world.

5. The ERCIM Competence

Each Institute has a strong team in database technology, with its own national projects backed by participation in EU projects. Each team has links to academics (through teaching / research) and links to commerce / industry (through projects and consultancy). The groups in the ERCIM institutes formed EDRG, ERCIM Database Research Group, in 1991. There have been 9 workshops to date, some joint projects and we obtained an EU-funded network of excellence (HCM programme). EDRG can put together approximately 200 quality DB researchers plus contacts in academia and industry.

6. The Way Forward

It is hoped that this workshop will define the requirements for the way forward: perhaps the requirement is for workshops to exchange experience and to motivate the initiation of joint projects. There is clearly a need to get to know teams and competencies on each side of the Mediterranean. Hopefully joint projects can be initiated in order to demonstrate the effectiveness of DB technology for the region.

Biological databases and their uses

Biological databases are libraries of life sciences information, collected from scientific experiments, published literature, high-throughput experiment technology, and computational analyses. They contain information from research areas including genomics, proteomics, metabolomics, microarray gene expression, and phylogenetics. Information contained in biological databases includes gene function, structure, localization (both cellular and chromosomal), clinical effects of mutations as well as similarities of biological sequences and structures.

Relational database concepts of computer science and Information retrieval concepts of digital libraries are important for understanding biological databases. Biological database design, development, and long-term management is a core area of the discipline of bioinformatics. Data contents include gene sequences, textual descriptions, attributes and ontology classifications, citations, and tabular data. These are often described as semi-structured data, and can be represented as tables, key delimited records, and XML structures. Cross-references among databases are common, using database accession numbers

Overview

Biological databases are an important tool in assisting scientists to understand and explain a host of biological phenomena from the structure of biomolecules and their interaction, to the whole metabolism of organisms and to understanding the evolution of species. This knowledge helps facilitate the fight against diseases, assists in the development of medications and in discovering basic relationships amongst species in the history of life.

Biological knowledge is distributed amongst many different general and specialized databases. This sometimes makes it difficult to ensure the consistency of information. Biological databases cross-reference other databases with accession numbers as one way of linking their related knowledge together.

Output

Biological data comes in many formats. These formats include text, sequence data, protein structure and links. Each of these can be found from certain sources, for example:

- Text formats are provided by PubMed and OMIM.
- Sequence data are provide by GenBank, in terms of DNA, and UniProt, in terms of protein.
- Protein structures are provided by PDB, SCOP, and CATH.

Problems associated with protein databases

Since discovery in the area of protein structure has not evolved quite as quickly as discoveries in the area sequence data, due to the 3D nature of protein structure, less information is available for it. Nonetheless, data can be accessed through members of the wwPDB (PDBe, PDBj and RCSB PDB, SCOP-Structural Classification of Proteins- at, and CATH.

Species-specific databases

Species-specific databases are available for some species, mainly those that are often used in research. For example, Colibase is an *E. coli* database. Other popular species specific databases include, Flybase for *Drosophila*, and WormBase for the nematodes *Caenorhabditis elegans* and *Caenorhabditis briggsae*.

UNIT III PATTERN MATCHING & MACHINE LEANING

PAIRWISE SEQUENCE ALIGNMENT – LOCAL VS. GLOBAL ALIGNMENT

- **Optimal alignments**

  The alignment that is the best, given a defined set of rules and parameter values for comparing different alignments. There is no such thing as the single best alignment, since optimality always depends on the assumptions one bases the alignment on. For example, what penalty should gaps carry? All sequence alignment procedures make some such assumptions.

- **Global alignment**

  An alignment that assumes that the two proteins are basically similar over the entire length of one another. The alignment attempts to match them to each other from end to end, even though parts of the alignment are not very convincing. A tiny example:

  NLGPSTKDFGKISESREFDNQ
  | |||| |
  QLNQLERSFGKINMRLEDALV

- **Local alignment**

  An alignment that searches for segments of the two sequences that match well. There is no attempt to force entire sequences into an alignment, just those parts that appear to have good similarity, according to some criterion. Using the same sequences as above, one could get:

  NLGPSTKDDFGKILGPSTKDDQ
  ||||
  QNQLERSSNFGKINQLERSSNN

  It may seem that one should always use local alignments. However, it may be difficult to spot an overall similarity, as opposed to just a domain-to-domain similarity, if one uses only local alignment, so global alignment is useful in some cases. You can produce a global or a local alignment with the Emboss Pairwise global and local alignment tool.

EMBOSS-Align is the tool we will use to compare 2 sequences:

The EMBOSS-Align tool contains two programs each using a different algorithm:

- When you want an alignment that covers the whole length of both sequences, use the needle program.
- When you are trying to find the best region of similarity between two sequences, use the water program.

The following is a slightly more deatiled explanation of the two programs, needle and water, used in EMBOSS-Align:

- **Needle program** - This program uses the Needleman-Wunsch global alignment algorithm to find the optimum alignment (including gaps) of two sequences when considering their entire length.

  The Needleman-Wunsch algorithm is a member of the class of algorithms that can calculate the best score and alignment in the order of mn steps, (where 'n' and 'm' are the lengths of the two sequences). These dynamic programming algorithms were first developed for protein sequence comparison by Needleman and Wunsch, though similar methods were independently devised during the late 1960's and early 1970's for use in the fields of speech processing and computer science.

  What is the optimal alignment? Dynamic programming methods ensure the optimal global alignment by exploring all possible alignments and choosing the best. It does this by reading in a scoring matrix that contains values for every possible residue or nucleotide match. Needle finds an alignment with the maximum possible score where the score of an alignment is equal to the sum of the matches taken from the scoring matrix.

  An important problem is the treatment of gaps, i.e., spaces inserted to optimise the alignment score. A penalty is subtracted from the score for each gap opened (the 'gap open' penalty) and a penalty is subtracted from the score for the total number of gap spaces multiplied by a cost (the 'gap extension' penalty).

  Typically, the cost of extending a gap is set to be 5-10 times lower than the cost for opening a gap.

This is a true implementation of the Needleman-Wunsch algorithm and so produces a full path matrix. It therefore cannot be used with genome sized sequences unless you've a lot of memory and a lot of time.

Needle is for aligning two sequences over their entire length. This works best with closely related sequences. If you use needle to align very distantly-related sequences, it will produce a result but much of the alignment may have little or no biological significance.

A true Needleman Wunsch implementation like needle needs memory proportional to the product of the sequence lengths. For two sequences of length 10,000,000 and 1,000 it therefore needs memory proportional to 10,000,000,000 characters. Two arrays of this size are produced, one of integers and one of floats so multiply that figure by 8 to get the memory usage in bytes. That doesn't include other overheads. Therefore only use water and needle for accurate alignment of reasonably short sequences.

- **Water program** - Water uses the Smith-Waterman algorithm (modified for speed enhancements) to calculate the local alignment.

  The Smith-Waterman algorithm is a member of the class of algorithms that can calculate the best score and local alignment in the order of mn steps, (where 'n' and 'm' are the lengths of the two sequences). These dynamic programming algorithms were first developed for protein sequence comparison by Smith and Waterman, though similar methods were independently devised during the late 1960's and early 1970's for use in the fields of speech processing and computer science.

  A local alignment searches for regions of local similarity between two sequences and need not include the entire length of the sequences. Local alignment methods are very useful for scanning databases or other circumstances when you wish to find matches between small regions of sequences, for example between protein domains.

  Dynamic programming methods ensure the optimal local alignment by exploring all possible alignments and choosing the best. It does this by reading in a scoring matrix that contains values for every possible residue or nucleotide match. Water

finds an alignment with the maximum possible score where the score of an alignment is equal to the sum of the matches taken from the scoring matrix.

An important problem is the treatment of gaps, i.e., spaces inserted to optimise the alignment score. A penalty is subtracted from the score for each gap opened (the 'gap open' penalty) and a penalty is subtracted from the score for the total number of gap spaces multiplied by a cost (the 'gap extension' penalty).

Typically, the cost of extending a gap is set to be 5-10 times lower than the cost for opening a gap.

Water is a true implementation of the Smith-Waterman algorithm and so produces a full path matrix. It therefore cannot be used with genome sized sequences unless you have a lot of memory and a lot of time.

Local alignment methods only report the best matching areas between two sequences - there may be a large number of alternative local alignments that do not score as highly. If two proteins share more than one common region, for example one has a single copy of a particular domain while the other has two copies, it may be possible to "miss" the second and subsequent alignments. You will be able to see this situation if you have done a dotplot and your local alignment does not show all the features you expected to see.

Water is for aligning the best matching subsequences of two sequences. It does not necessarily align whole sequences against each other; you should use needle if you wish to align closely related sequences along their whole lengths.

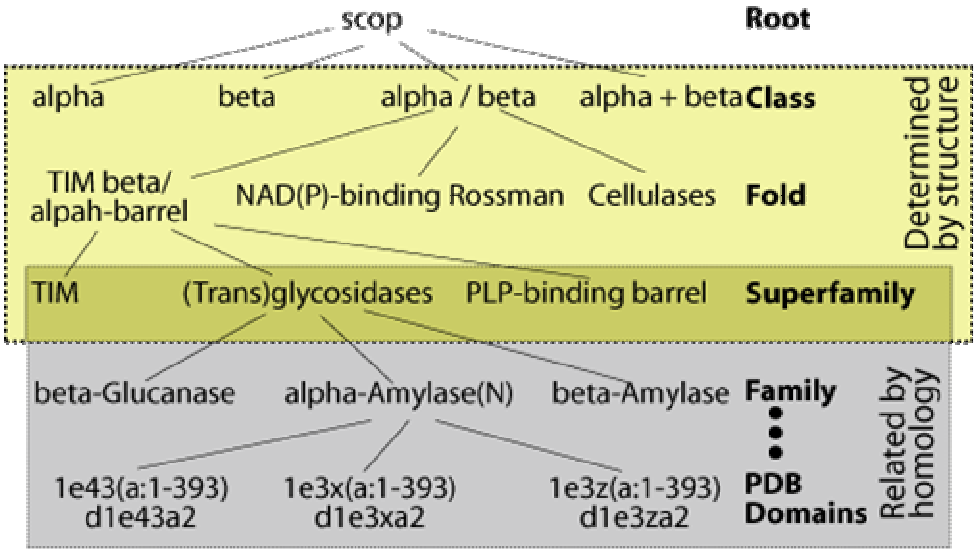
A true Smith Waterman implementation like water needs memory proportional to the product of the sequence lengths. For two sequences of length 10,000,000 and 1,000 it therefore needs memory proportional to 10,000,000,000 characters. Two arrays of this size are produced, one of integers and one of floating point (real) numbers so multiply that figure by 8 to get the memory usage in bytes. That doesn't include other overheads. Therefore only use water and needle for accurate alignment of reasonably short sequences.

Pairwise Sequence Comparison Evaluation

Introduction

Pairwise sequence comparison is the workhorse method of computational biology. There are several popular programs available for doing pairwise database sequence searches, like BLAST and FASTA. We would like to understand how well these methods perform relative to one another and in an absolute sense.

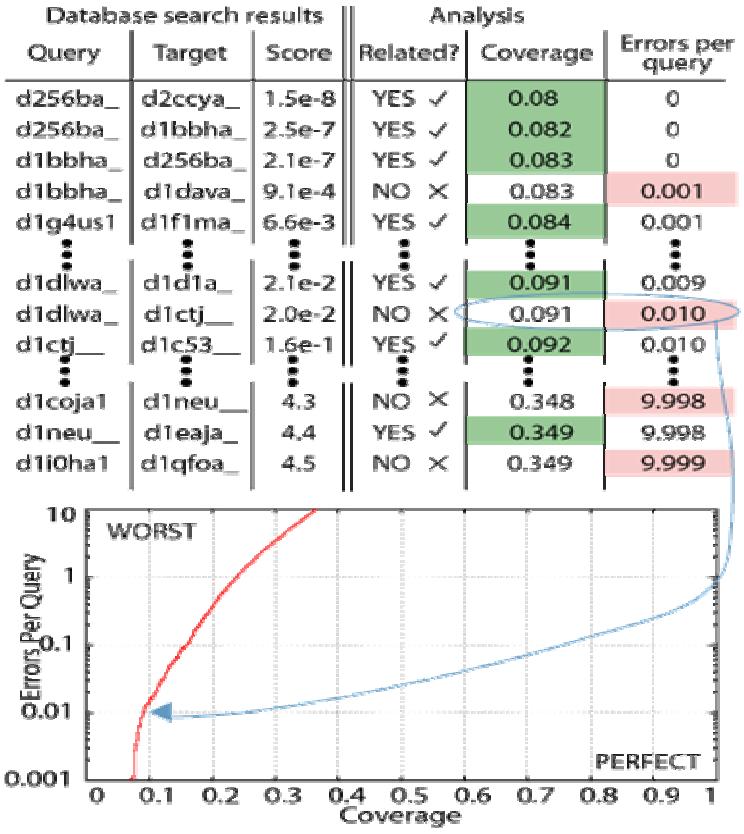
Additionally, we would like to know how best to use these methods in terms of user defined parameters and how sensitive each method is to parameter choice (like gap parameters and substitution matrix).



In order to assess database search methods, it is necessary to have a test dataset of sequences whose relationships are known. It is important that this knowledge be derived independently of the methods being tested. We use the structurally and evolutionarily derived relationships in the SCOP database for this purpose (figure). Specifically, the ASTRAL database provides SCOP sequences filtered at various levels of sequence identity. We use the ASTRAL genetic domain database filtered at 40% identity to make our evaluations specific for remote homologs. Sequences that are classified in the same superfamily are related both structurally and evolutionarily. Therefore, our tests evaluate a given method's ability to find the relations between superfamily members.

Methodology

Analyzing the results of a database search is always a matter of finding the best compromise between sensitivity and specificity. A given database search will likely yield a handful of very similar sequences that are homolgous and a larger handful of vaguely similar sequences some of which may be homologous. Homology (one of the most commonly misused words in biology) means sharing a common ancestor. Two sequences are homologous if there was a single sequence that gave rise to them both through duplication and divergence. If there is enough similarity between two sequences to be detected through pairwise comparison, it is usually safe to infer that they are homologous.

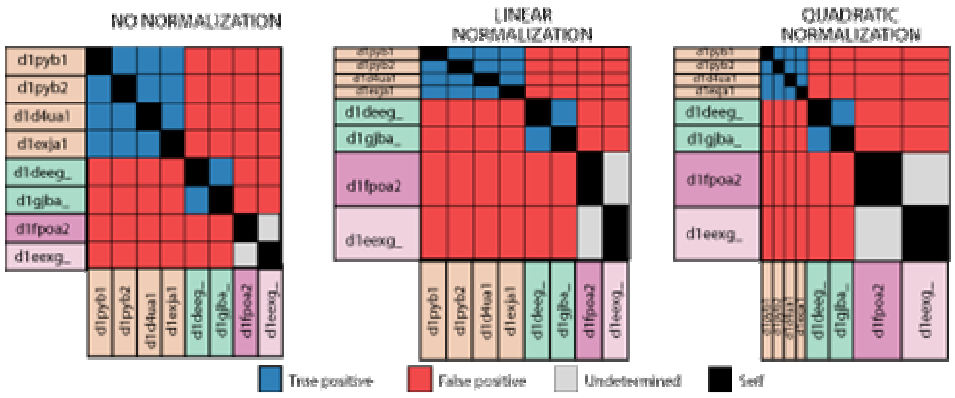


Database search results			Analysis		
Query	Target	Score	Related?	Coverage	Errors per query
d256ba_	d2ccya_	1.5e-8	YES ✓	0.08	0
d256ba_	d1bbha_	2.5e-7	YES ✓	0.082	0
d1bbha_	d256ba_	2.1e-7	YES ✓	0.083	0
d1bbha_	d1dava_	9.1e-4	NO ✗	0.083	0.001
d1g4us1	d1f1ma_	6.6e-3	YES ✓	0.084	0.001
⋮	⋮	⋮	⋮	⋮	⋮
d1dlwa_	d1d1a_	2.1e-2	YES ✓	0.091	0.009
d1dlwa_	d1ctj__	2.0e-2	NO ✗	0.091	0.010
d1ctj__	d1c53__	1.6e-1	YES ✓	0.092	0.010
⋮	⋮	⋮	⋮	⋮	⋮
d1coja1	d1neu__	4.3	NO ✗	0.348	9.998
d1neu__	d1eaja_	4.4	YES ✓	0.349	9.998
d1i0ha1	d1qfoa_	4.5	NO ✗	0.349	9.999

Frustratingly, there are many sequences with little or no detectable similarity that really are homolgous.

A good search method will find as many of the true homologs as possible in a database while cleanly separating them from the non-homologs. It will additionally be coupled to a good statistical scoring method that accurately reports how likely it is that any given match arose purely by chance and not because the sequences are really related.

In order to assess database search methods, we perform a database vs database search with our test sequences, derived from the ASTRAL database (figure). That is, we take each sequence, one by one, and use it as a query sequence to search the database. All of the hits from all of these searches are then pooled and sorted from most significant to least significant. Since we already know which sequences are related, we can then go through this list and see how many the method being evaluated got right. Ideally, there would be a clean separation between the true homologs, at the top of the list, and non-homologs at the bottom. In practice this never happens.

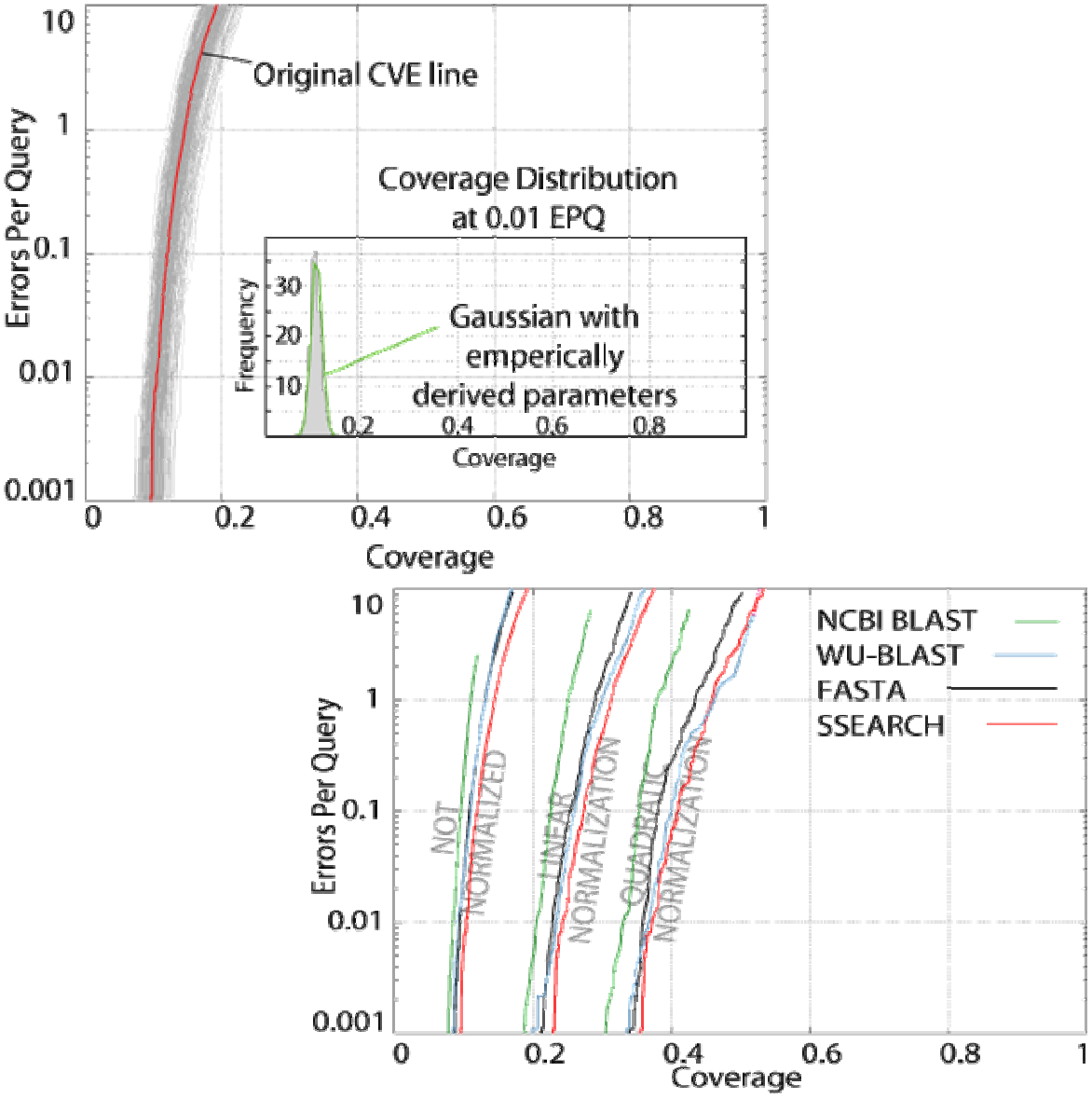


This data can be rendered in a Coverage versus Errors per Query (CVE) plot (figure). Coverage refers to how many of the true relations in the database were found. Erros per query is the number of times a given false relationship was reported, divided by the number of sequences in a database. The sorted list of all database search hits is traversed and at each significance level, a point on the CVE plot is generated. Looking at the CVE plot in the figure, you can see that as the coverage increases (as you find more

homologs), the number of errors increases. This is what is meant by a compromise between sensitivity and specificity.

CVE results from any two database search methods (or parameter sets, or scoring schemes, etc.) can be compared but it is not immediately obvious how significant any performance difference may be. To address this question, we use a technique pioneered by Brad Efron known as bootstrapping (figure). Bootstrapping is a method for estimating the distribution from which a given statistic derives, even when you can not resample from the population again. That is exactly the problem we are faced with here. There is a population of protein sequences out in nature and only a small fraction are sampled in the ASTRAL database. We can't easily throw out the ASTRAL database and have new ones generated. If we could, we would do that and recalculate CVE statistics for each one and then see its distribution directly. Instead, we bootstrap the data we do have. That is, we resample it by making new databases that derive from the original database. These resampled databases have the same number of sequences as the original database, but each sequence is represented a random number of times.

Another consideration in these analyses is that of the representational biases within our test set. Since our test sequences derive from solved structures, this problem is particularly acute. Only sequences that are amenable to structure determination and deemed interesting research subjects have their structures solved and make their way to the ASTRAL database. The ASTRAL database, filtered at 40% identity has a few very large superfamilies, like the immunoglobulins, and very many smaller superfamilies. To address this problem, we can normalize results by superfamily size, downweighting those that are in larger superfamilies. We employ two approaches to this. Linear normalization downweights each correctly identified relation in linear proportion to its superfamily size. In this way, larger superfamilies still contribute more to the overall results, but less than in unnormalized results. The other normalization scheme, quadratic normalization, weights the results from each superfamily equally, regardless of its size.

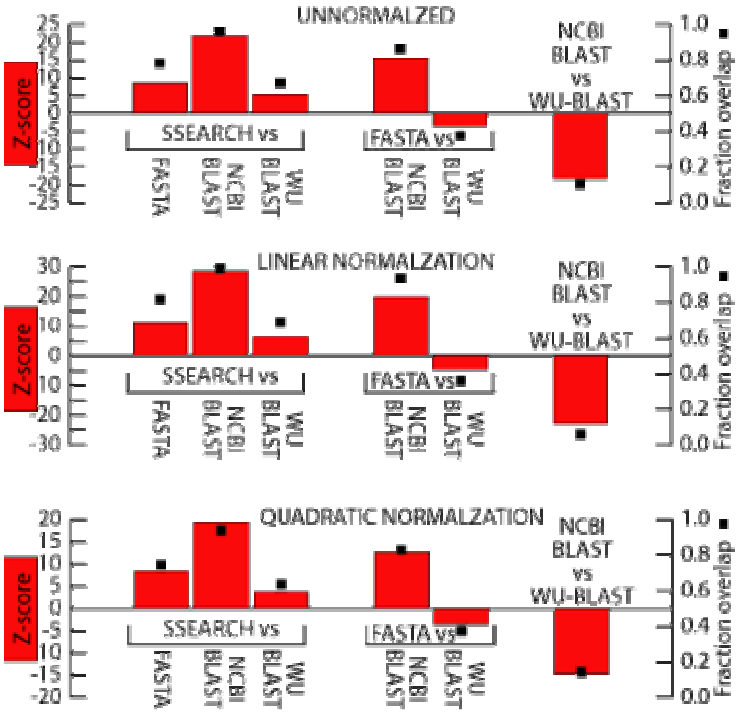


Result analysis

Great, so which pairwise method is the best? Well, it depends. The SSEARCH program, is a full implementation of the Smith-Waterman local alignment algorithm. It is guaranteed to find the optimal alignment under a given scoring scheme. The hueristic methods, like BLAST, run faster, but miss some of the best alignments. Therefore, they do not perform as well in CVE analysis.

The figure shows a CVE plot of four popular pairwise search programs, using parameters optimized for this test (see the "Bootstrapping and Normalization for Enhance Pairwise Sequence Comparison" manuscript for details). As you can see, all four methods fail to detect the majority of the evolutionary relationships in the database. This is what is meant by understanding how well the programs perform in an absolute sense. If our test database is anything like the real sequence databases biologists use (like GenBank), this means that when one does a database search, most of the sequences that are really related can't be detected. So, here's a good take-home message: If you clone and sequence a new gene, BLAST it against GenBank and fail to find any significant hits, **do not** write in your paper that your sequence is not homologous to any other known sequence. It very well may be, but you just can't detect it.

Another interesting observation is that for all four methods, if we normalize the results, the coverage increases. This means that when results from large superfamilies are discounted, the situation improves. Therefore, the pairwise relations in larger superfamiles are more difficult to detect, on the whole, than those in smaller superfamilies, in the ASTRAL database. Why should this be the case? I do not know.



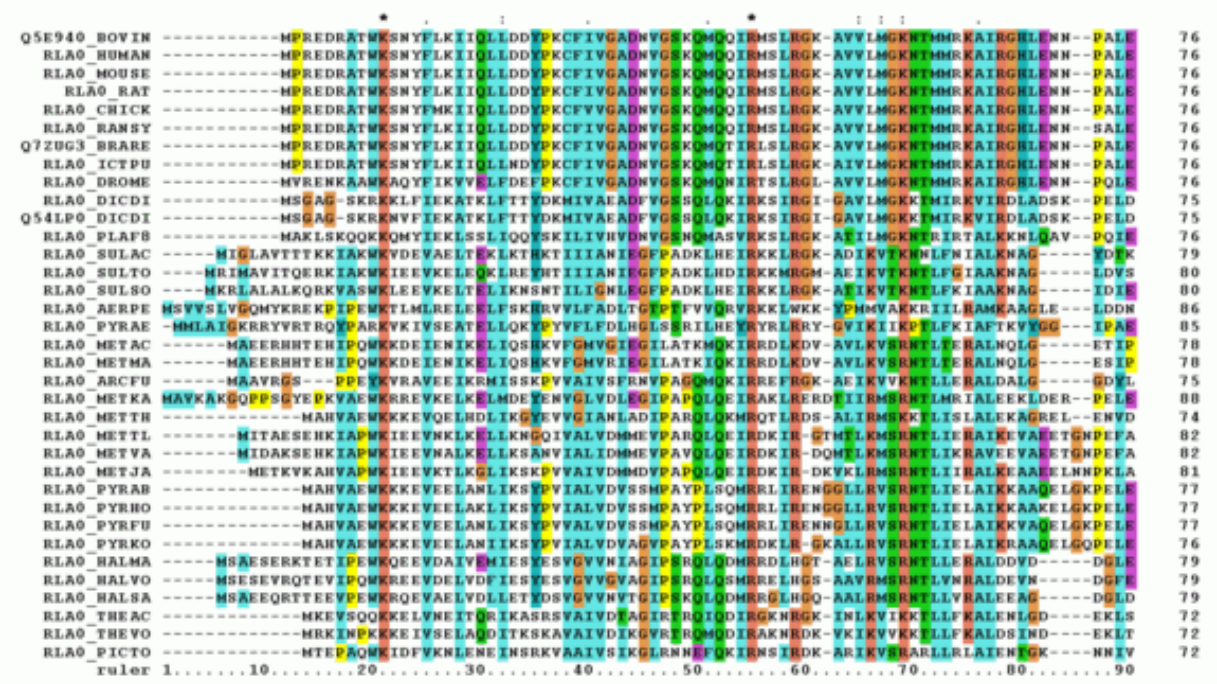
Using the bootstrap procedure we can determine whether the coverage difference we observe at a given error rate is significant. The figure to the right shows the 200 CVE lines generated when we bootstrap resample the database 200 times. Overlayed is the Original CVE line, that is, the CVE line from the unbootstrapped database. As you can see, the resampled databases are distributed around the original line. The inset shows a histogram of the 200 bootstrap coverage values at 1% error rate. If you calculate the standard deviation of these values and plug that into the Guassian formula, bootstrap theory predicts that it should fit the actual data pretty well. Looks like it does.

This is a figure showing how significant the differences between these methods are. The left axis is the Z-scores of a two sample parametric means test between the two methods using the bootstrapped standard error. Generally, a Z-score whose absolute value is greater than 2 or 3 means that two values are significantly different. Therefore, it looks like SSEARCH generates results that are significantly better than the heuristic

methods. The right axis is the fraction overlap between the two bootstrap distributions. This is a simple metric meant to give a general sense of how the two bootstrap distributions compare. It is generated by randomly drawing from one of the 200 bootstrap distributions of both methods and checking which had the greater coverage. This is repeated 1000 times. If the two distributions are equal, then the randomly drawn value from either one will be greater than the randomly drawn value from the other about half the time. This would be a fraction overlap value of 0.5. As you can see, this statistic roughly mirrors the Z-score of the two-sample parametric means test.

MULTIPLE SEQUENCE ALIGNMENT

A multiple sequence alignment (MSA) is a sequence alignment of three or more biological sequences, generally protein, DNA, or RNA. In many cases, the input set of query sequences are assumed to have an evolutionary relationship by which they share a lineage and are descended from a common ancestor. From the resulting MSA, sequence homology can be inferred and phylogenetic analysis can be conducted to assess the sequences' shared evolutionary origins. Visual depictions of the alignment as in the image at right illustrate mutation events such as point mutations (single amino acid or nucleotide changes) that appear as differing characters in a single alignment column, and insertion or deletion mutations (indels or gaps) that appear as hyphens in one or more of the sequences in the alignment. Multiple sequence alignment is often used to assess sequence conservation of protein domains, tertiary and secondary structures, and even individual amino acids or nucleotides.



First 90 positions of a protein multiple sequence alignment of instances of the acidic ribosomal protein P0 (L10E) from several organisms. Generated with ClustalX.

Multiple sequence alignment also refers to the process of aligning such a sequence set. Because three or more sequences of biologically relevant length can be difficult and are almost always time-consuming to align by hand, computational algorithms are used to produce and analyze the alignments. MSAs require more sophisticated methodologies than pairwise alignment because they are more computationally complex. Most multiple sequence alignment programs use heuristic methods rather than global optimization because identifying the optimal alignment between more than a few sequences of moderate length is prohibitively computationally expensive.

Dynamic programming and computational complexity

A direct method for producing an MSA uses the dynamic programming technique to identify the globally optimal alignment solution. For proteins, this method usually involves two sets of parameters: a gap penalty and a substitution matrix assigning scores or probabilities to the alignment of each possible pair of amino acids based on the similarity of the amino acids' chemical properties and the evolutionary probability of the mutation. For nucleotide sequences a similar gap penalty is used, but a much simpler

substitution matrix, wherein only identical matches and mismatches are considered, is typical. The scores in the substitution matrix may be either all positive or a mix of positive and negative in the case of a global alignment, but must be both positive and negative, in the case of a local alignment.

For *n* individual sequences, the naive method requires constructing the *n*-dimensional equivalent of the matrix formed in standard pairwise sequence alignment. The search space thus increases exponentially with increasing *n* and is also strongly dependent on sequence length. Expressed with the big O notation commonly used to measure computational complexity, a naïve MSA takes $O(Length^{Nseqs})$ time to produce. To find the global optimum for *n* sequences this way has been shown to be an NP-complete problem. In 1989, based on Carrillo-Lipman Algorithm, Altschul introduced a practical method that uses pairwise alignments to constrain the n-dimensional search space. In this approach pairwise dynamic programming alignments are performed on each pair of sequences in the query set, and only the space near the n-dimensional intersection of these alignments is searched for the n-way alignment. The MSA program optimizes the sum of all of the pairs of characters at each position in the alignment (the so-called *sum of pair* score) and has been implemented in a software program for constructing multiple sequence alignments.

Progressive alignment construction

The most widely used approach to multiple sequence alignments uses a heuristic search known as progressive technique (also known as the hierarchical or tree method), that builds up a final MSA by combining pairwise alignments beginning with the most similar pair and progressing to the most distantly related. All progressive alignment methods require two stages: a first stage in which the relationships between the sequences are represented as a tree, called a *guide tree*, and a second step in which the MSA is built by adding the sequences sequentially to the growing MSA according to the guide tree. The initial *guide tree* is determined by an efficient clustering method such as neighbor-joining or UPGMA, and may use distances based on the number of identical two letter sub-sequences (as in FASTA rather than a dynamic programming alignment).

Progressive alignments cannot be globally optimal. The primary problem is that when errors are made at any stage in growing the MSA, these errors are then propagated

through to the final result. Performance is also particularly bad when all of the sequences in the set are rather distantly related. Most modern progressive methods modify their scoring function with a secondary weighting function that assigns scaling factors to individual members of the query set in a nonlinear fashion based on their phylogenetic distance from their nearest neighbors. This corrects for non-random selection of the sequences given to the alignment program.

Progressive alignment methods are efficient enough to implement on a large scale for many (100s to 1000s) sequences. Progressive alignment services are commonly available on publicly accessible web servers so users need not locally install the applications of interest. The most popular progressive alignment method has been the Clustal family, especially the weighted variant ClustalW to which access is provided by a large number of web portals including GenomeNet, EBI, and EMBNet. Different portals or implementations can vary in user interface and make different parameters accessible to the user. ClustalW is used extensively for phylogenetic tree construction, in spite of the author's explicit warnings that unedited alignments should not be used in such studies and as input for protein structure prediction by homology modeling.

Another common progressive alignment method called T-Coffee is slower than Clustal and its derivatives but generally produces more accurate alignments for distantly related sequence sets. T-Coffee calculates pairwise alignments by combining the direct alignment of the pair with indirect alignments that aligns each sequence of the pair to a third sequence. It uses the output from Clustal as well as another local alignment program LALIGN, which finds multiple regions of local alignment between two sequences. The resulting alignment and phylogenetic tree are used as a guide to produce new and more accurate weighting factors.

Because progressive methods are heuristics that are not guaranteed to converge to a global optimum, alignment quality can be difficult to evaluate and their true biological significance can be obscure. A semi-progressive method that improves alignment quality and does not use a lossy heuristic while still running in polynomial time has been implemented in the program PSAlign.

Iterative methods

A set of methods to produce MSAs while reducing the errors inherent in progressive methods are classified as "iterative" because they work similarly to progressive methods but repeatedly realign the initial sequences as well as adding new sequences to the growing MSA. One reason progressive methods are so strongly dependent on a high-quality initial alignment is the fact that these alignments are always incorporated into the final result - that is, once a sequence has been aligned into the MSA, its alignment is not considered further. This approximation improves efficiency at the cost of accuracy. By contrast, iterative methods can return to previously calculated pairwise alignments or sub-MSAs incorporating subsets of the query sequence as a means of optimizing a general objective function such as finding a high-quality alignment score.

A variety of subtly different iteration methods have been implemented and made available in software packages; reviews and comparisons have been useful but generally refrain from choosing a "best" technique. The software package PRRN/PRRP uses a hill-climbing algorithm to optimize its MSA alignment score and iteratively corrects both alignment weights and locally divergent or "gappy" regions of the growing MSA. PRRP performs best when refining an alignment previously constructed by a faster method.

Another iterative program, DIALIGN, takes an unusual approach of focusing narrowly on local alignments between sub-segments or sequence motifs without introducing a gap penalty. The alignment of individual motifs is then achieved with a matrix representation similar to a dot-matrix plot in a pairwise alignment. An alternative method that uses fast local alignments as anchor points or "seeds" for a slower global-alignment procedure is implemented in the CHAOS/DIALIGN suite.

A third popular iteration-based method called MUSCLE (multiple sequence alignment by log-expectation) improves on progressive methods with a more accurate distance measure to assess the relatedness of two sequences. The distance measure is updated between iteration stages (although, in its original form, MUSCLE contained only 2-3 iterations depending on whether refinement was enabled).

Hidden Markov models

Hidden Markov models are probabilistic models that can assign likelihoods to all possible combinations of gaps, matches, and mismatches to determine the most likely

MSA or set of possible MSAs. HMMs can produce a single highest-scoring output but can also generate a family of possible alignments that can then be evaluated for biological significance. HMMs can produce both global and local alignments. Although HMM-based methods have been developed relatively recently, they offer significant improvements in computational speed, especially for sequences that contain overlapping regions.

Typical HMM-based methods work by representing an MSA as a form of directed acyclic graph known as a partial-order graph, which consists of a series of nodes representing possible entries in the columns of an MSA. In this representation a column that is absolutely conserved (that is, that all the sequences in the MSA share a particular character at a particular position) is coded as a single node with as many outgoing connections as there are possible characters in the next column of the alignment. In the terms of a typical hidden Markov model, the observed states are the individual alignment columns and the "hidden" states represent the presumed ancestral sequence from which the sequences in the query set are hypothesized to have descended. An efficient search variant of the dynamic programming method, known as the Viterbi algorithm, is generally used to successively align the growing MSA to the next sequence in the query set to produce a new MSA. This is distinct from progressive alignment methods because the alignment of prior sequences is updated at each new sequence addition. However, like progressive methods, this technique can be influenced by the order in which the sequences in the query set are integrated into the alignment, especially when the sequences are distantly related.

Several software programs are available in which variants of HMM-based methods have been implemented and which are noted for their scalability and efficiency, although properly using an HMM method is more complex than using more common progressive methods. The simplest is POA (Partial-Order Alignment); a similar but more generalized method is implemented in the packages SAM (Sequence Alignment and Modeling System) and HMMER. SAM has been used as a source of alignments for protein structure prediction to participate in the CASP structure prediction experiment and to develop a database of predicted proteins in the yeast species *S. cerevisiae*. HHsearch is a software package for the detection of remotely related protein sequences based on the

pairwise comparison of HMMs. A server running HHsearch (HHpred) was by far the fastest of the 10 best automatic structure prediction servers in the CASP7 and CASP8 structure prediction competitions.

Genetic algorithms and simulated annealing

Standard optimization techniques in computer science - both of which were inspired by, but do not directly reproduce, physical processes - have also been used in an attempt to more efficiently produce quality MSAs. One such technique, genetic algorithms, has been used for MSA production in an attempt to broadly simulate the hypothesized evolutionary process that gave rise to the divergence in the query set. The method works by breaking a series of possible MSAs into fragments and repeatedly rearranging those fragments with the introduction of gaps at varying positions. A general objective function is optimized during the simulation, most generally the "sum of pairs" maximization function introduced in dynamic programming-based MSA methods. A technique for protein sequences has been implemented in the software program SAGA (Sequence Alignment by Genetic Algorithm) and its equivalent in RNA is called RAGA.

The technique of simulated annealing, by which an existing MSA produced by another method is refined by a series of rearrangements designed to find better regions of alignment space than the one the input alignment already occupies. Like the genetic algorithm method, simulated annealing maximizes an objective function like the sum-of-pairs function. Simulated annealing uses a metaphorical "temperature factor" that determines the rate at which rearrangements proceed and the likelihood of each rearrangement; typical usage alternates periods of high rearrangement rates with relatively low likelihood (to explore more distant regions of alignment space) with periods of lower rates and higher likelihoods to more thoroughly explore local minima near the newly "colonized" regions. This approach has been implemented in the program MSASA (Multiple Sequence Alignment by Simulated Annealing).

Motif finding

Alignment of the seven Drosophila caspases colored by motifs as identified by MEME. When motif positions and sequence alignments are generated independently, they often correlate well but not perfectly, as in this example.

Motif finding, also known as profile analysis, is a method of locating sequence motifs in global MSAs that is both a means of producing a better MSA and a means of producing a scoring matrix for use in searching other sequences for similar motifs. A variety of methods for isolating the motifs have been developed, but all are based on identifying short highly conserved patterns within the larger alignment and constructing a matrix similar to a substitution matrix that reflects the amino acid or nucleotide composition of each position in the putative motif. The alignment can then be refined using these matrices. In standard profile analysis, the matrix includes entries for each possible character as well as entries for gaps. Alternatively, statistical pattern-finding algorithms can identify motifs as a precursor to an MSA rather than as a derivation. In many cases when the query set contains only a small number of sequences or contains only highly related sequences, pseudocounts are added to normalize the distribution reflected in the scoring matrix. In particular, this corrects zero-probability entries in the matrix to values that are small but nonzero.

Blocks analysis is a method of motif finding that restricts motifs to ungapped regions in the alignment. Blocks can be generated from an MSA or they can be extracted from unaligned sequences using a precalculated set of common motifs previously generated from known gene families. Block scoring generally relies on the spacing of high-frequency characters rather than on the calculation of an explicit substitution matrix. The BLOCKS server provides an interactive method to locate such motifs in unaligned sequences.

Statistical pattern-matching has been implemented using both the expectation-maximization algorithm and the Gibbs sampler. One of the most common motif-finding tools, known as MEME, uses expectation maximization and hidden Markov methods to generate motifs that are then used as search tools by its companion MAST in the combined suite MEME/MAST.

Visualization and editing tools

The necessary use of heuristics for multiple alignment means that for an arbitrary set of proteins, there is always a good chance that an alignment will contain errors. These can arise because of unique insertions into one or more regions of sequences, or through some more complex evolutionary process leading to proteins that do not align easily by

sequence alone. Multiple sequence alignment viewers enable alignments to be visually verified, often by inspecting the quality of alignment for annotated functional sites on two or more sequences. Many also enable the alignment to be edited to correct these (usually minor) errors, in order to obtain an optimal 'curated' alignment suitable for use in phylogenetic analysis or comparative modeling.

Use in phylogenetics

Multiple sequence alignments can be used to create a phylogenetic tree. This is made possible by two reasons. The first is because functional domains that are known in annotated sequences can be used for alignment in non-annotated sequences. The other is that conserved regions known to be functionally important can be found. This makes it possible for multiple sequence alignments to be used to analyze and find evolutionary relationships through homology between sequences. Point mutations and insertion or deletion events (called indels) can be detected.

Multiple sequence alignments can also be used to identify functionally important sites, such as binding sites, active sites, or sites corresponding to other key functions, by locating conserved domains. When looking at multiple sequence alignments, it is useful to consider different aspects of the sequences when comparing sequences. These aspects include identity, similarity, and homology. Identity means that the sequences have identical residues at their respective positions. On the other hand, similarity has to do with the sequences being compared having similar residues quantitatively. For example, in terms of nucleotide sequences, pyrimidines are considered similar to each other, as are purines. Similarity ultimately leads to homology, in that the more similar sequences are, the closer they are to being homologous. This homology in sequences, can then go on to help find common ancestry

DOT MATRIX ANALYSIS

Dot-matrix methods

A DNA dot plot of a human zinc finger transcription factor (GenBank ID NM_002383), showing regional self-similarity. The main diagonal represents the sequence's alignment with itself; lines off the main diagonal represent similar or repetitive patterns within the sequence. This is a typical example of a recurrence plot.

The dot-matrix approach, which implicitly produces a family of alignments for individual sequence regions, is qualitative and conceptually simple, though time-consuming to analyze on a large scale. In the absence of noise, it can be easy to visually identify certain sequence features—such as insertions, deletions, repeats, or inverted repeats—from a dot-matrix plot. To construct a dot-matrix plot, the two sequences are written along the top row and leftmost column of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match—this is a typical recurrence plot. Some implementations vary the size or intensity of the dot depending on the degree of similarity of the two characters, to accommodate conservative substitutions. The dot plots of very closely related sequences will appear as a single line along the matrix's main diagonal.

Problems with dot plots as an information display technique include: noise, lack of clarity, non-intuitiveness, difficulty extracting match summary statistics and match positions on the two sequences. There is also much wasted space where the match data is inherently duplicated across the diagonal and most of the actual area of the plot is taken up by either empty space or noise, and, finally, dot-plots are limited to two sequences. None of these limitations apply to Miropeats alignment diagrams but they have their own particular flaws.

Dot plots can also be used to assess repetitiveness in a single sequence. A sequence can be plotted against itself and regions that share significant similarities will appear as lines off the main diagonal. This effect can occur when a protein consists of multiple similar structural domains.

DYNAMIC PROGRAMMING

The technique of dynamic programming can be applied to produce global alignments via the Needleman-Wunsch algorithm, and local alignments via the Smith-Waterman algorithm. In typical usage, protein alignments use a substitution matrix to assign scores to amino-acid matches or mismatches, and a gap penalty for matching an amino acid in one sequence to a gap in the other. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty. (In standard dynamic programming, the score of each amino acid position is independent of the identity of its neighbors, and

therefore base stacking effects are not taken into account. However, it is possible to account for such effects by modifying the algorithm.) A common extension to standard linear gap costs, is the usage of two different gap penalties for opening a gap and for extending a gap. Typically the former is much larger than the latter, e.g. -10 for gap open and -2 for gap extension. Thus, the number of gaps in an alignment is usually reduced and residues and gaps are kept together, which typically makes more biological sense. The Gotoh algorithm implements affine gap costs by using three matrices.

Dynamic programming can be useful in aligning nucleotide to protein sequences, a task complicated by the need to take into account frameshift mutations (usually insertions or deletions). The framesearch method produces a series of global or local pairwise alignments between a query nucleotide sequence and a search set of protein sequences, or vice versa. Its ability to evaluate frameshifts offset by an arbitrary number of nucleotides makes the method useful for sequences containing large numbers of indels, which can be very difficult to align with more efficient heuristic methods. In practice, the method requires large amounts of computing power or a system whose architecture is specialized for dynamic programming. The BLAST and EMBOSS suites provide basic tools for creating translated alignments (though some of these approaches take advantage of side-effects of sequence searching capabilities of the tools). More general methods are available from both commercial sources, such as *FrameSearch*, distributed as part of the Accelrys GCG package, and Open Source software such as Genewise.

The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function; however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, it is prohibitively slow for large numbers of or extremely long sequences.

Word methods

Word methods, also known as *k*-tuple methods, are heuristic methods that are not guaranteed to find an optimal alignment solution, but are significantly more efficient than dynamic programming. These methods are especially useful in large-scale database searches where it is understood that a large proportion of the candidate sequences will have essentially no significant match with the query sequence. Word methods are best

known for their implementation in the database search tools FASTA and the BLAST family. Word methods identify a series of short, non overlapping subsequences ("words") in the query sequence that are then matched to candidate database sequences. The relative positions of the word in the two sequences being compared are subtracted to obtain an offset; this will indicate a region of alignment if multiple distinct words produce the same offset. Only if this region is detected do these methods apply more sensitive alignment criteria; thus, many unnecessary comparisons with sequences of no appreciable similarity are eliminated.

In the FASTA method, the user defines a value *k* to use as the word length with which to search the database. The method is slower but more sensitive at lower values of *k*, which are also preferred for searches involving a very short query sequence. The BLAST family of search methods provides a number of algorithms optimized for particular types of queries, such as searching for distantly related sequence matches. BLAST was developed to provide a faster alternative to FASTA without sacrificing much accuracy; like FASTA, BLAST uses a word search of length *k*, but evaluates only the most significant word matches, rather than every word match as does FASTA. Most BLAST implementations use a fixed default word length that is optimized for the query and database type, and that is changed only under special circumstances, such as when searching with repetitive or very short query sequences. Implementations can be found via a number of web portals, such as EMBL FASTA and NCBI BLAS

SUBSTITUTION MATRIX

In bioinformatics and evolutionary biology, a substitution matrix describes the rate at which one character in a sequence changes to other character states over time. Substitution matrices are usually seen in the context of amino acid or DNA sequence alignments, where the similarity between sequences depends on their divergence time and the substitution rates as represented in the matrix.

Background

In the process of evolution, from one generation to the next the amino acid sequences of an organism's proteins are gradually altered through the action of DNA mutations. For example, the sequence

ALEIRYLRD

could mutate into the sequence

ALEINYLRD

in one step, and possibly

AQEINYQRD

over a longer period of evolutionary time. Each amino acid is more or less likely to mutate into various other amino acids. For instance, a hydrophilic residue such as arginine is more likely to be replaced another hydrophilic residue such as glutamine, than it is to be mutated into a hydrophobic residue such as leucine. This is primarily due to redundancy in the genetic code, which translates similar codons into similar amino acids. Furthermore, mutating an amino acid to a residue with significantly different properties could affect the folding and/or activity of the protein. There is therefore usually strong selective pressure to remove such mutations quickly from a population.

If we have two amino acid sequences in front of us, we should be able to say something about how likely they are to be derived from a common ancestor, or homologous. If we can line up the two sequences using a sequence alignment algorithm such that the mutations required to transform a hypothetical ancestor sequence into both of the current sequences would be evolutionarily plausible, then we'd like to assign a high score to the comparison of the sequences.

To this end, we will construct a 20x20 matrix where the (i,j)th entry is equal to the probability of the ith amino acid being transformed into the jth amino acid in a certain amount of evolutionary time. There are many different ways to construct such a matrix, called a **substitution matrix**. Here are the most commonly used ones:

Identity matrix

The simplest possible substitution matrix would be one in which each amino acid is considered maximally similar to itself, but not able to transform into any other amino acid. This matrix would look like:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

This identity matrix will succeed in the alignment of very similar amino acid sequences but will be miserable at aligning two distantly related sequences. We need to figure out all the probabilities in a more rigorous fashion. It turns out that an empirical examination of previously aligned sequences works best.

Log-odds matrices

We express the probabilities of transformation in what are called log-odds scores. The scores matrix S is defined as

$$S_{i,j} = \log \frac{p_i \cdot M_{i,j}}{p_i \cdot p_j} = \log \frac{M_{i,j}}{p_j} = \log \frac{observed\ frequency}{expected\ frequency}$$

where $M_{i,j}$ is the probability that amino acid *i* transforms into amino acid *j* and p_i is the frequency of amino acid i. The base of the logarithm is not important, and you will often see the same substitution matrix expressed in different bases.

PAM

One of the first amino acid substitution matrices, the PAM *(Point Accepted Mutation)* matrix was developed by Margaret Dayhoff in the 1970s. This matrix is calculated by observing the differences in closely related proteins. The PAM1 matrix estimates what rate of substitution would be expected if 1% of the amino acids had changed. The PAM1 matrix is used as the basis for calculating other matrices by assuming that repeated mutations would follow the same pattern as those in the PAM1 matrix, and multiple substitutions can occur at the same site. Using this logic, Dayhoff derived matrices as high as PAM250. Usually the PAM 30 and the PAM70 are used.

A matrix for divergent sequences can be calculated from a matrix for closely related sequences by taking the second matrix to a power. For instance, we can roughly approximate the WIKI2 matrix from the WIKI1 matrix by saying $W_2 = W_1^2$ where W_1 is WIKI1 and W_2 is WIKI2. This is how the PAM250 matrix is calculated.

BLOSUM

Dayhoff's methodology of comparing closely related species turned out not to work very well for aligning evolutionarily divergent sequences. Sequence changes over long evolutionary time scales are not well approximated by compounding small changes that occur over short time scales. The BLOSUM *(BLOck SUbstitution Matrix)* series of matrices rectifies this problem. Henikoff and Henikoff constructed these matrices using multiple alignments of evolutionarily divergent proteins. The probabilities used in the matrix calculation are computed by looking at "blocks" of conserved sequences found in multiple protein alignments. These conserved sequences are assumed to be of functional importance within related proteins. To reduce bias from closely related sequences, segments in a block with a sequence identity above a certain threshold were clustered giving weight 1 to each such cluster (Henikoff and Henikoff). For the BLOSUM62 matrix, this threshold was set at 62%. Pairs frequencies were then counted between clusters, hence pairs were only counted between segments less than 62% identical. One would use a higher numbered BLOSUM matrix for aligning two closely related sequences and a lower number for more divergent sequences.

It turns out that the BLOSUM62 matrix does an excellent job detecting similarities in distant sequences, and this is the matrix used by default in most recent alignment applications such as BLAST.

Differences between PAM and BLOSUM

1. PAM matrices are based on an explicit evolutionary model (i.e. replacements are counted on the branches of a phylogenetic tree), whereas the BLOSUM matrices are based on an implicit model of evolution.
2. The PAM matrices are based on mutations observed throughout a global alignment, this includes both highly conserved and highly mutable regions. The BLOSUM matrices are based only on highly conserved regions in series of alignments forbidden to contain gaps.
3. The method used to count the replacements is different: unlike the PAM matrix, the BLOSUM procedure uses groups of sequences within which not all mutations are counted the same.

4. Higher numbers in the PAM matrix naming scheme denote larger evolutionary distance, while larger numbers in the BLOSUM matrix naming scheme denote higher sequence similarity and therefore smaller evolutionary distance. Example: PAM150 is used for more distant sequences than PAM100; BLOSUM62 is used for closer sequences than Blosum50.

Extensions and improvements

Many specialized substitution matrices have been developed that describe the amino acid substitution rates in specific structural or sequence contexts, such as in transmembrane alpha helices, for combinations of secondary structure states and solvent accessibility states, or for local sequence-structure contexts. These context-specific substitution matrices lead to generally improved alignment quality at some cost of speed but are not yet widely used. Recently, sequence context-specific amino acid similarities have been derived that do not need substitution matrices but that rely on a library of sequence contexts instead. Using this idea, a context-specific extension of the popular BLAST program has been demonstrated to achieve a twofold sensitivity improvement for remotely related sequences over BLAST at similar speeds (CS-BLAST).

ALGORITHM FOR SEQUENCE ALIGNMENT: DYNAMIC PROGRAMMING

Making an alignment by hand is possible, but tedious. In some cases, when one has a lot of information about the proteins, such as active site residues, secondary structure, 3D structure, mutations, etc, it may still be necessary to make a manual alignment (or at least edit an alignment) to fit all the data. The available automatic methods may not be able to produce a good enough alignment in such cases.

Of course, we would like to have a completely automatic method to perform sequence alignment. The method of choice is based on so-called **dynamic programming**, which is a general algorithm for solving certain optimization problems. The word "programming" does not mean that it has to be a computer program; this is just mathematical jargon for using a fixed set of rules to arrive at a solution.

For any automatic method to work, we need to be explicit about the assumptions that should go into it. We therefore need to have an explicit scheme for the gap penalties and for the substitution matrix (see the previous page). The chosen gap penalties and substitution matrices are often collectively called the **scoring scheme**.

There are clearly many different possible scoring schemes. One may also complicate things further by allowing position-specific scores: If one knows from other sources (3D structure) that a gap should absolutely not be allowed in a certain part of a sequence, then the gap-open penalty could be set to a very high value in that part.

Given a scoring scheme, how does an alignment algorithm work? Let us use the classical **Needleman-Wunsch-Sellers** algorithm to demonstrate how a dynamic-programming algorithm can work. Please note that there are other variants of dynamic programming in sequence analysis.

The Needleman-Wunsch-Sellers algorithm sets up a matrix where each sequence is placed along the sides of the matrix. Each element in the matrix represents the two residues of the sequences being aligned at that position. To calculate the score in every position (**i**, **j**) one looks at the alignment that has already been made up to that point, and finds the best way to continue. Having gone through the entire matrix in this way, one can go back and trace which way through the matrix gives the best alignment.

Let us use the following gap-penalty function, where **k** is the length of the gap, $\mathbf{c_{open}}$ the gap-open penalty constant, and $\mathbf{c_{length}}$ the gap-length penalty constant:

$W(k) = c_{open} + c_{length} * k$

The formulat describing the Needleman-Wunsch-Sellers method is recursive, and for the position (**i**, **j**) is as follows, where **D** is value of element (**i**, **j**) in the matrix and **subst** is the substitution matrix:

$D_{i,j} = \max \{$	$D_{i-1,j-1} + \text{subst}(A_i, B_j)$
	$D_{i-1,j-k} + W(k)$ (where $k = 1, ..., j-1$)
	$D_{i-k,j-1} + W(k)$ (where $k = 1, ..., i-1$)

After one has applied this to the matrix, one finds the optimal alignment by tracing backwards from the diagonal element backwards to the previous highest value, and so on.

I have found a good tutorial describing dynamic programming for sequence alignment of the Needleman-Wunsch variant. The tutorial was written by **Eric C. Rouchka** (Washington University in St. Louis), and walks through an example in detail.

1. Dynamic programming, a simple variant, with no gap penalties and a simple substitution scoring scheme.

2. Dynamic programming, a more advanced variant, with gap penalties and a slightly more complicated substitution scoring scheme.

Bayesian probability is one of the different interpretations of the concept of probability and belongs to the category of evidential probabilities. The Bayesian interpretation of probability can be seen as an extension of logic that enables reasoning with uncertain statements. To evaluate the probability of a hypothesis, the Bayesian probabilist specifies some prior probability, which is then updated in the light of new relevant data. The Bayesian interpretation provides a standard set of procedures and formulae to perform this calculation. Bayesian probability interprets the concept of probability as "a measure of a state of knowledge", in contrast to interpreting it as a frequency or a "propensity" of some phenomenon.

"Bayesian" refers to the 18th century mathematician and theologian Thomas Bayes (1702–1761), who provided the first mathematical treatment of a non-trivial problem of Bayesian inference. Nevertheless, it was the French mathematician Pierre-Simon Laplace (1749–1827) who pioneered and popularized what is now called Bayesian probability.

Broadly speaking, there are two views on Bayesian probability that interpret the *state of knowledge* concept in different ways. According to the *objectivist view*, the rules of Bayesian statistics can be justified by requirements of rationality and consistency and interpreted as an extension of logic. According to the *subjectivist view*, the state of knowledge measures a "personal belief" Many modern machine learning methods are based on objectivist Bayesian principles. In the Bayesian view, a probability is assigned to a hypothesis, whereas under the frequentist view, a hypothesis is typically tested without being assigned a probability.

BLAST

In bioinformatics, **B**asic **L**ocal **A**lignment **S**earch **T**ool, or **BLAST**, is an algorithm for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences. A BLAST search enables a researcher to compare a query sequence with a library or database of sequences, and identify library sequences that resemble the query sequence above a certain threshold. Different types of BLASTs are available according to the query

sequences. For example, following the discovery of a previously unknown gene in the mouse, a scientist will typically perform a BLAST search of the human genome to see if humans carry a similar gene; BLAST will identify sequences in the human genome that resemble the mouse gene based on similarity of sequence. The BLAST program was designed by Eugene Myers, Stephen Altschul, Warren Gish, David J. Lipman, and Webb Miller at the NIH and was published in the *Journal of Molecular Biology* in 1990.

Background

BLAST is one of the most widely used bioinformatics programs, because it addresses a fundamental problem and the algorithm emphasizes speed over sensitivity. This emphasis on speed is vital to making the algorithm practical on the huge genome databases currently available, although subsequent algorithms can be even faster.

Before fast algorithms such as BLAST and FASTA were developed, doing database searches for the protein or nucleic sequences was very time consuming by using a full alignment procedure like Smith-Waterman.

Indeed, BLAST is faster than Smith-Waterman, however, it cannot "guarantee the optimal alignments of the query and database sequences", as Smith-Waterman does, which "ensured the best performance on accuracy and the most precise results" at the expense of time and computer power.

BLAST is more time efficient than FASTA by searching only for the more significant patterns in the sequences, but with comparative sensitivity. This could be further realized by knowing the algorithm of BLAST introduced below.

Examples of other questions that researchers use BLAST to answer are:

- Which bacterial species have a protein that is related in lineage to a certain protein with known amino-acid sequence?
- Where does a certain sequence of DNA originate?
- What other genes encode proteins that exhibit structures or motifs such as ones that have just been determined?

BLAST is also often used as part of other algorithms that require approximate sequence matching.

The BLAST algorithm and the computer program that implements it were developed by Stephen Altschul, Warren Gish, and David Lipman at the U.S. National

Center for Biotechnology Information (NCBI), Webb Miller at the Pennsylvania State University, and Gene Myers at the University of Arizona. It is available on the web on the NCBI website. Alternative implementations include AB-BLAST (formerly known as WU-BLAST), FSA-BLAST (last updated in 2006), and ScalaBLAST.

Input

Input sequences should in FASTA format or Genbank format.

Output

BLAST output can be delivered in a variety of formats. These formats include HTML, plain text, and XML formatting. For NCBI's web-page, the default format for output is HTML. When performing a BLAST on NCBI, the results are given in a graphical format showing the hits found, a table showing sequence identifiers for the hits with scoring related data, as well as alignments for the sequence of interest and the hits received with corresponding BLAST scores for these. The easiest to read and most informative of these is probably the table.

If you are searching a proprietary sequence or simply one that is unavailable in databases available to the public through sources such as NCBI, there is a BLAST program available for download to any computer, at no cost. This can be found at BLAST+ executables. There are also commercial programs available for purchase. Databases can be found from the NCBI site, as well as from Index of BLAST databases (FTP).

Process

Using a heuristic method, BLAST finds homologous sequences, not by comparing either sequence in its entirety, but rather by locating short matches between the two sequences. This process of finding initial words is called seeding. It is after this first match that BLAST begins to make local alignments. While attempting to find homology in sequences, sets of common letters, known as words, are very important. For example, suppose that the sequence contains the following stretch of letters, GLKFA. If a BLASTp was being conducted under default conditions, the word size would be 3 letters. In this case, using the given stretch of letters, the searched words would be GLK, LKF, KFA. The heuristic algorithm of BLAST locates all common three-letter words between the sequence of interest and the hit sequence, or sequences, from the database. These results

will then be used to build an alignment. After making words for the sequence of interest, neighborhood words are also assembled. These words must satisfy a requirement of having a score of at least the threshold, T, when compared by using a scoring matrix. Along the lines of terms stated above, if a BLASTp were being conducted, the scoring matrix that would be used would most likely be BLOSUM62. Once both words and neighborhood words are assembled and compiled, they are compared to the sequences in the database in order to find matches. The *threshold score* T, determines whether a particular word will be included in the alignment or not. Once seeding has been conducted, the alignment, which is only 3 residues long, is extended in both directions by the algorithm used by BLAST. Each extension impacts the score of the alignment by either increasing or decreasing it. Should this score be higher than a pre-determined T, the alignment will be included in the results given by BLAST. However, should this score be lower than this pre-determined T, the alignment will cease to extend, preventing areas of poor alignment to be included in the BLAST results. Note, that increasing the T score limits the amount of space available to search, decreasing the number of neighborhood words, while at the same time speeding up the process of BLAST.

Algorithm

BLAST

To run, BLAST requires a query sequence to search for, and a sequence to search against (also called the target sequence) or a sequence database containing multiple such sequences. BLAST will find subsequences in the database which are similar to subsequences in the query. In typical usage, the query sequence is much smaller than the database, e.g., the query may be one thousand nucleotides while the database is several billion nucleotides.

The main idea of BLAST is that there are often high-scoring segment pairs (HSP) contained in a statistically significant alignment. BLAST searches for high scoring sequence alignments between the query sequence and sequences in the database using a heuristic approach that approximates the Smith-Waterman algorithm. The exhaustive Smith-Waterman approach is too slow for searching large genomic databases such as GenBank. Therefore, the BLAST algorithm uses a heuristic approach that is less accurate than the Smith-Waterman algorithm but over 50 times faster. The speed and relatively

good accuracy of BLAST are among the key technical innovations of the BLAST programs.

An overview of the BLASTP algorithm (a protein to protein search) is as follows:

1. **Remove low-complexity region or sequence repeats in the query sequence.**

   "Low-complexity region" means a region of a sequence composed of few kinds of elements. These regions might give high scores that confuse the program to find the actual significant sequences in the database, so they should be filtered out. The regions will be marked with an X (protein sequences) or N (nucleic acid sequences) and then be ignored by the BLAST program. To filter out the low-complexity regions, the SEG program is used for protein sequences and the program DUST is used for DNA sequences. On the other hand, the program XNU is used to mask off the tandem repeats in protein sequences.

2. **Make a k-letter word list of the query sequence.**

   Take k=3 for example, we list the words of length 3 in the query protein sequence (k is usually 11 for a DNA sequence) "sequentially", until the last letter of the query sequence is included. The method is illustrated in figure 1.

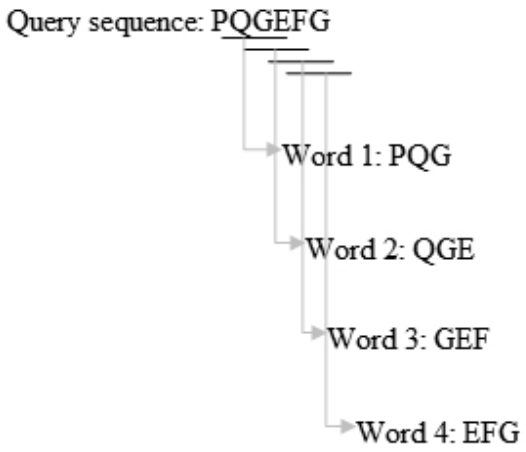


Fig. 1 The method to establish the k-letter query word list.

3. **List the possible matching words.**

   This step is one of the main differences between BLAST and FASTA. FASTA cares about all of the common words in the database and query sequences that are listed in step 2; however, BLAST only cares about the high-scoring words. The scores are

created by comparing the word in the list in step 2 with all the 3-letter words. By using the scoring matrix (substitution matrix) to score the comparison of each residue pair, there are 20^3 possible match scores for a 3-letter word. For example, the score obtained by comparing PQG with PEG and PQA is 15 and 12, respectively. For DNA words, a match is scored as +5 and a mismatch as -4, or as +2 and -3. After that, a neighborhood word score threshold T is used to reduce the number of possible matching words. The words whose scores are greater than the threshold T will remain in the possible matching words list, while those with lower scores will be discarded. For example, PEG is kept, but PQA is abandoned when T is 13.

4. **Organize the remaining high-scoring words into an efficient search tree.**

This is for the purpose that the program can rapidly compare the high-scoring words to the database sequences.

5. **Repeat step 3 to 4 for each k-letter word in the query sequence.**
6. **Scan the database sequences for exact matches with the remaining high-scoring words.**

The BLAST program scans the database sequences for the remaining high-scoring word, such as PEG, of each position. If an exact match is found, this match is used to seed a possible un-gapped alignment between the query and database sequences.

7. **Extend the exact matches to high-scoring segment pair (HSP).**

- The original version of BLAST stretches a longer alignment between the query and the database sequence in the left and right directions, from the position where the exact match occurred. The extension doesn't stop until the accumulated total score of the HSP begins to decrease. A simplified example is presented in figure 2.

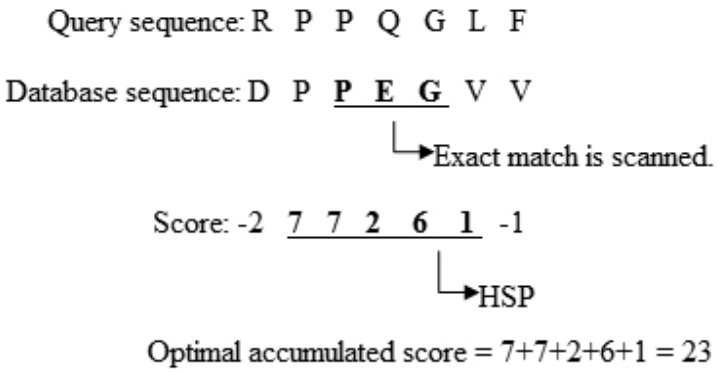


Fig. 2 The process to extend the exact match.

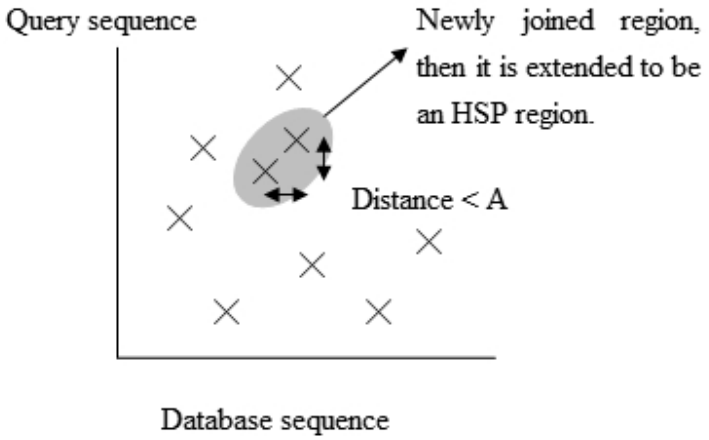


Fig. 3 The positions of the exact matches.

To save more time, a newer version of BLAST, called BLAST2 or gapped BLAST, has been developed. BLAST2 adopts a lower neighborhood word score threshold to maintain the same level of sensitivity for detecting sequence similarity. Therefore, the possible matching words list in step 3 becomes longer. Next, the exact matched regions, within distance A from each other on the same diagonal in figure 3, will be joined as a longer new region. Finally, the new regions are then extended by the same method as in the original version of BLAST, and the HSPs' (High-scoring segment pair) scores of the extended regions are then created by using a substitution matrix as before.

8. **List all of the HSPs in the database whose score is high enough to be considered.**

   We list the HSPs whose scores are greater than the empirically determined cutoff score S. By examining the distribution of the alignment scores modeled by

comparing random sequences, a cutoff score S can be determined such that its value is large enough to guarantee the significance of the remaining HSPs.

9. **Evaluate the significance of the HSP score.**

BLAST next assesses the statistical significance of each HSP score by exploiting the Gumbel extreme value distribution (EVD). (It is proved that the distribution of Smith-Waterman local alignment scores between two random sequences follows the Gumbel EVD. For local alignments containing gaps it is not proved.). In accordance with the Gumbel EVD, the probability p of observing a score S equal to or greater than x is given by the equation

$$p(S \geq x) = 1 - \exp\left(-e^{-\lambda(x-\mu)}\right)$$

where

$$\mu = {}^{[\log(Km'n')]}\!/_{\lambda}$$

The statistical parameters λ and K are estimated by fitting the distribution of the un-gapped local alignment scores, of the query sequence and a lot of shuffled versions (Global or local shuffling) of a database sequence, to the Gumbel extreme value distribution. Note that λ and K depend upon the substitution matrix, gap penalties, and sequence composition (the letter frequencies). The m' and n' is the effective length of the query and database sequence, respectively. The original sequence length is shortened to the effective length to compensate for the edge effect (an alignment start near the end of one of the query or database sequence is likely not to have enough sequence to build an optimal alignment). They can be calculated as

$$m' \approx m - {}^{(\ln Kmn)}\!/_{H}$$

$$n' \approx n - {}^{(\ln Kmn)}\!/_{H}$$

where H is the average expected score per aligned pair of residues in an alignment of two random sequences. Altschul and Gish gave the typical values, $\lambda = 0.318$, $K = 0.13$, and $H = 0.40$, for un-gapped local alignment using BLOSUM62 as the substitution matrix. Using the typical values for assessing the significance is called the lookup table methods, and is not accurate. The expect

score E of a database match is the number of times that an unrelated database sequence would obtain a score S higher than x by chance. The expectation E obtained in a search for a database of D sequences is given by

$$E \approx 1 - e^{-p(s>x)D}$$

Furthermore, when $p < 0.1$, E could be approximated by the Poisson distribution as

$$E \approx pD$$

Note that the E value accessing the significance of the HSP score here (for un-gapped local alignment) is not identical to the one in the later step to evaluate the final gapped local alignment score, due to the variation of the statistical parameters.

10. **Make two or more HSP regions into a longer alignment.**

    Sometimes, we find two or more HSP regions in one database sequence that can be made into a longer alignment. This provides additional evidence of the relation between the query and database sequence. There are two methods, the Poisson method and the sum-of scores method, to compare the significance of the newly combined HSP regions. Suppose that there are two combined HSP regions with the sets of score (65, 40) and (52, 45), respectively. The Poisson method gives more significance to the set with the lower score of each set is higher (45>40). However, the sum-of-scores method prefers the first set, because 65+40 (105) is greater than 52+45(97). The original BLAST uses the Poisson method; gapped BLAST and the WU-BLAST use the sum-of scores method.

11. **Show the gapped Smith-Waterman local alignments of the query and each of the matched database sequences.**

    - The original BLAST only generates un-gapped alignments including the initially found HSPs individually, even when there is more than one HSP found in one database sequence.
    - BLAST2 versions produce a single alignment with gaps that can include all of the initially found HSP regions. Note that the computation of the score and its corresponding E score is involved with the adequate gap penalties.

12. **Report the matches whose expect score is lower than a threshold parameter E.**

Parallel BLAST

Parallel BLAST versions are implemented using MPI and Pthreads, and have been ported to various platforms including Windows, Linux, Solaris, Mac OS X, and AIX. Popular approaches to parallelize BLAST include query distribution, hash table segmentation, computation parallelization, and database segmentation (partition).

Program

The BLAST program can either be downloaded and run as a command-line utility "blastall" or accessed for free over the web. The BLAST web server, hosted by the NCBI, allows anyone with a web browser to perform similarity searches against constantly updated databases of proteins and DNA that include most of the newly sequenced organisms.

The BLAST program is based on an open-source format, giving everyone access to it and enabling them to have the ability to change the program code. This has led to the creation of several BLAST "spin-offs".

There are now a handful of different BLAST programs available, which can be used depending on what one is attempting to do and what they are working with. These different programs vary in query sequence input, the database being searched, and what is being compared. These programs and their details are listed below:

BLAST is actually a family of programs (all included in the blastall executable). These include:

Nucleotide-nucleotide BLAST (blastn)

This program, given a DNA query, returns the most similar DNA sequences from the DNA database that the user specifies.

Protein-protein BLAST (blastp)

This program, given a protein query, returns the most similar protein sequences from the protein database that the user specifies.

Position-Specific Iterative BLAST (PSI-BLAST)

This program is used to find distant relatives of a protein. First, a list of all closely related proteins is created. These proteins are combined into a general "profile" sequence, which summarises significant features present in these

sequences. A query against the protein database is then run using this profile, and a larger group of proteins is found. This larger group is used to construct another profile, and the process is repeated.

By including related proteins in the search, PSI-BLAST is much more sensitive in picking up distant evolutionary relationships than a standard protein-protein BLAST.

Nucleotide 6-frame translation-protein (blastx)

This program compares the six-frame conceptual translation products of a nucleotide query sequence (both strands) against a protein sequence database.

Nucleotide 6-frame translation-nucleotide 6-frame translation (tblastx)

This program is the slowest of the BLAST family. It translates the query nucleotide sequence in all six possible frames and compares it against the six-frame translations of a nucleotide sequence database. The purpose of tblastx is to find very distant relationships between nucleotide sequences.

Protein-nucleotide 6-frame translation (tblastn)

This program compares a protein query against the all six reading frames of a nucleotide sequence database.

Large numbers of query sequences (megablast)

When comparing large numbers of input sequences via the command-line BLAST, "megablast" is much faster than running BLAST multiple times. It concatenates many input sequences together to form a large sequence before searching the BLAST database, then post-analyze the search results to glean individual alignments and statistical values.

Of these programs, BLASTn and BLASTp are the most commonly used because they use direct comparisons, and do not require translations. However, since protein sequences are better conserved evolutionarily than nucleotide sequences, tBLASTn, tBLASTx, and BLASTx, produce more reliable and accurate results when dealing with coding DNA. They also enable one to be able to directly see the function of the protein sequence, since by translating the sequence of interest before searching often gives you annotated protein hits.

Alternative versions

An extremely fast but considerably less sensitive alternative to BLAST that compares nucleotide sequences to the genome is BLAT (**B**last **L**ike **A**lignment **T**ool). A version designed for comparing multiple large genomes or chromosomes is BLASTZ.

CS-BLAST (context-specific BLAST) is an extended version of BLAST for searching protein sequences that finds twice as many remotely related sequences as BLAST at the same speed and error rate. In CS-BLAST, the mutation probabilities between amino acids depend not only on the single amino acid, as in BLAST, but also on its local sequence context (the six left and six right sequence neighbors).

Accelerated versions

- There are two main field-programmable gate array (FPGA) implementations of the BLAST algorithm. Progeniq is up to 100x faster than a software implementation running on the same processor. TimeLogic offers a FPGA BLAST package called Tera-BLAST.
- The Mitrion-C Open Bio Project is an ongoing effort to port blast to run on Mitrion FPGAs. It is available on SourceForge.
- A CUDA GPU accelerated version of BLASTP that runs up to 10x faster than NCBI BLAST is now available CUDA-BLASTP

Uses of BLAST

BLAST can be used for several purposes. These include identifying species, locating domains, establishing phylogeny, DNA mapping, and comparison.

Identifying Species

With the use of BLAST, you can possibly correctly identify a species and/or find homologous species. This can be useful, for example, when you are working with a DNA sequence from an unknown species.

Locating Domains

When working with a protein sequence you can input it into BLAST, to locate known domains within the sequence of interest.

Establishing Phylogeny

Using the results received through BLAST you can create a phylogenetic tree using the BLAST web-page. It should be noted that phylogenies based on BLAST

alone are less reliable than other purpose-built computational phylogenetic methods, so should only be relied upon for "first pass" phylogenetic analyses.

DNA Mapping

When working with a known species, and looking to sequence a gene at an unknown location, BLAST can compare the chromosomal position of the sequence of interest, to relevant sequences in the database(s).

Comparison

When working with genes, BLAST can locate common genes in two related species, and can be used to map annotations from one organism to another.

Comparing BLAST and the Smith-Waterman Process

While both Smith-Waterman and BLAST are used to find homologous sequences by searching and comparing a query sequence with those in the databases, they do have their differences.

Due to the fact that BLAST is based on a heuristic algorithm, the results received through BLAST, in terms of the hits found, may not be the best possible results, as it will not provide you with all the hits within the database. BLAST misses hard to find matches.

A better alternative in order to find the best possible results would be to use the Smith-Waterman algorithm. This method varies from the BLAST method in two areas, accuracy and speed. The Smith-Waterman option provides better accuracy, in that it finds matches that BLAST cannot, because it does not miss any information. Therefore, it is necessary for remote homology. However, when compared to BLAST, it is more time consuming, not to mention that it requires large amounts of computer usage and space. Fortunately, technologies to speed up the Smith-Waterman process have been found to improve the time necessary to perform a search dramatically. These technologies include FPGA chips and SIMD technology.

In order to receive better results from BLAST, the settings can be changed from their default settings. However, there is no given or set way of changing these settings in order to receive the best results for a given sequence. The settings available for change are E-Value, gap costs, filters, word size, and substitution matrix. Note, that the algorithm used for BLAST was developed off the algorithm used for Smith-Waterman. BLAST

employs an alignment which finds "local alignments between sequences by finding short matches and from these initial matches (local) alignments are created".

Summary

The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between protein or nucleotide sequences. The program compares nucleotide or protein sequences to sequence in a database and calculates the statistical significance of the matches. This chapter first provides an introduction to BLAST and then describes the practical application of different BLAST programs based on the BLAST Quick Start mini-course (www.ncbi.nlm.nih.gov/Class/minicourses). In each example, emphasis is placed on practical step-by-step procedures, although relevant theory is also given where it affects the choice of BLAST program, parameters, and database.

Introduction

BLAST is an acronym for Basic Local Alignment Search Tool and refers to a suite of programs used to generate alignments between a nucleotide or protein sequence, referred to as a “query” and nucleotide or protein sequences within a database, referred to as “subject” sequences. The original BLAST program used a protein “query” sequence to scan a protein sequence database. A version operating on nucleotide query” sequences and a nucleotide sequence database soon followed. The introduction of an intermediate layer in which nucleotide sequences are translated into their corresponding protein sequences according to a specified genetic code allows cross-comparisons between nucleotide and protein sequences. Specialized variants of BLAST allow fast searches of nucleotide databases with very large query sequences, or the generation of alignments between a single pair of sequences. Both the standalone and web version of BLAST are available from the National Center for Biotechnology Information (www.ncbi.nlm.nih.gov). The web version provides searches of the complete genomes of Homo sapiens as well as those of many model organisms, including mouse, rat, fruit fly, and Arabidopsis thaliana, allowing BLAST alignments to be seen in a full genomic context.

Query and Database Sequence Formats

BLAST “query” sequences are given as character strings of single letter nucleotide or amino acid codes, preceded by a definition line, beginning with a “>”

symbol and containing identifiers and descriptive information. This format is known as FASTA. BLAST databases are constructed from concatenated FASTA formatted sequences using a program called "formatdb" that produces a mixture of binary- and ascii-encoded files containing the sequences and indexing information used during the BLAST search.

Scoring of Alignments and Substitution Matrices

A BLAST alignment consists of a pair of sequences, in which every letter in one sequence is paired with, or "aligned to," exactly one letter or a gap in the other. The alignment score is computed by assigning a value to each aligned pair of letters and then summing these values over the length of the alignment. For protein sequence alignments, scores for every possible amino acid letter pair are given in a "substitution matrix" where likely substitutions have positive values and unlikely substitutions have negative values. By default, BLAST uses the "blosum62" matrix, a member of the most commonly used series of substitution matrix , however, several members of the PAM series are also available. For nucleotide alignments, BLAST uses a reward of +2 for aligned pairs of identical letters and a penalty of −3 for each nonidentical aligned pair. The creation of a gap in an alignment results in a negative "gap-creation" penalty, with each extension of a preexisting gap incurring a lesser penalty. For a detailed treatment of the theory of alignment scoring.

Overview of the Algorithm

BLAST begins a search by indexing all character strings of a certain length within the "query" by their starting position in the query. The length of the string to index, called the "wordsize" is configurable by the user. The allowable range for the "wordsize" varies according to the BLAST program used; typical values are 3 for protein-to-protein sequence searches and 11 for nucleotide to nucleotide searches. BLAST then scans the database looking for matches between the "words" indexed in the "query" and strings found within the database sequences. For nucleotide-to-nucleotide searches, these matches must be exact; for protein-to-protein searches, the score of the match as determined using a substitution matrix, must exceed a specified threshold. When a word match is found, two nearby words in the case of protein searches, BLAST attempts to extend both forward and backward from the match to produce an alignment. BLAST will

continue this extension as long as the alignment score continues to increase or until it drops by a critical amount owing to the negative scores given by mismatches. This critical amount is known as the "dropoff." The methods BLAST uses to initiate refine alignments are given more fully in refs.

Statistical Significance

The alignments found by BLAST during a search are scored, as previously described, and assigned a statistical value, called the "Expect Value." The "Expect Value" is the number of times that an alignment as good or better than that found by BLAST would be expected to occur by chance, given the size of the database searched. An "Expect Value" threshold, set by the user, determines which alignments will be reported. A higher "Expect Value" threshold is less stringent and the BLAST default of "10" is designed to ensure that no biologically significant alignment is missed. However, "Expect Values" in the range of 0.001 to 0.0000001 are commonly used to restrict the alignments shown to those of high quality.

Course and Website

This BLAST Quickstart chapter illustrates the use of the principal BLAST programs to solve problems that arise in the analysis of protein and nucleotide sequences. Each section provides a succinct description of a protocol with two problems that serve as practical examples. Relevant theory is given when it affects the selection of a search strategy or search parameter, however, the emphasis is on the procedure itself. The sections follow closely the structure of the BLAST QuickStart Mini-Course found at www.ncbi.nlm.nih.gov/Class/minicourses. The BLAST QuickStart is one of 10 2-h format Mini-Courses offered by NCBI on campus at the National Institutes of Health and at locations around the country to over 4000 students a year. The courses use a paired problems approach in which the first of two similar problems or problem sets is solved by the instructor during the first hour on a computer linked to a projection system, while the students watch; in the second hour, the students tackle the second problem, or set of problems at their own computers. These courses have been effective as practical introductions to bioinformatics procedures. To get the most from the sections next, it will be necessary to navigate to the URL previously listed and click on the "BLAST

Quickstart" link to reach the online exercises, although the liberal collection of screen shots will allow the reader follow along for the most part without web access.

Nucleotide BLAST

Nucleotide BLAST refers to the use of a member of the BLAST suite of programs, such as "blastn" to search with a nucleotide "query" against a database of nucleotide "subject" sequences.

Available Nucleotide-Level Searches

There are two members of the BLAST suite of programs that are designed to make nucleotide-to-nucleotide alignments. The first is the original BLAST nucleotide search program known as "blastn." The "blastn" program is a general purpose nucleotide search and alignment program that is sensitive and can be used to align tRNA or rRNA sequences as well as mRNA or genomic DNA sequences containing a mix of coding and noncoding regions. A more recently developed nucleotide-level BLAST program called MegaBLAST is about 10 times faster than "blastn" but is designed to align sequences that are nearly identical, differing by only a few percent from one another. MegaBLAST allows the rapid mapping of a transcript onto a typical 3 billion base mammalian genome in seconds, and is useful for processing large batches of sequences. A refinement of MegaBLAST, known as discontiguous MegaBLAST, uses a discontiguous template to define an initial "word" in which characters in some positions, such as those in the wobble base position of codons, need not match. Discontiguous MegaBLAST allows rapid cross-species mappings involving coding regions in cases where species differences in codon usage would prevent alignments using the original MegaBLAST program.

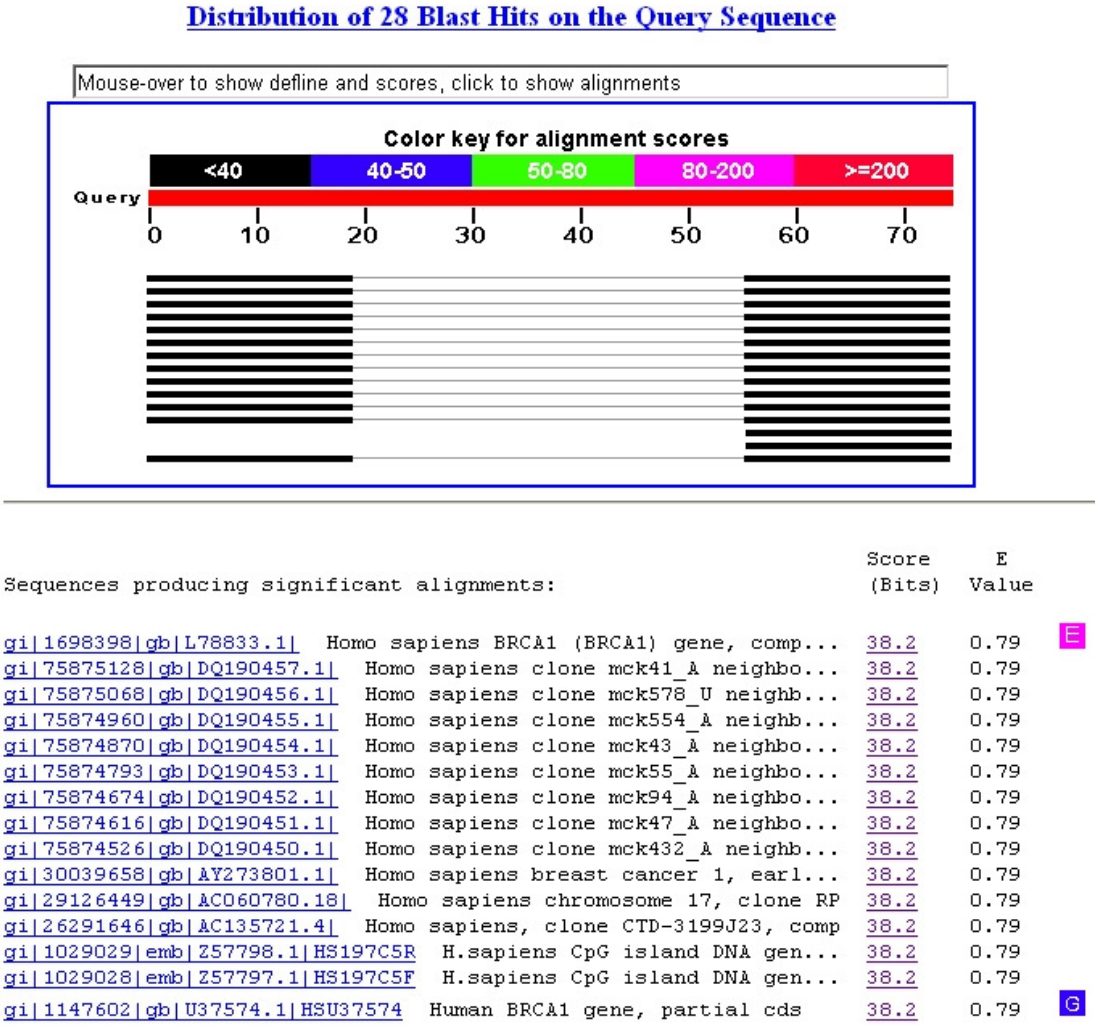
Examples of Nucleotide BLAST Searches

Problem 1

Click on the link indicated by "P" next to the "Nucleotide-nucleotide BLAST (blastn)" to access the problem. This problem demonstrates how to use BLAST to find human sequences in GenBank that can be amplified with a particular primer pair. Access the nucleotide–nucleotide BLAST page (by clicking on the Nucleotide–nucleotide BLAST link). Paste both the forward and reverse primers into the BLAST input box. Insert a string of about 30 N's after the first primer sequence to separate the two sequences to be found in separate, not overlapping alignments. Limit your search to

human sequences by selecting “Homo sapiens” from the “All organisms” pull down menu under the Options for advanced blasting and click the BLAST! link. Retrieve results by clicking on the “Format” button. Look for two hits to the same database sequence.

In this result, shown in Fig. 1, there are 13 GenBank entries that align to both the forward and reverse primers at different locations (indicated by thick bars) with a gap in between (indicated by a thin gray bar). There are two GenBank entries that align only to the reverse primer. One alignment of the primer pair to the GenBank entry L78833.1 is shown in Fig. 2. The forward primer aligns to the sequence L78833.1 on the forward strand (as indicated by Strand Plus/Plus) at nucleotides 3252..3270. The reverse primer aligns to the reverse strand (as indicated by Strand Plus/Minus) at nucleotides 3475..3457. Thus, the two primers will amplify the sequence from nucleotides 3252..3475 of the entry L78833.1. Retrieve the entry L78833.1 in Entrez, by clicking on it. The annotation shows that the amplified region covers the Exon 1a and the upstream sequence of the BRCA1 gene. Refer to the Note 1 for the multiple hits. You may perform similar search against the human genome BLAST database.



Sequences producing significant alignments:		Score (Bits)	E Value
gi|1698398|gb|L78833.1|	Homo sapiens BRCA1 (BRCA1) gene, comp...	38.2	0.79
gi|75875128|gb|DQ190457.1|	Homo sapiens clone mck41_A neighbo...	38.2	0.79
gi|75875068|gb|DQ190456.1|	Homo sapiens clone mck578_U neighb...	38.2	0.79
gi|75874960|gb|DQ190455.1|	Homo sapiens clone mck554_A neighb...	38.2	0.79
gi|75874870|gb|DQ190454.1|	Homo sapiens clone mck43_A neighbo...	38.2	0.79
gi|75874793|gb|DQ190453.1|	Homo sapiens clone mck55_A neighbo...	38.2	0.79
gi|75874674|gb|DQ190452.1|	Homo sapiens clone mck94_A neighbo...	38.2	0.79
gi|75874616|gb|DQ190451.1|	Homo sapiens clone mck47_A neighbo...	38.2	0.79
gi|75874526|gb|DQ190450.1|	Homo sapiens clone mck432_A neighb...	38.2	0.79
gi|30039658|gb|AY273801.1|	Homo sapiens breast cancer 1, earl...	38.2	0.79
gi|29126449|gb|AC060780.18|	Homo sapiens chromosome 17, clone RP	38.2	0.79
gi|26291646|gb|AC135721.4|	Homo sapiens, clone CTD-3199J23, comp	38.2	0.79
gi|1029029|emb|Z57798.1|HS197C5R	H.sapiens CpG island DNA gen...	38.2	0.79
gi|1029028|emb|Z57797.1|HS197C5F	H.sapiens CpG island DNA gen...	38.2	0.79
gi|1147602|gb|U37574.1|HSU37574	Human BRCA1 gene, partial cds	38.2	0.79

Fig. 1

Graphical overview of primer hits from the nucleotide–nucleotide search problem 1. The result shows that 13 GenBank entries align to the forward and reverse primers at different locations (indicated by thick bars) with a gap in between (indicated

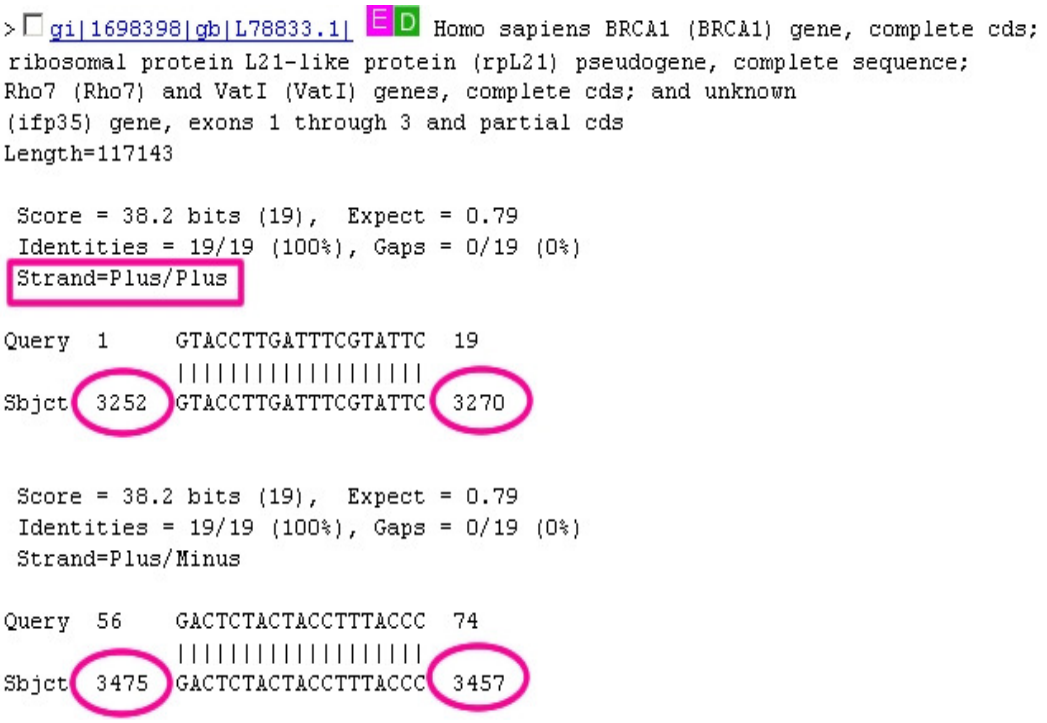


Fig. 2

Alignment view of one of the primer hits to the GenBank entry L78833.1. The forward primer (query nucleotides 1..19) aligns to the sequence L78833.1 on the forward strand (indicated by Strand Plus/Plus) at nucleotides 3252..3270. The reverse primer (query

Problem 2

Next to the Nucleotide–nucleotide BLAST (blastn) to access the problem. This problem describes how to obtain single-nucleotide polymorphism (SNP) information in similar sequences in the database. Hermankova et al. studied the HIV-1 drug resistance profiles in children and adults receiving combination drug therapy. To identify the SNPs in the HIV-1 isolates from these patients, or other similar sequences in the database, use the sequence from one of the patients given next and run a nucleotide–nucleotide BLAST search as described in the problem previously listed. Format the results using the “Flat Query with Identities” option from the “Alignment View” pull down menu under the “Format” options. Identify the SNP observed at alignment position 6 (query nucleotide number 10) in Fig. 3. There is an A/G SNP in many of thedatabase sequences.

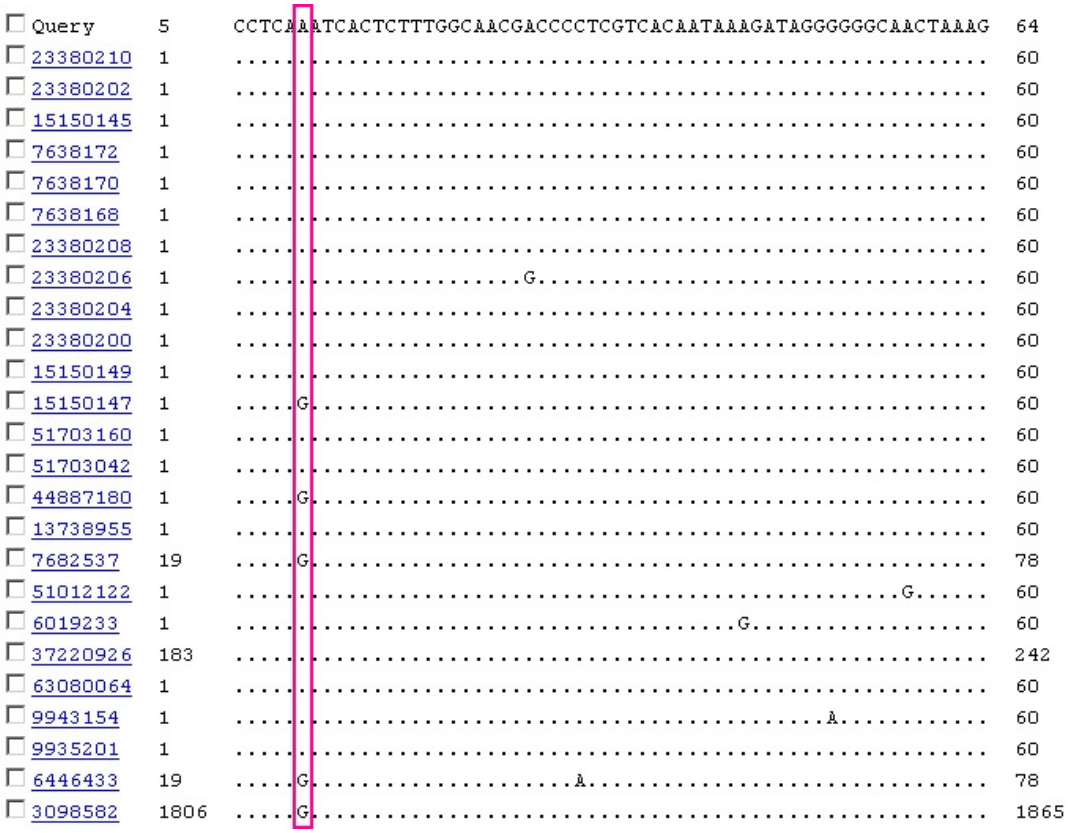


Fig. 3

Query-anchored alignment view for finding single-nucleotide polymorphism (SNP) from the nucleotide–nucleotide search problem 2. Nucleotides in the database sequences identical to the query HIV-1 sequence are indicated by dots and SNPs are indicated

Protein BLAST

Protein-to-protein sequence searches are performed using the original member of the BLAST suite of programs, known as “blastp.”

Available Protein-Level Searches

The default wordsize for a blastp search is three; the default substitution matrix is the blosum62 matrix. Changing the wordsize from three to two increases the sensitivity of the search. Using a different substitution matrix can also have an effect on search

sensitivity. During a "blastp" search, low-complexity regions of the query sequence are filtered to reduce the construction of spurious alignments and enhance search speed.

Examples of Protein BLAST Searches

Problem 1

Next to "Protein–protein BLAST (blastp)" to access the problem. It describes how to use blastp to determine the type of protein. For this purpose, we will choose the database containing the curated and annotated protein sequences, such as RefSeq or Swissprot. Use the query sequence provided in the problem. This sequence was generated by translating a *5 exon* gene from *Drosophila*. To determine the nature of this protein, run a blastp search. Access the "Protein–protein BLAST (blastp)" page by clicking on the link, paste in the query sequence, select the Swissprot database from the "Choose database" pull down menu and click on the BLAST! link. For each protein–protein search, the query is also searched against the Conserved Domain Database. Retrieve results by clicking on the "Format" button. The protein is similar to a number of aspartate amino transferases.

Translated BLAST

Translated BLAST searches use a genetic code to translate either the "query," database "subjects," or both, into protein sequences, which are then aligned as in "blastp." The translations are performed in the three forward as well as the three reverse reading frames so that no possible translation is missed.

Available Translated Searches

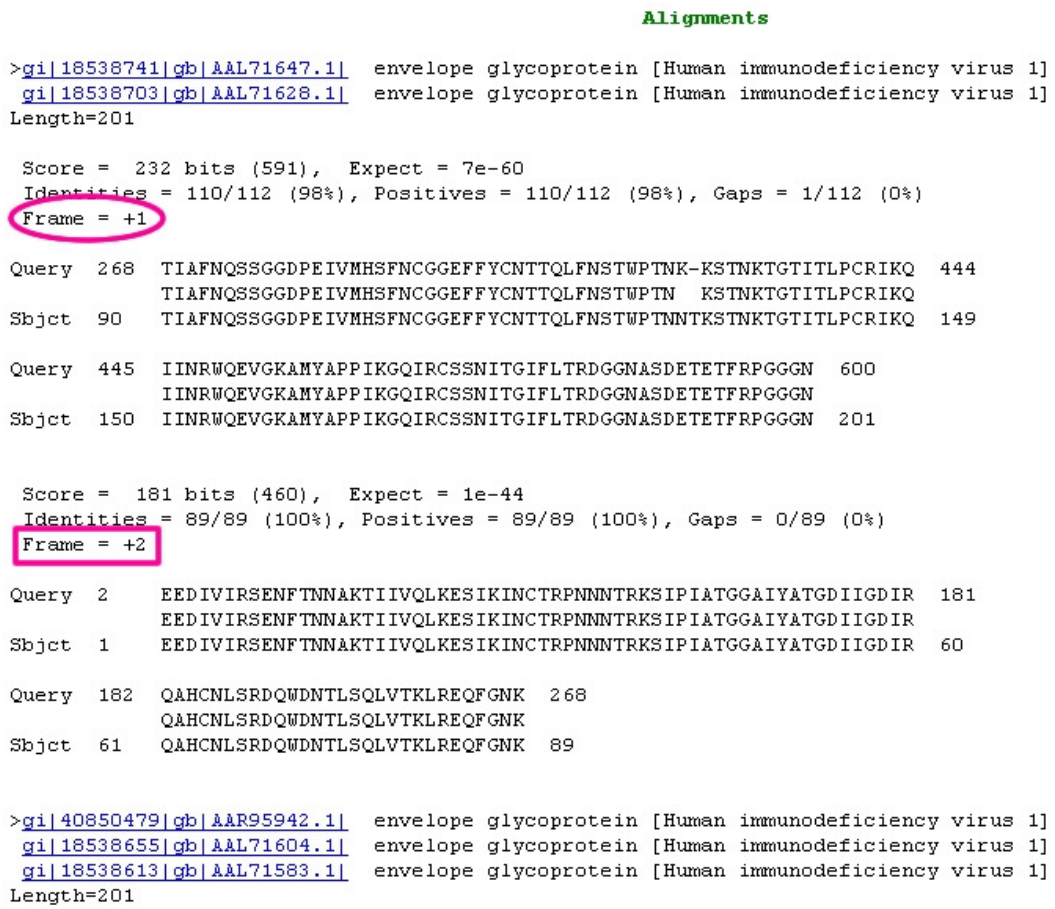
There are three varieties of translated BLAST search; "tblastn," "blastx," and "tblastx." In the first variant, "tblastn," a protein sequence query is compared to the six-frame translations of the sequences in a nucleotide database. In the second variant, "blastx," a nucleotide sequence query is translated in six reading frames, and the resulting six-protein sequences are compared, in turn, to those in a protein sequence database. In the third variant, "tblastx," both the "query" and database "subject" nucleotide sequences are translated in six reading frames, after which 36 (6 × 6) protein "blastp" comparisons are made. Protein sequences are better conserved than their corresponding nucleotide sequences. Because the translated searches make their comparisons at the level of protein sequences, they are more sensitive than direct nucleotide sequence searches. A common

use of the “tblastn” and “blastx” programs is to help annotate coding regions on a nucleotide sequence; they are also useful in detecting frame-shifts in these coding regions. The “tblastx” program provides a sensitive way to compare transcripts to genomic sequences without the knowledge of any protein translation, however, it is very computationally intensive. MegaBLAST can often achieve sufficient sensitivity at a much greater speed in searches between the sequences of closely related species and is preferred for batch analysis of short transcript sequences such as expressed sequence tags.

Examples of Translated BLAST Searches

Problem 1

Next to the “Translated query vs protein database (blastx)” to access the problem. This problem describes how to identify a frame shift in a nucleotide sequence by comparing its translated amino acid sequence to a similar protein in the database. Access the Blastx page by clicking on the link “Translated query vs protein database (blastx),” paste the nucleotide sequence provided in the problem in the query box and run the Blast search. The translation of the query sequence is similar to the sequences of envelope glycoproteins in the database. Compared to the similar proteins in the results, there appears to be a frame shift around nucleotide 268 as seen in Fig. 4. The query whentranslated in reading frame 2 (as indicated by a rectangle) up to nucleotide 268 is similar to only the first 89 amino acids of the database protein AAL71647.1. The translation of the query needs to be shifted to reading frame 1 (as indicated by an oval) to find similarity to the rest of the protein sequence.



```
                                                Alignments

>gi|18538741|gb|AAL71647.1|  envelope glycoprotein [Human immunodeficiency virus 1]
 gi|18538703|gb|AAL71628.1|  envelope glycoprotein [Human immunodeficiency virus 1]
Length=201

 Score =  232 bits (591),  Expect = 7e-60
 Identities = 110/112 (98%), Positives = 110/112 (98%), Gaps = 1/112 (0%)
 Frame = +1

Query  268  TIAFNQSSGGDPEIVMHSFNCGGEFFYCNTTQLFNSTWPTNK-KSTNKTGTITLPCRIKQ  444
            TIAFNQSSGGDPEIVMHSFNCGGEFFYCNTTQLFNSTWPTN  KSTNKTGTITLPCRIKQ
Sbjct  90   TIAFNQSSGGDPEIVMHSFNCGGEFFYCNTTQLFNSTWPTNNTKSTNKTGTITLPCRIKQ  149

Query  445  IINRWQEVGKAMYAPPIKGQIRCSSNITGIFLTRDGGNASDETETFRPGGGN  600
            IINRWQEVGKAMYAPPIKGQIRCSSNITGIFLTRDGGNASDETETFRPGGGN
Sbjct  150  IINRWQEVGKAMYAPPIKGQIRCSSNITGIFLTRDGGNASDETETFRPGGGN  201


 Score =  181 bits (460),  Expect = 1e-44
 Identities = 89/89 (100%), Positives = 89/89 (100%), Gaps = 0/89 (0%)
 Frame = +2

Query  2    EEDIVIRSENFTNNAKTIIVQLKESIKINCTRPNNNTRKSIPIATGGAIYATGDIIGDIR  181
            EEDIVIRSENFTNNAKTIIVQLKESIKINCTRPNNNTRKSIPIATGGAIYATGDIIGDIR
Sbjct  1    EEDIVIRSENFTNNAKTIIVQLKESIKINCTRPNNNTRKSIPIATGGAIYATGDIIGDIR  60

Query  182  QAHCNLSRDQWDNTLSQLVTKLREQFGNK  268
            QAHCNLSRDQWDNTLSQLVTKLREQFGNK
Sbjct  61   QAHCNLSRDQWDNTLSQLVTKLREQFGNK  89


>gi|40850479|gb|AAR95942.1|  envelope glycoprotein [Human immunodeficiency virus 1]
 gi|18538655|gb|AAL71604.1|  envelope glycoprotein [Human immunodeficiency virus 1]
 gi|18538613|gb|AAL71583.1|  envelope glycoprotein [Human immunodeficiency virus 1]
Length=201
```

Fig. 4

Detecting frame shifts using a translated search: Blastx problem 1. The query, when translated in reading frame 2 (highlighted by a rectangle) up to nucleotide 268, is similar to only the first 89 amino acids of the database protein AAL71647.1. The translation

Problem 2

Next to "Translated query vs protein database (blastx)" to access the problem. Paste in the sequence provided in the problem and run the blastx search to obtain a result similar to that shown in Fig. 5. The translation of the querysequence 1..564 in reading frame 1 (as indicated by an oval) to find similarity to the rest of the protein sequence. There is a frame shift in the query nucleotide around 564. To find out the nucleotide difference around 564.

```
>  gi|19568959|gb|AAL91985.1|  envelope glycoprotein [Human immunodeficiency virus type 1]
Length=514

 Score =  561 bits (1445),  Expect = 3e-158
 Identities = 301/306 (98%), Positives = 304/306 (99%), Gaps = 0/306 (0%)
 Frame = +2

Query  566   DIVPIDNDRNDSTSYRLLSCNTSVITQACPKVSFEPIPIHYCAPAGFAILKCNNKTFSGT  745
             DIVPIDNDRNDSTSYRLLSCNTSVITQACPKVSFEPIPIHYCAPAGFAILKCNNKTFSGT
Sbjct  185   DIVPIDNDRNDSTSYRLLSCNTSVITQACPKVSFEPIPIHYCAPAGFAILKCNNKTFSGT  244

Query  746   GPCTNVSTVQCTHGIRPIVSTQLLLNGSLAEEGIVIRYENITDNAKSIIIQLNETVQINC  925
             GPCTNVSTVQCTHGIRPIVSTQLLLNGSLAEEGIVIRYENITDNAKSII+QLNETVQINC
Sbjct  245   GPCTNVSTVQCTHGIRPIVSTQLLLNGSLAEEGIVIRYENITDNAKSIIVQLNETVQINC  304

Query  926   TRPNNNTRKSIPIGPGRAFYATGDIIGDIRKAYCNISGAKWNNTLKRIAYKLKEQFPNKT  1105
             TRPNNNTRKSIPIGPGRAFYATGDIIGDIRKAYCNISGAKWNNTLKRIAYKLKEQFPNKT
Sbjct  305   TRPNNNTRKSIPIGPGRAFYATGDIIGDIRKAYCNISGAKWNNTLKRIAYKLKEQFPNKT  364

Query  1106  IVFKPSSGGDPEIVMHSFNCRGEFFYCNTTKLFDSSWDXXXXXXXXXXXXXXXXSIILPC  1285
             IVFKPSSGGDPEIVMHSFNCRGEFFYCNTTKLFDSSWDNTNLNKTWNNTWNKNNSIILPC
Sbjct  365   IVFKPSSGGDPEIVMHSFNCRGEFFYCNTTKLFDSSWDNTNLNKTWNNTWNKNNSIILPC  424

Query  1286  RIKQIINMWQEVGKAMYAPPIEGPLYCLSNITGLILTRXXXXXXXXXXXXXXXXFRPGGG  1465
             RIKQIINMWQEVGKAMYAPPIEGP+YCLSNITGLILTRDGGNETDGNNT G ETFRPGGG
Sbjct  425   RIKQIINMWQEVGKAMYAPPIEGPIYCLSNITGLILTRDGGNETDGNNTSGKETFRPGGG  484

Query  1466  NMRDNW  1483
             +MRDNW
Sbjct  485   DMRDNW  490


 Score =  315 bits (806),  Expect = 4e-84
 Identities = 165/188 (87%), Positives = 173/188 (92%), Gaps = 4/188 (2%)
 Frame = +1

Query  1     MRVKEKYQHLWRWGTMLLGLLMIRSAADQLWVTVYYGVPVWKEATTTPFCASDAKAYDTE  180
             MRVKEKYQHLWRWGTMLLGLLMI SAADQLWVTVYYGVPVWKEATTT FCASDAKAYDTE
Sbjct  1     MRVKEKYQHLWRWGTMLLGLLMICSAADQLWVTVYYGVPVWKEATTTLFCASDAKAYDTE  60
```

Fig. 5

Detecting frame shifts using a translated search: blastx problem 2. The translation of the query sequence 1..564 in reading frame 1 (highlighted by a rectangle) is similar to the first 184 amino acids of the database protein AAL91985.1.

Genome BLAST

Genome BLAST refers to the application of any of the BLAST search programs to the complete genomic sequence of an organism or the transcript and protein sequences derived from its annotation.

Available Genome-Wide Searches

Genome BLAST services are available at NCBI for a variety of organisms including human, mouse, rat, fruit fly, and many others in a growing list. At a minimum, MegaBLAST and "blastn" searches against the complete genome are supported. These are usually offered in conjunction with "tblastn" searches against the genome, "blastp" and "blastx" searches against the proteins annotated on the genome and MegaBLAST, "blastn" and "tblastn" searches against collections of transcript sequences that have been mapped to the genome. Hits to the genome are displayed graphically within NCBI's MapViewer to show their genomic context.

Examples of Genome-Wide BLAST Searches

Problem 1

Next to mouse genome BLAST to access the problem. This problem describes how to use mouse genome blast to identify the Hoxb homologues encoded by the mouse genomic assembly sequence. Translated searches or protein–protein searches are more sensitive for identifying similarity in the coding regions than the nucleotide–nucleotide searches. Within the translated or protein–protein searches, tblastn will be more appropriate than blastx or blastp for this problem. Both latter programs will use protein databases consisting of already identified protein sequences whereas tblastn will be useful for identifying unannotated coding regions as well.

The sensitivity of tblastn as compared to the nucleotide–nucleotide search to identify a similarity to a coding region by running two searches: MegaBLAST the query mRNA sequence, NM_008268, against the mouse genomic sequence and tblastn the query protein sequence, NP_032294, against the mouse genomic sequence. Access the mouse genome BLAST page, by clicking on the "mouse" link under the Genomes panel. For the first search, paste the accession number NM_008268 into the query box, accept the default MegaBLAST option, and select the "genome (reference only)" as the database. The results, shown in Figs. 6 and 7, contain only four hits, two to the two Hoxb5 coding exons and one each to the Hoxb3 and Hoxd3 genes. Pay attention to the "Refer to Features in this part of subject sequence." Three of these hits, two to the Hoxb5 and one to the Hoxb3 genes, are on the Contig NT_096135.3 placed on chromosome 11. For the second search, paste the protein accession number NP_032294 into the mouse

genome search page, select “genome (reference only)” as the database and tblastn as the program. The result should appear similar to that shown in Fig. 8. This search gives several more hits than the earlier MegaBLAST search. Pay attention to the “Refer to Features in this part of subject sequence.” There is a complete hit to the homeobox B5 protein, shown in Fig. 9, and to the homeodomains of the other members of the homeobox B family, seen in Fig. 10 (corresponding to the amino acids 195..253 in the query), such as B6, B4, B3, B2, B13, and so on, onchromosome 11, homeobox A family members on chromosome 6, and homeobox C family members on chromosome 15 (refer to Note 8 for the locations of conserved domain).

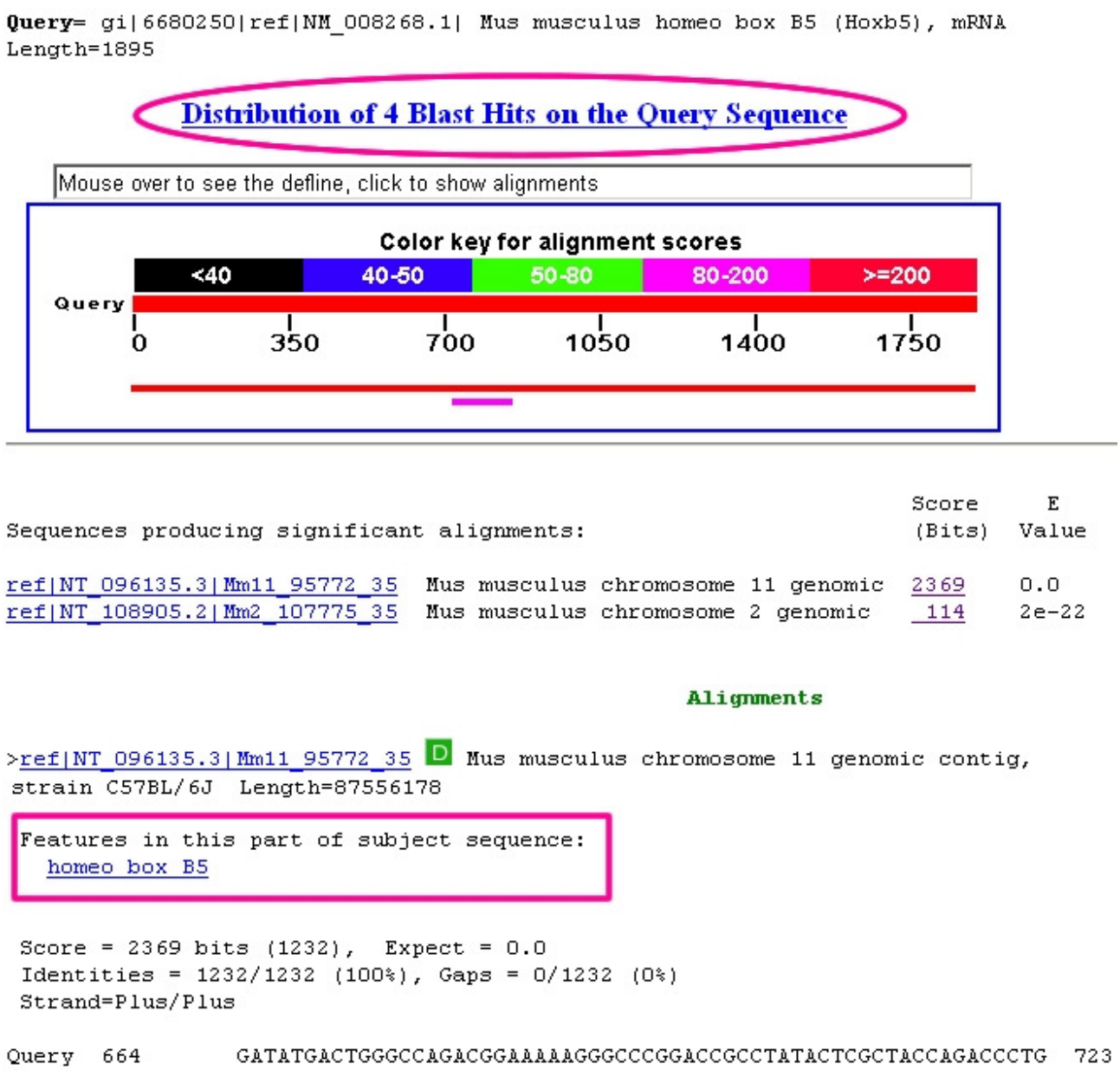
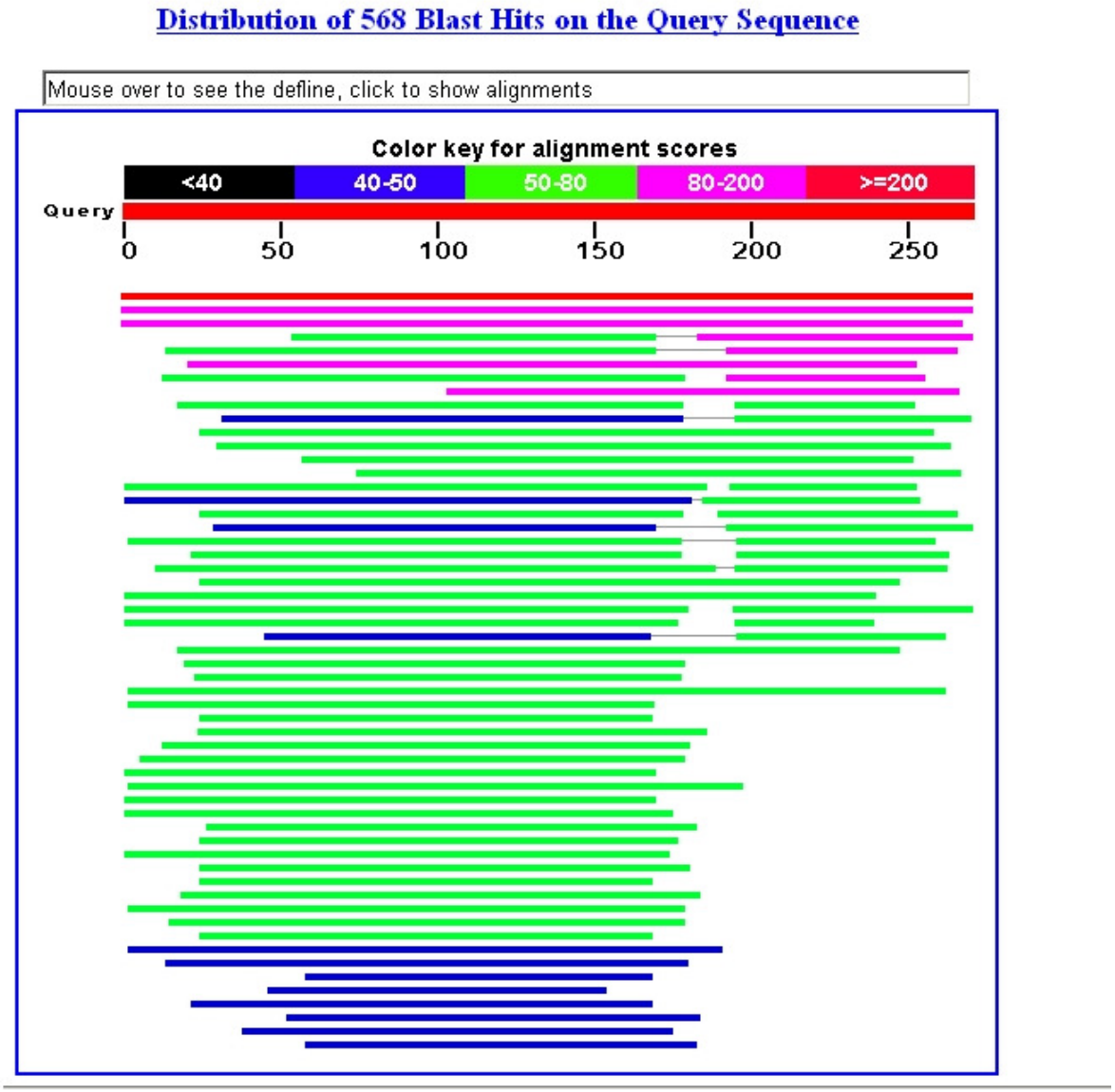


Fig. 6

Genomic MegaBLAST against the mouse genome problem 1: graphical overview. The mRNA query NM_008268.1 gets four hits to the mouse genome as highlighted by an oval. A part of the alignment view of the hit to the homeobox B5 coding region is also displayed

Fig. 7

Genomic MegaBLAST against the mouse genome problem 1: alignment view. Two of the BLAST hits, for the query NM_008268.1, shown in Fig. 6, are to the homeobox B3 and D3 coding regions (as highlighted by rectangles).



Sequences producing significant alignments:		Score (Bits)	E Value
ref|NT_096135.3|Mm11_95772_35	Mus musculus chromosome 11 genomic	376	7e-103
ref|NT_039343.5|Mm6_39383_35	Mus musculus chromosome 6 genomic c	159	2e-37
ref|NT_039621.5|Mm15_39661_35	Mus musculus chromosome 15 genomic	128	3e-28
ref|NT_108905.2|Mm2_107775_35	Mus musculus chromosome 2 genomic	125	2e-27
ref|NT_039324.5|Mm5_39364_35	Mus musculus chromosome 5 genomic c	92.8	2e-17
ref|NT_039299.5|Mm5_39339_35	Mus musculus chromosome 5 genomic c	87.8	5e-16
ref|NT_039306.5|Mm5_39346_35	Mus musculus chromosome 5 genomic c	85.1	3e-15

Fig. 8

Genomic tblastn against the mouse genome problem 1: graphical overview. The protein query, NP_032294, on performing tblastn against the mouse genome gets several more

hits than the MegaBLAST of the corresponding mRNA against the mouse genome as shown.

Fig. 9

Genomic tblastn against the mouse genome problem 1: alignment view. The protein query, NP_032294, on performing tblastn against the mouse genome, aligns completely to the two coding exons of the homeobox *B5* gene annotated on the mouse genome contig NT_096135.3

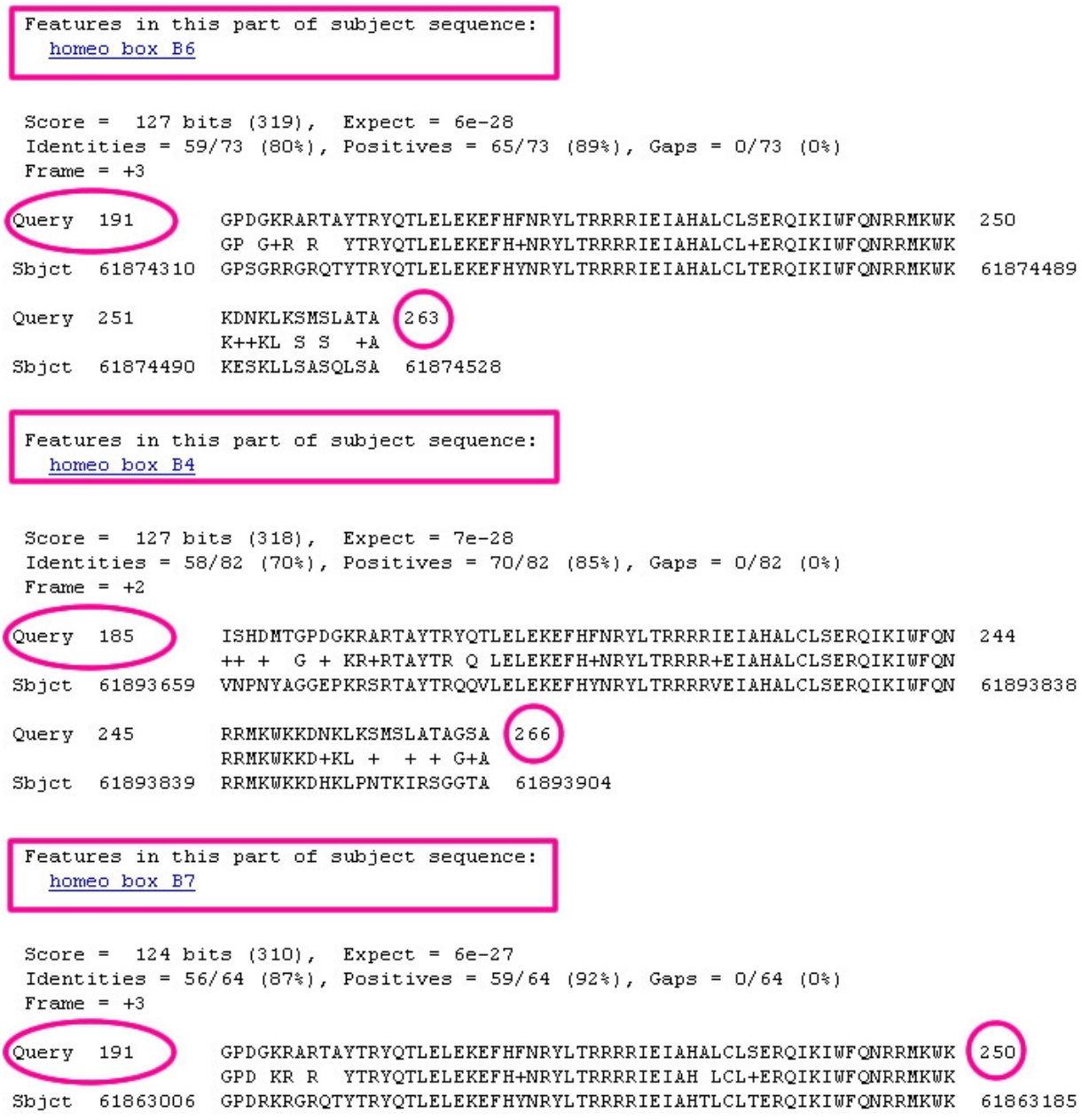


Fig. 10

Genomic tblastn against mouse genome problem 1: alignment view (continued). Some of the hits in the region of the conserved homeodomain (amino acid residues 195..253) of the query protein NP_032294 to other members of the homeobox protein family are displayed.

Problem 2

Next to mouse genome BLAST to access the problem. This problem describes how to use mouse genome blast to identify the protocadherin β homologues encoded by the mouse genomic sequence, tblastn will be useful for identifying unannotated homologues also. Access the mouse genome BLAST page, by clicking on the "mouse" link under Genomic BLAST. Paste the accession number for protocadehrin β 1 protein, AAK26059, in the query box, select "genome(reference only)" as the database and tblastn as the program. The result contains a complete hit to the protocadehrin β 1 protein and to other members of the protocadehrin β and γ subfamily A on chromosome 18

BLAST2 Sequences

BLAST2Sequences is used to compare two sequences, protein or nucleotide, using any one of the principal BLAST variants, "blastp," "blastn," "tblastn," "blastx," "tblastx," or MegaBLAST.

Comparisons Between Two Sequences

The output of BLAST2Sequences consists of a set of the traditional pairwise alignments generated by the principal BLAST programs it uses, supplemented with a dot plot representation of these alignments. The dot plot is useful for highlighting deletions and duplications of segments between two sequences. The translated variants of BLAST2Sequences are useful for the detection of exons.

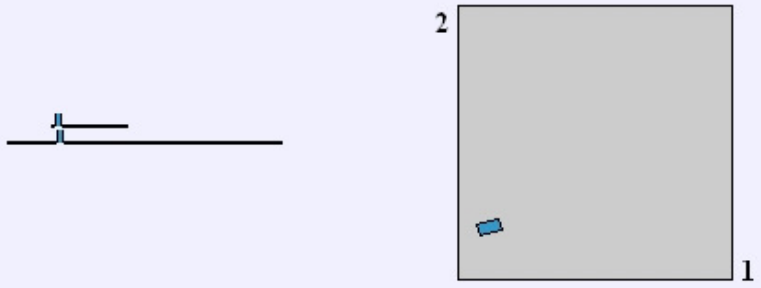
Examples of Two Sequence Comparisons

Problem 1

This problem describes the comparison of two nucleotide sequences. The problem provides a genomic sequence and an mRNA (cDNA) sequence. The genomic sequence is a piece from a GenBank HTG record that contains part of the Werner's syndrome gene *WRN*. This Gene contains 35 exons. The figure in the problem on the BLAST QuickStart website shows the mapping of exons to the cDNA coordinates. We will use BLAST2Sequences to determine which exon, if any, is contained in the supplied HTG sequence by comparing it against the *WRN* gene cDNA sequence. Access the BLAST2Sequences page by clicking on the link "Align two sequences (bl2seq)." Paste the HTG sequence in the top box under "Sequence1" and the cDNA sequence in the bottom box under "Sequence2." Click on the "Align" button to obtain the outputof Fig.

11. The query (genomic sequence) nucleotides 116..233 are similar to "Sbjct" (cDNA sequence) nucleotides 954..1071. Compare the cDNA coordinates to the exon coordinates provided in the problem. The cDNA coordinates 954..1071 correspond to the exon number 8. Thus, the provided genomic DNA contains exon 8 of the *WRN* gene.

Sequence 2: lcl|cDNAof WRNgene
Length = 5189 (1 .. 5189)



NOTE:Bitscore and expect value are calculated based on the size of the nr database.

NOTE:If protein translation is reversed, please repeat the search with reverse strand of the query sequence.

```
 Score =  227 bits (118),  Expect = 8e-56
 Identities = 118/118 (100%), Gaps = 0/118 (0%)
 Strand=Plus/Plus

Query  116   AGAGGAAGAAATCCTACTTAGCGACATGAACAAACAGTTGACTTCAATCTCTGAGGAAGT  175
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  954   AGAGGAAGAAATCCTACTTAGCGACATGAACAAACAGTTGACTTCAATCTCTGAGGAAGT  1013

Query  176   GATGGATCTGGCTAAGCATCTTCCTCATGCTTTCAGTAAATTGGAAAACCCACGGAGG  233
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  1014  GATGGATCTGGCTAAGCATCTTCCTCATGCTTTCAGTAAATTGGAAAACCCACGGAGG  1071



CPU time:     0.03 user secs.       0.01 sys. secs          0.04 total secs.

Lambda      K      H
    1.33    0.621     1.12
```

Fig. 11

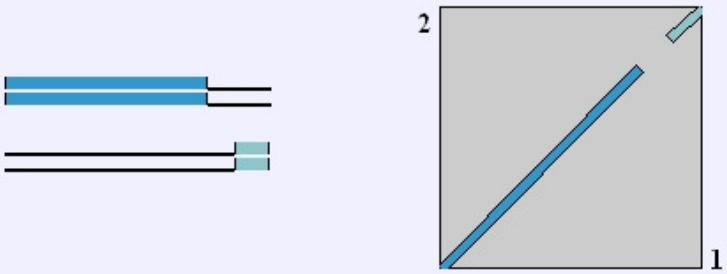
BLAST2Sequences problem 1 output: detecting an exon on unannotated genomic sequence. The query (genomic sequence) nucleotides 116..233 are similar to “Sbjct” (cDNA sequence) nucleotides 954..1071. Compare the cDNA coordinates to the exon.

Problem 2

This problem describes the importance of one of the BLAST parameters. The problem gives one DNA sequence. Paste the sequence in both the Sequence 1 and Sequence 2 windows in the BLAST2Sequences page, and click on Align to reach a display similar to that of Fig. 12. Why is the alignment broken into two parts? The sequence between the nucleotides 655..752 is missing in the alignment of the sequence to itself. This is because of the default Low complexity filter option. Unclick the “Filter” option and perform the search again. Now the query sequence aligns completely to itself (as it should). BLAST masked the nucleotides 655..752, missing in the Alignment View of Fig. 13, as it is a low complexity region.

Sequence 1: lcl|seq_1
Length = 863 (1 .. 863)

Sequence 2: lcl|seq_2
Length = 863 (1 .. 863)



NOTE:Bitscore and expect value are calculated based on the size of the nr database.

NOTE:If protein translation is reversed, please repeat the search with reverse strand of the query sequence.

```
 Score = 1258 bits (654),  Expect = 0.0
 Identities = 654/654 (100%), Gaps = 0/654 (0%)
 Strand=Plus/Plus

Query  1    AGCCCCCTCACCTCACTCCGCAGCCATACAGCCCCAGAGGCTCCCGATGGCGAGATTATG  60
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  1    AGCCCCCTCACCTCACTCCGCAGCCATACAGCCCCAGAGGCTCCCGATGGCGAGATTATG  60

Query  61   GTGCCTTGGCTATCATCATGGCAGGAATTGCATTTGGCTTTCACCAACTCTACAAGAGGT  120
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  61   GTGCCTTGGCTATCATCATGGCAGGAATTGCATTTGGCTTTCACCAACTCTACAAGAGGT  120
```

Fig. 12

BLAST2Sequences problem 2 output: the mystery of the missing piece. The alignment of the query sequence to itself is broken into two parts.

Fig. 13

BLAST2Sequences problem 2: alignment view at the junction. The sequence between the nucleotides 655..752 is missing in the alignment of the sequence to itself. The nucleotides at the junction of the two alignments are highlighted by rectangles.

FASTA

FASTA is a DNA and protein sequence alignment software package first described (as FASTP) by David J. Lipman and William R. Pearson in 1985.

History

The original FASTP program has designed for protein sequence similarity searching. FASTA added the ability to do DNA:DNA searches, translated protein:DNA

searches, and also provided a more sophisticated shuffling program for evaluating statistical significance. There are several programs in this package that allow the alignment of protein sequences and DNA sequences.

Usage

FASTA is pronounced "fast A", and stands for "FAST-All", because it works with any alphabet, an extension of "FAST-P" (protein) and "FAST-N" (nucleotide) alignment.

The current FASTA package contains programs for protein:protein, DNA:DNA, protein:translated DNA (with frameshifts), and ordered or unordered peptide searches. Recent versions of the FASTA package include special translated search algorithms that correctly handle frameshift errors (which six-frame-translated searches do not handle very well) when comparing nucleotide to protein sequence data.

In addition to rapid heuristic search methods, the FASTA package provides SSEARCH, an implementation of the optimal Smith-Waterman algorithm.

A major focus of the package is the calculation of accurate similarity statistics, so that biologists can judge whether an alignment is likely to have occurred by chance, or whether it can be used to infer homology. The FASTA package is available from fasta.bioch.virginia.edu.

The web-interface to submit sequences for running a search of the European Bioinformatics Institute (EBI)'s online databases is also available using the FASTA programs.

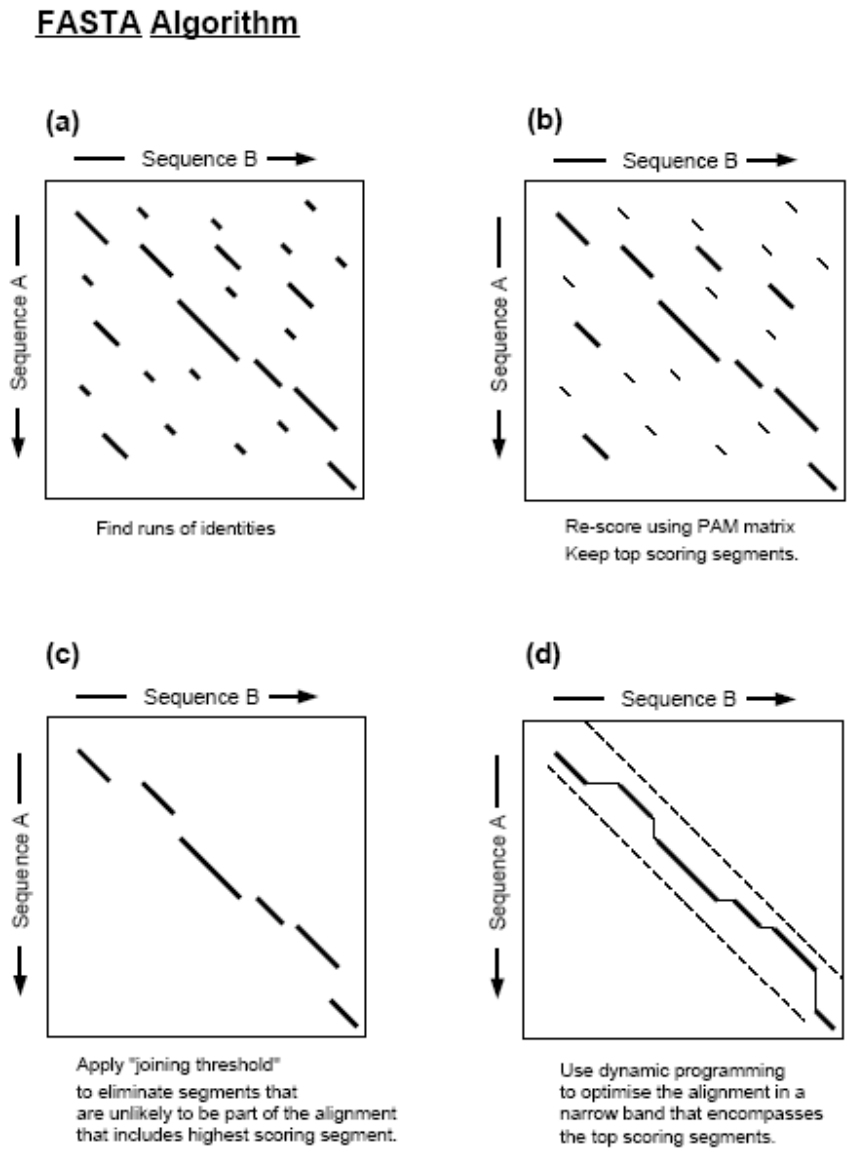
The FASTA file format used as input for this software is now largely used by other sequence database search tools (such as BLAST) and sequence alignment programs (Clustal, T-Coffee, etc).

Search method

FASTA takes a given nucleotide or amino-acid sequence and searches a corresponding sequence database by using local sequence alignment to find matches of similar database sequences.

The FASTA program follows a largely heuristic method which contributes to the high speed of its execution. It initially observes the pattern of word hits, word-to-word matches of a given length, and marks potential matches before performing a more time-consuming optimized search using a Smith-Waterman type of algorithm.

The size taken for a word, given by the parameter ktup, controls the sensitivity and speed of the program. Increasing the ktup value decreases number of background hits that are found. From the word hits that are returned the program looks for segments that contain a cluster of nearby hits. It then investigates these segments for a possible match.



There are some differences between fastn and fastp relating to the type of sequences used but both use four steps and calculate three scores to describe and format the sequence similarity results. These are:

- Identify regions of highest density in each sequence comparison. Taking a ktup to equal 1 or 2.

  In this step all or a group of the identities between two sequences are found using a look up table. The ktup value determines how many consecutive identities are required for a match to be declared. Thus the lesser the ktup value: the more sensitive the search. ktup=2 is frequently taken by users for protein sequences and ktup=4 or 6 for nucleotide sequences. Short oligonucleotides are usually run with ktup = 1. The program then finds all similar **local regions**, represented as diagonals of a certain length in a dot plot, between the two sequences by counting ktup matches and penalizing for intervening mismatches. This way, **local regions** of highest density matches in a diagonal are isolated from background hits. For protein sequences BLOSUM50 values are used for scoring ktup matches. This ensures that groups of identities with high similarity scores contribute more to the local diagonal score than to identities with low similarity scores. Nucleotide sequences use the identity matrix for the same purpose. The best 10 local regions selected from all the diagonals put together are then saved.
- Rescan the regions taken using the scoring matrices. trimming the ends of the region to include only those contributing to the highest score.

  Rescan the 10 regions taken. This time use the relevant scoring matrix while rescoring to allow runs of identities shorter than the ktup value. Also while rescoring conservative replacements that contribute to the similarity score are taken. Though protein sequences use the BLOSUM50 matrix, scoring matrices based on the minimum number of base changes required for a specific replacement, on identities alone, or on an alternative measure of similarity such as PAM, can also be used with the program. For each of the diagonal regions rescanned this way, a subregion with the maximum score is identified. The initial scores found in step1 are used to rank the library sequences. The highest score is referred to as *init1* score.
- In an alignment if several initial regions with scores greater than a CUTOFF value are found, check whether the trimmed initial regions can be joined to form an approximate alignment with gaps. Calculate a similarity score that is the sum of

the joined regions penalising for each gap 20 points. This initial similarity score (*initn*) is used to rank the library sequences. The score of the single best initial region found in step 2 is reported (*init1*).

Here the program calculates an optimal alignment of initial regions as a combination of compatible regions with maximal score. This optimal alignment of initial regions can be rapidly calculated using a dynamic programming algorithm. The resulting score initn is used to rank the library sequences.This joining process increases sensitivity but decreases selectivity. A carefully calculated cut-off value is thus used to control where this step is implemented, a value that is approximately one standard deviation above the average score expected from unrelated sequences in the library. A 200-residue query sequence with ktup2 uses a value 28.

- Use a banded Smith-Waterman algorithm to calculate an optimal score for alignment.

  This step uses a banded Smith-Waterman algorithm to create an optimised score (*opt*) for each alignment of query sequence to a database(library) sequence. It takes a band of 32 residues centered on the *init1* region of step2 for calculating the optimal alignment. After all sequences are searched the program plots the initial scores of each database sequence in a histogram, and calculates the statistical significance of the "opt" score. For protein sequences, the final alignment is produced using a full Smith-Waterman alignment. For DNA sequences, a banded alignment is provided.

MACHINE LEARNING

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables. A major focus of machine learning research is to

automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data). Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases.

Definition

Tom M. Mitchell provided a widely quoted definition: A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Generalization

The core objective of a learner is to generalize from its experience. The training examples from its experience come from some generally unknown probability distribution and the learner has to extract from them something more general, something about that distribution, that allows it to produce useful answers in new cases.

Human interaction

Some machine learning systems attempt to eliminate the need for human intuition in data analysis, while others adopt a collaborative approach between human and machine. Human intuition cannot, however, be entirely eliminated, since the system's designer must specify how the data is to be represented and what mechanisms will be used to search for a characterization of the data.

Algorithm types

Machine learning algorithms are organized into a taxonomy, based on the desired outcome of the algorithm.

- **Supervised learning** generates a function that maps inputs to desired outputs (also called **labels**, because they are often provided by human experts labeling the training examples). For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- **Unsupervised learning** models a set of inputs, like clustering.
- **Semi-supervised learning** combines both labeled and unlabeled examples to generate an appropriate function or classifier.

- **Reinforcement learning** learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.
- **Transduction** tries to predict new outputs based on training inputs, training outputs, and test inputs.
- **Learning to learn** learns its own inductive bias based on previous experience.

Theory

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield absolute guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

There are many similarities between machine learning theory and statistics, although they use different terms.

Approaches

Decision tree learning

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value.

Association rule learning

Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial neural networks

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is inspired by the structure and/or functional aspects of biological neural networks. Computations are structured in terms of an interconnected

group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

Genetic programming

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task. It is a specialization of genetic algorithms (GA) where each individual is a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task.

Inductive logic programming

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program which entails all the positive and none of the negative examples.

Support vector machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Clustering

Cluster analysis or clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Bayesian networks

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

Reinforcement learning

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Representation learning

Several learning algorithms, mostly unsupervised learning algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include principal components analysis and clustering. Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing to reconstruct the inputs coming from the unknown data generating distribution, while not being necessarily faithful for input configurations that are implausible under that distribution. Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse (has many zeros). Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an ideal representation is one that disentangles the underlying factors of variation that explain the observed data.

Applications

Applications for machine learning include:

- machine perception
- computer vision
- natural language processing
- syntactic pattern recognition
- search engines
- medical diagnosis
- bioinformatics
- brain-machine interfaces
- cheminformatics
- Detecting credit card fraud
- stock market analysis
- Classifying DNA sequences
- speech and handwriting recognition
- object recognition in computer vision
- game playing
- software engineering
- adaptive websites
- robot locomotion
- structural health monitoring.

In 2006, the on-line movie company Netflix held the first "Netflix Prize" competition to find a program to better predict user preferences and beat its existing Netflix movie recommendation system by at least 10%. The AT&T Research Team BellKor beat out several other teams with their machine learning program "Pragmatic Chaos". After winning several minor prizes, it won the grand prize competition in 2009 for $1 million.

The term **neural network** was traditionally used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus the term has two distinct usages:

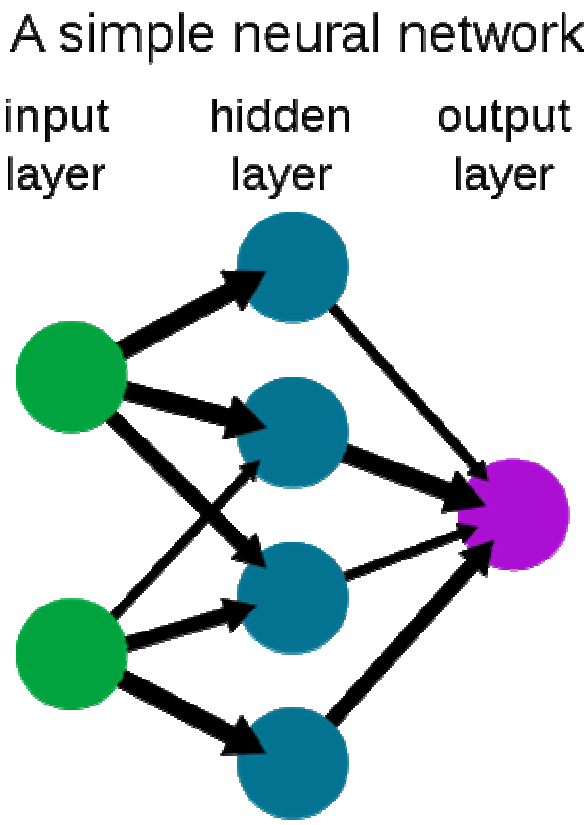
1. Biological neural networks are made up of real biological neurons that are connected or functionally related in the peripheral nervous system or the central

nervous system. In the field of neuroscience, they are often identified as groups of neurons that perform a specific physiological function in laboratory analysis.

2. Artificial neural networks are composed of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of a real biological system. The real, biological nervous system is highly complex: artificial neural network algorithms attempt to abstract this complexity and focus on what may hypothetically matter most from an information processing point of view. Good performance (e.g. as measured by good predictive ability, low generalization error), or performance mimicking animal or human error patterns, can then be used as one source of evidence towards supporting the hypothesis that the abstraction really captured something important from the point of view of information processing in the brain. Another incentive for these abstractions is to reduce the amount of computation required to simulate artificial neural networks, so as to allow one to experiment with larger networks and train them on larger data sets.

NEURAL NETWORKS

This topic focuses on the relationship between the two concepts; for detailed coverage of the two different concepts refer to the separate articles: Biological neural network and Artificial neural network.



Overview

A biological neural network is composed of a group or groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic microcircuits and other connections are possible. Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion, which have an effect on electrical signaling. As such, neural networks are extremely complex.

Artificial intelligence and cognitive modeling try to simulate some properties of biological neural networks. While similar in their techniques, the former has the aim of solving particular tasks, while the latter aims to build mathematical models of biological neural systems.

In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to

construct software agents (in computer and video games) or autonomous robots. Most of the currently employed artificial neural networks for artificial intelligence are based on statistical estimation, optimization and control theory.

The cognitive modelling field involves the physical or mathematical modeling of the behaviour of neural systems; ranging from the individual neural level (e.g. modelling the spike response curves of neurons to a stimulus), through the neural cluster level (e.g. modelling the release and effects of dopamine in the basal ganglia) to the complete organism (e.g. behavioural modelling of the organism's response to stimuli). Artificial intelligence, cognitive modelling, and neural networks are information processing paradigms inspired by the way biological neural systems process data.

History of the neural network analogy

In the brain, spontaneous order arises out of decentralized networks of simple units (neurons). In the late 1940s Donald Hebb made one of the first hypotheses of learning with a mechanism of neural plasticity called Hebbian learning. Hebbian learning is considered to be a 'typical' unsupervised learning rule and its later variants were early models for long term potentiation. These ideas started being applied to computational models in 1948 with Turing's B-type machines and the perceptron.

The perceptron is essentially a linear classifier for classifying data $x \in \mathbb{R}^n$ specified by parameters $w \in \mathbb{R}^n, b \in \mathbb{R}$ and an output function f = w'x + b. Its parameters are adapted with an ad-hoc rule similar to stochastic steepest gradient descent. Because the inner product is a linear operator in the input space, the perceptron can only perfectly classify a set of data for which different classes are linearly separable in the input space, while it often fails completely for non-separable data. While the development of the algorithm initially generated some enthusiasm, partly because of its apparent relation to biological mechanisms, the later discovery of this inadequacy caused such models to be abandoned until the introduction of non-linear models into the field.

The cognitron (1975) designed by Kunihiko Fukushima was an early multilayered neural network with a training algorithm. The actual structure of the network and the methods used to set the interconnection weights change from one neural strategy to another, each with its advantages and disadvantages. Networks can propagate information in one direction only, or they can bounce back and forth until self-activation at a node occurs

and the network settles on a final state. The ability for bi-directional flow of inputs between neurons/nodes was produced with the Hopfield's network (1982), and specialization of these node layers for specific purposes was introduced through the first hybrid network.

The parallel distributed processing of the mid-1980s became popular under the name connectionism.

The rediscovery of the backpropagation algorithm was probably the main reason behind the repopularisation of neural networks after the publication of "Learning Internal Representations by Error Propagation" in 1986 (Though backpropagation itself dates from 1969). The original network utilized multiple layers of weight-sum units of the type $f = g(w'x + b)$, where g was a sigmoid function or logistic function such as used in logistic regression. Training was done by a form of stochastic gradient descent. The employment of the chain rule of differentiation in deriving the appropriate parameter updates results in an algorithm that seems to 'backpropagate errors', hence the nomenclature. However it is essentially a form of gradient descent. Determining the optimal parameters in a model of this type is not trivial, and local numerical optimization methods such as gradient descent can be sensitive to initialization because of the presence of local minima of the training criterion. In recent times, networks with the same architecture as the backpropagation network are referred to as multilayer perceptrons. This name does not impose any limitations on the type of algorithm used for learning.

The backpropagation network generated much enthusiasm at the time and there was much controversy about whether such learning could be implemented in the brain or not, partly because a mechanism for reverse signaling was not obvious at the time, but most importantly because there was no plausible source for the 'teaching' or 'target' signal. However, since 2006, several unsupervised learning procedures have been proposed for neural networks with one or more layers, using so-called deep learning algorithms. These algorithms can be used to learn intermediate representations, with or without a target signal, that capture the salient features of the distribution of sensory signals arriving at each layer of the neural network.

The brain, neural networks and computers

Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain, even though the relation between this model and brain biological architecture is debated, as little is known about how the brain actually works.

A subject of current research in theoretical neuroscience is the question surrounding the degree of complexity and the properties that individual neural elements should have to reproduce something resembling animal intelligence.

Historically, computers evolved from the von Neumann architecture, which is based on sequential processing and execution of explicit instructions. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems, which may rely largely on parallel processing as well as implicit instructions based on recognition of patterns of 'sensory' input from external sources. In other words, at its very heart a neural network is a complex statistical processor (as opposed to being tasked to sequentially process and execute).

Neural coding is concerned with how sensory and other information is represented in the brain by neurons. The main goal of studying neural coding is to characterize the relationship between the stimulus and the individual or ensemble neuronal responses and the relationship among electrical activity of the neurons in the ensemble. It is thought that neurons can encode both digital and analog information.

Neural networks and artificial intelligence

A neural network (NN), in the case of artificial neurons called artificial neural network (ANN) or simulated neural network (SNN), is an interconnected group of natural or artificial neurons that uses a mathematical or computational model for information processing based on a connectionistic approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms neural networks are non-linear statistical data modeling or decision making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

However, the paradigm of neural networks - i.e., implicit, not explicit , learning is stressed - seems more to correspond to some kind of natural intelligence than to the traditional symbol-based Artificial Intelligence, which would stress, instead, rule-based learning.

Background

An artificial neural network involves a network of simple processing elements (artificial neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. Artificial neurons were first proposed in 1943 by Warren McCulloch, a neurophysiologist, and Walter Pitts, an MIT logician.[2] One classical type of artificial neural network is the recurrent Hopfield net.

In a neural network model simple nodes, which can be called variously "neurons", "neurodes", "Processing Elements" (PE) or "units", are connected together to form a network of nodes — hence the term "neural network". While a neural network does not have to be adaptive per se, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow.

In modern software implementations of artificial neural networks the approach inspired by biology has more or less been abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks, or parts of neural networks (such as artificial neurons), are used as components in larger systems that combine both adaptive and non-adaptive elements.

The concept of a neural network appears to have first been proposed by Alan Turing in his 1948 paper "Intelligent Machinery".

Applications of natural and of artificial neural networks

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations and also to use it. Unsupervised neural networks can also be used to learn representations of the input that capture the salient characteristics of the input distribution, e.g., see the Boltzmann machine (1983), and more recently, deep learning algorithms, which can implicitly learn the distribution function of the observed data. Learning in neural networks is particularly useful in applications where the complexity of the data or task makes the design of such functions by hand impractical.

The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind signal separation and compression.

Application areas of ANNs include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition, etc.), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

Neural networks and neuroscience

Theoretical and computational neuroscience is the field concerned with the theoretical analysis and computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behaviour, the field is closely related to cognitive and behavioural modeling.

The aim of the field is to create models of biological neural systems in order to understand how biological systems work. To gain this understanding, neuroscientists strive to make a link between observed biological processes (data), biologically plausible mechanisms for neural processing and learning (biological neural network models) and theory (statistical learning theory and information theory).

Types of models

Many models are used in the field, each defined at a different level of abstraction and trying to model different aspects of neural systems. They range from models of the short-term behaviour of individual neurons, through models of how the dynamics of neural circuitry arise from interactions between individual neurons, to models of how behaviour can arise from abstract neural modules that represent complete subsystems. These include

models of the long-term and short-term plasticity of neural systems and its relation to learning and memory, from the individual neuron to the system level.

Current research

While initially research had been concerned mostly with the electrical characteristics of neurons, a particularly important part of the investigation in recent years has been the exploration of the role of neuromodulators such as dopamine, acetylcholine, and serotonin on behaviour and learning.

Biophysical models, such as BCM theory, have been important in understanding mechanisms for synaptic plasticity, and have had applications in both computer science and neuroscience. Research is ongoing in understanding the computational algorithms used in the brain, with some recent biological evidence for radial basis networks and neural backpropagation as mechanisms for processing data.

Computational devices have been created in CMOS for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these efforts could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.

Criticism

A common criticism of neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. This is not surprising, since any learning machine needs sufficient representative examples in order to capture the underlying structure that allows it to generalize to new cases. Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become overtrained (if, for example, it is presented with a series of right turns – it should not learn to always turn right). These issues are common in neural networks that

must decide from amongst a wide variety of responses, but can be dealt with in several ways, for example by randomly shuffling the training examples, by using a numerical optimization algorithm that does not take too large steps when changing the network connections following an example, or by grouping examples in so-called mini-batches.

A. K. Dewdney, a former Scientific American columnist, wrote in 1997, "Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised anyone takes them seriously as a general problem-solving tool." (Dewdney, p. 82)

Arguments for Dewdney's position are that to implement large and effective software neural networks, much processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on Von Neumann technology may compel a NN designer to fill many millions of database rows for its connections - which can lead to abusive RAM and HD necessities. Furthermore, the designer of NN systems will often need to simulate the transmission of signals through many of these connections and their associated neurons - which must often be matched with incredible amounts of CPU processing power and time. While neural networks often yield effective programs, they too often do so at the cost of time and money efficiency.

Arguments against Dewdney's position are that neural nets have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft to detecting credit card fraud.

Technology writer Roger Bridgman commented on Dewdney's statements about neural nets:

Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn't?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be "an opaque, unreadable table...valueless as a scientific resource". In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having.

In response to this kind of criticism, one should note that although it is true that analyzing what has been learned by an artificial neural network is difficult, it is much easier to do so than to analyze what has been learned by a biological neural network. Furthermore, researchers involved in exploring learning algorithms for neural networks are gradually uncovering generic principles which allow a learning machine to be successful. For example, see the discussions regarding local vs non-local learning, as well as shallow vs deep architecture, in an article by Bengio and LeCun (2007).

Some other criticisms came from believers of hybrid models (combining neural networks and symbolic approaches). They advocate the intermix of these two approaches and believe that hybrid models can better capture the mechanisms of the human mind (Sun and Bookman 1990).

HIDDEN MARKOV MODELS

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. An HMM can be considered as the simplest dynamic Bayesian network.

In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; even if the model parameters are known exactly, the model is still 'hidden'.

Hidden Markov models are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

Description in terms of urns

In its discrete form, a hidden Markov process can be visualized as a generalization of the familiar Urn problem. For instance, from Rabiner 1989: A genie is in a room that is not visible to the researcher. It is drawing balls labeled y1, y2, y3, ... from the urns X1, X2, X3, ... in that room and putting the balls on a conveyor belt, where the researcher can observe the sequence of the balls but not the sequence of urns from which they were chosen. The genie has some procedure to choose urns; the choice of the urn for the n-th ball depends upon only a random number and the choice of the urn for the (n – 1)-th ball. Because the choice of urn does not directly depend on the urns further previous, this is called a Markov process. It can be described by the upper part of the diagram at the top of this article.

Because the Markov process itself cannot be observed, and only the sequence of labeled balls can be observed, this arrangement is called a "hidden Markov process". This is illustrated by the lower part of the diagram above, where one can see that balls y1, y2, y3, y4 can be drawn at each state. Even if the researcher knows the composition of the urns and has just observed a sequence of three balls, e.g. y1, y1 and y1 on the conveyor belt, the researcher still cannot be sure from which urn (i.e., at which state) the genie has drawn the third ball. However, the researcher can work out other details, such as the identity of the urn the genie is most likely to have drawn the third ball from.

Architecture of a hidden Markov model

The diagram below shows the general architecture of an instantiated HMM. Each oval shape represents a random variable that can adopt any of a number of values. The random variable x(t) is the hidden state at time t (with the model from the above diagram, $x(t) \in \{ x_1, x_2, x_3 \}$). The random variable y(t) is the observation at time t ($y(t) \in \{ y_1, y_2, y_3, y_4 \}$). The arrows in the diagram (often called a trellis diagram) denote conditional dependencies.

From the diagram, it is clear that the conditional probability distribution of the hidden variable x(t) at time t, given the values of the hidden variable x at all times, depends only on the value of the hidden variable x(t – 1): the values at time t – 2 and before have no influence. This is called the Markov property. Similarly, the value of the

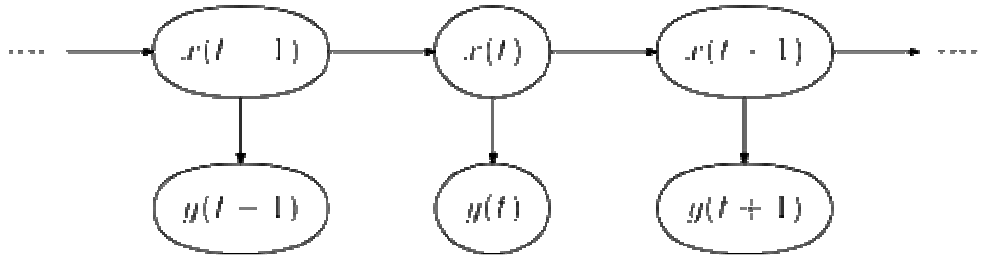
observed variable y(t) only depends on the value of the hidden variable x(t) (both at time t).

In the standard type of hidden Markov model considered here, the state space of the hidden variables is discrete, while the observations themselves can either be discrete (typically generated from a categorical distribution) or continuous (typically from a Gaussian distribution). The parameters of a hidden Markov model are of two types, transition probabilities and emission probabilities (also known as output probabilities). The transition probabilities control the way the hidden state at time t is chosen given the hidden state at time t − 1.

The hidden state space is assumed to consist of one of N possible values, modeled as a categorical distribution. (See the section below on extensions for other possibilities.) This means that for each of the N possible states that a hidden variable at time t can be in, there is a transition probability from this state to each of the N possible states of the hidden variable at time t + 1, for a total of N^2 transition probabilities. (Note, however, that the set of transition probabilities for transitions from any given state must sum to 1, meaning that any one transition probability can be determined once the others are known, leaving a total of N(N − 1) transition parameters.)

In addition, for each of the N possible states, there is a set of emission probabilities governing the distribution of the observed variable at a particular time given the state of the hidden variable at that time. The size of this set depends on the nature of the observed variable. For example, if the observed variable is discrete with M possible values, governed by a categorical distribution, there will be M − 1 separate parameters, for a total of N(M − 1) emission parameters over all hidden states. On the other hand, if the observed variable is an M-dimensional vector distributed according to an arbitrary multivariate Gaussian distribution, there will be M parameters controlling the means and M(M + 1) / 2 parameters controlling the covariance matrix, for a total of $N(M + \frac{M(M+1)}{2}) = NM(M+3)/2 = O(NM^2)$ emission parameters. (In such a case, unless the value of M is small, it may be more practical to restrict the nature of the covariances between individual elements of the observation vector, e.g. by

assuming that the elements are independent of each other, or less restrictively, are independent of all but a fixed number of adjacent elements.)



Mathematical description of a hidden Markov model

General description

A basic, non-Bayesian hidden Markov model can be described as follows:

$$
\begin{array}{lcl}
N & = & \text{number of states} \\
T & = & \text{number of observations} \\
\theta_{i=1\ldots N} & = & \text{emission parameter for an observation associated with state } i \\
\phi_{i=1\ldots N, j=1\ldots N} & = & \text{probability of transition from state } i \text{ to state } j \\
\boldsymbol{\phi}_{i=1\ldots N} & = & N\text{-dimensional vector, composed of } \phi_{i,1\ldots N}\text{; must sum to 1} \\
x_{t=1\ldots T} & = & \text{state of observation at time } t \\
y_{t=1\ldots T} & = & \text{observation at time } t \\
F(y|\theta) & = & \text{probability distribution of an observation, parametrized on } \theta \\
x_{t=2\ldots T} & \sim & \text{Categorical}(\boldsymbol{\phi}_{x_{t-1}}) \\
y_{t=1\ldots T} & \sim & F(\theta_{x_t})
\end{array}
$$

Note that, in the above model (and also the one below), the prior distribution of the initial state x_1 is not specified. Typical learning models correspond to assuming a discrete uniform distribution over possible states (i.e. no particular prior distribution is assumed).

In a Bayesian setting, all parameters are associated with random variables, as follows:

N, T = as above

$\theta_{i=1...N}, \phi_{i=1...N, j=1...N}, \boldsymbol{\phi}_{i=1...N}$ = as above

$x_{t=1...T}, y_{t=1...T}, F(y|\theta)$ = as above

α = shared hyperparameter for emission parameters

β = shared hyperparameter for transition parameters

$H(\theta|\alpha)$ = prior probability distribution of emission parameters, parametrized on α

$\theta_{i=1...N} \sim H(\alpha)$

$\boldsymbol{\phi}_{i=1...N} \sim \text{Symmetric-Dirichlet}_N(\beta)$

$x_{t=2...T} \sim \text{Categorical}(\boldsymbol{\phi}_{x_{i-t}})$

$y_{t=1...T} \sim F(\theta_{x_t})$

These characterizations use F and H to describe arbitrary distributions over observations and parameters, respectively. Typically H will be the conjugate prior of F. The two most common choices of F are Gaussian and categorical.

Compared with a simple mixture model

As mentioned above, the distribution of each observation in a hidden Markov model is a mixture density, with the states of the HMM corresponding to mixture components. It is useful to compare the above characterizations for an HMM with the corresponding characterizations, of a mixture model, using the same notation.

A non-Bayesian mixture model:

N = number of mixture components

T = number of observations

$\theta_{i=1...N}$ = parameter of distribution of observation associated with component i

$\phi_{i=1...N}$ = mixture weight, i.e. prior probability of a particular component i

$\boldsymbol{\phi}$ = N-dimensional vector composed of all the individual $\phi_{1...N}$; must sum to 1

$x_{i=1...T}$ = component of observation i

$y_{i=1...T}$ = observation i

$F(y|\theta)$ = probability distribution of an observation, parametrized on θ

$x_{i=1...T} \sim \text{Categorical}(\boldsymbol{\phi})$

$y_{i=1...T} \sim F(\theta_{x_i})$

A Bayesian mixture model:

$$
\begin{array}{lcl}
N, T & = & \text{as above} \\
\theta_{i=1\ldots N}, \phi_{i=1\ldots N}, \boldsymbol{\phi} & = & \text{as above} \\
x_{i=1\ldots T}, y_{i=1\ldots T}, F(y|\theta) & = & \text{as above} \\
\alpha & = & \text{shared hyperparameter for component parameters} \\
\beta & = & \text{shared hyperparameter for mixture weights} \\
H(\theta|\alpha) & = & \text{prior probability distribution of component parameters, parametrized on } \alpha \\
\theta_{i=1\ldots N} & \sim & H(\alpha) \\
\boldsymbol{\phi} & \sim & \text{Symmetric-Dirichlet}_N(\beta) \\
x_{i=1\ldots T} & \sim & \text{Categorical}(\boldsymbol{\phi}) \\
y_{i=1\ldots T} & \sim & F(\theta_{x_i})
\end{array}
$$

Examples of HMMs

The following mathematical descriptions are fully written out and explained, for ease of implementation.

A typical non-Bayesian HMM with Gaussian observations looks like this:

$$
\begin{array}{lcl}
N & = & \text{number of states} \\
T & = & \text{number of observations} \\
\phi_{i=1\ldots N, j=1\ldots N} & = & \text{probability of transition from state } i \text{ to state } j \\
\boldsymbol{\phi}_{i=1\ldots N} & = & N\text{-dimensional vector, composed of } \phi_{i,1\ldots N}\text{; must sum to 1} \\
\mu_{i=1\ldots N} & = & \text{mean of observations associated with state } i \\
\sigma^2_{i=1\ldots N} & = & \text{variance of observations associated with state } i \\
x_{t=1\ldots T} & = & \text{state of observation at time } t \\
y_{t=1\ldots T} & = & \text{observation at time } t \\
x_{t=2\ldots T} & \sim & \text{Categorical}(\boldsymbol{\phi}_{x_{t-1}}) \\
y_{t=1\ldots T} & \sim & \mathcal{N}(\mu_{x_t}, \sigma^2_{x_t})
\end{array}
$$

A typical Bayesian HMM with Gaussian observations looks like this:

$$
\begin{aligned}
N &= \text{number of states} \\
T &= \text{number of observations} \\
\phi_{i=1\ldots N, j=1\ldots N} &= \text{probability of transition from state } i \text{ to state } j \\
\boldsymbol{\phi}_{i=1\ldots N} &= N\text{-dimensional vector, composed of } \phi_{i,1\ldots N}\text{; must sum to 1} \\
\mu_{i=1\ldots N} &= \text{mean of observations associated with state } i \\
\sigma^2_{i=1\ldots N} &= \text{variance of observations associated with state } i \\
x_{t=1\ldots T} &= \text{state of observation at time } t \\
y_{t=1\ldots T} &= \text{observation at time } t \\
\beta &= \text{concentration hyperparameter controlling the density of the transition matrix} \\
\mu_0, \lambda &= \text{shared hyperparameters of the means for each state} \\
\nu, \sigma_0^2 &= \text{shared hyperparameters of the variances for each state} \\
\boldsymbol{\phi}_{i=1\ldots N} &\sim \text{Symmetric-Dirichlet}_N(\beta) \\
x_{t=2\ldots T} &\sim \text{Categorical}(\boldsymbol{\phi}_{x_{t-1}}) \\
\mu_{i=1\ldots N} &\sim \mathcal{N}(\mu_0, \lambda\sigma_i^2) \\
\sigma^2_{i=1\ldots N} &\sim \text{Inverse-Gamma}(\nu, \sigma_0^2) \\
y_{t=1\ldots T} &\sim \mathcal{N}(\mu_{x_t}, \sigma^2_{x_t})
\end{aligned}
$$

A typical non-Bayesian HMM with categorical observations looks like this:

$$
\begin{aligned}
N &= \text{number of states} \\
T &= \text{number of observations} \\
\phi_{i=1\ldots N, j=1\ldots N} &= \text{probability of transition from state } i \text{ to state } j \\
\boldsymbol{\phi}_{i=1\ldots N} &= N\text{-dimensional vector, composed of } \phi_{i,1\ldots N}\text{; must sum to 1} \\
V &= \text{dimension of categorical observations, e.g. size of word vocabulary} \\
\theta_{i=1\ldots N, j=1\ldots V} &= \text{probability for state } i \text{ of observing the } j\text{th item} \\
\boldsymbol{\theta}_{i=1\ldots N} &= V\text{-dimensional vector, composed of } \theta_{i,1\ldots V}\text{; must sum to 1} \\
x_{t=1\ldots T} &= \text{state of observation at time } t \\
y_{t=1\ldots T} &= \text{observation at time } t \\
x_{t=2\ldots T} &\sim \text{Categorical}(\boldsymbol{\phi}_{x_{t-1}}) \\
y_{t=1\ldots T} &\sim \text{Categorical}(\boldsymbol{\theta}_{x_t})
\end{aligned}
$$

A typical Bayesian HMM with categorical observations looks like this:

N = number of states
T = number of observations
$\phi_{i=1...N,j=1...N}$ = probability of transition from state i to state j
$\boldsymbol{\phi}_{i=1...N}$ = N-dimensional vector, composed of $\phi_{i,1...N}$; must sum to 1
V = dimension of categorical observations, e.g. size of word vocabulary
$\theta_{i=1...N,j=1...V}$ = probability for state i of observing the jth item
$\boldsymbol{\theta}_{i=1...N}$ = V-dimensional vector, composed of $\theta_{i,1...V}$; must sum to 1
$x_{t=1...T}$ = state of observation at time t
$y_{t=1...T}$ = observation at time t
α = shared concentration hyperparameter of $\boldsymbol{\theta}$ for each state
β = concentration hyperparameter controlling the density of the transition matrix
$\boldsymbol{\phi}_{i=1...N} \sim \text{Symmetric-Dirichlet}_N(\beta)$
$\boldsymbol{\theta}_{1...V} \sim \text{Symmetric-Dirichlet}_V(\alpha)$
$x_{t=2...T} \sim \text{Categorical}(\boldsymbol{\phi}_{x_{t-1}})$
$y_{t=1...T} \sim \text{Categorical}(\boldsymbol{\theta}_{x_t})$

Note that in the above Bayesian characterizations, β (a concentration parameter) controls the density of the transition matrix. That is, with a high value of β (significantly above 1), the probabilities controlling the transition out of a particular state will all be similar, meaning there will be a significantly probability of transitioning to any of the other states. In other words, the path followed by the Markov chain of hidden states will be highly random. With a low value of β (significantly below 1), only a small number of the possible transitions out of a given state will have significant probability, meaning that the path followed by the hidden states will be somewhat predictable.

A two-level Bayesian HMM

An alternative for the above two Bayesian examples would be to add another level of prior parameters for the transition matrix. That is, replace the lines

β = concentration hyperparameter controlling the density of the transitio
$\boldsymbol{\phi}_{i=1...N} \sim \text{Symmetric-Dirichlet}_N(\beta)$

with the following:

γ = concentration hyperparameter controlling how many states are intrinsically likely
β = concentration hyperparameter controlling the density of the transition matrix
$\boldsymbol{\eta}$ = N-dimensional vector of probabilities, specifying the intrinsic probability of a given state
$\boldsymbol{\eta} \sim \text{Symmetric-Dirichlet}_N(\gamma)$
$\boldsymbol{\phi}_{i=1...N} \sim \text{Dirichlet}_N(\beta N \boldsymbol{\eta})$

What this means is the following:

1. $\boldsymbol{\eta}$ is a probability distribution over states, specifying which states are inherently likely. The greater the probability of a given state in this vector, the more likely is a transition to that state (regardless of the starting state).
2. γ controls the density of $\boldsymbol{\eta}$. Values significantly above 1 cause a dense vector where all states will have similar prior probabilities. Values significantly below 1 cause a sparse vector where only a few states are inherently likely (have prior probabilities significantly above 0).
3. β controls the density of the transition matrix, or more specifically, the density of the N different probability vectors $\boldsymbol{\phi}_{i=1...N}$ specifying the probability of transitions out of state i to any other state.

Imagine that the value of β is significantly above 1. Then the different $\boldsymbol{\phi}$ vectors will be dense, i.e. the probability mass will be spread out fairly evenly over all states. However, to the extent that this mass is unevenly spread, $\boldsymbol{\eta}$ controls which states are likely to get more mass than others.

Now, imagine instead that β is significantly below 1. This willl make the $\boldsymbol{\phi}$ vectors sparse, i.e. almost all the probability mass is distributed over a small number of states, and for the rest, a transition to that state will be very unlikely. Notice that there are different $\boldsymbol{\phi}$ vectors for each starting state, and so even if all the vectors are sparse, different vectors may distribute the mass to different ending states. However, for all of the vectors, $\boldsymbol{\eta}$ controls which ending states are likely to get mass assigned to them. For example, if β is 0.1, then each $\boldsymbol{\phi}$ will be sparse and, for any given starting state i, the set of states $\mathbf{J}_i$ to which transitions are likely to occur will be very small, typically having only one or two members. Now, if the probabilities in $\boldsymbol{\eta}$ are all the same (or equivalently, one of the above models without $\boldsymbol{\eta}$ is used), then for different i, there will be different states in the corresponding $\mathbf{J}_i$, so that all states are equally likely to occur in any given

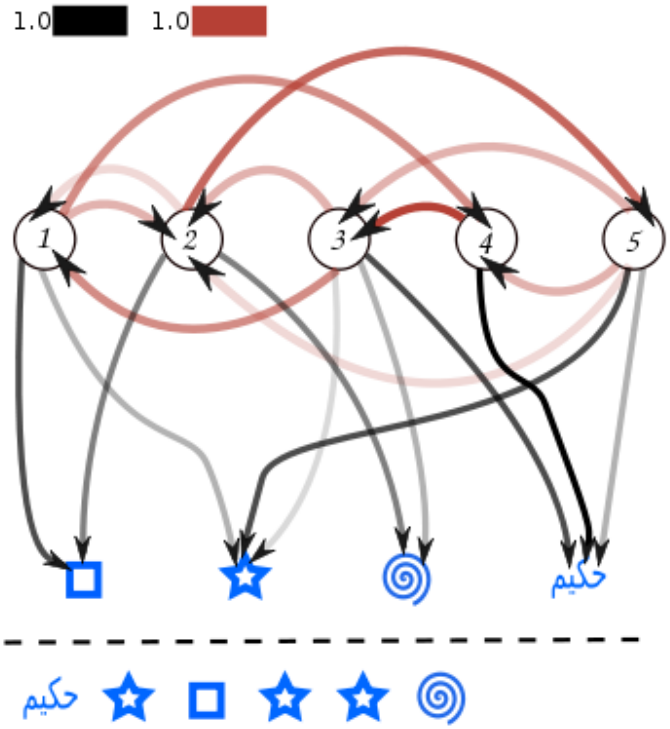
$\mathbf{J}_i$. On the other hand, if the values in $\boldsymbol{\eta}$ are unbalanced, so that one state has a much higher probability than others, almost all $\mathbf{J}_i$ will contain this state; hence, regardless of the starting state, transitions will nearly always occur to this given state.

Hence, a two-level model such as just described allows independent control over (1) the overall density of the transition matrix, and (2) the density of states to which transitions are likely (i.e. the density of the prior distribution of states in any particular hidden variable x_i). In both cases this is done while still assuming ignorance over which particular states are more likely than others. If it is desired to inject this information into the model, the probability vector $\boldsymbol{\eta}$ can be directly specified; or, if there is less certainty about these relative probabilities, a non-symmetric Dirichlet distribution can be used as the prior distribution over $\boldsymbol{\eta}$. That is, instead of using a symmetric Dirichlet distribution with a single parameter γ (or equivalently, a general Dirichlet with a vector all of whose values are equal to γ), use a general Dirichlet with values that are variously greater or less than γ, according to which state is more or less preferred.

Learning

The parameter learning task in HMMs is to find, given an output sequence or a set of such sequences, the best set of state transition and output probabilities. The task is usually to derive the maximum likelihood estimate of the parameters of the HMM given the set of output sequences. No tractable algorithm is known for solving this problem exactly, but a local maximum likelihood can be derived efficiently using the Baum-Welch algorithm or the Baldi-Chauvin algorithm. The Baum-Welch algorithm is an example of a forward-backward algorithm, and is a special case of the Expectation-maximization algorithm.

Inference



The state transition and output probabilities of an HMM are indicated by the line opacity in the upper part of the diagram. Given that we have observed the output sequence in the lower part of the diagram, we may be interested in the most likely sequence of states that could have produced it. Based on the arrows that are present in the diagram, the following state sequences are candidates:

5 3 2 5 3 2
4 3 2 5 3 2
3 1 2 5 3 2

We can find the most likely sequence by evaluating the joint probability of both the state sequence and the observations for each case (simply by multiplying the probability values, which here correspond to the opacities of the arrows involved). In general, this

type of problem (i.e. finding the most likely explanation for an observation sequence) can be solved efficiently using the Viterbi algorithm.

Several inference problems are associated with hidden Markov models, as outlined below.

Filtering

The task is to compute, given the model's parameters and a sequence of observations, the distribution over hidden states at the end of the sequence, i.e. to compute $P(x(t) \mid y(1), \ldots, y(t))$. This problem can be handled efficiently using the forward algorithm.

Probability of an observed sequence

The task is to compute, given the parameters of the model, the probability of a particular output sequence. This requires summation over all possible state sequences:

The probability of observing a sequence

$$Y = y(0), y(1), \ldots, y(L-1)$$

of length L is given by

$$P(Y) = \sum_X P(Y \mid X) P(X),$$

where the sum runs over all possible hidden-node sequences

$$X = x(0), x(1), \ldots, x(L-1).$$

Applying the principle of dynamic programming, this problem, too, can be handled efficiently using the forward algorithm.

Most likely explanation

The task is to compute, given the parameters of the model and a particular output sequence, the state sequence that is most likely to have generated that output sequence (see illustration on the right). This requires finding a maximum over all possible state sequences, but can similarly be solved efficiently by the Viterbi algorithm.

Smoothing

The task is to compute, given the parameters of the model and a particular output sequence up to time t, the probability distribution over hidden states for a point in time in the past, i.e. to compute $P(x(k) \mid y(1), \ldots, y(t))$ for some k < t. The forward-

backward algorithm is an efficient method for computing the smoothed values for all hidden state variables.

Statistical significance

For some of the above problems, it may also be interesting to ask about statistical significance. What is the probability that a sequence drawn from some null distribution will have an HMM probability (in the case of the forward algorithm) or a maximum state sequence probability (in the case of the Viterbi algorithm) at least as large as that of a particular output sequence? When an HMM is used to evaluate the relevance of a hypothesis for a particular output sequence, the statistical significance indicates the false positive rate associated with accepting the hypothesis for the output sequence.

A concrete example

Consider two friends, Alice and Bob, who live far apart from each other and who talk together daily over the telephone about what they did that day. Bob is only interested in three activities: walking in the park, shopping, and cleaning his apartment. The choice of what to do is determined exclusively by the weather on a given day. Alice has no definite information about the weather where Bob lives, but she knows general trends. Based on what Bob tells her he did each day, Alice tries to guess what the weather must have been like.

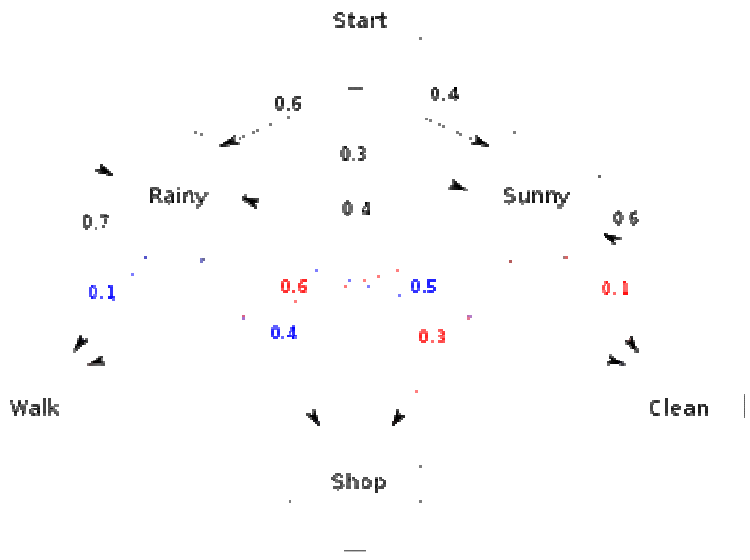
Alice believes that the weather operates as a discrete Markov chain. There are two states, "Rainy" and "Sunny", but she cannot observe them directly, that is, they are hidden from her. On each day, there is a certain chance that Bob will perform one of the following activities, depending on the weather: "walk", "shop", or "clean". Since Bob tells Alice about his activities, those are the observations. The entire system is that of a hidden Markov model (HMM).

Alice knows the general weather trends in the area, and what Bob likes to do on average. In other words, the parameters of the HMM are known. They can be represented as follows in the Python programming language:

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')
```

```
start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
  'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
  'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
  }

emission_probability = {
  'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
  'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
  }
```

In this piece of code, start_probability represents Alice's belief about which state the HMM is in when Bob first calls her (all she knows is that it tends to be rainy on average). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately {'Rainy': 0.57, 'Sunny': 0.43}. The transition_probability represents the change of the weather in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow will be sunny if today is rainy. The emission_probability represents how likely Bob is to perform a certain activity on each day. If it is rainy, there is a 50% chance that he is cleaning his apartment; if it is sunny, there is a 60% chance that he is outside for a walk.



This example is further elaborated in the Viterbi algorithm page.

Applications of hidden Markov models

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable (but other data that depends on the sequence is). Common applications include:

- Cryptanalysis
- Speech recognition
- Part-of-speech tagging
- Machine translation
- Partial discharge
- Gene prediction
- Alignment of bio-sequences
- Activity recognition
- Protein folding

History

Hidden Markov Models were first described in a series of statistical papers by Leonard E. Baum and other authors in the second half of the 1960s. One of the first applications of HMMs was speech recognition, starting in the mid-1970s.

In the second half of the 1980s, HMMs began to be applied to the analysis of biological sequences, in particular DNA. Since then, they have become ubiquitous in the field of bioinformatics.

Types of hidden Markov models

Hidden Markov models can model complex Markov processes where the states emit the observations according to some probability distribution. One such example of distribution is Gaussian distribution, in such a Hidden Markov Model the states output is represented by a Gaussian distribution.

Moreover it could represent even more complex behavior when the output of the states is represented as mixture of two or more Gaussians, in which case the probability of generating an observation is the product of the probability of first selecting one of the Gaussians and the probability of generating that observation from that Gaussian.

Extensions

In the hidden Markov models considered above, the state space of the hidden variables is discrete, while the observations themselves can either be discrete (typically generated from a categorical distribution) or continuous (typically from a Gaussian distribution). Hidden Markov models can also be generalized to allow continuous state spaces. Examples of such models are those where the Markov process over hidden variables is a linear dynamical system, with a linear relationship among related variables and where all hidden and observed variables follow a Gaussian distribution. In simple cases, such as the linear dynamical system just , exact inference is tractable (in this case, using the Kalman filter); however, in general, exact inference in HMMs with continuous latent variables is infeasible, and approximate methods must be used, such as the extended Kalman filter or the particle filter.

Hidden Markov models are generative models, in which the joint distribution of observations and hidden states, or equivalently both the prior distribution of hidden states (the transition probabilities) and conditional distribution of observations given states (the emission probabilities), is modeled. The above algorithms implicitly assume a uniform prior distribution over the transition probabilities. However, it is also possible to create hidden Markov models with other types of prior distributions. An obvious candidate, given the categorical distribution of the transition probabilities, is the Dirichlet distribution, which is the conjugate prior distribution of the categorical distribution. Typically, a symmetric Dirichlet distribution is chosen, reflecting ignorance about which states are inherently more likely than others. The single parameter of this distribution (termed the concentration parameter) controls the relative density or sparseness of the resulting transition matrix. A choice of 1 yields a uniform distribution. Values greater than 1 produce a dense matrix, in which the transition probabilities between pairs of states are likely to be nearly equal. Values less than 1 result in a sparse matrix in which, for each given source state, only a small number of destination states have non-negligible transition probabilities. It is also possible to use a two-level prior Dirichlet distribution, in which one Dirichlet distribution (the upper distribution) governs the parameters of another Dirichlet distribution (the lower distribution), which in turn governs the transition probabilities. The upper distribution governs the overall distribution of states,

determining how likely each state is to occur; its concentration parameter determines the density or sparseness of states. Such a two-level prior distribution, where both concentration parameters are set to produce sparse distributions, might be useful for example in unsupervised part-of-speech tagging, where some parts of speech occur much more commonly than others; learning algorithms that assume a uniform prior distribution generally perform poorly on this task. The parameters of models of this sort, with non-uniform prior distributions, can be learned using Gibbs sampling or extended versions of the expectation-maximization algorithm.

An extension of the previously-described hidden Markov models with Dirichlet priors uses a Dirichlet process in place of a Dirichlet distribution. This type of model allows for an unknown and potentially infinite number of states. It is common to use a two-level Dirichlet process, similar to the previously-described model with two levels of Dirichlet distributions. Such a model is called a hierarchical Dirichlet process hidden Markov model, or HDP-HMM for short.

A different type of extension uses a discriminative model in place of the generative model of standard HMM's. This type of model directly models the conditional distribution of the hidden states given the observations, rather than modeling the joint distribution. An example of this model is the so-called maximum entropy Markov model (MEMM), which models the conditional distribution of the states using logistic regression (also known as a "maximum entropy model"). The advantage of this type of model is that arbitrary features (i.e. functions) of the observations can be modeled, allowing domain-specific knowledge of the problem at hand to be injected into the model. Models of this sort are not limited to modeling direct dependencies between a hidden state and its associated observation; rather, features of nearby observations, of combinations of the associated observation and nearby observations, or in fact of arbitrary observations at any distance from a given hidden state can be included in the process used to determine the value of a hidden state. Furthermore, there is no need for these features to be statistically independent of each other, as would be the case if such features were used in a generative model. Finally, arbitrary features over pairs of adjacent hidden states can be used rather than simple transition probabilities. The disadvantages of such models are: (1) The types of prior distributions that can be placed on hidden states

are severely limited; (2) It is not possible to predict the probability of seeing an arbitrary observation. This second limitation is often not an issue in practice, since many common usages of HMM's do not require such predictive probabilities.

A variant of the previously described discriminative model is the linear-chain conditional random field. This uses an undirected graphical model (aka Markov random field) rather than the directed graphical models of MEMM's and similar models. The advantage of this type of model is that it does not suffer from the so-called label bias problem of MEMM's, and thus may make more accurate predictions. The disadvantage is that training can be slower than for MEMM's.

Yet another variant is the factorial hidden Markov model, which allows for a single observation to be conditioned on the corresponding hidden variables of a set of K independent Markov chains, rather than a single Markov chain. Learning in such a model is difficult, as dynamic-programming techniques can no longer be used to find an exact solution; in practice, approximate techniques must be used.

All of the above models can be extended to allow for more distant dependencies among hidden states, e.g. allowing for a given state to be dependent on the previous two or three states rather than a single previous state; i.e. the transition probabilities are extended to encompass sets of three or four adjacent states (or in general K adjacent states). The disadvantage of such models is that dynamic-programming algorithms for training them have an $O(N^K T)$ running time, for K adjacent states and T total observations (i.e. a length-T Markov chain).

HOMOLOGY MODELING

Homology modeling, also known as comparative modeling of protein refers to constructing an atomic-resolution model of the "target" protein from its amino acid sequence and an experimental three-dimensional structure of a related homologous protein (the "template"). Homology modeling relies on the identification of one or more known protein structures likely to resemble the structure of the query sequence, and on the production of an alignment that maps residues in the query sequence to residues in the template sequence. It has been shown that protein structures are more conserved than protein sequences amongst homologues, but sequences falling below a 20% sequence identity can have very different structure.

Evolutionarily related proteins have similar sequences and naturally occurring homologous proteins have similar protein structure. It has been shown that three-dimensional protein structure is evolutionarily more conserved than expected due to sequence conservation.

The sequence alignment and template structure are then used to produce a structural model of the target. Because protein structures are more conserved than DNA sequences, detectable levels of sequence similarity usually imply significant structural similarity.

The quality of the homology model is dependent on the quality of the sequence alignment and template structure. The approach can be complicated by the presence of alignment gaps (commonly called indels) that indicate a structural region present in the target but not in the template, and by structure gaps in the template that arise from poor resolution in the experimental procedure (usually X-ray crystallography) used to solve the structure. Model quality declines with decreasing sequence identity; a typical model has ~1-2 Å root mean square deviation between the matched C^{α} atoms at 70% sequence identity but only 2-4 Å agreement at 25% sequence identity. However, the errors are significantly higher in the loop regions, where the amino acid sequences of the target and template proteins may be completely different.

Regions of the model that were constructed without a template, usually by loop modeling, are generally much less accurate than the rest of the model. Errors in side chain packing and position also increase with decreasing identity, and variations in these packing configurations have been suggested as a major reason for poor model quality at low identity. Taken together, these various atomic-position errors are significant and impede the use of homology models for purposes that require atomic-resolution data, such as drug design and protein-protein interaction predictions; even the quaternary structure of a protein may be difficult to predict from homology models of its subunit(s). Nevertheless, homology models can be useful in reaching qualitative conclusions about the biochemistry of the query sequence, especially in formulating hypotheses about why certain residues are conserved, which may in turn lead to experiments to test those hypotheses. For example, the spatial arrangement of conserved residues may suggest whether a particular residue is conserved to stabilize the folding, to participate in binding some small molecule, or to foster association with another protein or nucleic acid.

Homology modeling can produce high-quality structural models when the target and template are closely related, which has inspired the formation of a structural genomics consortium dedicated to the production of representative experimental structures for all classes of protein folds. The chief inaccuracies in homology modeling, which worsen with lower sequence identity, derive from errors in the initial sequence alignment and from improper template selection. Like other methods of structure prediction, current practice in homology modeling is assessed in a biannual large-scale experiment known as the Critical Assessment of Techniques for Protein Structure Prediction, or CASP.

Motive

The method of homology modeling is based on the observation that protein tertiary structure is better conserved than amino acid sequence. Thus, even proteins that have diverged appreciably in sequence but still share detectable similarity will also share common structural properties, particularly the overall fold. Because it is difficult and time-consuming to obtain experimental structures from methods such as X-ray crystallography and protein NMR for every protein of interest, homology modeling can provide useful structural models for generating hypotheses about a protein's function and directing further experimental work.

There are exceptions to the general rule that proteins sharing significant sequence identity will share a fold. For example, a judiciously chosen set of mutations of less than 50% of a protein can cause the protein to adopt a completely different fold. However, such a massive structural rearrangement is unlikely to occur in evolution, especially since the protein is usually under the constraint that it must fold properly and carry out its function in the cell. Consequently, the roughly folded structure of a protein (its "topology") is conserved longer than its amino-acid sequence and much longer than the corresponding DNA sequence; in other words, two proteins may share a similar fold even if their evolutionary relationship is so distant that it cannot be discerned reliably. For comparison, the function of a protein is conserved much less than the protein sequence, since relatively few changes in amino-acid sequence are required to take on a related function.

Steps in model production

The homology modeling procedure can be broken down into four sequential steps: template selection, target-template alignment, model construction, and model assessment. The first two steps are often essentially performed together, as the most common methods of identifying templates rely on the production of sequence alignments; however, these alignments may not be of sufficient quality because database search techniques prioritize speed over alignment quality. These processes can be performed iteratively to improve the quality of the final model, although quality assessments that are not dependent on the true target structure are still under development.

Optimizing the speed and accuracy of these steps for use in large-scale automated structure prediction is a key component of structural genomics initiatives, partly because the resulting volume of data will be too large to process manually and partly because the goal of structural genomics requires providing models of reasonable quality to researchers who are not themselves structure prediction experts.

Template selection and sequence alignment

The critical first step in homology modeling is the identification of the best template structure, if indeed any are available. The simplest method of template identification relies on serial pairwise sequence alignments aided by database search techniques such as FASTA and BLAST. More sensitive methods based on multiple sequence alignment - of which PSI-BLAST is the most common example - iteratively update their position-specific scoring matrix to successively identify more distantly related homologs. This family of methods has been shown to produce a larger number of potential templates and to identify better templates for sequences that have only distant relationships to any solved structure. Protein threading, also known as fold recognition or 3D-1D alignment, can also be used as a search technique for identifying templates to be used in traditional homology modeling methods. When performing a BLAST search, a reliable first approach is to identify hits with a sufficiently low E-value, which are considered sufficiently close in evolution to make a reliable homology model. Other factors may tip the balance in marginal cases; for example, the template may have a function similar to that of the query sequence, or it may belong to a homologous operon. However, a template with a poor E-value should generally not be chosen, even if it is the

only one available, since it may well have a wrong structure, leading to the production of a misguided model. A better approach is to submit the primary sequence to fold-recognition servers or, better still, consensus meta-servers which improve upon individual fold-recognition servers by identifying similarities (consensus) among independent predictions.

Often several candidate template structures are identified by these approaches. Although some methods can generate hybrid models from multiple templates, most methods rely on a single template. Therefore, choosing the best template from among the candidates is a key step, and can affect the final accuracy of the structure significantly. This choice is guided by several factors, such as the similarity of the query and template sequences, of their functions, and of the predicted query and observed template secondary structures. Perhaps most importantly, the coverage of the aligned regions: the fraction of the query sequence structure that can be predicted from the template, and the plausibility of the resulting model. Thus, sometimes several homology models are produced for a single query sequence, with the most likely candidate chosen only in the final step.

It is possible to use the sequence alignment generated by the database search technique as the basis for the subsequent model production; however, more sophisticated approaches have also been explored. One proposal generates an ensemble of stochastically defined pairwise alignments between the target sequence and a single identified template as a means of exploring "alignment space" in regions of sequence with low local similarity. "Profile-profile" alignments that first generate a sequence profile of the target and systematically compare it to the sequence profiles of solved structures; the coarse-graining inherent in the profile construction is thought to reduce noise introduced by sequence drift in nonessential regions of the sequence.

Model generation

Given a template and an alignment, the information contained therein must be used to generate a three-dimensional structural model of the target, represented as a set of Cartesian coordinates for each atom in the protein. Three major classes of model generation methods have been proposed.

Fragment assembly

The original method of homology modeling relied on the assembly of a complete model from conserved structural fragments identified in closely related solved structures. For example, a modeling study of serine proteases in mammals identified a sharp distinction between "core" structural regions conserved in all experimental structures in the class, and variable regions typically located in the loops where the majority of the sequence differences were localized. Thus unsolved proteins could be modeled by first constructing the conserved core and then substituting variable regions from other proteins in the set of solved structures. Current implementations of this method differ mainly in the way they deal with regions that are not conserved or that lack a template. The variable regions are often constructed with the help of fragment libraries.

Segment matching

The segment-matching method divides the target into a series of short segments, each of which is matched to its own template fitted from the Protein Data Bank. Thus, sequence alignment is done over segments rather than over the entire protein. Selection of the template for each segment is based on sequence similarity, comparisons of alpha carbon coordinates, and predicted steric conflicts arising from the van der Waals radii of the divergent atoms between target and template.

Satisfaction of spatial restraints

The most common current homology modeling method takes its inspiration from calculations required to construct a three-dimensional structure from data generated by NMR spectroscopy. One or more target-template alignments are used to construct a set of geometrical criteria that are then converted to probability density functions for each restraint. Restraints applied to the main protein internal coordinates - protein backbone distances and dihedral angles - serve as the basis for a global optimization procedure that originally used conjugate gradient energy minimization to iteratively refine the positions of all heavy atoms in the protein.

This method had been dramatically expanded to apply specifically to loop modeling, which can be extremely difficult due to the high flexibility of loops in proteins in aqueous solution. A more recent expansion applies the spatial-restraint model to electron density maps derived from cryoelectron microscopy studies, which provide low-

resolution information that is not usually itself sufficient to generate atomic-resolution structural models. To address the problem of inaccuracies in initial target-template sequence alignment, an iterative procedure has also been introduced to refine the alignment on the basis of the initial structural fit. The most commonly used software in spatial restraint-based modeling is MODELLER and a database called ModBase has been established for reliable models generated with it.

Loop modeling

Regions of the target sequence that are not aligned to a template are modeled by loop modeling; they are the most susceptible to major modeling errors and occur with higher frequency when the target and template have low sequence identity. The coordinates of unmatched sections determined by loop modeling programs are generally much less accurate than those obtained from simply copying the coordinates of a known structure, particularly if the loop is longer than 10 residues. The first two sidechain dihedral angles (χ_1 and χ_2) can usually be estimated within 30° for an accurate backbone structure; however, the later dihedral angles found in longer side chains such as lysine and arginine are notoriously difficult to predict. Moreover, small errors in χ_1 (and, to a lesser extent, in χ_2) can cause relatively large errors in the positions of the atoms at the terminus of side chain; such atoms often have a functional importance, particularly when located near the active site.

Model assessment

Assessment of homology models without reference to the true target structure is usually performed with two methods: statistical potentials or physics-based energy calculations. Both methods produce an estimate of the energy (or an energy-like analog) for the model or models being assessed; independent criteria are needed to determine acceptable cutoffs. Neither of the two methods correlates exceptionally well with true structural accuracy, especially on protein types underrepresented in the PDB, such as membrane proteins.

Statistical potentials are empirical methods based on observed residue-residue contact frequencies among proteins of known structure in the PDB. They assign a probability or energy score to each possible pairwise interaction between amino acids and combine these pairwise interaction scores into a single score for the entire model. Some

such methods can also produce a residue-by-residue assessment that identifies poorly scoring regions within the model, though the model may have a reasonable score overall. These methods emphasize the hydrophobic core and solvent-exposed polar amino acids often present in globular proteins. Examples of popular statistical potentials include Prosa and DOPE. Statistical potentials are more computationally efficient than energy calculations.

Physics-based energy calculations aim to capture the interatomic interactions that are physically responsible for protein stability in solution, especially van der Waals and electrostatic interactions. These calculations are performed using a molecular mechanics force field; proteins are normally too large even for semi-empirical quantum mechanics-based calculations. The use of these methods is based on the energy landscape hypothesis of protein folding, which predicts that a protein's native state is also its energy minimum. Such methods usually employ implicit solvation, which provides a continuous approximation of a solvent bath for a single protein molecule without necessitating the explicit representation of individual solvent molecules. A force field specifically constructed for model assessment is known as the Effective Force Field (EFF) and is based on atomic parameters from CHARMM.

A very extensive model validation report can be obtained using the Radboud Universiteit Nijmegen "What Check" software which is one option of the Radboud Universiteit Nijmegen "What If" software package; it produces a many page document with extensive analyses of nearly 200 scientific and administrative aspects of the model. "What Check" is available as a free server; it can also be used to validate experimentally determined structures of macromolecules.

One newer method for model assessment relies on machine learning techniques such as neural nets, which may be trained to assess the structure directly or to form a consensus among multiple statistical and energy-based methods. Very recent results using support vector machine regression on a jury of more traditional assessment methods outperformed common statistical, energy-based, and machine learning methods.

Structural comparison methods

The assessment of homology models' accuracy is straightforward when the experimental structure is known. The most common method of comparing two protein

structures uses the root-mean-square deviation (RMSD) metric to measure the mean distance between the corresponding atoms in the two structures after they have been superimposed. However, RMSD does underestimate the accuracy of models in which the core is essentially correctly modeled, but some flexible loop regions are inaccurate. A method introduced for the modeling assessment experiment CASP is known as the global distance test (GDT) and measures the total number of atoms whose distance from the model to the experimental structure lies under a certain distance cutoff. Both methods can be used for any subset of atoms in the structure, but are often applied to only the alpha carbon or protein backbone atoms to minimize the noise created by poorly modeled side chain rotameric states, which most modeling methods are not optimized to predict.

Benchmarking

Several large-scale benchmarking efforts have been made to assess the relative quality of various current homology modeling methods. CASP is a community-wide prediction experiment that runs every two years during the summer months and challenges prediction teams to submit structural models for a number of sequences whose structures have recently been solved experimentally but have not yet been published. Its partner CAFASP has run in parallel with CASP but evaluates only models produced via fully automated servers. Continuously running experiments that do not have prediction 'seasons' focus mainly on benchmarking publicly available webservers. LiveBench and EVA run continuously to assess participating servers' performance in prediction of imminently released structures from the PDB. CASP and CAFASP serve mainly as evaluations of the state of the art in modeling, while the continuous assessments seek to evaluate the model quality that would be obtained by a non-expert user employing publicly available tools.

Accuracy

The accuracy of the structures generated by homology modeling is highly dependent on the sequence identity between target and template. Above 50% sequence identity, models tend to be reliable, with only minor errors in side chain packing and rotameric state, and an overall RMSD between the modeled and the experimental structure falling around 1 Â. This error is comparable to the typical resolution of a structure solved by NMR. In the 30-50% identity range, errors can be more severe and

are often located in loops. Below 30% identity, serious errors occur, sometimes resulting in the basic fold being mis-predicted. This low-identity region is often referred to as the "twilight zone" within which homology modeling is extremely difficult, and to which it is possibly less suited than fold recognition methods.

At high sequence identities, the primary source of error in homology modeling derives from the choice of the template or templates on which the model is based, while lower identities exhibit serious errors in sequence alignment that inhibit the production of high-quality models. It has been suggested that the major impediment to quality model production is inadequacies in sequence alignment, since "optimal" structural alignments between two proteins of known structure can be used as input to current modeling methods to produce quite accurate reproductions of the original experimental structure.

Attempts have been made to improve the accuracy of homology models built with existing methods by subjecting them to molecular dynamics simulation in an effort to improve their RMSD to the experimental structure. However, current force field parameterizations may not be sufficiently accurate for this task, since homology models used as starting structures for molecular dynamics tend to produce slightly worse structures. Slight improvements have been observed in cases where significant restraints were used during the simulation.

Sources of error

The two most common and large-scale sources of error in homology modeling are poor template selection and inaccuracies in target-template sequence alignment. Controlling for these two factors by using a structural alignment, or a sequence alignment produced on the basis of comparing two solved structures, dramatically reduces the errors in final models; these "gold standard" alignments can be used as input to current modeling methods to produce quite accurate reproductions of the original experimental structure. Results from the most recent CASP experiment suggest that "consensus" methods collecting the results of multiple fold recognition and multiple alignment searches increase the likelihood of identifying the correct template; similarly, the use of multiple templates in the model-building step may be worse than the use of the single correct template but better than the use of a single suboptimal one. Alignment errors may be minimized by the use of a multiple alignment even if only one template is used, and by

the iterative refinement of local regions of low similarity. A lesser source of model errors are errors in the template structure. The PDBREPORT database lists several million, mostly very small but occasionally dramatic, errors in experimental (template) structures that have been deposited in the PDB.

Serious local errors can arise in homology models where an insertion or deletion mutation or a gap in a solved structure result in a region of target sequence for which there is no corresponding template. This problem can be minimized by the use of multiple templates, but the method is complicated by the templates' differing local structures around the gap and by the likelihood that a missing region in one experimental structure is also missing in other structures of the same protein family. Missing regions are most common in loops where high local flexibility increases the difficulty of resolving the region by structure-determination methods. Although some guidance is provided even with a single template by the positioning of the ends of the missing region, the longer the gap, the more difficult it is to model. Loops of up to about 9 residues can be modeled with moderate accuracy in some cases if the local alignment is correct. Larger regions are often modeled individually using ab initio structure prediction techniques, although this approach has met with only isolated success.

The rotameric states of side chains and their internal packing arrangement also present difficulties in homology modeling, even in targets for which the backbone structure is relatively easy to predict. This is partly due to the fact that many side chains in crystal structures are not in their "optimal" rotameric state as a result of energetic factors in the hydrophobic core and in the packing of the individual molecules in a protein crystal. One method of addressing this problem requires searching a rotameric library to identify locally low-energy combinations of packing states. It has been suggested that a major reason that homology modeling so difficult when target-template sequence identity lies below 30% is that such proteins have broadly similar folds but widely divergent side chain packing arrangements.

Utility

Uses of the structural models include protein-protein interaction prediction, protein-protein docking, molecular docking, and functional annotation of genes identified in an organism's genome. Even low-accuracy homology models can be useful for these

purposes, because their inaccuracies tend to be located in the loops on the protein surface, which are normally more variable even between closely related proteins. The functional regions of the protein, especially its active site, tend to be more highly conserved and thus more accurately modeled.

Homology models can also be used to identify subtle differences between related proteins that have not all been solved structurally. For example, the method was used to identify cation binding sites on the Na^+/K^+ ATPase and to propose hypotheses about different ATPases' binding affinity. Used in conjunction with molecular dynamics simulations, homology models can also generate hypotheses about the kinetics and dynamics of a protein, as in studies of the ion selectivity of a potassium channel. Large-scale automated modeling of all identified protein-coding regions in a genome has been attempted for the yeast Saccharomyces cerevisiae, resulting in nearly 1000 quality models for proteins whose structures had not yet been determined at the time of the study, and identifying novel relationships between 236 yeast proteins and other previously solved structures.

UNIT IV PHYLOGENY

In biology, phylogenetics is the study of evolutionary relatedness among groups of organisms (e.g. species, populations), which is discovered through molecular sequencing data and morphological data matrices. The term *phylogenetics* derives from the Greek terms *phyle* (φυλή) and *phylon* (φῦλον), denoting “tribe” and “race”; and the term *genetikos* (γενετικός), denoting “relative to birth”, from *genesis* (γένεσις) “birth”.

Taxonomy, the classification, identification, and naming of organisms, is richly informed by phylogenetics, but remains methodologically and logically distinct. The fields of phylogenetics and taxonomy overlap in the science of phylogenetic systematics — one methodology, cladism (also cladistics) shared derived characters (synapomorphies) used to create ancestor-descendant trees (cladograms) and delimit taxa (clades). In biological systematics as a whole, phylogenetic analyses have become essential in researching the evolutionary tree of life.

MUTATIONS; IRRELEVANT MUTATIONS; CONTROLS; MUTATIONS AS A MEASURE OF TIME

Mutations

Differences among species are the key to reconstructing the phylogenetic tree. Species differ in the characteristics, also called characters. The characters may be observable and measurable properties of the individuals. For instance, among mammals, the numbers of the different kinds of teeth that the individuals of the species have has been a successful character to classify mammals. This character has been especially important among extinct species since fossilized teeth are commonly found.

Any characters can be used to classify species and reconstruct a phylogenetic tree of species, but some are more useful than others. If a species depends on a character for its continued survival, that character will not change as any mutations of it will be eliminated. Call such characters *essential.* And most visible characters are essential for the species. This means that if we choose essential characters, any differences should count as very significant. There are, however, some difficulties with considering essential characters. If one species evolves by changing an essential characteristic, whatever ecological forces supported that change may also apply to other species, and that could lead to parallel evolution. Thus, differences or similarities in essential characters are very relevant to the reconstruction of the general shape of the phylogenetic tree, but they really can't be used to determine the relative lengths of the lines within the tree. Some species have been stable for millions of years. Others evolve very fast.

Irrelevant mutations. We could, on the other hand, consider nonessential characters. Changes in nonessential characters are effected by mutations, mutations that we can call irrelevant. The rate of change of irrelevant mutations should be fairly uniform among species, especially among species that are fairly closely related. Unfortunately, if a character really doesn't matter, it should be very difficult to perceive the value of that character in a species.

Much of the genome sequence of an organism is irrelevant. For example, there are 64 (4^3) different codons for 20 amino acids. Some amino acids are coded by up to four different codons. For these multiply coded amino acids, typically the third nucleotide can take any of the four possible values. In other words, a mutation in this third nucleotide is irrelevant. The DNA can mutate at this site and the resulting protein doesn't change.

By concentrating on irrelevant mutations, not only can the shape of the phylogenetic tree be reconstructed, but the relative lengths of the lines within the phylogenetic tree can also be estimated.

Controls. The diagram above shows a phylogenetic tree. You can ask for a new tree and change the number of extant species as before. These are the controls in the yellow portion of the control panel; they determine the random selection of a model phylogenetic tree.

The pink portion of the control panel allows you to control mutations. There are a number of characters being tracked, 40 by default. Each character has a number of alternate values, 4 by default. You can also set the mutation rate, 100 by default. A value of 100 means that the mutation rate for each character is 100/1000 mutations per unit time interval. That means that along a line in the model tree of length 1 unit, there will be about 1 mutation per 10 characters. If you hike the mutation rate up too far (like 10000), you'll see that it looks like the values of the characters are completely unrelated among the various species. If you set the mutation rate down to 1, you may not see any mutations at all (but any you see will be very important in reconstructing the tree).

The side window of gene sequences. In a resizable side window, you can see the characters of each of the the extant species in a column, the rows being the different characters. The alternative values are shown in different colors. For instance, if all the species have the same color in a row, then that character hasn't mutated, or else they've all mutated to the same new value. (Incidentally, all characters start out with the same character value, denoted red in the window.) If in some row, two of the species show one green, but three show red, then that's some indication that the two species are more closely related to each other than they are to the other three, while the other three species are more closely related to each other then they are to the first two. Of course, that's a statistical indication. The variance in all the characters should be taken into consideration to reconstruct the tree.

Mutations as a measure of time. Let's concentrate on one character to begin with. Our first questions are: What is the probability $p(t)$ that the character has some value at the beginning of a time interval of length t as it does at the end? What is the probability $q(t)$

that the character has one value at the beginning of a time interval of length t but a different value at the end of the interval?

Suppose that there are m different possible alternate values, and suppose that the mutation rate is r mutations per unit time interval.

Some statistical analyisis (which we'll skip) gives us the answers to these questions.

$$p(t) = 1 - \frac{m-1}{m}(1 - e^{-rt})$$

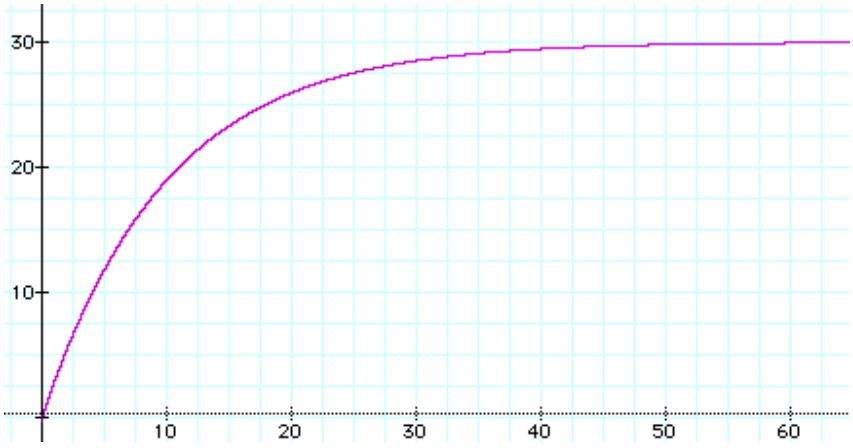
$$q(t) = \frac{1}{m}(1 - e^{-rt})$$

Note that initially, when $t = 0$, $p(0)$ is 1, while $q(0)$ is 0 since there are no mutations in no time. Also, as t approaches infinity, $p(t)$ and $q(t)$ both approach $1/m$, which means that in the long run, each of the m alternative values are equally probable.

Now let's assume that there are n different characters, not just one. Then $E(t)$, the expected number of characters that are not the same at the end of a time interval of length t as they were at the beginning, is $n(m-1)\,q(t)$, that is,

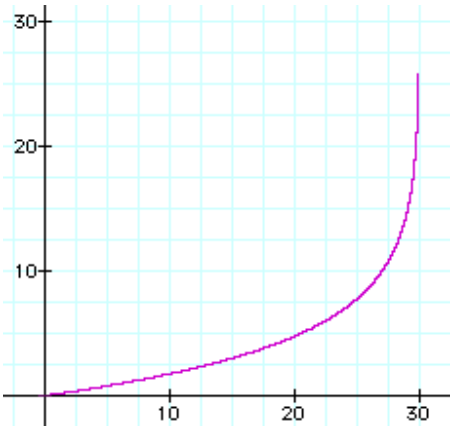
$$E(t) = \frac{n(m-1)}{m}(1 - e^{-rt})$$

Here's the graph of that function when there are $m = 4$ alternate values for each character, there are $n = 40$ characters, and the mutation rate is $r = 0.1$. Time t is shown on the horizontal axis, while the vertical axis gives y, the expected number of character differences.



Note that when t gets large, the expected number of character differences approaches 30.

We can take the inverse function of $y = E(t)$, that is, turn this graph around, to give us an estimate for time t in terms of the observed number of character differences. Let g denote the inverse function. Then



$$t = g(y) = -\tfrac{1}{r}\log\left(1 - \tfrac{m}{n(m-1)}y\right)$$

The base of the logarithm function here is *e*.

The graph of $t = g(y)$ is shown to the right with the same parameter values $m = 4$, $n = 40$, and $r = 0.1$. Note that as the number of expected differences approaches 30, the corresponding time approaches infinity. The observed number of differences may be near the expected number, but it's usually more or less. So the observed number of differences could easily be greater than 30. Should that happen, the best conclusion to make is that the time is very great, but can't be estimated. It would be prudent not to estimate the time when the number of differences is slightly less than 30, too.

Genetic Distances and Reconstruction of Phylogenetic Trees Recently many investigators have used microsatellite DNA loci for studying the evolutionary relationships of closely related populations or species, and some authors proposed new

genetic distance measures for this purpose. However, the efficiencies of these distance measures in obtaining the correct tree topology remains unclear. We therefore investigated the probability of obtaining the correct topology (P(C)) for these new distances as well as traditional distance measures by using computer simulation. We used both the infinite-allele model (IAM) and the stepwise mutation model (SMM), which seem to be appropriate for classical markers and microsatellite loci, respectively. The results show that in both the IAM and SMM CAVALLI-SFORZA and EDWARDS' chord distance (D(C)) and NEI et al.'s D(A) distance generally show higher P(C) values than other distance measures, whether the bottleneck effect exists or not. For estimating evolutionary times, however, NEI's standard distance and GOLDSTEIN et al.'s ($\delta\mu$)(2) are more appropriate than other distances. Microsatellite DNA seems to be very useful for clarifying the evolutionary relationships of closely related populations.

ESTIMATING TIME INTERVALS FROM DISTANCES.

Suppose all we know is how far apart the species are as measured by the number of differences in their characters, that is, the entries in the difference matrix. How can we reconstruct the phylogenetic tree?

First, we can convert the differences to times. The conversion is given by the formula

$$t = g(y) = -\frac{1}{r}\log\left(1 - \frac{m}{n(m-1)}y\right)$$

as explained in the page on mutations.

Of course, we might not be able to reconstruct the actual tree, since the mutations are random and need not reflect the actual distances between species. So a better question is: how can we reconstruct the most likely phylogenetic tree? This question could be made precise using statistical theory, but there is no guarantee that the problem is tractable. We can come up with some promising algorithms that should give trees that aren't too far away from the most likely tree.

A solution: the "minimum" reconstruction method. It seems reasonable that the two species that share the greatest number of characters are the most closely related. That is, the smallest entry in the mutation matrix indicates which two species diverged most

recently. Also, the next smallest entry should indicate which two species diverged just before that. And so forth. That's the idea of the algorithm, but it needs a little clarification. Suppose species 1 and 2 are closest with 4 differences in their characters, and species 1 and 3 are next closest with 6 differences. Then since we conclude that species 1 and 2 diverged most recently, it won't be species 1 and 3 that diverged just before that, rather it will be the ancestral species of 1 and 2 that diverged from species 3 just before that.

Here you see a distance matrix (which is actually a mutation matrix based on a hidden model phylogenetic tree) and the reconstructed phylogenetic tree build from this minimum reconstruction method. You should be able to see how that tree can be derived from the matrix.

If you like, you can ask for a different matrix based on different mutations. Note how radically the reconstructed tree varies with each set of mutations. If there are more characters, then you'll see that the reconstructed matrix depends less on which actual mutations occur.

Two other solutions: the "average" and "maximum" methods. There are two algorithms that are very similar to the minimum reconstruction method. They start out exactly the same by joining the two species that share the most characters. To explain these methods, suppose that species 1 and 2 are closest. Name their ancestral species as species 6. With the minimum method, we effectively determined that the distance between species 6 and any other species such as species 3 was the minimum of the distance from 1 to 3 and the distance from 2 to 3. With the maximum method, instead take the distance from species 6 to species 3 to be the maximum of these two distances. And, of course, for the average method, take the average of those two distances.

These three algorithms go by various names. The one using averages is often called "weighted pair group method using arithmetic averages," **WPGMS**. (There is a closely related algorithm called the "unweighted pair group method using arithmetic averages," **UPGMS**.) The algorithm using minimum distance is called a "single linkage method," and the one using maximum distance is called a "complete linkage method." All three algorithms use a nearest-neighbor technique to construct a tree.

You can select which of these three reconstruction methods to use in the control panel. It's surprising how similar the trees resulting from these three algorithms are. You can see for yourself by selecting a different algorithm.

Estimating time intervals from distance intervals. When there aren't very many mutations, that is, when the numbers in the difference matrix are small, then you expect time to be proportional to the differences. That is, if one pair of species differ by 3 characters, but another pair of species differ by 6 characters, then we would estimate that the common ancestor of the first pair is twice as recent as the common ancestor of the second pair. That means that the length of the line in the tree (as measured by the elapsed time between the beginning of the line and the end of the line) is about proportional to the number of mutations that occur in the evolving species that that line represents.

That's not quite correct, however. When there is more than one mutation in one character for that species during that time interval, only one mutation can be observed. For short time intervals, double mutations are so rare that it makes no difference, but for very long time intervals, they should be taken into consideration. The algorithm used in these demonstrations assumes that the time intevals are short enough to ignore double mutations. If you turn up the mutation rate very high, you'll see the reconstructed tree squished near the top due to these multiple mutations.

UNIT V ADVANCED TOPICS IN BIOINFORMATICS

BIOMOLECULAR COMPUTING

Biomolecular computing, ‘computations performed by biomolecules’, is challenging traditional approaches to computation both theoretically and technologically. Often placed within the wider context of ‘natural’ or even ‘unconventional’ computing, the study of natural and artificial molecular computations is adding to our understanding both of biology and computer science well beyond the framework of neuroscience. The papers in this special theme document only a part of an increasing involvement of Europe in this far reaching undertaking. In this introduction, I wish to outline the current scope of

the field and assemble some basic arguments that biomolecular computation is of central importance to both computer science and biology. Readers will also find arguments for not dismissing DNA Computing as limited to exhaustive search and for a qualitatively distinctive advantage over all other types of computation including quantum computing.

The idea that molecular systems can perform computations is not new and was indeed more natural in the pre-transistor age. Most computer scientists know of von Neumann's discussions of self-reproducing automata in the late 1940s, some of which were framed in molecular terms. Here the basic issue was that of bootstrapping: can a machine construct a machine more complex than itself?

Important was the idea, appearing less natural in the current age of dichotomy between hardware and software, that the computations of a device can alter the device itself. This vision is natural at the scale of molecular reactions, although it may appear utopic to those running huge chip production facilities. Alan Turing also looked beyond purely symbolic processing to natural bootstrapping mechanisms in his work on self-structuring in molecular and biological systems. Purely chemical computers have been proposed by Ross and Hjelmfelt extending this approach. In biology, the idea of molecular information processing took hold starting from the unraveling of the genetic code and translation machinery and extended to genetic regulation, cellular signaling, protein trafficking, morphogenesis and evolution - all of this independently of the development in the neurosciences. For example, because of the fundamental role of information processing in evolution, and the ability to address these issues on laboratory time scales at the molecular level, I founded the first multi-disciplinary Department of Molecular Information Processing in 1992. In 1994 came Adleman's key experiment demonstrating that the tools of laboratory molecular biology could be used to program computations with DNA *in vitro*. The huge information storage capacity of DNA and the low energy dissipation of DNA processing lead to an explosion of interest in massively parallel DNA Computing. For serious proponents of the field however, there really never was a question of brute search with DNA solving the problem of an exponential growth in the number of alternative solutions indefinitely. In a new field, one starts with the simplest algorithms and proceeds from there: as a number of contributions and patents

have shown, DNA Computing is not limited to simple algorithms or even, as we argue here, to a fixed hardware configuration.

After 1994, universal computation and complexity results for DNA Computing rapidly ensued (recent examples of ongoing projects here are reported in this collection by Rozenberg, and Csuhaj-Varju). The laboratory procedures for manipulating populations of DNA were formalized and new sets of primitive operations proposed: the connection with recombination and so called splicing systems was particularly interesting as it strengthened the view of evolution as a computational process. Essentially, three classes of DNA Computing are now apparent: intramolecular, intermolecular and supramolecular. Cutting across this classification, DNA Computing approaches can be distinguished as either homogeneous (ie well stirred) or spatially structured (including multi-compartment or membrane systems, cellular DNA computing and dataflow like architectures using microstructured flow systems) and as either *in vitro* (purely chemical) or *in vivo* (ie inside cellular life forms). Approaches differ in the level of programmability, automation, generality and parallelism (eg SIMD vs MIMD) and whether the emphasis is on achieving new basic operations, new architectures, error tolerance, evolvability or scalability. The Japanese Project lead by Hagiya focuses on intramolecular DNA Computing, constructing programmable state machines in single DNA molecules which operate by means of intramolecular conformational transitions. Intermolecular DNA Computing, of which Adleman's experiment is an example, is still the dominant form, focusing on the hybridization between different DNA molecules as a basic step of computations and this is common to the three projects reported here having an experimental component (McCaskill, Rozenberg and Amos). Beyond Europe, the group of Wisconsin are prominent in exploiting a surface based approach to intermolecular DNA Computing using DNA Chips. Finally, supramolecular DNA Computing, as pioneered by Eric Winfree, harnesses the process of self-assembly of rigid DNA molecules with different sequences to perform computations. The connection with nanomachines and nanosystems is then clear and will become more pervasive in the near future.

- it opens the possibility of a simultaneous bootstrapping solution of future computer design, construction and efficient computation

- It provides programmable access to nanosystems and the world of molecular biology, extending the reach of computation
- it admits complex, efficient and universal algorithms running on dynamically constructed dedicated molecular hardware
- it can contribute to our understanding of information flow in evolution and biological construction
- it is opening up new formal models of computation, extending our understanding of the limits of computation.

The difference with Quantum Computing is dramatic. Quantum Computing involves high physical technology for the isolation of mixed quantum states necessary to implement (if this is scalable) efficient computations solving combinatorially complex problems such as factorization. DNA Computing operates in natural noisy environments, such as a glass of water. It involves an evolvable platform for computation in which the computer construction machinery itself is embedded. Embedded computing is possible without electrical power in microscopic, error prone and real time environments, using mechanisms and technology compatible with our own make up. Because DNA Computing is linked to molecular construction, the computations may eventually also be employed to build three dimensional self-organizing partially electronic or more remotely even quantum computers. Moreover, DNA Computing opens computers to a wealth of applications in intelligent manufacturing systems, complex molecular diagnostics and molecular process control.

The papers in this section primarily deal with Biomolecular Computing. The first contribution outlines the European initiative in coordinating Molecular Computing (EMCC). Three groups present their multidisciplinary projects involving joint theoretical and experimental work. Two papers are devoted to extending the range of formal models of computation. The collection concludes with a small sampler from the more established approach to biologically inspired computation using neural network models. It is interesting that one of these contributions addresses the application of neural modelling to symbolic information processing. However, the extent to which informational biomolecules play a specific role in long term memory and the structuring of the brain, uniting neural and molecular computation, still awaits clarification.

CELLULAR COMPUTING

The recent completion of the first draft of the human genome has led to an explosion of interest in genetics and molecular biology. The view of the genome as a network of interacting computational components is well-established, but researchers are now trying to reverse the analogy, by using living organisms to construct logic circuits.

The cellular computing project is the result of collaboration between teams at the University of Liverpool (Alan Gibbons, Martyn Amos and Paul Sant) and the University of Warwick (David Hodgson and Gerald Owenson). The field emerged in 1994 with the publication of Adleman's seminal article, in which he demonstrated for the first time how a computation may be performed at a molecular level. Our group contributed to the development of the area by describing a generalization of Adleman's approach, proposing methods for assessing the complexity of molecular algorithms, and carrying out experimental investigations into error-resistant laboratory methods. This work quickly confirmed that the massively-parallel random search employed by Adleman would greatly restrict the scalability of that approach. We therefore proposed an alternative method, by demonstrating how Boolean logic circuits may be simulated using operations on strands of DNA.

Our original intention was to implement the Boolean circuit in vitro (ie in a laboratory 'test tube'). However, after much consideration we decided to attempt a rather more ambitious approach, harnessing genetic regulatory mechanisms *in vivo*, within the living *E. coli.* bacterium.

The central dogma of molecular biology is that DNA (information storage) is copied, producing an RNA message (information transfer). This RNA then acts as the template for protein synthesis. The basic 'building blocks' of genetic information are known as genes. Each gene codes for a specific protein which may be turned on (expressed) or off (repressed) when required. In order for the DNA sequence to be converted into a protein molecule, it must be read (transcribed) and the transcript converted (translated) into a protein. Each step of the conversion from stored information (DNA) to protein synthesis (effector) is itself affected or catalyzed by other molecules. These molecules may be enzymes or other compounds (for example, sugars) that are required for a process to continue. Consequently, a loop is formed, where products of one

gene are required to produce further gene products, and may even influence that gene's own expression.

The interaction of various components of the E. coli genome during development may be described in terms of a logic circuit. For example, a gene's expression may require the presence of two particular sugars. Thus, we may view this gene in terms of the Boolean AND function, where the presence or absence of the two sugars represent the two inputs to the function, and the expression or repression of the gene corresponds to the function's output. That gene's product may then be required for the expression (or repression) of another different gene, so we can see how gene products act as 'wires', carrying signals between different 'gates' (genes).

In order to implement our chosen logic circuit, we select a set of genes to represent gates, ensuring that the inter-gene dependencies correctly reflect the connectivity of the circuit. We then insert these genes into a bacterium, using standard tools of molecular biology. These insertions form the most time-consuming component of the entire experimental process, as the insertion of even a single gene can be problematic. However, once the entire set of genes is present in a single colony of bacteria we have an unlimited supply of 'biological hardware' at our disposal. The inputs to the circuit are set by enforcing in the cell's environment the presence or absence of various compounds that affect the expression of the genes representing the first level gates. Then, essentially, the development of the cell and the complex regulatory processes involved, simulate the circuit, without any additional human intervention. This last point is crucial, as most existing proposals for molecular computing require a series of manipulations to be performed by a laboratory technician. Each manipulation reduces the probability of success for the experiment, so the ideal situation is a 'one-pot' reaction, such as the one we propose.

Postdoctoral researcher Gerald Owenson at the lab bench.

PCR equipment, used to amplify DNA samples.

We have recently begun work on simulating a small (3 gate) circuit of NAND gates in vivo. We believe that, within the next three years, the introduction of human-defined logic circuits into living bacteria will be a reality. Of course, such implementations will never rival existing silicon-based computers in terms of speed or efficiency. However, our goal differs from that of a lot of groups in the community, who insist that DNA-based computers may eventually rival existing machines in domains like encryption. Rather, we see the potential applications of introducing logic into cells as lying in fields such as medicine, agriculture and nano-technology. The currenct 'state of the art' in this area has resulted in the reprogramming of E. coli genetic expression to generate simple oscillators. This work, although 'blue sky' in nature, will advance the field to a stage where cells may be reprogrammed to give them simple 'decision making' capabilities.

MICRO ARRAY ANALYSIS

A DNA microarray is a multiplex technology used in molecular biology. It consists of an arrayed series of thousands of microscopic spots of DNA oligonucleotides, called features, each containing picomoles (10^{-12} moles) of a specific DNA sequence, known as *probes* (or *reporters*). These can be a short section of a gene or other DNA element that are used to hybridize a cDNA or cRNA sample (called *target*) under high-stringency conditions. Probe-target hybridization is usually detected and quantified by detection of fluorophore-, silver-, or chemiluminescence-labeled targets to determine relative

abundance of nucleic acid sequences in the target. Since an array can contain tens of thousands of probes, a microarray experiment can accomplish many genetic tests in parallel. Therefore arrays have dramatically accelerated many types of investigation.

In standard microarrays, the probes are attached via surface engineering to a solid surface by a covalent bond to a chemical matrix (via epoxy-silane, amino-silane, lysine, polyacrylamide or others). The solid surface can be glass or a silicon chip, in which case they are colloquially known as an *Affy chip* when an Affymetrix chip is used. Other microarray platforms, such as Illumina, use microscopic beads, instead of the large solid support. DNA arrays are different from other types of microarray only in that they either measure DNA or use DNA as part of its detection system.

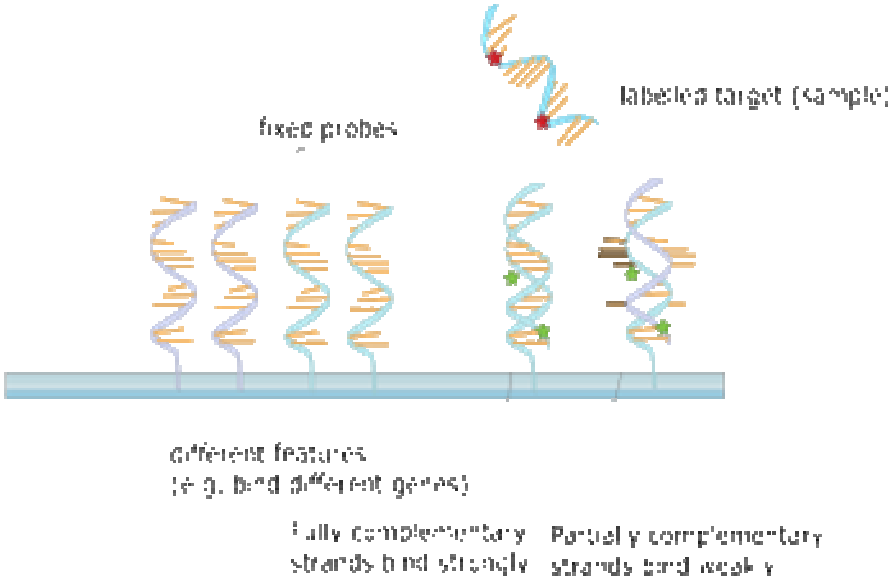
DNA microarrays can be used to measure changes in expression levels, to detect single nucleotide polymorphisms (SNPs), or to genotype or resequence mutant genomes. Microarrays also differ in fabrication, workings, accuracy, efficiency, and cost. Additional factors for microarray experiments are the experimental design and the methods of analyzing the data

History

Microarray technology evolved from Southern blotting, where fragmented DNA is attached to a substrate and then probed with a known gene or fragment. Nucleic Acids Res. 1992 Apr 11;20(7):1679-84. Oligonucleotide hybridizations on glass supports: a novel linker for oligonucleotide synthesis and hybridization properties of oligonucleotides synthesised in situ. Maskos U, Southern EM. The first reported use of this approach was the analysis of 378 arrayed lysed bacterial colonies each harboring a different sequence which were assayed in multiple replicas for expression of the genes in multiple normal and tumor tissue (Augenlicht and Kobrin, Cancer Research, 42, 1088–1093, 1982). This was expanded to analysis of more than 4000 human sequences with computer driven scanning and image processing for quantitative analysis of the sequences in human colonic tumors and normal tissue (Augenlicht *et al.*, Cancer Research, 47, 6017-6021, 1987) and then to comparison of colonic tissues at different genetic risk (Augenlicht *et al.*, Proceedings National Academy of Sciences, USA, 88, 3286-3289, 1991). The use of a collection of distinct DNAs in arrays for expression profiling was also described in 1987, and the arrayed DNAs were used to identify genes

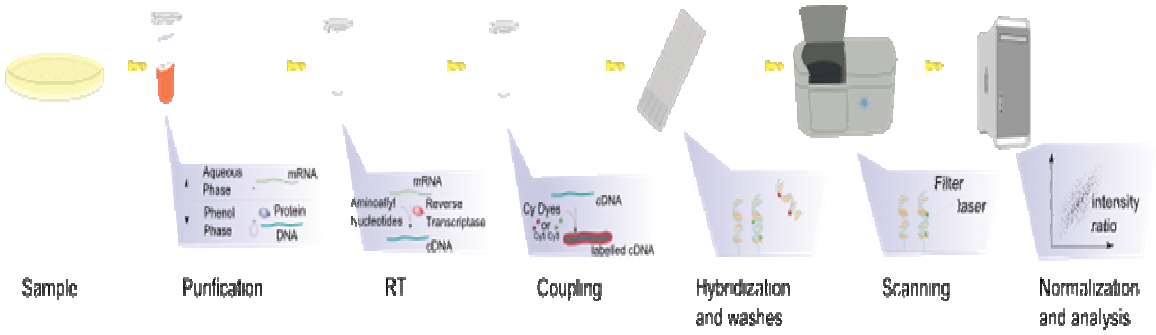
whose expression is modulated by interferon. These early gene arrays were made by spotting cDNAs onto filter paper with a pin-spotting device. The use of miniaturized microarrays for gene expression profiling was first reported in 1995, and a complete eukaryotic genome (*Saccharomyces cerevisiae*) on a microarray was published in 1997.

Principle



The core principle behind microarrays is hybridization between two DNA strands, the property of complementary nucleic acid sequences to specifically pair with each other by forming hydrogen bonds between complementary nucleotide base pairs. A high number of complementary base pairs in a nucleotide sequence means tighter non-covalent bonding between the two strands. After washing off of non-specific bonding sequences, only strongly paired strands will remain hybridized. So fluorescently labeled target sequences that bind to a probe sequence generate a signal that depends on the strength of the hybridization determined by the number of paired bases, the hybridization conditions (such as temperature), and washing after hybridization. Total strength of the signal, from a spot (feature), depends upon the amount of target sample binding to the probes present on that spot. Microarrays use relative quantization in which the intensity of a feature is compared to the intensity of the same feature under a different condition, and the identity of the feature is known by its position. An alternative to microarrays is

serial analysis of gene expression, where the transcriptome is sequenced allowing an absolute measurement.



The step required in a microarray experiment

Uses and types

Two Affymetrix chips

Many types of array exist and the broadest distinction is whether they are spatially arranged on a surface or on coded beads:

- The traditional solid-phase array is a collection of orderly microscopic "spots", called features, each with a specific probe attached to a solid surface, such as glass, plastic or silicon biochip (commonly known as a *genome chip*, *DNA chip* or *gene array*). Thousands of them can be placed in known locations on a single DNA microarray.
- The alternative bead array is a collection of microscopic polystyrene beads, each with a specific probe and a ratio of two or more dyes, which do not interfere with the fluorescent dyes used on the target sequence.

DNA microarrays can be used to detect DNA (as in comparative genomic hybridization), or detect RNA (most commonly as cDNA after reverse transcription) that may or may not be translated into proteins. The process of measuring gene expression via cDNA is called expression analysis or expression profiling. Applications include:

Application or technology	**Synopsis**
Gene expression profiling	In an mRNA or gene expression profiling experiment the expression levels of thousands of genes are simultaneously

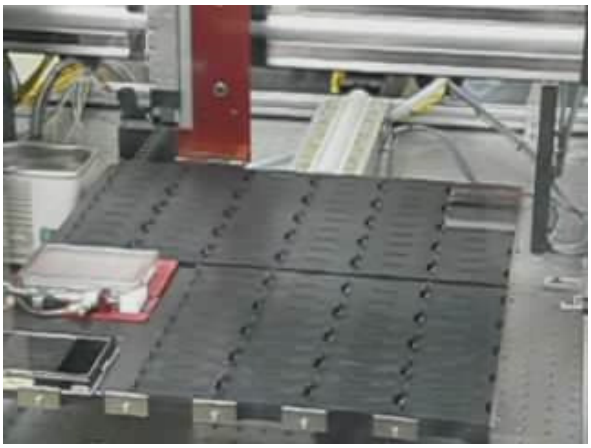
	monitored to study the effects of certain treatments, diseases, and developmental stages on gene expression. For example, microarray-based gene expression profiling can be used to identify genes whose expression is changed in response to pathogens or other organisms by comparing gene expression in infected to that in uninfected cells or tissues.
Comparative genomic hybridization	Assessing genome content in different cells or closely related organisms.
GeneID	Small microarrays to check IDs of organisms in food and feed (like GMO , mycoplasms in cell culture, or pathogens for disease detection, mostly combining PCR and microarray technology.
Chromatin immunoprecipitation on Chip	DNA sequences bound to a particular protein can be isolated by immunoprecipitating that protein (ChIP), these fragments can be then hybridized to a microarray (such as a tiling array) allowing the determination of protein binding site occupancy throughout the genome. Example protein to immunoprecipitate are histone modifications (H3K27me3, H3K4me2, H3K9me3, etc.), Polycomb-group protein (PRC2:Suz12, PRC1:YY1) and trithorax-group protein (Ash1) to study the epigenetic landscape or RNA Polymerase II to study the transcription landscape.
DamID	Analogously to ChIP, genomic regions bound by a protein of interest can be isolated and used to probe a microarray to determine binding site occupancy. Unlike ChIP, DamID does not require antibodies but makes use of adenine methylation near the protein's binding sites to selectively amplify those regions, introduced by expressing minute amounts of protein of interest fused to bacterial DNA adenine methyltransferase.
SNP detection	Identifying single nucleotide polymorphism among alleles within or between populations. Several applications of

	microarrays make use of SNP detection, including Genotyping, forensic analysis, measuring predisposition to disease, identifying drug-candidates, evaluating germline mutations in individuals or somatic mutations in cancers, assessing loss of heterozygosity, or genetic linkage analysis.
Alternative splicing detection	An '*exon junction array* design uses probes specific to the expected or potential splice sites of predicted exons for a gene. It is of intermediate density, or coverage, to a typical gene expression array (with 1-3 probes per gene) and a genomic tiling array (with hundreds or thousands of probes per gene). It is used to assay the expression of alternative splice forms of a gene. Exon arrays have a different design, employing probes designed to detect each individual exon for known or predicted genes, and can be used for detecting different splicing isoforms.
Fusion genes microarray	A Fusion gene microarray can detect fusion transcripts, *e.g.* from cancer specimens. The principle behind this is building on the alternative splicing microarrays. The oligo design strategy enables combined measurements of chimeric transcript junctions with exon-wise measurements of individual fusion partners.
Tiling array	Genome tiling arrays consist of overlapping probes designed to densely represent a genomic region of interest, sometimes as large as an entire human chromosome. The purpose is to empirically detect expression of transcripts or alternatively splice forms which may not have been previously known or predicted.

Fabrication

Microarrays can be manufactured in different ways, depending on the number of probes under examination, costs, customization requirements, and the type of scientific question being asked. Arrays may have as few as 10 probes or up to 2.1 million micrometre-scale probes from commercial vendors.

Spotted vs. in situ synthesised arrays



A DNA microarray being printed by a robot at the University of Delaware

Microarrays can be fabricated using a variety of technologies, including printing with fine-pointed pins onto glass slides, photolithography using pre-made masks, photolithography using dynamic micromirror devices, ink-jet printing, or electrochemistry on microelectrode arrays.

In *spotted microarrays*, the probes are oligonucleotides, cDNA or small fragments of PCR products that correspond to mRNAs. The probes are synthesized prior to deposition on the array surface and are then "spotted" onto glass. A common approach utilizes an array of fine pins or needles controlled by a robotic arm that is dipped into wells containing DNA probes and then depositing each probe at designated locations on the array surface. The resulting "grid" of probes represents the nucleic acid profiles of the prepared probes and is ready to receive complementary cDNA or cRNA "targets" derived from experimental or clinical samples. This technique is used by research scientists around the world to produce "in-house" printed microarrays from their own labs. These arrays may be easily customized for each experiment, because researchers can choose the probes and printing locations on the arrays, synthesize the probes in their own lab (or collaborating facility), and spot the arrays. They can then generate their own labeled samples for hybridization, hybridize the samples to the array, and finally scan the arrays with their own equipment. This provides a relatively low-cost microarray that may be customized for each study, and avoids the costs of purchasing often more expensive commercial arrays that may represent vast numbers of genes that are not of interest to the

investigator. Publications exist which indicate in-house spotted microarrays may not provide the same level of sensitivity compared to commercial oligonucleotide arrays, possibly owing to the small batch sizes and reduced printing efficiencies when compared to industrial manufactures of oligo arrays.

In *oligonucleotide microarrays*, the probes are short sequences designed to match parts of the sequence of known or predicted open reading frames. Although oligonucleotide probes are often used in "spotted" microarrays, the term "oligonucleotide array" most often refers to a specific technique of manufacturing. Oligonucleotide arrays are produced by printing short oligonucleotide sequences designed to represent a single gene or family of gene splice-variants by synthesizing this sequence directly onto the array surface instead of depositing intact sequences. Sequences may be longer (60-mer probes such as the Agilent design) or shorter (25-mer probes produced by Affymetrix) depending on the desired purpose; longer probes are more specific to individual target genes, shorter probes may be spotted in higher density across the array and are cheaper to manufacture. One technique used to produce oligonucleotide arrays include photolithographic synthesis (Affymetrix) on a silica substrate where light and light-sensitive masking agents are used to "build" a sequence one nucleotide at a time across the entire array. Each applicable probe is selectively "unmasked" prior to bathing the array in a solution of a single nucleotide, then a masking reaction takes place and the next set of probes are unmasked in preparation for a different nucleotide exposure. After many repetitions, the sequences of every probe become fully constructed. More recently, Maskless Array Synthesis from NimbleGen Systems has combined flexibility with large numbers of probes.

Two-channel vs. one-channel detection

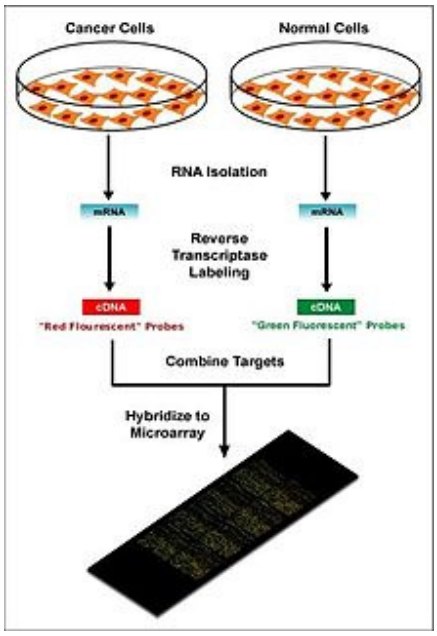


Diagram of typical dual-colour microarray experiment.

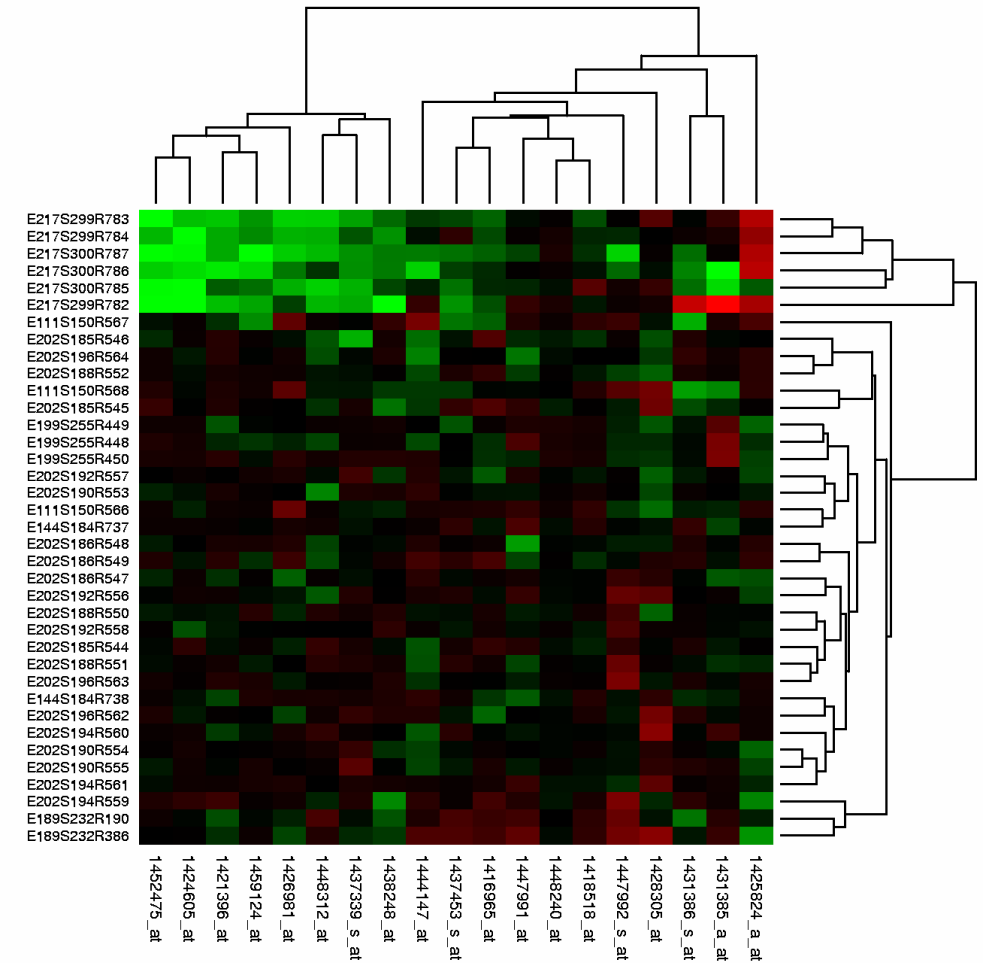
Two-color microarrays or *two-channel microarrays* are typically hybridized with cDNA prepared from two samples to be compared (e.g. diseased tissue versus healthy tissue) and that are labeled with two different fluorophores. Fluorescent dyes commonly used for cDNA labeling include Cy3, which has a fluorescence emission wavelength of 570 nm (corresponding to the green part of the light spectrum), and Cy5 with a fluorescence emission wavelength of 670 nm (corresponding to the red part of the light spectrum). The two Cy-labeled cDNA samples are mixed and hybridized to a single microarray that is then scanned in a microarray scanner to visualize fluorescence of the two fluorophores after excitation with a laser beam of a defined wavelength. Relative intensities of each fluorophore may then be used in ratio-based analysis to identify up-regulated and down-regulated genes.

Oligonucleotide microarrays often carry control probes designed to hybridize with RNA spike-ins. The degree of hybridization between the spike-ins and the control probes is used to normalize the hybridization measurements for the target probes. Although absolute levels of gene expression may be determined in the two-color array in rare instances, the relative differences in expression among different spots within a sample

and between samples is the preferred method of data analysis for the two-color system. Examples of providers for such microarrays includes Agilent with their Dual-Mode platform, Eppendorf with their DualChip platform for colorimetric Silverquant labeling, and TeleChem International with Arrayit.

In *single-channel microarrays* or *one-color microarrays*, the arrays provide intensity data for each probe or probe set indicating a relative level of hybridization with the labeled target. However, they do not truly indicate abundance levels of a gene but rather relative abundance when compared to other samples or conditions when processed in the same experiment. Each RNA molecule encounters protocol and batch-specific bias during amplification, labeling, and hybridization phases of the experiment making comparisons between genes for the same microarray uninformative. The comparison of two conditions for the same gene requires two separate single-dye hybridizations. Several popular single-channel systems are the Affymetrix "Gene Chip", Illumina "Bead Chip", Agilent single-channel arrays, the Applied Microarrays "CodeLink" arrays, and the Eppendorf "DualChip & Silverquant". One strength of the single-dye system lies in the fact that an aberrant sample cannot affect the raw data derived from other samples, because each array chip is exposed to only one sample (as opposed to a two-color system in which a single low-quality sample may drastically impinge on overall data precision even if the other sample was of high quality). Another benefit is that data are more easily compared to arrays from different experiments so long as batch effects have been accounted for. A drawback to the one-color system is that, when compared to the two-color system, twice as many microarrays are needed to compare samples within an experiment.

Microarrays and bioinformatics



Gene expression values from microarray experiments can be represented as heat maps to visualize the result of data analysis.

The advent of inexpensive microarray experiments created several specific bioinformatics challenges:

- the multiple levels of replication in experimental design (Experimental design)
- the number of platforms and independent groups and data format (Standardization)
- the treatment of the data (Statistical analysis)
- accuracy and precision (Relation between probe and gene)
- the sheer volume of data and the ability to share it (Data warehousing)

Experimental design

Due to the biological complexity of gene expression, the considerations of experimental design that are discussed in the expression profiling article are of critical importance if statistically and biologically valid conclusions are to be drawn from the data.

There are three main elements to consider when designing a microarray experiment. First, replication of the biological samples is essential for drawing conclusions from the experiment. Second, technical replicates (two RNA samples obtained from each experimental unit) help to ensure precision and allow for testing differences within treatment groups. The biological replicates include independent RNA extractions and technical replicates may be two aliquots of the same extraction. Third, spots of each cDNA clone or oligonucleotide are present as replicates (at least duplicates) on the microarray slide, to provide a measure of technical precision in each hybridization. It is critical that information about the sample preparation and handling is discussed, in order to help identify the independent units in the experiment and to avoid inflated estimates of statistical significance.

Standardization

Microarray data is difficult to exchange due to the lack of standardization in platform fabrication, assay protocols, and analysis methods. This presents an interoperability problem in bioinformatics. Various grass-roots open-source projects are trying to ease the exchange and analysis of data produced with non-proprietary chips:

- For example, the "Minimum Information About a Microarray Experiment" (MIAME) checklist helps define the level of detail that should exist and is being adopted by many journals as a requirement for the submission of papers incorporating microarray results. But MIAME does not describe the format for the information, so while many formats can support the MIAME requirements, as of 2007 no format permits verification of complete semantic compliance.
- The "MicroArray Quality Control (MAQC) Project" is being conducted by the US Food and Drug Administration (FDA) to develop standards

and quality control metrics which will eventually allow the use of MicroArray data in drug discovery, clinical practice and regulatory decision-making.

- The MGED Society has developed standards for the representation of gene expression experiment results and relevant annotations.

Statistical analysis

Microarray data sets are commonly very large, and analytical precision is influenced by a number of variables. Statistical challenges include taking into account effects of background noise and appropriate normalization of the data. Normalization methods may be suited to specific platforms and, in the case of commercial platforms, the analysis may be proprietary. Algorithms that affect statistical analysis include:

- Image analysis: gridding, spot recognition of the scanned image (segmentation algorithm), removal or marking of poor-quality and low-intensity features (called *flagging*).
- Data processing: background subtraction (based on global or local background), determination of spot intensities and intensity ratios, visualisation of data, and log-transformation of ratios, global or local normalization of intensity ratios.
- Identification of statistically significant changes: t-test, ANOVA, Bayesian method Mann–Whitney test methods tailored to microarray data sets, which take into account multiple comparisons or cluster analysis. These methods assess statistical power based on the variation present in the data and the number of experimental replicates, and can help minimize Type I and type II errors in the analyses.
- Network-based methods: Statistical methods that take the underlying structure of gene networks into account, representing either associative or causative interactions or dependencies among gene products.

Microarray data may require further processing aimed at reducing the dimensionality of the data to aid comprehension and more focused analysis. Other methods permit analysis of data consisting of a low number of biological or technical replicates; for example, the Local Pooled Error (LPE) test pools standard deviations of genes with similar expression levels in an effort to compensate for insufficient replication.

Relation between probe and gene

The relation between a probe and the mRNA that it is expected to detect is not trivial. Some mRNAs may cross-hybridize probes in the array that are supposed to detect another mRNA. In addition, mRNAs may experience amplification bias that is sequence or molecule-specific. Thirdly, probes that are designed to detect the mRNA of a particular gene may be relying on genomic EST information that is incorrectly associated with that gene.

Data warehousing

Microarray data was found to be more useful when compared to other similar datasets. The sheer volume (in bytes), specialized formats (such as MIAME), and curation efforts associated with the datasets require specialized databases to store the data.

SYSTEMS BIOLOGY

Systems biology is a term used to describe a number of trends in bioscience research, and a movement which draws on those trends. Proponents describe systems biology as a biology-based inter-disciplinary study field that focuses on complex interactions in biological systems, claiming that it uses a new perspective (holism instead of reduction). Particularly from year 2000 onwards, the term is used widely in the biosciences, and in a variety of contexts. An often stated ambition of systems biology is the modeling and discovery of emergent properties, properties of a system whose theoretical description is only possible using techniques which fall under the remit of systems biology.

Overview

Systems biology can be considered from a number of different aspects:

- As a **field of study**, particularly, the study of the interactions between the components of *biological systems*, and how these interactions give rise to the function and behavior of that system (for example, the enzymes and metabolites in a metabolic pathway).
- As a **paradigm**, usually defined in antithesis to the so-called reductionist paradigm (biological organisation), although fully consistent with the scientific method. The distinction between the two paradigms is referred to in these quotations:

"The reductionist approach has successfully identified most of the components and many of the interactions but, unfortunately, offers no convincing concepts or methods to understand how system properties emerge...the pluralism of causes and effects in biological networks is better addressed by observing, through quantitative measures, multiple components simultaneously and by rigorous data integration with mathematical models" Science

"Systems biology...is about putting together rather than taking apart, integration rather than reduction. It requires that we develop ways of thinking about integration that are as rigorous as our reductionist programmes, but different....It means changing our philosophy, in the full sense of the term" Denis Noble

- As a series of **operational protocols used for performing research**, namely a cycle composed of theory, analytic or computational modelling to propose specific testable hypotheses about a biological system, experimental validation, and then using the newly acquired quantitative description of cells or cell processes to refine the computational model or theory. Since the objective is a model of the interactions in a system, the experimental techniques that most suit systems biology are those that are system-wide and attempt to be as complete as possible. Therefore, transcriptomics, metabolomics, proteomics and high-throughput techniques are used to collect quantitative data for the construction and validation of models.
- As the application of dynamical systems theory to molecular biology.
- As a **socioscientific phenomenon** defined by the strategy of pursuing integration of complex data about the interactions in biological systems from diverse experimental sources using interdisciplinary tools and personnel.

This variety of viewpoints is illustrative of the fact that systems biology refers to a cluster of peripherally overlapping concepts rather than a single well-delineated field. However the term has widespread currency and popularity as of 2007, with chairs and institutes of systems biology proliferating worldwide.

History

Systems biology finds its roots in:

- the quantitative modeling of enzyme kinetics, a discipline that flourished between 1900 and 1970,

- the mathematical modeling of population growth,
- the simulations developed to study neurophysiology, and
- control theory and cybernetics.

One of the theorists who can be seen as one of the precursors of systems biology is Ludwig von Bertalanffy with his general systems theory. One of the first numerical simulations in biology was published in 1952 by the British neurophysiologists and Nobel prize winners Alan Lloyd Hodgkin and Andrew Fielding Huxley, who constructed a mathematical model that explained the action potential propagating along the axon of a neuronal cell. Their model described a cellular function emerging from the interaction between two different molecular components, a potassium and a sodium channels, and can therefore be seen as the beginning of computational systems biology. In 1960, Denis Noble developed the first computer model of the heart pacemaker.

The formal study of systems biology, as a distinct discipline, was launched by systems theorist Mihajlo Mesarovic in 1966 with an international symposium at the Case Institute of Technology in Cleveland, Ohio entitled "Systems Theory and Biology."

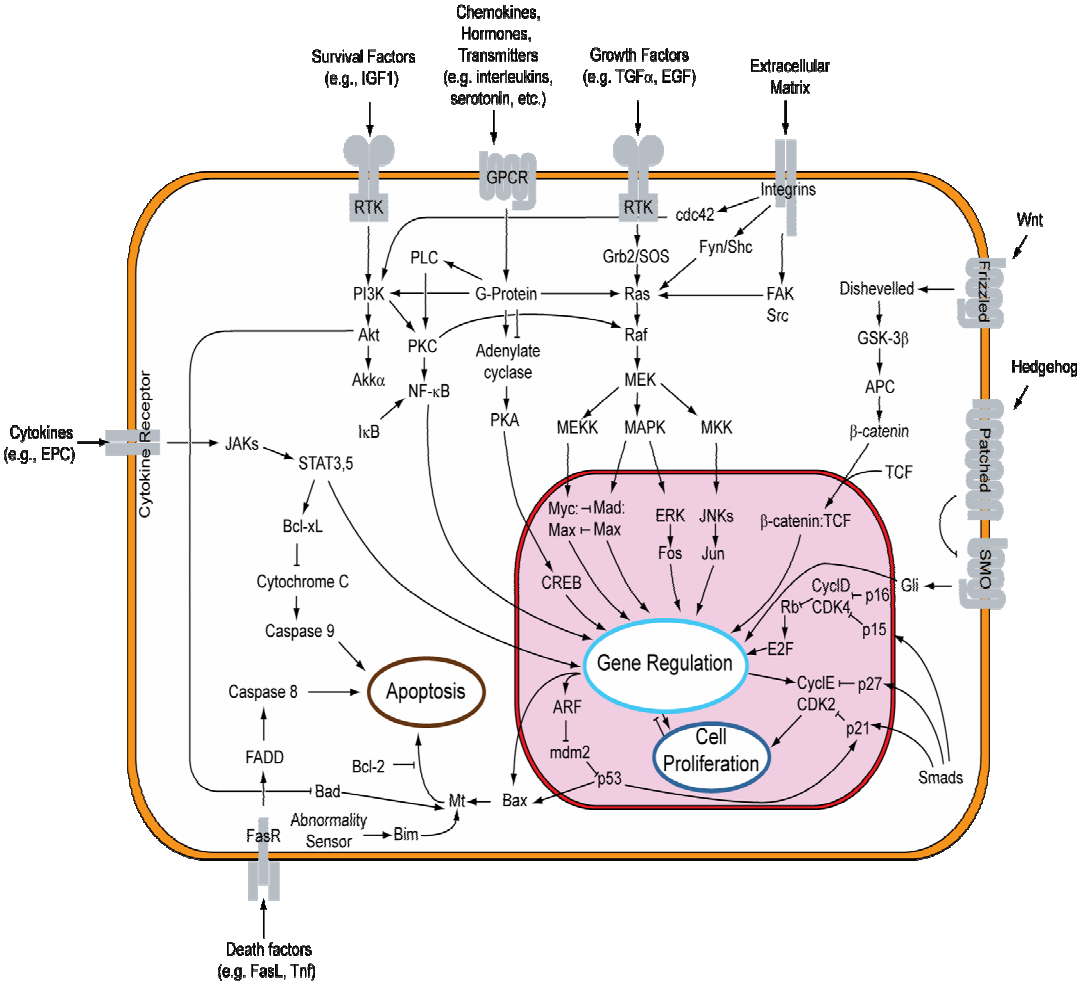
The 1960s and 1970s saw the development of several approaches to study complex molecular systems, such as the Metabolic Control Analysis and the biochemical systems theory. The successes of molecular biology throughout the 1980s, coupled with a skepticism toward theoretical biology, that then promised more than it achieved, caused the quantitative modelling of biological processes to become a somewhat minor field.[1]

However the birth of functional genomics in the 1990s meant that large quantities of high quality data became available, while the computing power exploded, making more realistic models possible. In 1997, the group of Masaru Tomita published the first quantitative model of the metabolism of a whole (hypothetical) cell.

Around the year 2000, after Institutes of Systems Biology were established in Seattle and Tokyo, systems biology emerged as a movement in its own right, spurred on by the completion of various genome projects, the large increase in data from the omics (e.g. genomics and proteomics) and the accompanying advances in high-throughput experiments and bioinformatics. Since then, various research institutes dedicated to systems biology have been developed. As of summer 2006, due to a shortage of people in

systems biology several doctoral training centres in systems biology have been established in many parts of the world.

Disciplines associated with systems biology



Overview of signal transduction pathways

According to the interpretation of Systems Biology as the ability to obtain, integrate and analyze complex data from multiple experimental sources using interdisciplinary tools, some typical technology platforms are:

- Phenomics: Organismal variation in phenotype as it changes during its life span.
- Genomics: Organismal deoxyribonucleic acid (DNA) sequence, including intra-organisamal cell specific variation. (i.e. Telomere length variation etc.).

- Epigenomics / Epigenetics: Organismal and corresponding cell specific transcriptomic regulating factors not empirically coded in the genomic sequence. (i.e. DNA methylation, Histone Acetelation etc.).
- Transcriptomics: Organismal, tissue or whole cell gene expression measurements by DNA microarrays or serial analysis of gene expression
- Interferomics: Organismal, tissue, or cell level transcript correcting factors (i.e. RNA interference)
- Translatomics / Proteomics: Organismal, tissue, or cell level measurements of proteins and peptides via two-dimensional gel electrophoresis, mass spectrometry or multi-dimensional protein identification techniques (advanced HPLC systems coupled with mass spectrometry). Sub disciplines include phosphoproteomics, glycoproteomics and other methods to detect chemically modified proteins.
- Metabolomics: Organismal, tissue, or cell level measurements of all small-molecules known as metabolites.
- Glycomics: Organismal, tissue, or cell level measurements of carbohydrates.
- Lipidomics: Organismal, tissue, or cell level measurements of lipids.

In addition to the identification and quantification of the above given molecules further techniques analyze the dynamics and interactions within a cell. This includes:[1]

- Interactomics: Organismal, tissue, or cell level study of interactions between molecules. Currently the authoritative molecular discipline in this field of study is protein-protein interactions (PPI), although the working definition does not preclude inclusion of other molecular disciplines such as those defined here.
- Fluxomics: Organismal, tissue, or cell level measurements of molecular dynamic changes over time.
- Biomics: systems analysis of the biome.

The investigations are frequently combined with large scale perturbation methods, including gene-based (RNAi, mis-expression of wild type and mutant genes) and chemical approaches using small molecule libraries. Robots and automated sensors enable such large-scale experimentation and data acquisition. These technologies are still emerging and many face problems that the larger the quantity of data produced, the lower the quality. A wide variety of quantitative scientists (computational biologists,

statisticians, mathematicians, computer scientists, engineers, and physicists) are working to improve the quality of these approaches and to create, refine, and retest the models to accurately reflect observations.

The systems biology approach often involves the development of mechanistic models, such as the reconstruction of dynamic systems from the quantitative properties of their elementary building blocks. For instance, a cellular network can be modelled mathematically using methods coming from chemical kinetics and control theory. Due to the large number of parameters, variables and constraints in cellular networks, numerical and computational techniques are often used.

Other aspects of computer science and informatics are also used in systems biology. These include:

- New forms of computational model, such as the use of process calculi to model biological processes (notable approaches include stochastic π-calculus, BioAmbients, Beta Binders, BioPEPA and Brane calculus) and constraint-based modeling.
- Integration of information from the literature, using techniques of information extraction and text mining.
- Development of online databases and repositories for sharing data and models, approaches to database integration and software interoperability via loose coupling of software, websites and databases, or commercial suits.
- Development of syntactically and semantically sound ways of representing biological models