

2019

Theoretical Biology and Bioinformatics
Utrecht University

Bas E. Dutilh & Can Keşmir



[BIOINFORMATIC DATA ANALYSIS]

Reader for the Bioinformatics part of the Systems Biology course.

A PDF of this reader can be downloaded for free and in full color at: <http://tbb.bio.uu.nl/BDA>

You can also find the errata through the link above.

Contents

1.	Computers in biology: systems biology and bioinformatics.....	5
1.1.	The Bioinformatic Data Analysis course.....	5
1.2.	Biology as a data science.....	6
1.3.	Reproducibility	6
1.4.	DNA sequencing.....	6
1.5.	File formats.....	8
1.6.	Databases: data and metadata	10
1.7.	Types of computational biologists	13
1.7.1.	Noriko Cassman, PhD student, Microbial Ecology, Netherlands Institute of Ecology	14
1.7.2.	Jayne Hehir-Kwa, assistant professor, Translational Genomics, Radboudumc.....	15
1.7.3.	Mattias de Hollander, bioinformatician, Netherlands Institute of Ecology	15
1.7.4.	Robin van der Lee, PhD student, Comparative and Integrative Genomics, Radboudumc ...	16
1.7.5.	Walter Pirovano, head, Bioinformatics Department, BaseClear.....	16
1.7.6.	Linsey Raaijmakers, PhD student, Mass Spectrometry and Proteomics, UU.....	17
1.7.7.	Daan Speth, PhD student, Microbiology, Radboud University	17
1.7.8.	Wouter Touw, PhD student, Centre for Molecular and Biomolecular Informatics	18
1.7.9.	Marcel van Verk, Senior scientist, Bioinformatics / Plant-Microbe Interactions, UU.....	18
1.7.10.	Renske Vroomans, PhD student, Computational Developmental Biology, UU	19
1.7.11.	Juline Walter, PhD student, Marine Microbiology, Federal University of Rio de Janeiro...	19
1.8.	Reading and videos	19
1.9.	Questions and exercises.....	20
2.	Talking to computers.....	23
2.1.	Algorithms and variables.....	23
2.2.	Computer languages	24
2.2.1.	Simple questions with logical answers: TRUE or FALSE?	25
2.2.2.	Functions	26
2.3.	Data types	28
2.3.1.	Numbers	28
2.3.2.	Text	28
2.3.3.	Variables that can contain multiple values	29
2.4.	Examples	29
2.4.1.	Minimum	29
2.4.2.	Euclid's algorithm.....	30
2.5.	R-specific functions	31
2.5.1.	Help.....	31
2.5.2.	Reading a table.....	31
2.5.3.	Plotting data.....	32
2.6.	Computer power: CPU and RAM.....	33
2.7.	Learning more about scripting.....	33
2.8.	Questions and exercises.....	34
3.	Dimensions of data.....	35
3.1.	Second generation DNA sequencing as a profiling technology.....	35
3.2.	Visualization of data	36
3.3.	Similarity between relative abundance profiles	38
3.4.	Distance measures.....	39
3.5.	Correlation.....	40
3.6.	Clustering algorithms	41
3.7.	Newick tree format.....	42

3.8.	Reading and videos	44
3.9.	Exercises	44
4.	Phylogenetic trees and genetic relatedness	48
4.1.	Phylogenetic trees	48
4.2.	Reading phylogenetic trees	50
4.3.	Rooting: providing a direction to the time axis	51
4.4.	Phylogenetic distance and the molecular clock	53
4.5.	Types of homology	54
4.6.	Gene trees and species trees	55
4.7.	Orthology and the biological function of genes	56
4.8.	Horizontal gene transfer (HGT)	57
4.9.	Reading and videos	58
4.10.	Questions and exercises	58
5.	Sequence conservation	62
5.1.	Sequence alignments	62
5.2.	The amino acids	63
5.3.	Sequence conservation and the genetic code	64
5.4.	Consensus sequences and ambiguity characters	65
5.5.	Sequence logos	66
5.6.	Microsatellites	68
5.7.	Reading	69
5.8.	Questions and exercises	69
6.	Quantifying sequence similarity	73
6.1.	Sequence identity and the identity matrix	73
6.2.	Blocks substitution matrix (BLOSUM)	73
6.3.	Point accepted mutations (PAM) matrices	76
6.4.	Transition and transversion mutations in DNA	76
6.5.	A substitution matrix is a model of evolution	77
6.6.	Mutational saturation and evolutionary distance	77
6.7.	Reading	79
6.8.	Questions and exercises	79
7.	Algorithms for sequence alignment	81
7.1.	Visualizing genome evolution using dot-plots	81
7.2.	Gaps	82
7.3.	Needleman-Wunsch global alignment algorithm	83
7.4.	Smith-Waterman local alignment algorithm	86
7.5.	Multiple sequence alignment	87
7.6.	How to use sequence alignment?	89
7.7.	Reading	89
7.8.	Questions and exercises	89
8.	Searching for similar sequences	93
8.1.	The Basic Local Alignment Search Tool BLAST	93
8.2.	Masking low-complexity regions	94
8.3.	Sequence searches: blastn, (discontiguous) megablast, and blastp	94
8.4.	Sensitive homology searches with sequence profiles	96
8.5.	Translated searches: blastx, tblastn, and tblastx	97
8.6.	Ultra-fast search tools that skip alignment extension	98
8.7.	Expectation values (E-values)	98

8.8.	Reading and videos	99
8.9.	Questions and exercises.....	100
9.	Phylogenetic inference	105
9.1.	Clustering using a distance matrix	105
9.2.	Unweighted Pair Group Method with Arithmetic mean (UPGMA)	105
9.3.	Neighbor joining (NJ)	107
9.4.	Maximum parsimony (MP)	109
9.5.	Maximum Likelihood (ML)	110
9.6.	How to search all possible trees?	110
9.7.	Bootstrapping.....	111
9.8.	Reading.....	113
9.9.	Questions and exercises.....	113
10.	Glossary of bioinformatic jargon	117
11.	References	123
12.	Answers to the questions	126
12.1.	Answers to questions in Section 1.9.....	126
12.2.	Answers to questions in Section 3.9.....	129
12.3.	Answers to questions in Section 4.10.....	133
12.4.	Answers to questions in Section 5.8.....	136
12.5.	Answers to questions in Section 6.8.....	138
12.6.	Answers to questions in Section 7.8.....	140
12.7.	Answers to questions in Section 8.9.....	144
12.8.	Answers to questions in Section 9.9.....	147

1. Computers in biology: systems biology and bioinformatics

The term systems biology (see http://en.wikipedia.org/wiki/Systems_biology*) emphasizes the fact that to fully understand the functioning of biological systems such as cells, organs, organisms, or ecosystems, it is not enough to simply assign functions to the components that make up these systems, such as molecules, cells, organs, or species, respectively. Instead, to fully understand the complexity of life at all these different levels, we need to analyze the organization, interactions, and control of the many smaller components of a biological system in an integrated way. These systems can respond dynamically, and often have nonlinear or even complex behavior.

The term bioinformatics was first defined at Utrecht University by Ben Hesper and Paulien Hogeweg as “the study of informatic processes in biotic systems” (Hesper and Hogeweg 1970), which can be seen as bioinformatics in the broad sense (*sensu lato*) because they viewed the whole biological world as a huge information integrating system (if you want to learn more about this, you should attend the Computational Biology course by Prof. Hogeweg, which she teaches to bachelor and master students). Today, the word bioinformatics is more often used in the stricter sense (*sensu stricto*) as the application of computational techniques (informatics) to analyze biological data.

1.1. The Bioinformatic Data Analysis course

In the Bioinformatic Data Analysis section of the Systems Biology course, we will teach you how to deal with biological data by introducing you to the field of bioinformatics. Because we will mainly work with sequence data, we will first think about how sequences evolve. Then, we will discuss several critical bioinformatic data analysis techniques that are important for all biologists, including nucleotide and protein sequence alignments, database searches, data clustering, and phylogenetic analysis. A good biologist should be able to understand, and ultimately ask complex questions in a structured and logical way. This is very important because if there is one thing that characterizes the biological world, it is the presence of noise (that, plus exceptions that prove a rule). Thus, we will discuss what it means to “ask questions” in the most structured way possible, such that they can be answered by a computer. To do this, we will introduce you to data analysis scripts. Scripting allows you to make a computer do tedious repetitive work for you, powering your analyses to the limit of your imagination – and the available data of course. While computer power may have limited some research questions in the past, recent technological developments have triggered biologists to ask ever bolder questions. Thus, smart bioinformatic scripts and new data analysis methods will always be driving the frontiers of biological science. Finally, Chapter 10 contains a glossary of bioinformatic jargon. Like most scientific fields, bioinformatics has a lot of terms that you need to know to be able to communicate with peers. You will probably know many of the terms already because they are often not only used in bioinformatics but also in other fields of biology. Together, the goal of this course is to enable you to understand and apply bioinformatic data analysis when you need it in your future career.

* Wikipedia and Google are NOT primary sources of information and should never be cited as such in scientific literature. They are, however, extremely useful tools to search for primary sources of information, or to quickly catch up on a certain topic. Not to get lost in the face of an overload of information is one of the most important skills of a bioinformatician, apply it when browsing the Internet!

1.2. Biology as a data science

Thanks to technological advances, high-throughput measurements of many different types of biological information can now be made. Complete genomes of many organisms are available, ranging in size from the smallest known viruses (*Circoviridae* genomes consist of ~1,800 base pairs (bp), and encode only two proteins) to the 22,061,874,510 bp genome of the loblolly pine (*Pinus taeda*). Metagenomics is charting microbial communities such as the human microbiome or the microbiomes of the world's oceans that consist of hundreds to thousands of interacting microbial and viral species. And sequences are not the only type of biological data that are blooming; the data "big bang" is expanding rapidly in almost every subfield of biology. As a result, biological databases are growing rapidly, some doubling their size in a matter of months or years. For example, the database of the International Nucleotide Sequence Database Collaboration (INSDC) doubles in size approximately every 18 months.

As data collection projects continue to advance, the emphasis progressively switches from the accumulation of data to its interpretation. Now more than ever, our ability to make new biological discoveries depends on our ability to combine diverse data sets. For example, sequence data must be integrated with structure and function data, gene expression data, pathway data, phenotypic and clinical data, and so forth. All that is where bioinformatic data analysis comes in. In today's biology, the challenge is no longer to get the data, but rather to analyze and integrate it in meaningful and innovative ways, obtaining new insights and understanding the biological system as a whole.

1.3. Reproducibility

In most scientific fields, data that represents measurements or inferences of sorts are critical to falsify, or provide support for theories about the way the world functions. This data needs to be consistently and reproducibly analyzed, and shared with the world in an understandable way so that everyone knows they can trust the results, and use the knowledge to benefit their work. By using computers for their analyses, bioinformaticians are able to perform hundreds, thousands, or even billions of analyses just as easily as two or five, as long as they are all performed in the same way – because that is where computers excel: in doing the exact same thing over and over again. This is, of course, the pinnacle of reproducibility, and one of the major reasons for the success of computers in biology and in the rest of the world. Finally, computers are not only indispensable to consistently analyze and integrate big data sets, but often also allow results to be obtained much faster than traditional wet-lab experiments.

1.4. DNA sequencing

So, where does all this data come from? As mentioned, much of the data that a bioinformatician will see is sequence data, and this is also the main type of data that we discuss in this course. Thanks to technological advances (Figure 1), it is now relatively cheap to determine the sequence of DNA in a biological sample. For just a few hundred, sometimes a few thousand euros, you can sequence millions, sometimes billions of nucleotides. The length of these sequencing reads depends on the sequencing platform used (Figure 11), and ranges from several tens (SOLiD), to several hundred (Illumina, Ion Torrent), up to tens of thousands (Nanopore, PacBio) of nucleotides per read. A read is a stretch of sequence that is read from a single molecule of DNA by a sequencing machine.

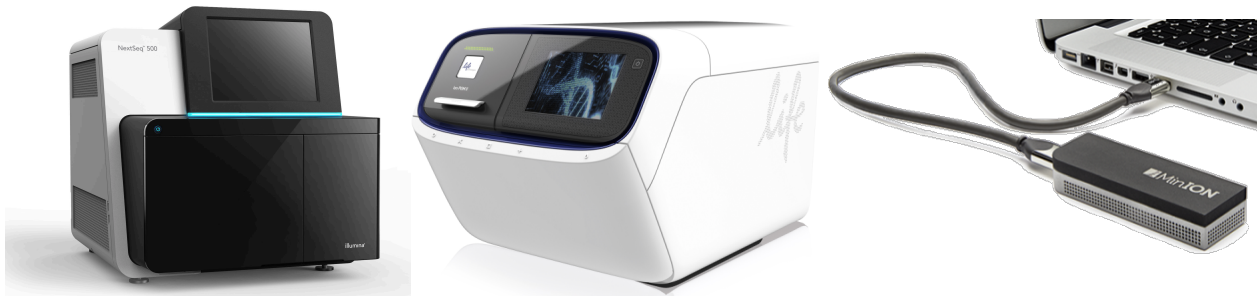


Figure 1. DNA sequencing machines. From left to right: Illumina, Ion Torrent, and Nanopore.

As shown in Figure 2, an important application of DNA sequencing is to obtain complete genomes that can be obtained after many short reads are assembled into longer sequences. However, the types of information that can be obtained with DNA sequencers are much more diverse than complete genomes alone. For example, in Chapter 3 we will discuss transcriptomics and metagenomics. In transcriptomics, RNA is isolated from a sample, and then reverse-transcribed to DNA, which is subsequently sequenced to determine which genes were expressed and in what relative abundances. In metagenomics, the DNA from a whole microbial community is isolated at once, and allows the genome sequences of the micro-organisms to be identified directly in their natural environment. Many of these naturally occurring micro-organisms are only distantly related to the model species that we know well, like the bacterium *Escherichia coli* (Hug et al. 2016).

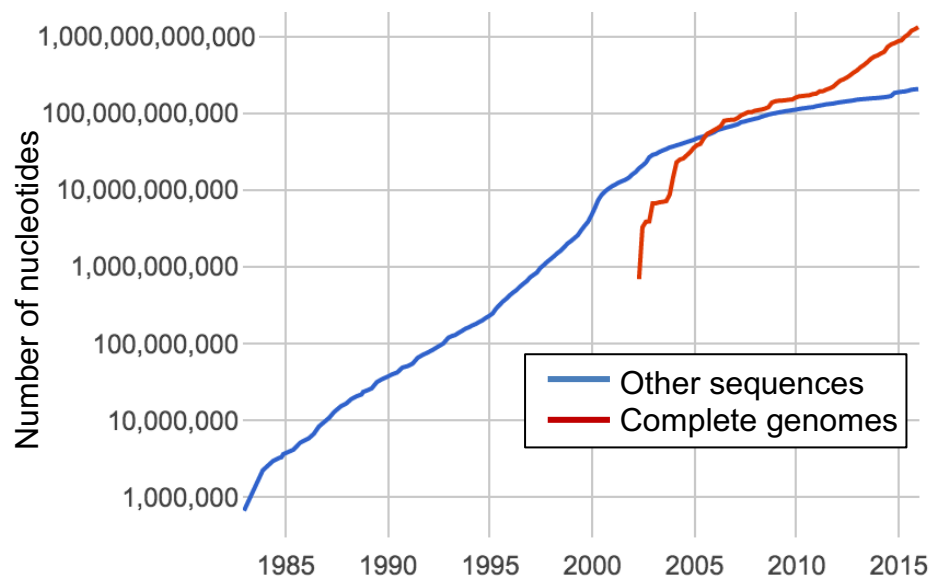


Figure 2. Volume of DNA information in Genbank. Source: <http://www.ncbi.nlm.nih.gov/genbank/statistics>.

Data tends to be highly skewed. We have a lot of knowledge on some model systems and model organisms, such as human (25 million sequences in Genbank), mouse (2.7 million), *Arabidopsis thaliana* (2.3 million), and *Escherichia coli* (4.8 million), while other organisms are much less investigated or even completely unknown. Often, many of these are sequences of the same gene, for example if the gene has been included in a population level diversity study it is possible that hundreds or thousands of sequences of that gene in the same species exist in the database. It is important to realize this when investigating data, especially if you want to draw conclusions about the abundance of certain genes, species, or functions in the biosphere.

1.5. File formats

Bioinformaticians spend a lot of their time analyzing and interpreting the biological data that results from high-throughput measurements, biological experiments, or other sources (see Section 1.7). Analyses often consist of multiple steps, for example: (1) “quality control” of DNA or RNA sequencing results; (2) identifying which genes or organisms were sequenced; (3) identifying differences between samples in a case/control analysis; (4) assessing statistical significance of the result. While there are computer programs available for many of these steps, a good bioinformatician will always keep a close eye on the data as it is being processed, trying to understand which observations represent an interesting trend or result, and which are biases in the experiment or artifacts of the analysis. This means they are always looking at the information in files.

Different types of biological data or information are often stored in files with a specific format. A file format is a standard way that information is encoded for storage in a computer file. Many file formats in bioinformatics are based on plain text, which means that you can easily look at the information in the file by opening it in a plain text editor, such as Notepad++ (Windows), or Sublime Text (Mac). There are two main file formats for sequence information: FASTA and FASTQ. Sequence information is one of the most basic types of biological information which we will use a lot during this course. FASTA files contain one or more sequences, each sequence with a unique identifier (ID) that allows it to be found (Figure 3). According to the central dogma, sequence information flows from DNA to RNA to protein, and all three sequence types can be stored in the FASTA format. The file extension of FASTA files is not fixed, and could include .fasta, .fa, .fas, .fna (for FASTA Nucleic Acids: DNA or RNA sequences), .faa (for FASTA Amino Acids: protein sequences), and even .txt. FASTQ files also contain sequences with an ID, but include quality information about every single nucleotide. File extensions are mostly .fastq or .fq.

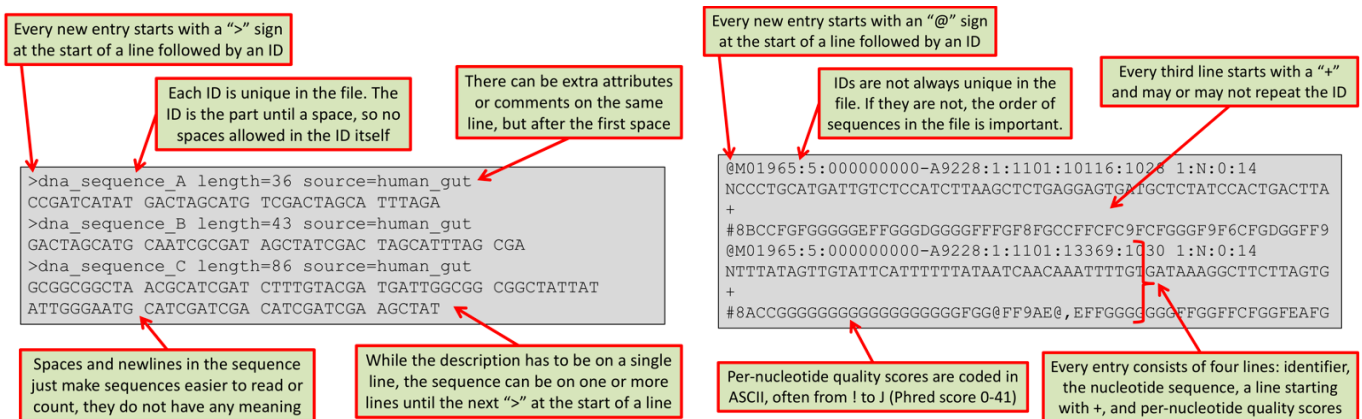


Figure 3. Some of the most important properties of the FASTA (left) and FASTQ (right) file formats.

FASTQ files may contain reads obtained from a DNA sequencing machine (Figure 1). In this case, the quality scores represent the confidence with which the nucleotide was measured by the machine. For example, in Illumina sequencing the four nucleotides A, C, G, and T are encoded with four different colors, and these colors are measured by the camera in four different channels

(<http://youtu.be/jFCD8Q6qSTM>). Software in the sequencing machine estimates the probability that a nucleotide was wrongly measured, based on the purity of the color signal in these channels. This probability is represented by the Phred score, a score between 0 (probability of 10^{-0} that the nucleotide is wrong, i.e. a 100% error rate) and 41 (probability of $10^{-4.1}$ that the nucleotide is wrong, i.e. a 0.0079% error rate). In FASTQ files, Phred scores are encoded with one letter per nucleotide, for which the ASCII code is used. In the most common encoding, the Phred score +33 is the value of the ASCII character (see Table 1), e.g. a Phred score of 30 (0.01% error rate) translates to an ASCII character with the value 63, i.e. a “?”.

Table 1. ASCII characters from ! to J (ASCII values 33-74) encode Phred quality scores from 0 to 41. This corresponds to a probability of 10^{-0} (100%) to $10^{-4.1}$ (<0.01%) that the nucleotide at that position is wrong. Phred+33 encoding is e.g. used by traditional Sanger sequencing and by the latest Illumina machines.

!	33	0	'	39	6	-	45	12	3	51	18	9	57	24	?	63	30	E	69	36
"	34	1	(40	7	.	46	13	4	52	19	:	58	25	@	64	31	F	70	37
#	35	2)	41	8	/	47	14	5	53	20	;	59	26	A	65	32	G	71	38
\$	36	3	*	42	9	0	48	15	6	54	21	<	60	27	B	66	33	H	72	39
%	37	4	+	43	10	1	49	16	7	55	22	=	61	28	C	67	34	I	73	40
&	38	5	,	44	11	2	50	17	8	56	23	>	62	29	D	68	35	J	74	41

ASCII character	ASCII value	Phred score
-----------------	-------------	-------------

Another important file format in data science is the tab-separated values format, which has the file extension .tsv or .txt (and sometimes erroneously .csv, which actually stands for comma-separated values). Also known as tab-delimited text files, .tsv files have a very simple format that stores data as a matrix of values, where the values on the same row are separated by tabs. Sometimes, the first row of the file contains column headers, and/or the first column may contain row headers or identifiers. If every row has the same number of tab-separated columns, the data matrix is said to be “rectangular”. Spreadsheet programs such as Excel or Open Office should be able to easily import .tsv files, and export spreadsheets to the .tsv format. Because .tsv files are based on raw text, you can also edit and view them in plain text editor, but the columns may seem scrambled if the word in a given column is wider than the display width of a tab (as you can see in the example in Figure 4). Note that there is nothing wrong with this file, it is just a bit hard for a human to view the information.

Name	First_Name	Last_Name	Age	Weight	Gender	Married
Patient01	Adriana	Mattos	35	64.5	F	TRUE
Patient02	Matthias	Uyttenhaghe	56	78.5	M	FALSE
Patient03	Jan	Aerntsz	34	50.3	M	TRUE
Patient04	Amin	Abboud	67	89.6	M	TRUE
Patient05	Janet	Thomlinson	31	87.5	F	FALSE
Patient06	Frederique	Vos	73	69.4	F	TRUE

Figure 4. Example of a tab-delimited text file. Note how spaces are avoided in the column heads by replacing them with underscores (_). This file can be downloaded from <http://tbb.bio.uu.nl/BDA/fig4.tsv>.

* Two often used, and often confused Latin abbreviations are i.e. that stands for *id est*, meaning “that is”, and e.g. that stands for *exempli gratia*, meaning “for example”. Both are a way of introducing additional information, but while e.g. offers an example, i.e. explains or rewords what was said before. See <http://www.elearnenglishlanguage.com/blog/english-mistakes/eg-vs-ie/> for a discussion of their correct usage.

In general, there seems to be a trade-off between a file being readable by humans, and it being able to store a lot of data in a small file size. The example of the FASTQ file format illustrates how bioinformaticians encode information to decrease the file size and save disk space. With the continued exponential growth of biological data (Figure 2), file formats are increasingly compressed, meaning they are transformed into a binary file format that is no longer readable by humans, but often takes much less space on the hard disk (see Section 2.6).

1.6. **Databases: data and metadata**

Data files are collected in databases where the information is stored in a computer readable format, and can be retrieved by researchers or by other computers. Some databases are also called data repositories if they function as a place where primary datasets can be stored and retrieved, such as the raw output from DNA sequencing machines, for example in FASTQ file format (1.5). Often, databases are made openly accessible online so they can be accessed by any interested researcher around the world. Such publically accessible databases are important instruments in science, and research funding agencies including the Netherlands Organization for Scientific Research (NWO) require that data that is generated through funded research projects is made openly accessible at the end of the project. Data sharing between scientists accelerates the speed of scientific progress, because it allows new investigations to build on previous discoveries. This is known as the "Fourth Paradigm of Data-Driven Scientific Discovery" (Hey, Tansley, and Tolle 2009).

A key aspect of data storage is the availability of suitable metadata: information about the dataset that allows anyone to understand how the data was generated, even if they were not involved in generating the data. As an example, Table 2 lists some of the metadata fields that are minimally required for a genome sequence. Although metadata collection is not always as well organized, similar minimum requirements have been defined for several other types of biological data as well.

Table 2. Selected fields from the Minimum Information about a (Meta) Genome Sequence (MIGS/MIMS) metadata standard as specified by the Genomic Standards Consortium (GSC) (Field et al. 2008).

Item	Definition
Submitted to INSDC	Sequences must be submitted to one of the databases of the International Nucleotide Sequence Database Collaboration (INSDC).
Investigation type	Eukaryote, bacteria, virus, plasmid, organelle, or metagenome.
Genetic lineage	Name and taxonomic identifier of organism (not for metagenome).
Project name	Name of the project within which the sequencing was organized.
Geographic location	The geographical origin of the sample as defined by country, latitude and longitude, depth or altitude.
Collection date	The date and time of sampling.
Environment (biome)	Biomes are defined based on factors such as plant structures, leaf types, plant spacing, and other factors like climate.
Ploidy	The ploidy level of the genome (e.g. haploid, diploid, tetraploid).
Known pathogenicity	To what is the entity pathogenic.
Nucleic acid extraction	Link to a literature reference or a standard operating procedure.
Sequencing method	Sequencing method used; e.g. Sanger, Illumina, Nanopore, etc.

As a bioinformatician, the information contained in biological databases may be your greatest resource. To answer their biological question, a data-minded biologist or bioinformatician immediately thinks about which data would help to answer it. For example, when asked whether a deletion mutation of a given gene X in a patient could contribute to an observed phenotype in the embryonic development of the limbs, you would want to consult OMIM, a database of known disease-related mutations, and Pubmed, a database of biomedical literature. There, you might discover if anything is already known about the involvement of gene X in limb formation.

However, it is possible that these searches come up negative, for example because gene X was never analyzed in detail before, or because the existing analyses still left you with questions. Knowing how to analyze large-scale datasets, you could do more to unravel the function of this gene. If you had a physiological mindset, you could check in which tissue the gene is expressed by consulting the Human Protein Atlas, a database that contains information about the tissue-specific localization of all human proteins. If you had an evolutionary mindset, as we will illustrate in Chapters 3 and 5, you might ask: is this gene present in other *Tetrapoda* (four-legged animals), and is its protein sequence conserved? To address this question, you could check a database of genome sequences such as RefSeq or Ensembl, to see whether gene X is present in other primates, mammals, or animals besides *Tetrapoda*. If you had a molecular mindset, you could identify protein domains in sequence X using the Pfam and SMART databases, giving you information on what molecular functions it could have, like kinase activity, or calcium binding (see Section 5.1). If you had a genetic mindset, you could investigate the expression of gene X with other genes by co-expression analysis (see Section 3.1). If other co-expressed genes are related to embryogenesis or limb formation, this would provide additional hints about the possible involvement of gene X in these processes as well. These are just some examples of bioinformatic answers to biological questions, that provide important hints when designing specific follow-up experiments to investigate gene X.

For some types of questions, bioinformaticians have made databases that can be easily searched online. There are also questions that require data mining, an important skill of a bioinformatician for which knowledge of scripting is necessary (Chapter 2). It is impossible to list all biological databases here. Often, searches on Google or Pubmed can be a great help to identify relevant sources of information, but it is also good to keep track of some of the most important databases in your field. Some examples of important databases are:

1. Refseq (<http://www.ncbi.nlm.nih.gov/refseq>) is a database of genome sequences that contains the high-quality genome sequences of important reference species like *Homo sapiens* and *Escherichia coli*.
2. The largest databases of nucleotide sequences (genes, genomes, and sequence fragments) are those of the three partners of the International Nucleotide Sequence Database Collaboration (INSDC) that serve three corners of the world: the European Bioinformatics Institute of the European Molecular Biology Laboratory (EBI-EMBL, <http://www.ebi.ac.uk/services>), Genbank at the National Center for Biotechnology Information (NCBI, <http://www.ncbi.nlm.nih.gov/genbank>), and the DNA Data Bank of Japan (DDBJ, <http://www.ddbj.nig.ac.jp>). The data in these databases is mirrored, and in 2014 consisted of over 654 billion nucleotides in sequences from over 280 thousand formally described species (Benson et al. 2014).

3. The raw output of DNA sequencing machines consists of vast datasets of unannotated reads that are stored in repositories like the European Nucleotide Archive (ENA, <http://www.ebi.ac.uk/ENA>). Researchers have often only analyzed one aspect of these datasets, so a lot remains to be discovered by so-called “data recycling”.
4. The UniProt KnowledgeBase (<http://www.uniprot.org>) contains protein sequences together with their functional annotation. EggNOG (<http://eggnoг.embl.de>) and Pfam (<http://pfam.xfam.org>) are also databases of proteins that are grouped in protein families, and contain multiple sequence alignments of each protein family. EggNOG groups the proteins according to their family tree into orthologous groups in a wide range of organisms (see Section 4.5).
5. Transcription factor binding sites can be found in JASPAR (<http://jaspar.genereg.net>).
6. The Protein Data Bank (PDB, <http://www.rcsb.org/pdb>) contains information about the three-dimensional structure of proteins.
7. The Kyoto Encyclopedia of Genes and Genomes (KEGG, <http://www.genome.jp/kegg>) contains information about groups of proteins that work together in a biological system, for example in metabolic pathways.
8. The NCBI Taxonomy database (<http://www.ncbi.nlm.nih.gov/Taxonomy>) contains information about the taxonomy of all organisms from viruses to humans.
9. Scientific literature can be searched in PubMed (<http://www.ncbi.nlm.nih.gov/pubmed>) or Google Scholar (<http://scholar.google.com>). You can search articles by author or by keyword.
10. The Search Tool for the Retrieval of Interacting of Genes/Proteins (STRING, <http://string-db.org>) contains information about interactions between proteins based on a range of evidence types (see Figure 5). Examples include experimental evidence, protein-protein links in other databases, links in scientific articles (text mining), and co-expression of the proteins across transcriptomes (see Section 3.3). This database exploits the concept of “guilt-by-association”, where unknown proteins can be associated to known functions by links with proteins whose function is known. This approach works quite well to discover new genes involved in similar processes.

The information in a database is stored in a computer readable format, so that it can be retrieved automatically when the database is accessed. To facilitate this access, each data element in the database has a unique ID or accession number that can also be used to refer to it. Optimally, the ID for a certain data element is conserved in time, so that the exact same element can be retrieved from the database at any later point in time. However, this is not always consistently done by all databases, so when accessing data in any database, it is always a good idea to record the date and version of the database, and mention this in scientific communications, for example when writing an article or a report.

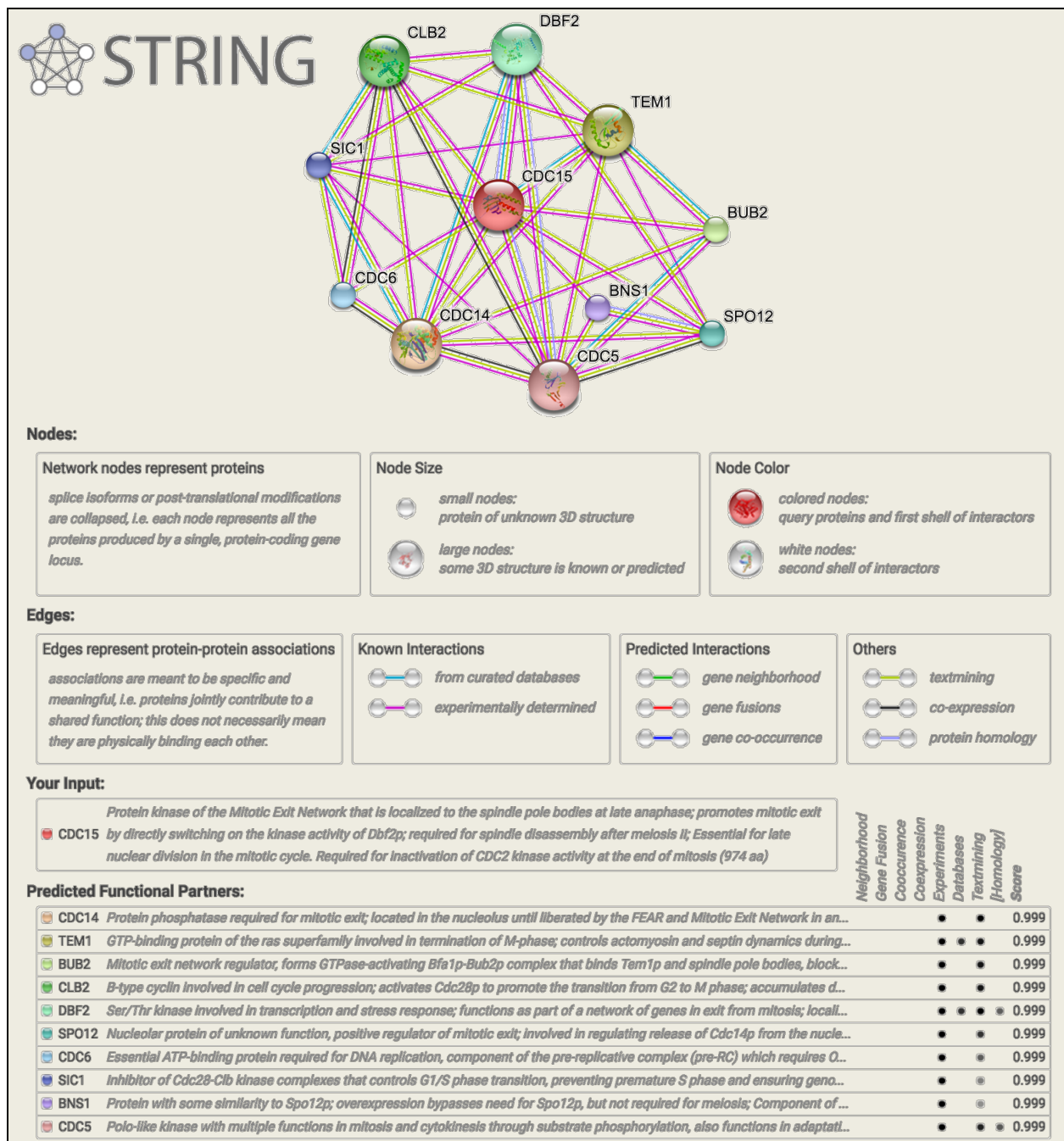


Figure 5. Output of a STRING search for proteins that interact with CDC15, a protein kinase in *Saccharomyces cerevisiae*. Only direct interactors (first shell) are shown.

1.7. Types of computational biologists

Although all computational biologists use computers for their research, there is no such thing as “the typical computational biologist” or “the typical bioinformatician”. Sure, there are specific organizations such as the Utrecht Bioinformatics Center (UBC) in Utrecht, the Bioinformatics and Systems Biology (BioSB) research school in The Netherlands, and the International Society for

Computational Biology (ISCB) internationally. Sure, there are specific journals, mailing lists, special interest groups, and blogs. But the computer in general, and more specifically bioinformatics, have become indispensable tools in all areas of biology, and as a result, computational biologists can have very different, sometimes cross-disciplinary roles. In the paragraphs below, we have asked a few of them from different areas to write a “self-portrait” of their job (dated early 2016), addressing questions like: What kind of lab are you in? What is the position you have in the lab? Which skills do you need? What is your main added value? Why do you love what you're doing? We hope these portraits will give you a better perspective on what a computational biologist does in everyday life, and that it will radiate some of the enthusiasm they have for their work.

This Section 1.7 is meant to give you an outlook on the career perspectives of young computational biologists. Read it for fun, but don't worry, you don't have to know this for the exam!

1.7.1. Noriko Cassman, PhD student, Microbial Ecology, Netherlands Institute of Ecology

My introduction to bioinformatics was during the second year of my Bachelor study in Biochemistry, when I took a course in general genetics. My professor mentioned off-hand that there would probably be lots of jobs coming up in this field because of new sequencing technologies that produce a tidal wave of information. This tidbit led me to take a course in computer science to see if I liked programming, which I did! Talking to computers and figuring out code puzzles was fun. Now that I think about it, maybe I liked it because I had already been tinkering with blogs and HTML back in



high school. When I had my B.Sc. in hand, I combined my fascination with microbiology and my penchant for programming, and completed a master's thesis in marine viral ecology, during which I got involved with metagenomics to study microbes and viruses in the wild. Now I work at the Netherlands Institute of Ecology (NIOO-KNAW), where I analyze metagenomic data from Brazilian sugarcane fields and Dutch grasslands to address microbial ecology questions like "How are bacteria that live in soils fertilized with nitrogen different from bacteria in soils without fertilizer?" and "Which soil bacteria, based on the potential function of their genes, can emit the greenhouse gas nitrous oxide?" What I like about bioinformatics is that the same concepts and tools can be applied to data from many different environments, making a little knowledge go a long way.



1.7.2. Jayne Hehir-Kwa, assistant professor, Translational Genomics, Radboudumc

The Human Genetics department is a multidisciplinary team which aims to bring technology innovations into diagnostics, for example new genome sequencing technologies. The group is made up of a dry lab (the bioinformaticians), a wet lab, and clinical geneticists. When a new application comes along, it is my job to investigate how the data analysis should be done, whether there are already existing data analysis tools in available or if new tools needs to be developed, and what needs to happen to make it robust enough for clinical diagnostics. This means that good communication skills are needed: I need to understand what is needed and explain this to people with diverse backgrounds. Solid computer programming skills are a must, because tools that we write go into production and are maintained by other team members. And sometimes I need to a bit of lateral thinking for new applications of existing data analysis, this is my added value to the group. What is great about the Translational Genomics group is that that we are busy with innovations and improving diagnostics, which can have a big and direct impact on patient's lives.



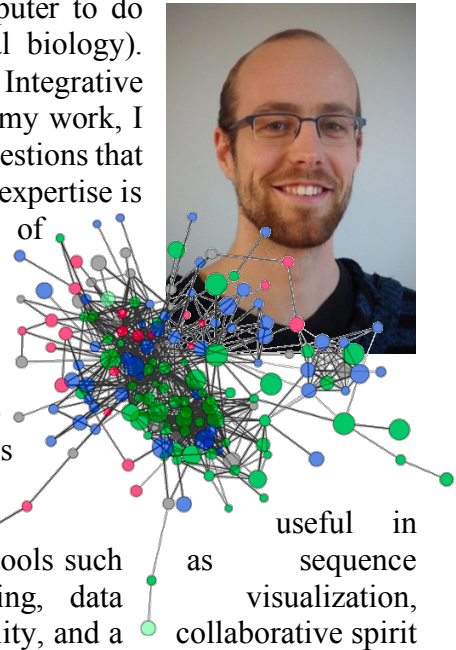
1.7.3. Mattias de Hollander, bioinformatician, Netherlands Institute of Ecology

I work as an embedded bioinformatician at the Netherlands Institute of Ecology (NIOO-KNAW) working together with one other bioinformatician and many molecular biologists, ecologists and microbiologists (PhDs/postdocs) from the Microbial Ecology department. After gaining a lot of hands-on knowledge on bioinformatics during my BSc at the University of Applied Sciences in Leiden, I continued in this field with a more environmental focus during my MSc in Wageningen. Luckily at that time (2010), high-throughput DNA sequencing was having a boost and after my internship at the NIOO there was enough sequencing data to stay appointed. Where we first mainly worked with pyrosequencing data (a DNA sequencing technology that is now no longer supported), nowadays we are mostly analyzing data from the more recent Illumina and Ion Torrent platforms. My main work is to build computational data analysis pipelines for metagenomics that people at the NIOO use to analyze environmental soil data. In these pipelines, I combine programs for quality control, clustering, comparing DNA sequences to databases, assembly, and gene annotation. On a daily basis I teach people how to use Linux on our bioinformatics server, I give advice for projects, keep all the software up-to-date, and add new tools and enable others to run their code via pre-made workflows, for example with R-studio. I enjoy helping people with their first steps into bioinformatics, work together in a project to bring it to a next level, and to solve bugs :)



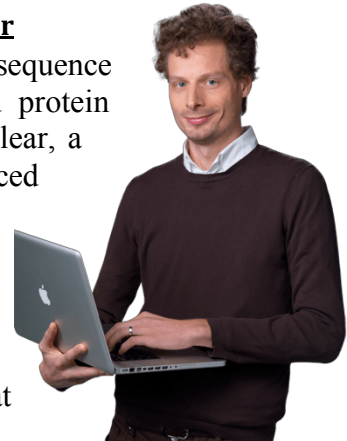
1.7.4. **Robin van der Lee, PhD student, Comparative and Integrative Genomics, Radboudumc**

Type of bioinformatician: “Molecular biologist using the computer to do research” (with the emphasis on the *biology* in computational biology). Profile: I am currently finishing my PhD in the Comparative and Integrative Genomics group under supervision of Prof. Martijn Huynen. In my work, I make full use of my training in biology to understand and solve questions that can only be solved by using computational approaches. My main expertise is the collection, handling, preparation, analysis and integration of large-scale molecular data sets such as genome and protein sequences, gene expression data, protein interactions and genetic variation data. I apply this information for example to identify novel components of biological systems like the antiviral immune response and the cilium organelle. Bioinformatics allows me to spend much of my time learning about new biology and new types of data, while developing new analysis techniques for investigating a wide range of biological systems. Skills that are useful in my work include familiarity with bioinformatics resources and tools such as sequence alignment and phylogenetic analysis, computer programming, data visualization, statistics, as well as the ability to organize my work well, flexibility, and a collaborative spirit because I work together with experimental biologists, geneticists, medical doctors, etc.



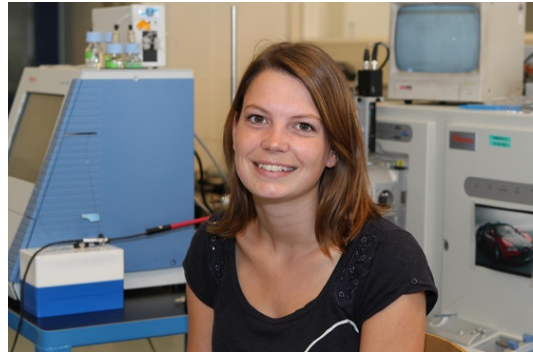
1.7.5. **Walter Pirovano, head, Bioinformatics Department, BaseClear**

During my Master and PhD (1999-2005), I mainly worked on protein sequence analysis, such as developing algorithms for sequence alignment and protein structure prediction. In 2006 I joined the DNA service provider BaseClear, a company that aimed to expand its portfolio with the recently introduced Next-Generation Sequencing (NGS) technology. Still sequence analysis, yet completely different! When the human genome project was concluded in 2003 and novel technologies appeared, the need for strong bioinformatics to handle big genomic datasets became apparent. I was thus appointed to take care of the data analysis and management. Of course, I could not do this alone, so currently we have a great bioinformatics team that analyzes thousands of samples per year on all major NGS platforms and builds tools to analyze these data. BaseClear is now recognized as a high-quality service lab, both in the national and European biotechnology market. We offer complete solutions to unravel viral, bacterial, and fungal (meta)genomes, from DNA isolation to data generation, and (importantly!) data interpretation. We offer customers interactive online solutions which are coupled to databases where their data is safely stored but can easily be retrieved. These systems are a great help for solving questions about comparative genomics, enzyme screening, or metagenomics. My job requires a technical knowledge from different perspectives, management skills and a vision to guide the bioinformatics team in the right direction, and the ability to maintain account relations with customers.



1.7.6. Linsey Raaijmakers, PhD student, Mass Spectrometry and Proteomics, UU

When I started looking for bachelor studies after finishing high school, I was very interested in biology. So as a logical decision, I picked the study Biomedical Sciences. It was during my internships that I noticed the large amount of data being generated by machines in the lab, and the big role of data analysis in scientific research. My interest grew in getting the most out of data acquired from lab experiments and I continued with a master's in bioinformatics. The combination of these two studies gave me an excellent knowledge for performing various analyses during my PhD. As a PhD student, I'm working in the Mass Spectrometry and Proteomics lab of Prof. Albert Heck, focusing mostly on melanoma and the question why patients become resistant to treatment after a couple of months. As one of the few bioinformaticians in the lab, I play a big role in different on-going projects, and my days can be very variable. I can be busy with small things like changing file formats or looking for patterns in files, or bigger things like statistical analysis and data visualization. And being a bioinformatician doesn't necessarily mean you spend whole days behind the computer – my experience from my bachelor allows me to perform some experiments in the lab as well.

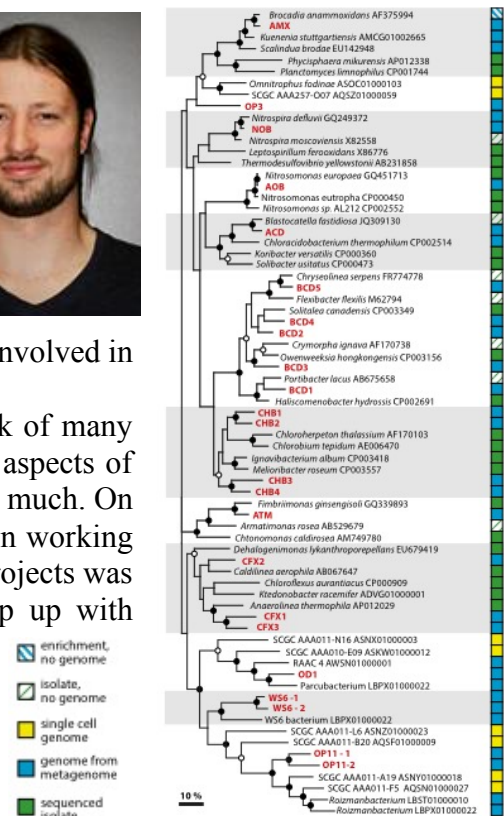


1.7.7. Daan Speth, PhD student, Microbiology, Radboud University

I work as one of the few bioinformaticians in a microbiology lab. Within microbiological research, bioinformatics has many different applications. Metagenomics, my specialty, is focused on discovery of novel microbial diversity. Over the past few years, metagenomic analyses have shown that there is a tremendous 'hidden diversity' that was not appreciated using other methods. This 'hidden diversity' has enormous potential and makes this an incredibly exciting time to be involved in microbiology.

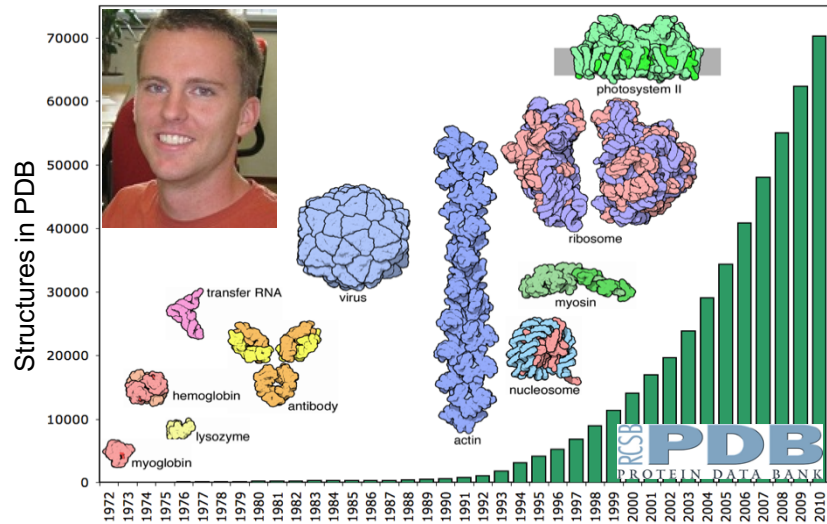


I have my own projects but am also involved in the work of many colleagues. On the one hand, as I am interested in many aspects of microbiology, I enjoy this diversity of research topics very much. On the other hand, as a PhD student the balancing act between working on my own projects or working with colleagues on their projects was always a challenge. On top of the regular work, I keep up with developments in bioinformatic analysis tools, test them, and make sure that the useful tools run on our lab compute server. Other than basic scripting, I don't write my own pipelines and computational tools, but for me this is sufficient for most applications.



1.7.8. Wouter Touw, PhD student, Centre for Molecular and Biomolecular Informatics

The Protein Data Bank (PDB) is a big database that currently holds more than 100,000 3D models of many different types of macromolecules. These structure models allow us to analyze things like the interactions between proteins and drugs, nucleic acids, or other proteins. In my PhD project, under supervision of Prof. Gert Vriend I study protein structures by integrating information from all the available 3D molecular



structures and discovering patterns. Using these patterns, I develop bioinformatic methods that improve the quality of protein structure models. This is great fun because it allows me to combine my knowledge of structural biology and data science: I need to be able to understand the molecular function of a protein, and I also need statistics and computer programming skills. As a PhD student I am also involved in teaching and supervising students. I enjoy that a lot!

1.7.9. Marcel van Verk, Senior scientist, Bioinformatics / Plant-Microbe Interactions, UU

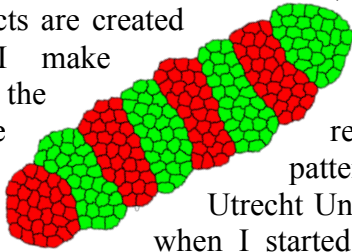
Already during my bachelor's in plant biology, I was very much interested in molecular biology and the complexity of genes and genomes. Because of this fascination I chose to do a PhD where I could study molecular regulation mechanisms of plants. During this period, I realized that a vast amount of data is deposited in databases (at that time mainly microarrays), and this data is directly available to help answering almost any biological question you might have. This has helped me during my PhD, to setup experiments with new target genes that we could have never discovered using conventional wet lab molecular biology. Since that moment I have continued to specialize in the analysis of large amounts of transcript and genomic data, with Next



Generation Sequencing (NGS) data in a wide variety of species from bacteria, yeast, plants, drosophila, zebrafish, to mouse and humans. Within my current position I perform my own research but also support many other groups with their experimental setup, sequencing data generation, analysis, and interpretation. It is therefore very important to have good social skills, a proper understanding of both biology and bioinformatics, and good organizational skills to manage the many projects and large amounts of data. To illustrate the large amounts of data I use, if I would accidentally print the data for a project that I work on, this would result in a stack of paper of 80 kilometers high! For me, what makes my work so fascinating is that I feel I am constantly solving extremely large and complicated puzzles using bioinformatics to help progress the understanding of biology. It's like finding a needle that we don't even know what it looks like in a gigantic haystack; and I will be first one to unravel this.

1.7.10. Renske Vroomans, PhD student, Computational Developmental Biology, UU

The general field I work in is often described as Evo-Devo – the evolution of development. I work in the Computational Developmental Biology group of Kirsten ten Tusscher, where I study the evolution of segmentation, the process by which the somites (vertebrae) of vertebrates and the segments of insects are created in the developing embryo. To do this, I make computer models that simulate the evolution of gene regulation in embryos, and the resulting tissue growth and segmental patterning. I did a bachelor's in biology here at Utrecht University, and I learned to write computer code when I started a Master project in the Theoretical Biology and Bioinformatics group. My skills are a combination of biological knowledge and computer programming. I love this work because modelling teaches you to look at problems from a different angle. In my case, it allows me to piece together information from several experimental studies into a comprehensive understanding of a developmental process. I certainly learnt that often, the answer to a biological question is counter-intuitive!



1.7.11. Juline Walter, PhD student, Marine Microbiology, Federal University of Rio de Janeiro

I have always wanted to get inside of the bioinformatics world, but this was not part of the biology curriculum in Rio de Janeiro. I was trained as a microbiologist and my work focuses on the coral reefs of the Abrolhos islands (South Bahia State, Brazil). Currently I am in the final step of my PhD under supervision of Prof. Fabiano Thompson. Computational challenges in biology frequently involve the analysis of big datasets, which we generated for my project and need to analyze in high-throughput. That is the current biology landscape. In the past year, I have had the great opportunity to work for one year in Utrecht together with the bioinformaticians, which has made me realize the computational challenges behind the genomics and metagenomics projects that I am involved in. I am always eager to learn and try new things, but I have to admit that for me bioinformatics was not easy at all. I am proud to have started and hope to keep working on the transition from being a wet-lab microbiologist to being a dry-lab microbiologist.



1.8. Reading and videos

- Obligatory: Introductory video about DNA sequencing: <http://youtu.be/jFCD8Q6qSTM>

- Obligatory: how to retrieve sequences for an organism using NCBI: <http://youtu.be/sK3ykyInU8o>
- Obligatory: how to download sequences (watch the first 2.5 minutes): <http://www.youtube.com/watch?v=OC74-DpkWjE&list=PL8FD4CC12DABD6B39>
- Obligatory: how to obtain genomic sequence around a gene using NCBI: <http://youtu.be/RHz2nZbzjpA>
- Obligatory: Wikipedia page about DNA sequencing: http://en.wikipedia.org/wiki/DNA_sequencing
- Optional: “All biology is computational biology” by Florian Markowetz: <http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.2002050>
- Optional: “Why has this really common virus only just been discovered?” by Ed Yong: <http://phenomena.nationalgeographic.com/2014/07/24/why-has-this-really-common-virus-only-just-been-discovered/>

1.9. **Questions and exercises**

1. Bioinformatics can include all the following, except:
 - a. Using computer programs to align DNA sequences.
 - b. Analyzing protein interactions in a species.
 - c. Searching the genomes of hundreds of bacteria for genes that are linked to pathogenesis.
 - d. Using molecular biological techniques to express specific gene products.
 - e. Development of computer-based tools for genome analysis.
 - f. Analyzing social interactions in insects.
 - g. Use of mathematical tools to make sense of biological systems.
 - h. Discovering a new species of virus living in the human intestinal tract.
 - i. Designing a good experimental setup with sufficient case/control samples.
2. What might be the most reasonable experimental approach to determine the probable function of a gene in humans?
 - a. Genetically engineer a mouse with a copy of this gene and examine its phenotype.
 - b. Look for a gene with a highly similar sequence in another species, prepare a knockout in that species and look for the phenotypic consequences.
 - c. Prepare a genetically engineered bacterial culture with the gene inserted and assess which new protein is synthesized.
 - d. Mate two organisms from another species that are heterozygous for the normal and mutated gene sequences and examine the phenotype of their offspring.
3. What might be the most reasonable bioinformatical approach to determine the probable function of a gene in humans?
 - a. Search the Human Proteome Atlas to discover in which tissue the protein that is encoded by the gene is expressed in humans.
 - b. Search Genbank for genes with a similar sequence in other species where the function of the gene has been experimentally determined.
 - c. Search Google for information about the function of the gene.
4. What is a read?

- a. A single sequence of nucleotides (DNA) or protein (amino acids) as present in a database such as Genbank.
 - b. A fragment of a genome or metagenome sequence obtained after DNA sequence assembly.
 - c. A stretch of sequence that is read from a single molecule of DNA by a sequencing machine.
5. Which of the following best describes metagenomics?
 - a. The science that combines many individual genomes to gain new knowledge about their function and evolution.
 - b. The science that studies very large genomes that were sequenced with next-generation sequencing.
 - c. The science that studies the effects of epigenetic markers on gene expression and ultimately on phenotype.
 - d. The science that studies the combined genetic material of microbial communities.
 6. Which of the following most correctly describes the Whole Genome Shotgun (WGS) approach for sequencing a genome?
 - a. Cloning large genome fragments into very large vectors such as Yeast Artificial Chromosomes (YACs), followed by sequencing.
 - b. Randomly sequencing genome fragments, followed by bioinformatic assembly and scaffolding.
 - c. Cloning fragments of various sizes into vectors, followed by sequencing and bioinformatic assembly.
 7. Place the following DNA sequencing machines in order from short to longest read lengths: Illumina, Ion Torrent, Nanopore, Sanger. How would you use the data coming from these DNA sequencing machines to get an entire genome sequence?
 8. What is proteomics?
 - a. The linkage of each gene to a particular protein.
 - b. The study of the full protein set in a genome or sample.
 - c. The totality of the functional possibilities of a single protein.
 - d. The study of how amino acids are ordered in a protein.
 - e. The study of how a single gene activates many proteins.
 9. One of the problems with the WGS approach is that it may underestimate the size of the genome. Which of the following might account for this? More than one answer is possible.
 - a. Some of the regions are not real biological sequences.
 - b. Some of the regions cannot be sequenced due to biases.
 - c. Highly divergent regions may not be assembled.
 - d. Highly similar sequences may be included only once in the assembly.
 10. Observe the following fragment of a Fasta file (right). Is it formatted correctly? If not, what should be changed?

```

>seq A
CCACGTGTGG
>seq B
CCCCTCGTGG
>seq C
CCACCACTGG
>seq D
TCACATGTGG
>seq E
CCACTGGTGG
>seq F
CCACACGTGA
>seq G
CCACTGGTGA
>seq H
TCACGCGTGG
>seq I
CCACATGTGG
>seq J
CCACTTGTGA

```

11. A genome sequence represents all the heritable information of an organism. Sequenced genomes are an important resource for bioinformaticians.
 - a. What is the difference between the genome sequences in the Genbank and Refseq databases?
 - b. What is the difference between a complete and a draft genome sequence?
12. For the organisms listed below, find the following information in online databases. (i) What is the taxonomy of this organism? Write down the species, genus, family, order, class, phylum, and superkingdom. (ii) What is the length of the genome sequence? (iii) How many different chromosomes does the organism have? (iv) Is the genome complete? (v) How many genes does the genome contain? (vi) Is it possible that there are more/less genes encoded than this?
 - a. Human papillomavirus
 - b. *Fusobacterium nucleatum*
 - c. *Arabidopsis thaliana*
 - d. *Drosophila melanogaster*
 - e. *Wolbachia pipientis* endosymbiont of *D. melanogaster*
 - f. *Saccharomyces cerevisiae*
 - g. *Homo sapiens*
13. Discuss this question with other students. Antibiotics resistant bacteria are a growing healthcare problem worldwide, but the origin of antibiotics resistance genes remains unclear. A researcher wants to know whether the bacterial communities in pristine coral reef ecosystems contain any antibiotics resistance genes.
 - a. What would be an appropriate approach to answer this question? Include as many bioinformatic steps as possible. Search online for possible databases that you could use.
 - b. While answering question a above, I hope that you found multiple databases of antibiotics resistance genes that you use. Which one would you use? Consider the following: How comprehensive is the database (i.e. how many resistance genes does it contain)? How up-to-date is it (i.e. how recently was it published)? Do other scientists use it (i.e. how often does it get cited)?
 - c. What do you expect will be the answer of this research question, and why do you think that?
14. Select one or more biological databases from the list in Section 1.6 (or from your own search on the internet) and come up with a biological research question that you could answer by using bioinformatic analysis of the data in this database. Be as detailed as you can about which bioinformatic analyses you could include. Discuss your plan with your fellow students.

2. Talking to computers

Computers are not very inventive. All they can do is what they have been told to do, but then they can do that very well: they are very fast, they are very good at remembering, and they almost never make mistakes (if they do it is usually because they have not been programmed properly). And once you have told one computer what to do, you can give the same list of commands to any number of computers, and they will do the exact same thing. That is why computers have been so successful in science: they are the pinnacle of reproducibility, and reproducibility is one of the holy grails of scientific research.

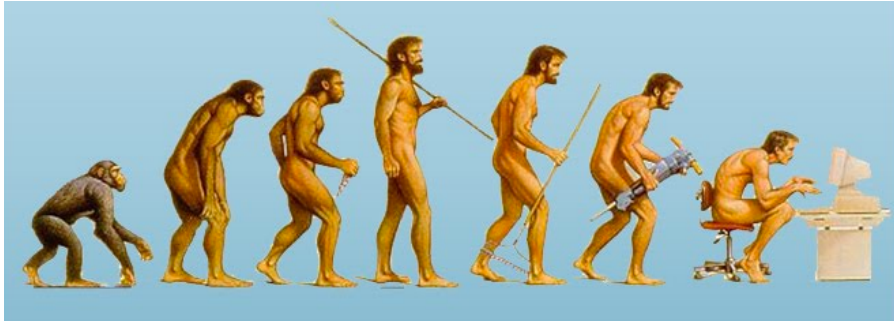


Figure 6. The evolution of computers. Figure taken from <https://shaunhall132.wordpress.com/2013/03/25/the-evolution-of-computers-50-years-of-it/>.

It is a skill to be able to tell computers what to do. Computer programming is a whole separate study, and as a field, it provides millions of people all over the world with work. Perhaps computer programming is not the first thing you think of when you think about what a biologist does every day. Still, as you have read in the self portraits in Section 1.7, many biologists do consider computer programming as one of their key skills, and they use it to answer real biological questions that sometimes cannot even be answered by using wet-lab experiments. We will not teach you computer programming in this course, but if you do not know it already, we would definitely stimulate you to discover if you like it, and see if you have a talent. There are numerous online courses available that will teach you the basics of scripting languages such as Python or R (some links are provided in Section 2.7) and we are 100% certain that learning these will help you in your career as a biologist. Life on earth is a very complex system, and one of the things you learn while becoming a biologist is how to address this complexity in a structured way. We believe that this skill can be trained by learning computer programming, because only if you ask your questions in a very structured way, by breaking them down into smaller and smaller sub-questions, will they be simple enough for a computer to understand them. It is important to realize that in this sense, computer programming is the ideal tool to study systems biology (see Section 1), where only by studying the collective interactions of many components, you can understand the larger complex as a whole. Depending on your high-school background, some students might find some aspects of programming easier than others. For example, those of you who learned algebra will find that this helps a lot in understanding and structuring variables, while those who learned logic will find that this helps in coming up with inventive algorithms.

2.1. Algorithms and variables

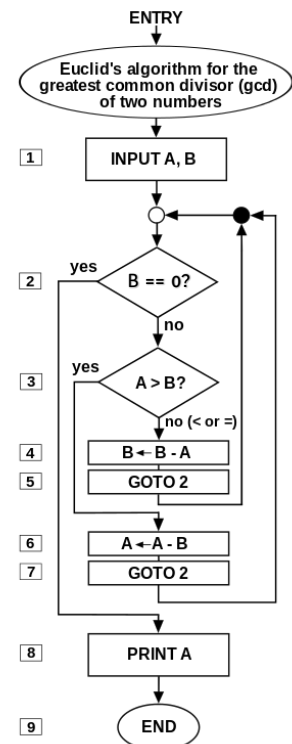
Abū ‘Abdallāh Muḥammad ibn Mūsā al-Khwārizmī (780-850, Islamic Golden Age) was a Persian mathematician, astronomer, and geographer who presented the first systematic way to

solve linear and quadratic equations by using a step-by-step set of operations. This approach of breaking a problem into simple steps made it possible to solve very complex equations in a simple way. Although this idea was not new (this was already used in ancient Greece, see Figure 7) al-Khwārizmī's work had a great influence on mathematics, and today his name lives on in the word “algorithm”.

Algorithms are step-by-step sets of operations that are used for complex calculations, data processing, and automated reasoning. They can perhaps best be compared to recipes in cooking or protocols in wet-lab experiments. A main difference is that algorithms frequently involve the use of variables that can be seen as little boxes or storage locations, where a number value, a logical value (TRUE/FALSE), a word, or even a whole list of values can be temporarily stored (see Section 2.3). Temporarily, because the values can easily be replaced as the algorithm progresses – that is why it is called a “variable”. Even a general protocol to isolate DNA from different kinds of tissue could contain variables. For example, one of the variables might correspond to the starting material, and in some places the protocol could involve different steps depending on the value of the variable (= the type of material): “If the *starting_material* is a hard tissue, grind it in liquid nitrogen; else dissolve it in buffer.”

Figure 7. Flow chart of Euclid's algorithm (~300 BC) for calculating and printing the greatest common divisor of two numbers. Figure taken from <https://en.wikipedia.org/wiki/Algorithm>.

Figure 7 shows the example of Euclid's algorithm for calculating the greatest common divisor of two arbitrary values. First [1] the two values are placed in variables A and B. Then, [2] WHILE $B \neq 0$ (note that this is the same as IF $B \neq 0$, and returning to the question after answering it in [5]/[7]), the algorithm proceeds by successive subtractions in two loops: [3] IF the test $A > B$ yields “no” (or FALSE), then the algorithm specifies: [4] $B \leftarrow B - A$, meaning that the value in B minus the value in A replaces the old value in B, and [5] go back to assessing the value in B. [3] ELSE (that is, IF the test $A > B$ yields “yes” or TRUE), then [6] $A \leftarrow A - B$, and [7] go back to assessing the value in B. This loop terminates when finally, the value in B is 0 in step [2], yielding the value of the greatest common divisor in A. [8] This value is printed and then [9] the algorithm terminates.



2.2. Computer languages

The example of Euclid's algorithm illustrates how a complex problem can be broken down into tiny steps that can be written as simple statements (while, if, else, print), tests ($B \neq 0$, $A > B$), and assignments ($B \leftarrow B - A$, $A \leftarrow A - B$). If they are written in the correct language with the correct syntax (\approx grammar), these are the kinds of tiny steps that can be understood by a computer. The example also shows how variables (A, B) can be used as storage locations for values, and that the value in a variable can be replaced as the algorithm progresses (in the assignment steps $B \leftarrow B - A$, $A \leftarrow A - B$). There are many programming and scripting languages available, some quite simple

and some more difficult to learn, that give you control over what a computer does. The Utrecht Bioinformatics Centre (UBC) has selected two of these, Python and R, that are very commonly used in computational biology. If you decide to continue with bioinformatics in your curriculum, these will be the languages that you will learn.

Computer languages typically use a similar set of operators for steps such as assignments, arithmetic calculations, comparisons, and logical operations. The most important operators are listed in Table 3. Note that the syntax for these operators, as well as for statements and functions, differs between computer languages. Here we will mostly adhere to the syntax of R.

Table 3. List of operators used in most computer languages (in this case R).

Operator	Function
<- (←) or =	Assignment: places the value on the right into the variable on the left
-> (→)	Assignment: places the value on the left into the variable on the right
+	Addition: reports the sum of two values, for example: $3 + 2 = 5$
-	Subtraction: reports the difference between two values, for example: $3 - 2 = 1$
/	Division: reports the fraction between two values, for example: $3 / 2 = 1.5$
*	Multiplication: reports the product of two values, for example: $3 * 2 = 6$
** or ^	Exponentiation: reports the exponent of two values, for example: $3 ** 2 = 9$
%%	Modulo: returns the remainder of an integer division, for example: $9 \% 5 = 4$
:	Colon operator: creates a range, for example 1:5 is equivalent to 1, 2, 3, 4, 5
!	Logical negation (not): returns TRUE if value is zero or FALSE; and FALSE otherwise
==	Equal test: returns TRUE if values are equal; and FALSE otherwise
!= (≠)	Not equal test: returns FALSE if values are equal; and TRUE otherwise
>	Greater than test: returns TRUE if left value is greater than right value; FALSE otherwise
<	Less than test: returns TRUE if right value is greater than left value; FALSE otherwise
>= (≥)	Greater or equal: returns FALSE if right value is larger than left value; TRUE otherwise
<= (≤)	Less or equal: returns FALSE if left value is larger than right value; TRUE otherwise
&	Logical “and”: returns TRUE if left and right values are both TRUE; FALSE otherwise
 	Logical “or”: returns FALSE if left and right values are both FALSE; TRUE otherwise

2.2.1. Simple questions with logical answers: TRUE or FALSE?

You see that many of the operators in Table 3 return “TRUE” or “FALSE” values. This is because they can actually be phrased as little questions. For example, “**a >= b**” can be read as: “Is the value in variable *a* greater or equal to the value in variable *b*?”, or “**! x**” can be read as: “Is the value in variable *x* zero (i.e. FALSE)?” These are simple yes/no questions, that the computer can quickly answer with the logical values TRUE (yes) or FALSE (no). In the computer, this binary decision is encoded in the simplest possible way, as a 1 or a 0, respectively, forming the essence

of computer coding. In fact, a computer considers any number to be TRUE, except 0 which is FALSE.

To make it slightly more complicated, logical questions can also be combined, which is what the last two operators do: **&** (ampersand) and **|** (pipe). These operators can also be phrased as questions, for example “**tu & vw**” can be read as: “Are the both values in the variables *tu* and *vw* TRUE?”, while “**grr | mmm**” can be read as: “Is at least one of the values in the variables *grr* or *mmm* TRUE?” Like the simple operator statements above, these questions can also be answered with the logical values TRUE (yes) or FALSE (no). By combining many of these very simple elements, computer algorithms can quickly become very complex. Moreover, by using variables to store the actual data (see Section 2.1), algorithms can be applied to very diverse input, and answer a diverse range of questions.

2.2.2. Functions

The challenge in designing an algorithm is to break down the complex question or data analysis into tiny steps so that they can be handled by operators and functions. Conversely, a complex analysis needs to be reconstructed from initially simple, but increasingly complex steps. This build-up is known as a modular architecture, i.e. the composition of complex structures from smaller components or modules (Figure 8). The beauty of modular architecture is that you can replace or add any one component (module) without affecting the rest of the system. This may be useful if you invent a better way to perform a step, or if you want to use part of your program in another analysis.

If an algorithm requires the same operation to be performed many times, computer code often uses functions, that are like little algorithms that perform a simple operation. Most computer languages know hundreds of functions that can do something, such as convert your input data into a different format, go through a list of values one by one, generate a plot, etcetera. We have already seen **while**, **if**, and **else** above, and give a few more examples in Table 4.

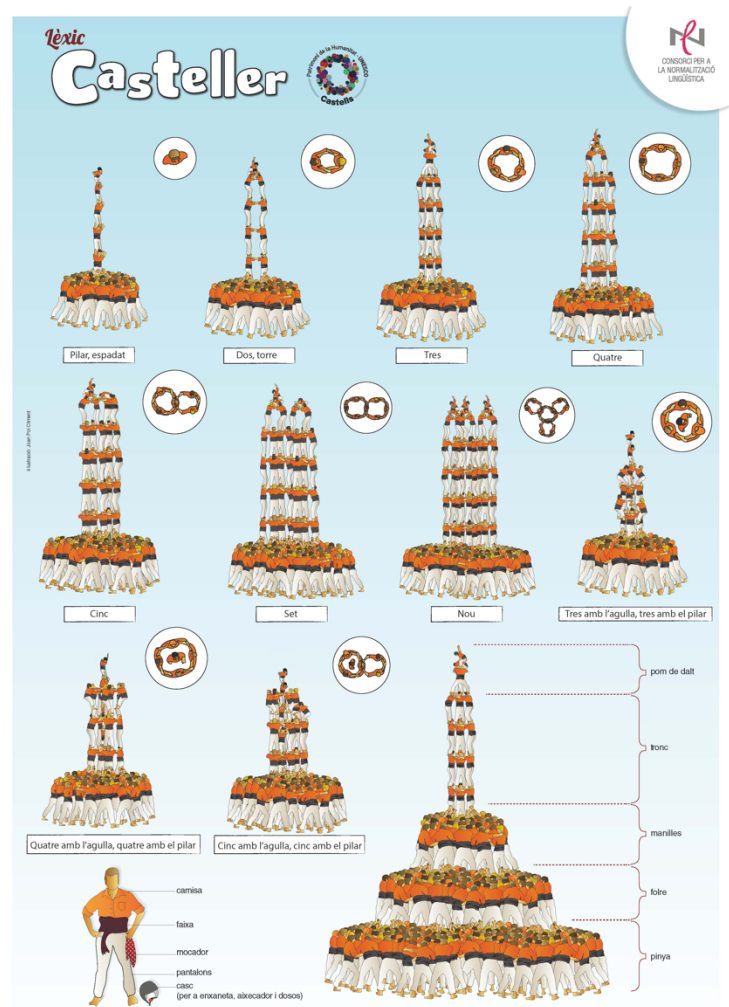


Figure 8. Figure from the **Coordinadora de Colles Castelleres de Catalunya** showing the modular architecture of human towers. Complex structures can be built by combining individual units in different ways. Figure from http://www.cccc.cat/contents/noticies/noticies-4808-ang_4808_72.

Table 4. Some examples of important statements used in computer scripts in the language R.

Statement	Function
print (<i>whatever</i>)	Prints contents of the variable <i>whatever</i> . The output may be printed on the screen or in a file (this can be specified).
c (<i>a</i> , <i>b</i> , <i>c</i> , <i>d</i> , ...)	Combines the values in the variables <i>a</i> , <i>b</i> , <i>c</i> , <i>d</i> , ... into a single vector containing many values. There can be as many as you want.
paste (<i>wrđ1</i> , <i>wrđ2</i> , ...)	Concatenates the values in the list, by default separated by spaces.
nchar (<i>yolo</i>)	Returns the number of characters in the variable <i>yolo</i> . Typically used for variables containing words, sentences, or DNA sequences.
length (<i>hair</i>)	Returns the number of elements in the vector-variable <i>hair</i> .
function (<i>bla</i>) { ... return (<i>bli</i>) }	Function: this is a function used in R to create new functions. A function can receive input (given in the variable <i>bla</i>), performs some operations on the input (statements in {} indicated by "..."), and finally returns some output as indicated by the return statement (in the variable <i>bli</i>). return has to be inside the {}.
if (<i>logical</i>) {} else {}	If the value or test in <i>logical</i> is TRUE, then any statements in the first set of {} are done. An else always follows an if : only the statements in the second set of {} are done if <i>logical</i> was FALSE.
while (<i>logix</i>) {}	While-loop: while the value of the variable, or the result of the test in <i>logix</i> is TRUE, any statements in {} are done and then <i>logix</i> is reassessed to see if the loop will be done again.
for (<i>i</i> in <i>long_vct</i>) {}	For-loop: any statements in {} are done <i>n</i> times, depending on the number of values in the vector variable <i>long_vct</i> . This variable <i>long_vct</i> could be created by the c (...) function (above). Every time, the next value in <i>long_vct</i> is placed into the variable <i>i</i> . The variable <i>i</i> can be used in the loop (inside the {}).

As said, this is just a small set of R functions, and very similar functions exist in many computer languages. In many cases, computational biologists will write their own functions for specific computational operations in a script or program. (Note that two terms for computer code are “scripts” or “programs”, the difference being that programs require an explicit compilation step.) For example, the function “**minimum** (*coin1*, *coin2*, *coin3*, *coin4*)” could return the lowest value in a vector of variables named *coin1*, *coin2*, ... (see Section 2.4.1), while the function “**euclid** (*a*, *b*)” could return the largest common denominator between the values in the variables named *a* and *b* (see Section 2.4.2). It is very important to note that you cannot use functions in your code without defining them. Without telling the computer what to do when it encounters a word like “minimum” or “euclid”, it will have no idea what this word means. To illustrate this, a function that returns the lowest value in the vector of variables might just as well be called “**uvwxyz** (*coin1*, *coin2*, *coin3*, ...)”, as long as it is defined in the right way

(for example, as in Section 2.4.1). When writing computer code, functions can be defined within the same computer program or script where they are used, but often they are also defined elsewhere and can be called again and again from many different programs. Functions can even be written by completely different people and made available through the internet. For example, BioPython and Bioconductor are two repositories for common functions used in computational biology in Python and R, respectively.

2.3. Data types

As introduced above in Section 2.1, variables are storage locations where data can be temporarily stored. We have seen different types of data above, think of the logical values that we discussed in Section 2.2.1, or the numbers and vectors mentioned in Section 2.2.2. R recognizes a finite number of data types, and we will discuss some of the most important ones here. In R you can always discover the data type of a given variable by typing “`str(variable_name)`”, where “`str`” stands for structure.

2.3.1. Numbers

We will skip the logical values (data type: `logi`, TRUE and FALSE) because we already discussed them in Section 2.2.1, plus they are not really numbers anyway. We just mention them here to remind you that (i) FALSE and TRUE are often seen as equivalents of the binary values 0 and 1, respectively, and (ii) all non-zero numbers are TRUE while zero is FALSE.

Integers (data type: `int`) are whole numbers, i.e. numbers that can be written without a fractional component. For example, 35, 2, 0, and -9151 are integers, while 91.05, 2½, and √7 are not. Instead, fractional numbers can be stored as doubles (data type: `num`). Most of the time, data values will be stored as doubles, because they obviously have a much higher precision than integers, given that they have multiple decimal numbers. Integers allow for faster computation than floats because they occupy less computer memory (see Section 2.6).

2.3.2. Text

To the computer, text basically just a string of characters. The data type `chr`, also known as “string” if there are more than one in a row, can store anything that is written, like “A”, “B”, “C”, “curriculum”, “...”, “GCTGGCCCAGAGCGTTCACGCC”, “8#4@?:#”, etc. Note that strings are often delimited by quotes “” to indicate that they are strings.

Although they are invisible, whitespace characters can also be (part of) strings. The most common whitespace character is the space “ ”. We have already seen the tab, which is often used to delimit columns in .tsv files (see Section 1.5) and is encoded in the computer as “\t”. Although this looks like two characters, it is actually just the tab*. The newline character is perhaps better known as the “hard return” at the end of a line and is encoded in computer languages as “\n”.

* The backslash in “\t” for tabs, in “\n” for newlines, and in many other special characters, is called an escape character. It tells the computer that the following should be interpreted as a special character.

2.3.3. Variables that can contain multiple values

In Table 4, we have already seen that operators like “**c**” and “**paste**” can take multiple values as input. In Section 2.4.1, we will see an example of a function that can take a whole range of numbers as input. Sometimes, you may need to store a vector of values together in one variable, and this is possible with vectors (data type: **logi[]**, **int[]**, **num[]**, and **chr[]**). As you can see from the listed data types, one vector can contain one type of data, and it is indicated between the square brackets how many elements the vector contains. Elements in the vector can be accessed by specifying the number between the square brackets, e.g. **vect[4]** will access the fourth element in the vector “**vect**”. One useful command to fill a vector is “**seq**”, which outputs a range of numbers between a minimum and a maximum value, with a fixed step size. We will see several examples of vectors in the following sections and in the exercises.

Data frames (data type: **data.frame**) are rectangular tables with rows and columns (see Section 1.5). Each column in the data frame can contain a different data type, but the data type of all values in the column should be the same. If you type “**str(dfr)**” for a data frame called **dfr**, you will see in detail how it is built up, and the data type in each column will be listed. The command “**dim(dfr)**” will just show you how many rows and columns the data frame contains.

2.4. Examples

What we will do in this course, is show you some examples of code that illustrate how complex problems are broken up into tiny steps, and we hope that this will give you an intuition for what computers do. For some biologists, computer programs for bioinformatic data analysis are just black boxes, but it is important not to be satisfied with this: only by understanding what goes on inside the program can you master your data analysis completely and know what the results really mean. Moreover, because computers play such a large role in biological research, that it is critical for a biologist to be able to speak the language of computer scientists.

Below, we will present a few simple scripting examples, and explain what they do and how they do it. These pieces of code are written in R, and we would stimulate you to play with them on your own computer, so that you can get a better feel for how they work.

2.4.1. Minimum

The function “**minimum**” takes as input a vector of values, and outputs the lowest value in this vector. We do not know *a priori* how many values are in the input vector, so we want to use a for-loop to go through all the values, regardless of if there is just one value, or if there are one million.

```
minimum <- function (input_vector) {  
  the_lowest <- input_vector[1]  
  for (i in input_vector) {  
    if (i < the_lowest) {  
      the_lowest <- i  
    }  
  }  
  return (the_lowest)  
}
```

In the first line, the name of our function is defined as “**minimum**”. Note that this could also be any other name we want, because this is where this word is first defined. The name of any variable can be changed, those are printed on a gray background. Things that cannot be changed are printed on a white background, those are parts of the built-in statements and syntax of R that allow it to interpret the script. The function “**function**” is assigned to the name “**minimum**”, so that R knows what to expect from the thing we have given the name “**minimum**”. “**function**” is a pre-defined statement that tells R to create a function that we will be giving some input, and from which will be expecting some output. The input will be given in the vector variable “**input_vector**” and the output will be returned by the function “**return**” in line 7 (see Table 4).

We know that the algorithm will have to go through all the values stored in “**input_vector**” and remember the lowest one, so the first thing we do is define a new variable, “**the_lowest**”, where the lowest value can be remembered (stored). We begin by placing the first of the vector of input values into this storage location “**the_lowest**” in line 2. In a vector variable, the first value can be found in **input_vector[1]**, the second value is in **input_vector[2]**, etcetera. Next, in lines 3-6, the **for**-loop makes sure that all the values in “**input_vector**” are compared to “**the_lowest**” one by one. The statements “**for**” and “**in**” are pre-defined in R and together form a for-loop that places all elements in a vector (“**input_vector**” in this case) into a variable (“**i**” in this case) one by one, and then performs the statements between the curly brackets { } each time (see Table 4). These statements involve testing “**if (i < the_lowest)**”, and if this is TRUE, the value in “**i**” is assigned to “**the_lowest**”. Thus, after all the values in “**input_vector**” have been compared, the very lowest one will be remembered in that variable, and can be returned as output of the function by the “**return**” statement, and the function is finished.

2.4.2. Euclid’s algorithm

Figure 7 shows Euclid’s algorithm for finding the largest common denominator of two given numbers, displayed as a flow diagram. When written as a function in R, it looks like this.

```
euclid <- function (values) {
  A <- values[1]
  B <- values[2]
  while (B != 0) {
    if (A > B) {
      A <- A - B }
    else {
      B <- B - A }
  }
  return (A)
}
```

As above, the name of our function is defined in the first line as “**euclid**”. The function assumes that it will receive two values as input, these are present in the short vector called “**values**” that contains two elements. These two values are placed in the variables **A** and **B**. Note that if more than

two values are passed to the function, they will not be considered because only the first two elements in the vector “**values**” are used. The order of these two values does not matter at this point. The first question the function asks, is whether **B** is unequal to zero. Note that if **B** is equal to zero, the function could immediately continue to the last line, and return the largest common denominator **A** as the output of the function. However, while the question “(**B** != 0)” yields the answer TRUE, the function is not yet done and it iteratively subtracts the smaller of the two numbers **A** and **B** from the greater. This is performed by the four lines with the **if** and **else** statements within the while loop: if **A** is greater than **B**, subtract **B** from **A**; otherwise subtract **A** from **B**. This iteration continues until finally, **A** and **B** become equal. We cannot know when this will happen, it might be in one of the first iterations, or only after the **while** loop has been executed thousands of times. However, at some point **A** and **B** become equal, and when that is the case, **A** will be subtracted from **B** in the **else** statement, and **B** becomes zero. This ends the **while** loop, and leads the function to return the value of **A** that was last subtracted from **B**, which is the largest common denominator of the two original numbers. Note that this number is still present in the variable **A**, because even though it was subtracted from **B**, the value in variable **A** remained unchanged.

2.5. R-specific functions

In this section, we discuss some additional R functions that are commonly used in bioinformatics.

2.5.1. Help

R has so many operators and statements that no one can know them all. This is why every statement has a “help page” associated with it. You can easily access this page by typing a question mark in front of a command, for example “**?print**”. We will see this in some of the exercises in Section 2.8. Many commands allow different arguments for additional functionality, and those are explained in the help page. If you just want to see how to use the command, the help page often lists some useful examples.

2.5.2. Reading a table

High throughput technologies generate large amounts of data which are commonly saved in tab-separated values files (tsv, see Section 1.5). If these data are rectangular, we can think of the file as containing rows and columns like a table. R is well suited to work with this kind of data. For example, you could read the values in Figure 4 into a variable called “**cohort**” as follows:

```
cohort <- read.table("fig4.tsv", sep="\t", header=TRUE, stringsAsFactors=FALSE)
```

Our call to the “**read.table**” function has four arguments:

- 1) The location of the file. The command above works if you have a file called **fig4.tsv** in your working directory, but you can also specify the full file path between the quotes, for example*: “**C:\Documents\fig4.txt**”. In RStudio you can use the tab key to search

* The backslashes in “**C:\Documents\fig4.txt**” indicate subdirectories and depending on your operating system you may need to use backslash “\” or forward slash “/”.

through your files. Hit the tab key while typing the file name and select the correct directory in the popup by using the up/down arrows. Typically, the file you want to load will be in your file system, but it can also be a web address if you have access to the internet. The file in Figure 4 is available at <http://tbb.bio.uu.nl/BDA/fig4.tsv> so you could also enter that location directly between the quotes: "`http://tbb.bio.uu.nl/BDA/fig4.tsv`".

- 2) The "**sep**" argument specifies what character is used to separate the items in the table. As explained in Section 1.5, this is a tab in .tsv files, but in principle you could also specify other characters here. The tab is indicated by "`\t`" in R (and in most other computer languages as well). Although this is written as two characters "`\`" and "`t`", it is really just one character and the backslash tells R that it is a special character.
- 3) The "**header**" argument states if the first line in the file contains column names or not.
- 4) The "**stringsAsFactors=FALSE**" statement makes sure that all the characters in the file are read correctly as strings. (Factors are yet another data type in R (see Section 2.3.2) that we rarely use but is kept for historical reasons.)

The result of using **read.table** is a data frame that is stored in the variable named **cohort**. A data frame is R's version of a rectangular table: an object that has the same number of columns in each row, and *vice versa*. Note that the first argument, the filename, is not labelled in the command above. This means that R expects the file name to be provided as the first argument of the **read.table** command. The order of the remaining arguments does not matter since they are provided with explicit names: "**sep=**", "**header=**", and "**stringsAsFactors=**".

2.5.3. Plotting data

The results of an experiment are often shown in the form of plots and graphs that visualize the data. R has many powerful functions to create plots, two of which we describe here. The first is the function "**plot()**", which you can use to make scatter plots of one feature or measurement against another (e.g. Figure 11). If you check the help page of this function (see Section 2.5.1), you will see that this function has many options and arguments, so that you can change things like the line colors, the axis labels, titles, etc. Let's try it out. After reading the data from Figure 4 into a data frame **cohort** (see Section 2.5.2), the following command generates a scatter plot of age versus weight (Figure 9):

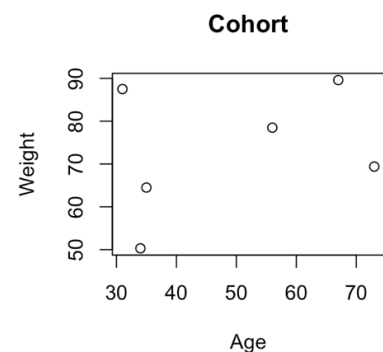


Figure 9. Scatter plot of Weight versus Age using the data from Figure 4.

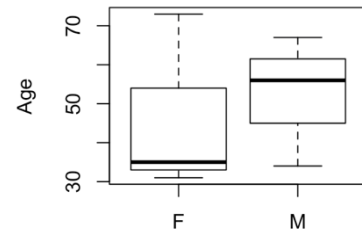
```
plot(x=cohort$Age, y=cohort$Weight, main="Cohort", ylab="Weight", xlab="Age")
```

The first two arguments are the X- and Y-axis variables (**x=cohort\$Age**, **y=cohort\$Weight**). The "**main**" argument provides the plot title. The X- and Y-axis labels are assigned with the parameters **xlab** and **ylab**, respectively. Note that the title and axis labels are strings of characters and therefore needs to be given between quotes (see Section 2.3.2).

The second function we will discuss is “**boxplot()**”. Box-and-whisker plots allow you to compare the value distribution for different categories or classes of data. For example, let’s compare the age distribution of males and females in our cohort from Figure 4:

```
boxplot(cohort[cohort$Gender=="F", "Age"],
        cohort[cohort$Gender=="M", "Age"], ylab="Age", names=c("F", "M"))
```

Figure 10. Box-and-whiskers plot of the age distribution in females and males using the data from Figure 4.



This generates the plot in Figure 10. The bold line in the middle of each box represents the median, and the edges of the box represent the first and the third quartiles. The whiskers represent either the minimum and maximum of the data, or 1.5 times the distance between the first and the third quartiles (whichever is smaller, check <http://onetipperday.sterding.com/2012/06/about-boxplot.html> for a further explanation).

2.6. Computer power: CPU and RAM

Computers are essential tools in bioinformatics. The performance of a computer depends on two main elements. The first is the number and speed of central processing units (CPUs), also known as cores. This determines how fast the computer can perform simple comparisons (like those we have seen in Section 2.2.1), how many tasks it can perform in parallel.

The second important element is the size and speed of the random-access memory (RAM). This indicates how much information the computer can access at any given time. For example, if we want to compare two sequences with lengths m and n , then we need at least $m \cdot n$ spaces in the RAM to hold all the information (see Chapter 7). Besides RAM, there is also a second type of computer memory called the hard disk drive (HDD). HDD space is much cheaper than RAM (about 10c versus \$5 per gigabyte), but accessing it is much slower than accessing the RAM because the time needed to access the data (i.e. to read or write it) depends on its physical location on the disk. In contrast, data stored in the RAM can be accessed in almost the same amount of time irrespective of the physical location of the data inside the computer. Consequently, loading information from the HDD into the RAM also takes a long time, so if you want to compare a lot of data at once, it is important that your computer has a lot of RAM available. Good computer programmers try to make their script as efficient as possible by reducing the amount of HDD access to a minimum and making optimal use of the RAM. We will see an example of this in Chapter 8.

2.7. Learning more about scripting

The paragraphs above have introduced some of R’s operators and statements and shown examples of functions and scripts. R has many more possibilities: it is one of the most powerful platforms for data analysis and researchers around the world are developing methods that they share online.

The best way to learn this is by doing it yourself and discovering that talking to computers is fun! Do you have a talent for scripting? Discover using one of the numerous courses and tutorials that are available online. Below is a list of useful links for R and Python, the two languages most used in bioinformatic data analysis and supported by the Utrecht Bioinformatics Centre (<http://ubc.uu.nl>). Since many computational biologists run their scripts on Linux computer servers, I have also included some links for learning to use the command line, the interface with a Linux computer. Finally a request: if you are into scripting and find other useful links that I could include in this Reader for next year's students, please let me know! You can find my email address on my website or contact me via Blackboard.

Useful links for learning R:

- Torfs & Brauer: <https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf>
- Datacamp: <https://www.datacamp.com/courses/free-introduction-to-r>
- Codeschool: <http://tryr.codeschool.com>
- Swirl: <http://swirlstats.com>
- Bioconductor: <https://www.bioconductor.org/>

Useful links for learning Python:

- Codecademy: <https://www.codecademy.com/tracks/python>
- CMBI Python course: <http://www.cmbi.ru.nl/pythoncourse>
- Think Python: <http://www.greenteapress.com/thinkpython/thinkpython.pdf>

Useful links for learning to use the command line:

- Make your computer dual bootable with Windows and Linux:
<https://www.howtogeek.com/128347/5-ways-to-try-out-and-install-ubuntu-on-your-computer/>
- Utrecht University Theoretical Biology and Bioinformatics group, Introduction to Linux:
<http://www-binf.bio.uu.nl/manuals/introlinux>
- Codecademy: <https://www.codecademy.com/learn/learn-the-command-line>
- Command-line bootcamp: http://rik.smith-unna.com/command_line_bootcamp

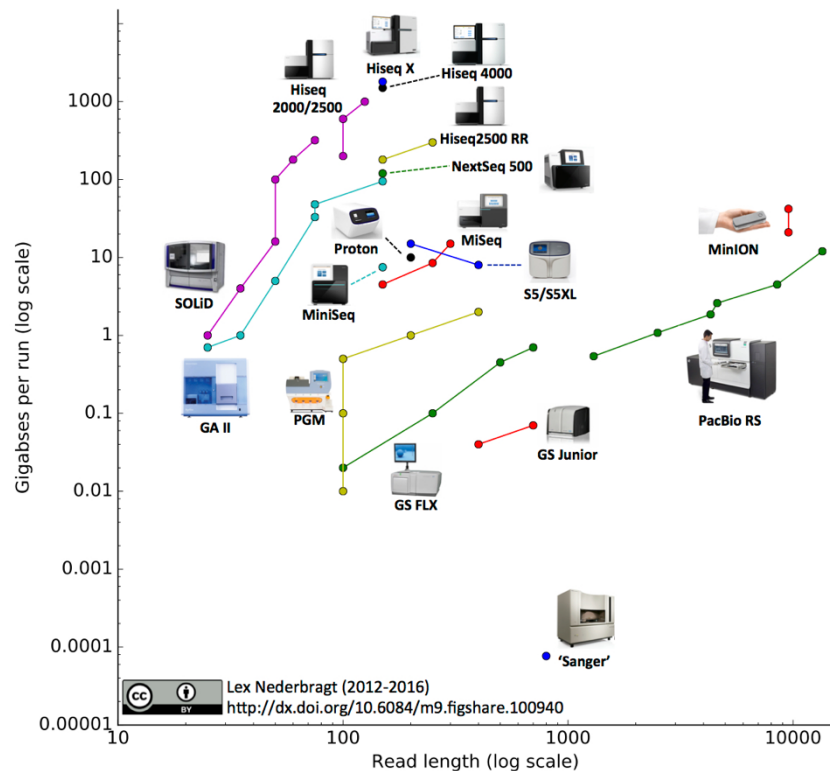
2.8. Questions and exercises

For the R questions, please refer to the additional section at the very end of this reader.

3. Dimensions of data

As introduced in Section 1.4, next-generation DNA sequencing platforms yield massive amounts of short reads. Figure 11 shows the recent developments in DNA sequencing technologies. Each differently colored line in the figure represents a different sequencing machine, and the points on the line show how in subsequent years, the performance of the machines improve, including both the length of individual sequencing reads, and the total number of nucleotides that the machine can deliver in a single run. Not all lines are equally long because new machines are developed, and older ones are taken out of the market.

Figure 11. Developments in DNA sequencing technologies until July 2016. This figure includes a first generation DNA sequencer (chain termination method: Sanger), second generation DNA sequencers (high-throughput sequencing: Illumina [GA II/MiSeq/Miniseq/Nextseq/Hiseq], Ion Torrent [PGM/Proton], 454 [GS Junior/GS FLX], SOLiD), and third generation sequencers (single molecule sequencing: PacBio, Oxford Nanopore MinION). Figure from <http://flxlexblog.wordpress.com>.



3.1. Second generation DNA sequencing as a profiling technology

First generation DNA sequencing was the most widely used sequencing method from its development in 1970s until well into the 2000s, and still one of the most accurate methods today (Sanger and Coulson 1975). Second generation sequencing generates thousands to millions of relatively short sequences from randomly amplified segments of the DNA in a sample, and includes most of the machines shown in Figure 11. Third generation sequencing generates long reads from individual fragments of DNA.

Of course, DNA sequencing can be used to determine the genome sequence of individual organisms. However, the advance of especially second generation DNA sequencing technologies has led to a different use of sequencing, i.e. as a way to generate a random sample of the sequences in a sample. We will discuss two applications of this below, i.e. transcriptomics and metagenomics. Both these approaches generate an overview of the sequences that are present in a sample, with the purpose of profiling the sample composition.

Transcriptomics analyzes which genes are transcribed in a sample, such as a cell culture or a tissue sample. Ideally, the researcher would like to have a long table of all the genes in the genome, and for each of them list how much of the total RNA in the sample belongs to that particular gene, i.e. how much that gene is transcribed in the sample. This is called a transcriptome of the cell culture or tissue, and can be used to answer many different research questions. Because second generation DNA sequencing generates up to millions of reads from random fragments of the DNA in a sample, this approach is ideal to generate the information the researcher needs. Using fast, heuristic sequence similarity search tools (see Chapter 8), each sequencing read can be assigned to one of the genes in the genome, and when all the reads are assigned, the relative expression of each of the genes can be calculated. This list of numbers is called a gene expression profile.

Metagenomics analyzes the micro-organisms in a sample, such as an environmental or medical sample. Like for transcriptomics above, the researcher would like to have a long table of all the species in microbial community, and for each of them list how many there are in that environment. Sometimes, the question is also about the functions of the microbes in the sample, so then the table might also list all the metabolic functions and their abundances. Metagenomics exploits second generation DNA sequencing in combination with fast, heuristic sequence similarity search tools to determine what species and functions are represented by the DNA in a sample, and determine their relative abundances (Figure 12). This list of numbers is called a species abundance profile.

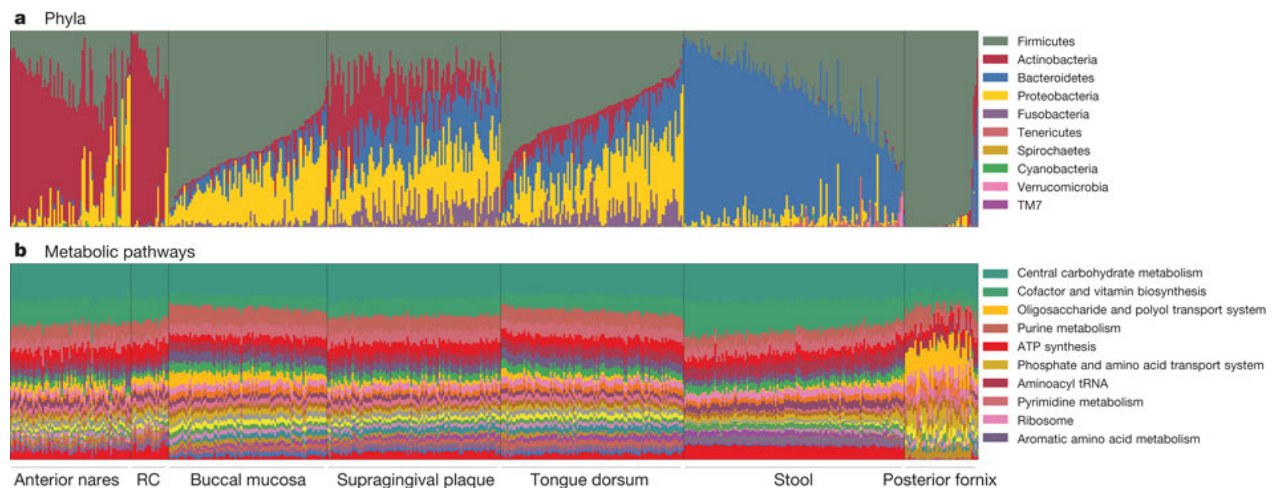


Figure 12. Metagenomics exploits second generation DNA sequencing to profile the human microbiome. Each column of stacked bars represents the relative abundances of microbial taxa (a, phyla) and functions (b, metabolic pathways) in one microbiome sample, sorted by body site. Figure from (Huttenhower et al. 2012).

3.2. Visualization of data

Gene expression and metagenome data are often multi-dimensional. For example, gene expression may be measured at several time points, or the composition of a microbial community might be measured under different environmental conditions. If we perform N experiments, we sequence N transcriptomes or metagenomes, and the abundance of a gene (or microbe) g across all these experiments can be represented by a N -dimensional vector of values. That is, we can represent a gene g with the following vector: $\vec{g} = (g_1, g_2, \dots, g_N)$. This means that each gene is basically a point in N -dimensional space.

Consider the response of yeast to glucose starvation. To find out which genes are down- and up-regulated, we measure the transcriptome every half hour during the first 6 hours. This makes every gene a data point in a 12-dimensional space. It is obviously not possible to visualize these data using all 12 dimensions. However, data always contain redundant information, and by removing some of this redundancy, we can attempt to visualize the data in fewer dimensions, like the two dimensions on a sheet of paper. For example, this redundancy might result from the response of yeast being delayed during the starvation experiment. If nothing changes in the first hour, then the gene expression patterns at $t = 0$, $t = 30$, and $t = 60$ minutes should be almost identical, so we could ignore these redundant values or average the expression values at $t = 0$, $t = 30$, and $t = 60$ minutes without a significant loss of information.

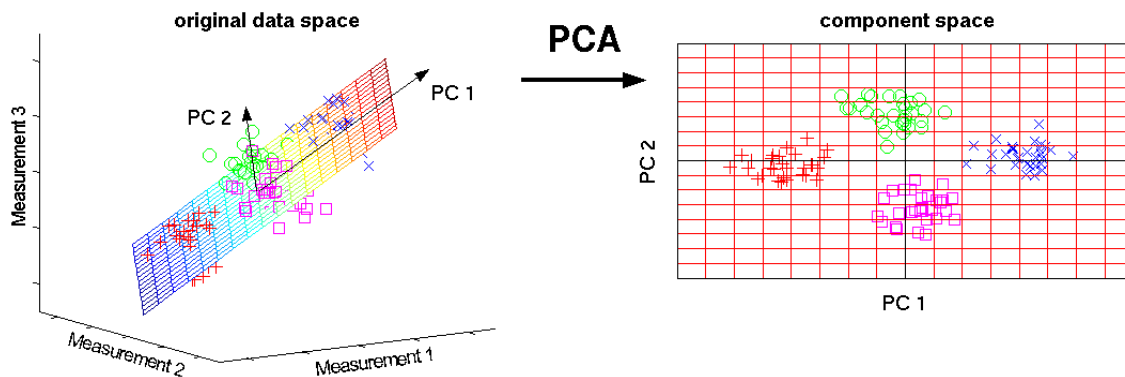


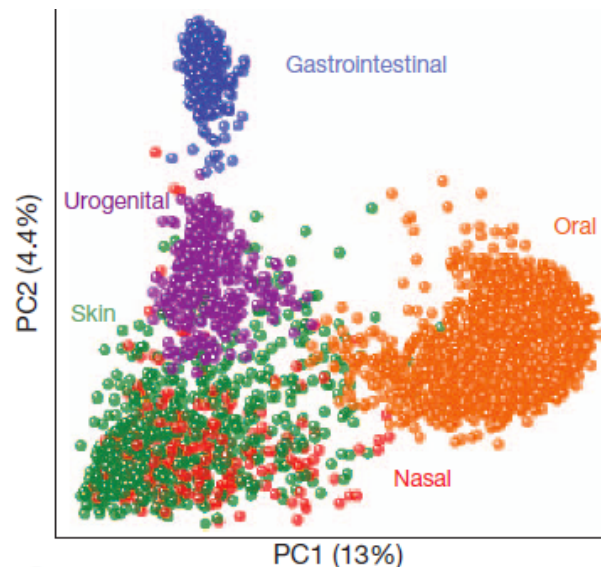
Figure 13. Principal Component Analysis (PCA) is a visualization technique that projects multi-dimensional data into lower-dimensional "component space" that maximizes the variation in the data. Figure from <http://www.nl pca.org>. See <http://setosa.io/ev/principal-component-analysis/> for a further visualization of PCA.

Principal Component Analysis (PCA) (also called singular value decomposition) is an often-used visualization tool to reduce the dimensionality of data. For example, the first two principal components represent just two dimensions, and two-dimensional data can easily be printed on a sheet of paper. PCA is a mathematical technique that picks out patterns in the data while minimizing the loss of information. As a result, the dimensionality of the gene-expression space is reduced. The mathematics behind PCA is rather complex, but the basic principles are easy to follow. Imagine taking a three-dimensional cloud of data points and rotating it so that you can view it from different perspectives, giving you different 2D views on your 3D cloud of data points ("original data space" in Figure 13). You can imagine that certain views would allow you to better separate the data than other views. PCA finds those views (i.e. lower-dimensional representations) that give you the maximal separation of the higher-dimensional data points. For example, in Figure 13 we can see how the genes that clustered in the original high-dimensional data space (here defined by the three measurements 1, 2, and 3) still cluster in the low-dimensional PCA space (here defined by the two principal components PC1 and PC2). Thus, by going from many to two dimensions, we do not lose essential information, while at the same time we can now easily visualize the individual genes, as well as their clusters in a two-dimensional graph.

In most implementations of PCA, it is difficult to define accurately the precise boundaries of distinct clusters in the data, i.e. to define which data points (e.g. genes or experiments) belong to each cluster. Well-defined clusters, as shown in the example in Figure 13, are a rare situation. For example, Figure 14 shows a PCA of the same human microbiome profiles shown in Figure 12, showing a strong similarity between samples from similar tissues, but also overlap between them.

To objectively divide high-dimensional data points into clusters, we need two things: (i) a measure of the similarity or distance between two high-dimensional data points (vectors, Section 3.4), and (ii) a clustering algorithm (Section 3.6).

Figure 14. PCA plot showing variation among human microbiome samples from different body sites. Figure from (Huttenhower et al. 2012).



3.3. Similarity between relative abundance profiles

To a bioinformatician, the profiles generated by transcriptomics or metagenomics are equivalent data types. Very similar approaches can be used to answer very different biological questions. For example, if gene expression profiles from a range of tumors and healthy tissues are compared, we can investigate if the profiles of tumor samples are more similar to each other than to the profiles of healthy tissues. With very similar methods, we can test if the oral microbiome of the human tongue is significantly different from that of the cheek (buccal mucosa, see Figure 15).

When similar profiling datasets are available for several conditions, they can be used to define clusters of data points that behave similarly. For example, genes that are functionally related may be up- or down-regulated together, resulting in a similar relative abundance pattern across different transcriptomic datasets. Such genes may be co-regulated because they are part of a common pathway, or they may respond to the same environmental stimuli. Thus, if one of the genes in such a co-regulated group is known to be associated with a disease or phenotype, it may indicate that other genes in the group are also associated with that disease or phenotype, providing leads for further experimental follow-up (see also Figure 5). Similarly, identifying micro-organisms that have a similar relative abundance pattern across metagenomic datasets may provide valuable information about the ecological interactions between these species, for example they may depend on similar environmental resources, or they may be dependent on each other for the exchange of metabolites (Figure 15).

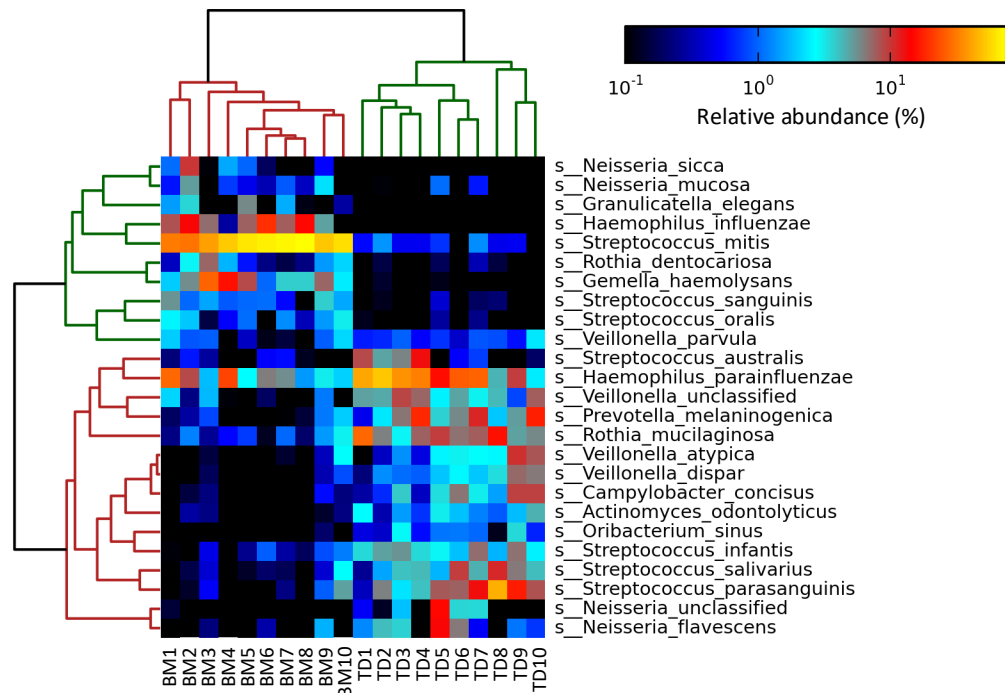


Figure 15. Bi-clustered heatmap showing the relative abundance of the 25 most abundant species found in ten human inner cheek samples (buccal mucosa, BM1-10) and ten tongue dorsum samples (TD1-10), analyzed with metagenomics. The clustering is performed with average linkage. Each microbial species has its preferred habitat, and the tongue and cheek samples clearly separate into two groups. Figure from https://bitbucket.org/nsegata/metaphlan/wiki/MetaPhlAn_Pipelines_Tutorial.

The two paragraphs above both describe information that can be gained by comparing relative abundance profiles. In the first paragraph, the profiles of genes or microbes *within* a dataset are compared between the datasets. In the second paragraph, the profiles of the genes or microbes *across* samples are compared to identify which ones are present or absent together. This change in perspective is visualized in Figure 15, where the same data is compared both as rows and as columns. Both orientations provide us with different information. The comparison between the columns tells us that the microbiomes of the cheek is different from the microbiomes of the tongue. The comparison between the rows tells us which microbial species tend to occur together. Together, the whole picture shows us which species occur on the inner cheek, and which species prefer the tongue. Importantly, the rows and columns are hierarchically clustered by similarity to aid in our interpretation. These clusters are indicated as tree-like graphs at the top and left of the heatmap. We will use R to create a bi-clustering heatmap in the exercises in Section 3.9.

3.4. Distance measures

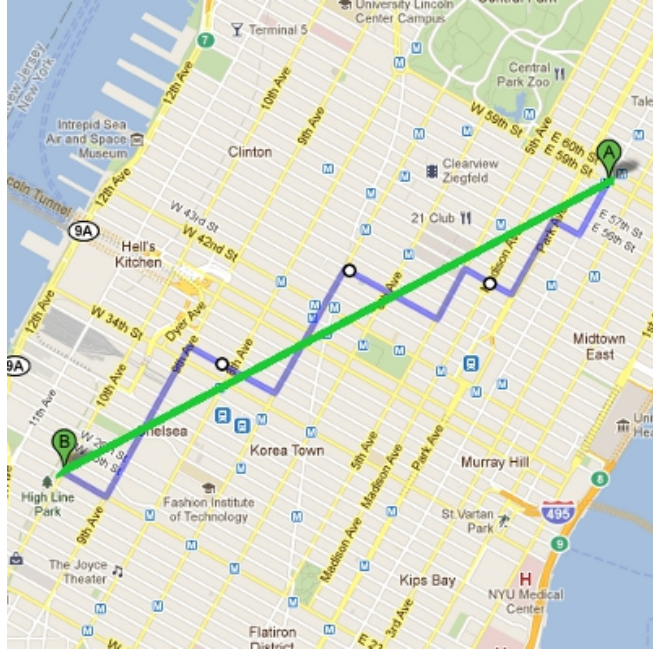
The first step of clustering is to calculate a distance matrix that lists the distances between all the observations. As introduced in Section 3.2, lists of data containing e.g. the expression values of genes g across N samples can be represented by an N -dimensional vector: $\vec{g} = (g_1, g_2, \dots, g_N)$. The distance of a point to itself is always 0, whatever the number of dimensions you have. But what is the distance between two different points in N -dimensional space? The distance between vectors \vec{g} can be calculated as the distance between the points in this N -dimensional space. Here, we will discuss some of the most popular distance measures that are used in bioinformatics.

The simplest measure is the Manhattan distance, also known as city block distance. This measure received its name because calculating the distance between two points x and y is like walking the distance on the map of Manhattan (Figure 16). The Manhattan distance is calculated as the sum of the absolute distances between the components of each vector (Equation 1).

Equation 1. The Manhattan distance, also known as city-block distance.

$$D_{xy} = \sum_{i=1}^N |x_i - y_i|$$

Figure 16. Two types of distances between A and B displayed on the map of Manhattan. The Manhattan distance between A and B is the distance covered by the blue line. The Euclidean distance is the distance covered by the green line. Note that the dimensions of the vectors corresponding to A and B are defined by the grid of roads on the city map. While there are two dimensions on the map of Manhattan (streets NW-SE and avenues NE-SW), the calculation can be extrapolated to multiple dimensions according to Equation 1 and Equation 2, respectively. Figure from <http://www.newyorkgeek.com>.



Another distance measure is the Euclidean distance, in which you may recognize Pythagoras' formula that has been extended to deal with an N -dimensional vector (Equation 2).

Equation 2. The Euclidean distance, which is equivalent to Pythagoras' formula in N dimensions.

$$D_{xy} = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}$$

3.5. Correlation

Another commonly used measure, this time a similarity measure, is the Pearson correlation coefficient r (Equation 3).

Equation 3. Pearson's correlation coefficient r . Here μ_x and μ_y are the mean expression values of genes x and y .

$$r = \frac{\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^N (x_i - \mu_x)^2} \sqrt{\sum_{i=1}^N (y_i - \mu_y)^2}}$$

Pearson's r is especially popular in the analysis of gene expression data, because it assesses the similarity in the "shape" of a profile of numbers, rather than its magnitude which weigh heavily in the Manhattan and Euclidean distance measures. The shape is important if the genes (in the case of transcriptomes) or micro-organisms (in metagenomes) respond to similar regulatory or environmental stimuli, but some respond more strongly than others. While Pearson's r is effective to detect similarities between gene expressions, it is not very robust to outliers in the data, and may be likely to generate false positives. For example, if two profiles have a common peak or valley at a single experimental condition (maybe an experimental error), then Pearson's r will be dominated by this condition, although the profiles at remaining conditions may not be similar at all.

Spearman's rank correlation coefficient or Spearman's ρ is the final similarity measure that we will discuss here. It is a rank correlation, which means that the data is first transformed into ranks before calculating a correlation. This means that the lowest observed value is replaced by the number 1, the second-lowest observed value is replaced by the number 2, and so on until the highest observed value is replaced by the number N . As a result, Spearman's ρ is much less sensitive to extreme outliers than Pearson's r , but it is also less sensitive for detecting weaker correlations.

Equation 4. Spearman's rank correlation coefficient ρ . X_i and Y_i are the ranks of values x_i and y_i in the vector.

$$\rho = 1 - \frac{6 \sum_{i=1}^N (X_i - Y_i)^2}{N(N^2 - 1)}$$

Note that correlation is a similarity measure, not a distance measure! Distance and similarity are each other's complement: if you calculate a distance measure, it can easily be converted into a similarity measure by rescaling (for example, if the numbers are scaled between 0 and 1, $D = 1 - S$). The values of the correlation coefficient r range between -1 and +1, with $r = 1$ when the two vectors are identical (perfect correlation), $r = -1$ when the two vectors are exact opposites (perfect anti-correlation), and $r = 0$ when the two vectors are completely independent (uncorrelated or orthogonal vectors). We will devise a formula to convert from correlation to distance in the exercises in Section 3.9.

3.6. Clustering algorithms

Once we have a distance matrix between all data points/vectors, we can proceed with the clustering. There are several clustering algorithms available to group them based on these distance scores. Clustering algorithms can be hierarchical or partitional. Partitional clustering algorithms determine clusters by looking at all distance scores together and then calculating the optimal cluster partitioning at once. Hierarchical clustering algorithms find clusters by taking the two closest entities (i.e. those with the lowest value in the distance matrix) and joining them into a cluster, and then re-calculating the distance matrix using a special set of rules for the entities that are already in a previously established cluster. The three main hierarchical clustering algorithms are:

- Single linkage clustering (Figure 17A). In this method the distance between two clusters is determined by the distance of the two closest objects (nearest neighbors) in the different clusters. This rule will string objects together to form clusters, and the resulting clusters tend to represent long "chains" (Figure 17D).

- Complete linkage clustering (Figure 17B). Here the distances between clusters are determined by the largest distance between any two objects in the different clusters (i.e. by the "furthest neighbors"). This method performs quite well in cases when the objects actually form naturally distinct "clumps" (Figure 17E). If the clusters are of a "chain" type nature, this method is inappropriate.
- Centroid clustering (Figure 17C, also known as average linkage clustering). The distance between two clusters is calculated as the average distance between all pairs of objects in the two different clusters. This method is also very good when the objects form natural distinct "clumps", but it also performs well with elongated "chain" type clusters (Figure 17F). Note that this is the same approach as UPGMA (see Section 9.2).

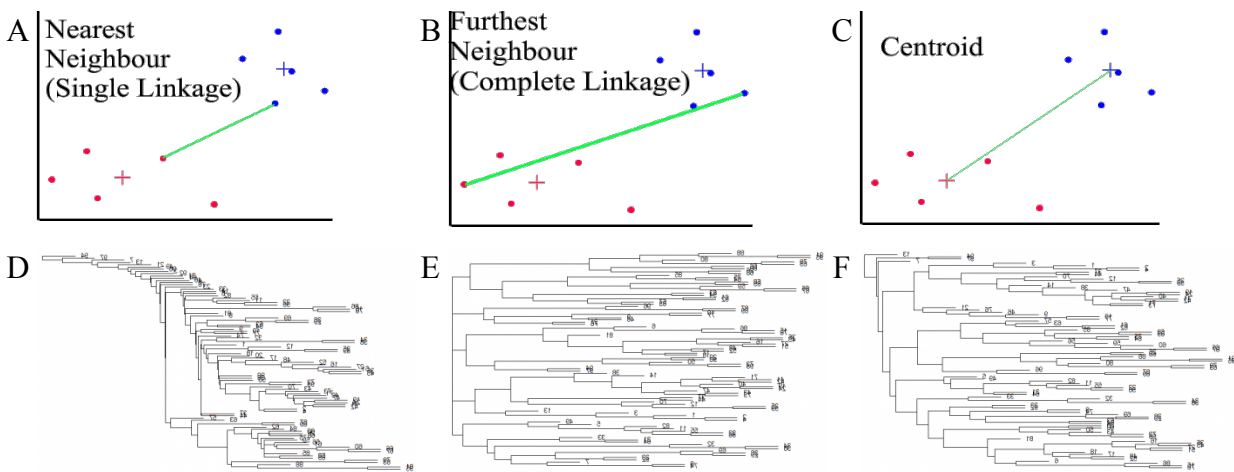
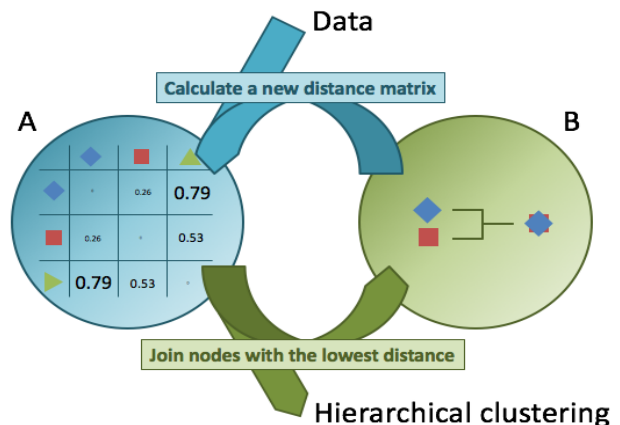


Figure 17. The difference between these hierarchical clustering algorithms lies in the way they define the distance between two clusters: A) single linkage, B) complete linkage, and C) centroid. D-F show typical clusterings, illustrating the chaining (D) and clumping (E) behavior in single and complete linkage, respectively.

In practice, hierarchical clustering a dataset is a process that iterates between two steps: (A) creating a distance matrix as discussed in Sections 3.4 and 3.5, and (B) joining the two most similar nodes as discussed in Section 3.6. As shown in Figure 18, the final result is a hierarchical clustering of all the data.

Figure 18. The process of hierarchical clustering iterates between (A) creating a distance matrix, and (B) joining the two most similar nodes, until there is only one node left: the hierarchical clustering.

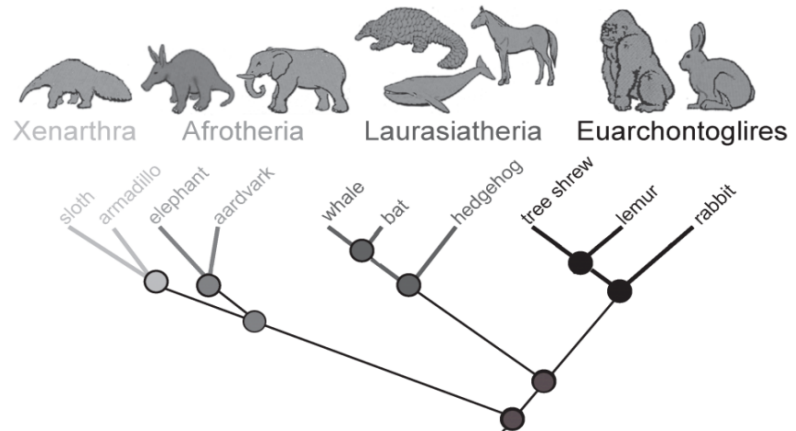


3.7. Newick tree format

We can record a clustered dataset by using a the so-called Newick tree format or bracket-notation. In this format, all items that occur together are listed between a pair of brackets. For example, the notation $((W, X), (Y, Z))$; indicates a dataset of four items, W, X, Y, and Z, that form two separate

clusters: (W, X), and (Y, Z). Clusters can contain any number of items. Moreover, items can, in turn, contain clusters, allowing the definition of nested sub-clusters within the dataset. Thus, this kind of notation allows the complete representation of a dataset in clusters at all levels. This is most easily explained by observing Figure 19, that shows the major groups of mammals represented as clusters in a cladogram.

Figure 19. Cladogram representing groups of mammals as hierarchical or nested clusters. Each cluster is represented by a circle. Figure from <http://www.palaeocast.com/episode-26-the-tree-of-mammals>.



The nested clusters in Figure 19 can be represented as a bracket-notation as follows:

```
((sloth, armadillo), (elephant, aardvark)), ((whale, bat), hedgehog), ((tree shrew, lemur), rabbit));
```

Figure 20. Newick tree format of the nested clustering in Figure 19.

This bracket-notation can also be represented as code that visualizes the nested clusters (Figure 21). The so-called white-space characters (such as spaces, tabs, and newlines) are not taken into account in this code, so a bracket-notation can be written either way, as in Figure 20 or as in Figure 21. Note the semi-colon at the end, indicating that the code is complete. Don't forget to add it at the exam for full marks 😊 !

Figure 21. Code for the Newick tree format of the nested clustering in Figure 19.

```
(
  (
    (
      sloth,
      armadillo
    ),
    (
      elephant,
      aardvark
    )
  ),
  (
    (
      (
        whale,
        bat
      ),
      hedgehog
    ),
    (
      (
        tree shrew,
        lemur
      ),
      rabbit
    )
  )
);
```


The names of sub-clusters, or internal nodes in the cladogram can also be represented in the Newick tree format by including them immediately behind the closing bracket. The bracket-notation of the cladogram in Figure 19 would then become:

```
((sloth, armadillo)Xenarthra, (elephant, aardvark)Afrotheria), (((whale, bat), hedgehog)Laurasiatheria, ((tree shrew, lemur), rabbit)Euarchontoglires))mammals;
```

Figure 22. Newick tree format of the nested clustering in Figure 19, including names for four internal nodes.

Often, for example when studying the clusters in genome-wide gene expression profiles, there are far too many items in the dataset to write the clusters as a bracket-notation. In these cases, the items are simply stored in a file containing a long list with two columns, the first column containing the ID of the clustered object (e.g. the gene ID), and the second column containing the cluster number or cluster ID.

3.8. Reading and videos

- Obligatory: NCBI Tutorial on the analysis of gene expression data: <http://www.youtube.com/watch?v=EUPmGWS8ik0&list=PL8FD4CC12DABD6B39&index=1>
- Optional: (Campbell et al., 2011) chapter 12.
- Optional: (Higgs and Attwood 2005) chapter 13.
- Optional: (Quackenbush 2001): Computational analysis of microarray data.
- Optional: Wikipedia page of the Newick tree format: http://en.wikipedia.org/wiki/Newick_format.

3.9. Exercises

1.
 - a. Observe the PCA plot in Figure 13. Two principal components (PCs) are shown in the figure, but how many dimensions does the underlying data have?
 - b. Observe the PCA plot in Figure 14. Two principal components (PCs) are shown in the figure, but how many dimensions does the underlying data have?
 - c. The figure suggests that the microbiomes in nose and skin cannot be distinguished. Explain why this is the case based on the data. Give a possible biological interpretation.
 - d. Now explain why this is NOT necessarily the case based on the data.
2. Using R, we can easily calculate many different distance/similarity measures.
 - a. Which distance measure(s) would you use to cluster vectors with a similar magnitude?
 - b. If two vectors have a very different magnitude (the numbers in one vector are much higher than in the other), how could you still use the distance measure(s) under a?
 - c. Which distance measure(s) would you use to cluster vectors with a similar profile?
 - d. Give a formula to convert correlation r to distance D . (HINT: see Section 3.5.)
3. We use metagenomics to determine the microbiome in poo of four alpacas (*Vicugna pacas*) from different zoos around the world.
 - a. Download the file <http://tbb.bio.uu.nl/BDA/alpacas.txt> that contains the number of sequencing reads from bacteria A-G in the four metagenomes.

- b. Start RStudio. Import the table into data frame “**alp**” by combining the functions **as.matrix()** and **read.table()** as follows.

```
alp <- as.matrix(read.table('alpacas.txt', row.names=1, header=TRUE))
```

- c. Take a look at the data matrix by typing **alp**. How many rows and columns are there?
d. For which alpaca was the most DNA sequenced? (Hint: use the function **rowSums**.) Does this tell you something about the alpaca in question?
e. Plot the data using the command below. Which alpacas have the most similar microbiome?

```
heatmap(alp, distfun = dist, hclustfun = hclust, Rowv=NA, Colv = NA)
```

- f. What does the following command do? (HINT: Print the new matrix and its row sums.)

```
alp_norm <- alp / rowSums(alp)
```

- g. Plot a heatmap of the matrix **alp_norm**. Does it differ from the one in e? Why (not)?
h. Run the following command to get the Euclidean distance between the alpaca microbiomes in the data frame **deuc**. Which alpacas have the most similar microbiomes? What is the Euclidean distance between the microbiomes of Alpaca1 and Alpaca3? Confirm this by calculating it with pen and paper. (Hint: use Equation 2.)

```
deuc <- dist(alp_norm, method='euclidean')
```

- i. Using R, we can easily calculate many different distance measures. Calculate the Manhattan distance between the alpaca microbiomes in a new data frame. (Hint: give the new data frame a meaningful name.) Manually confirm the Manhattan distance between Alpaca1 and Alpaca3 that you find using Equation 1.
j. Are the Manhattan distances between the microbiome pairs always larger than the Euclidean distances?
k. What do you expect will be in the variable **dpear** if you type the following command? Explain the command by writing out the conversion formula. (Hint: see Section 3.5.)

```
dpear <- as.dist(0.5 - cor(t(alp_norm), method='pearson') / 2)
```

- l. Take the Euclidean distance matrix. Cluster the alpacas using the **hclust()** command and put the result into a new data structure with the name “**calp**” (this name is important for an exercise below). By default, **hclust** uses complete linkage. Display the clustering using the **plot()** command and write this clustering using a bracket-notation.
m. Which Alpacas have the most similar microbiome? Compare the result with your answer under e, and with the distance matrix under h. Give a biological hypothesis about these alpacas.

- n. Use the function `str()` to look at the data structure returned by `hclust`. Now take a look at `calp$height`. These are the actual branch lengths of the clustering. Use pen and paper to confirm the first two values (0.1491943, 0.2192003).
- o. Cluster the alpacas using single linkage and display the clustering using the `plot()` command. (Hint: read `hclust` help page to see how you should do this.) Did the clustering change? Did the branch lengths change? Use pen and paper to confirm.
- p. Now that we know how to do the clustering, we can easily cluster the bacteria as well. Look at the following commands, and explain per command what happens. (Hint: the function `t()` transposes a matrix.)

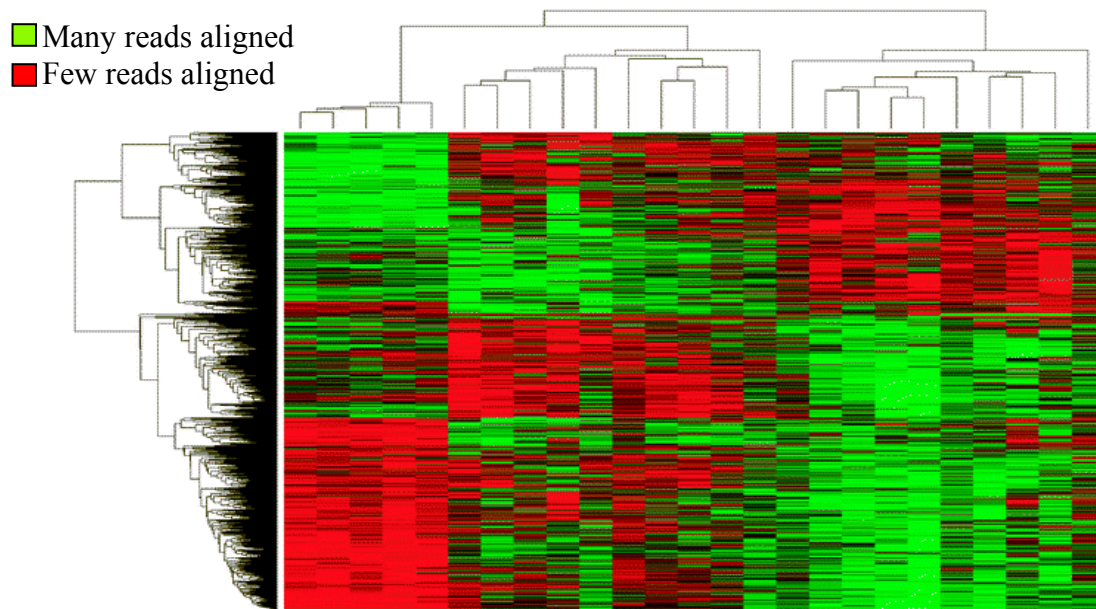
```
alpt <- t (alp)
alpt_norm <- alpt / rowSums(alpt)
dalpt_norm <- dist (alpt_norm, method='euclidean')
cbac <- hclust (dalpt_norm, method='complete')
plot (cbac)
```

- q. Come up with a possible biological interpretation why bacteria D and G cluster together.
- r. Finally, we can make a bi-clustered heatmap of this same data as follows.

```
heatmap(alp_norm, Rowv=as.dendrogram(calp), Colv=as.dendrogram(cbac))
```

Look at the clustering of rows and columns. Do they look familiar? Compare this with the heatmaps and clusterings you made above.

4. The figure below is derived from transcriptomic sequencing of a range of human cancers.
 - a. What is transcriptomics?
 - b. How many cancers were included in the study?
 - c. Explain in some detail how this figure was obtained. Include as many experimental and bioinformatic steps as you can think of.
 - d. Indicate in the figure which 3 cancers have the most similar gene expression profiles.
 - e. Give a possible explanation why some genes may have similar expression patterns across cancers.



4. Phylogenetic trees and genetic relatedness

Baffled by the variety of life, one of man's first biological activities has been to classify it. Since Darwin drew the first image of connected evolutionary lineages (Figure 23) (Darwin 1859), many biologists have been involved in the quest to obtain a hierarchical classification of all species that is consistent with their evolutionary relationships, also known as the Tree of Life. This makes the construction of trees one of the central activities of biologists, not only to reconstruct the tree of life, but also to understand the functional similarities between organisms in terms of their relationships, because "nothing in biology makes sense except in the light of evolution" (Dobzhansky 1973).

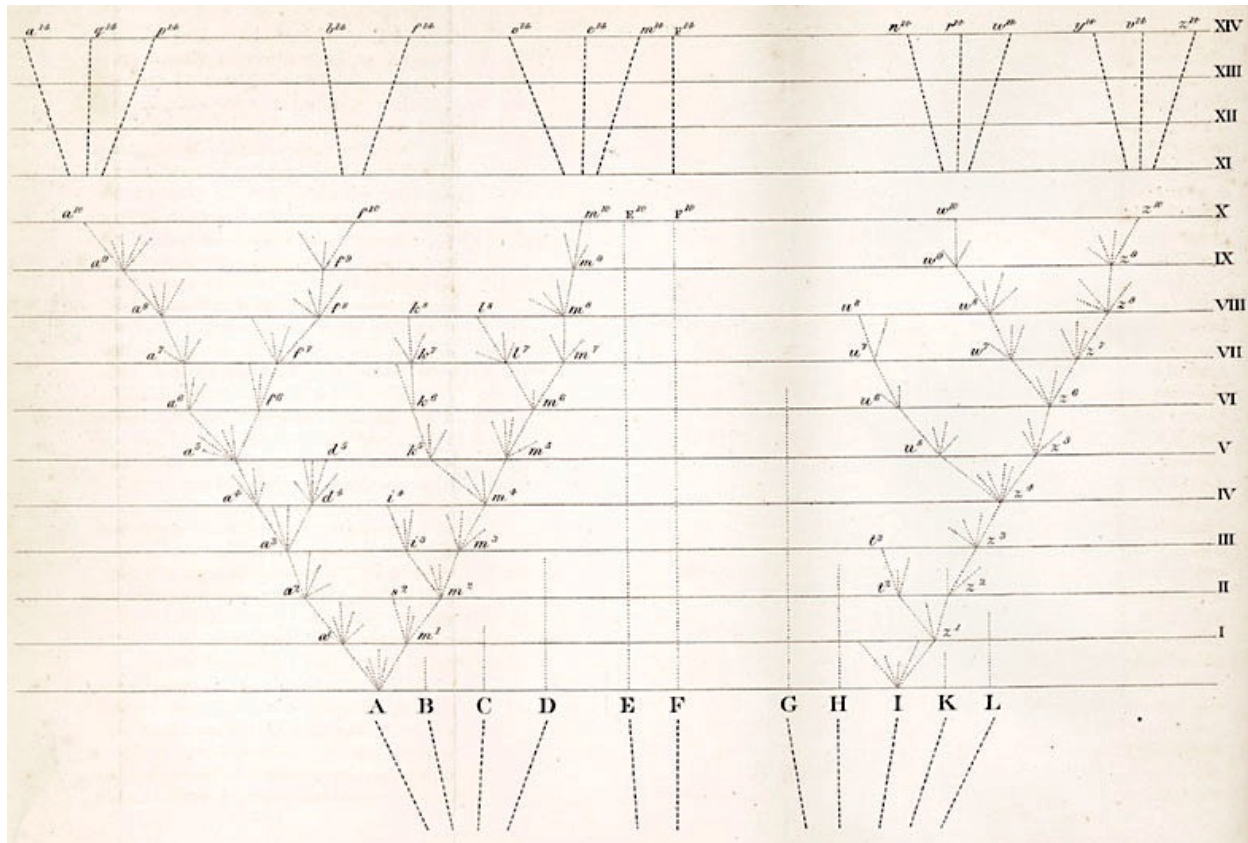


Figure 23. Diagram showing Darwin's conception of evolution as a branching, or tree-like, process (Darwin 1859). The lines reaching the top of the diagram are terminal nodes that contain currently surviving species. Terminal nodes ending lower than the top are extinct lineages, revealing Darwin's hunch, based on fossil records, that extinction rates are high.

4.1. Phylogenetic trees

Evolution requires only two basic ingredients: reproduction (i) with variation (ii). A third important ingredient, selection (iii), is not even required: research has shown that a lot of evolution is in fact neutral, meaning that things may change without necessarily getting fitter – or less fit. All things that evolve possess these ingredients, and conversely everything that has these ingredients can, in principle, evolve. In reproducing (i) biological entities such as genes, cells, organisms, or even populations, variation (ii) tends to occur at the most basic level by mutations in the DNA sequence, while selection (iii) takes place at a higher level. At the level of the organism for example, most

from their sequences. In fact, molecular phylogenies are the golden standard for inferring biological relationships whenever sequences can be obtained.

4.2. Reading phylogenetic trees

Phylogenies can be displayed in different ways. When observing a phylogenetic tree, there are several important things to look out for. First are the terminal nodes, also known as "branch tips" or "leaves" of the tree. The terminal nodes contain the observations that were used to construct the tree, generally sequences representing biological entities like species or genes. (We will discuss how to reconstruct a phylogeny in Chapter 9.)

Second are the evolutionary lineages, which are represented as lines in the cladogram. These lineages represent evolving organisms that slowly change in evolutionary time, and sometimes split into two daughter lineages, for example by a gene duplication event where a gene is copied within a chromosome, or at a higher level by a speciation event where a new species arises (more about this in Section 4.5). The genes or species that are derived from a single ancestor form a clade, i.e. a single branch in the cladogram. Properties that only occur in a single clade are called monophyletic, while properties that occur in several clades are called polyphyletic. It is thought that evolution is fundamentally a bifurcating process, which means that one ancestral lineage always splits into two daughter lineages, and never more than that at once. However, sometimes it can be difficult to resolve the order of bifurcations, in which case a tree can be drawn where one ancestral lineage splits into three or more daughter lineages (these are called multifurcations).

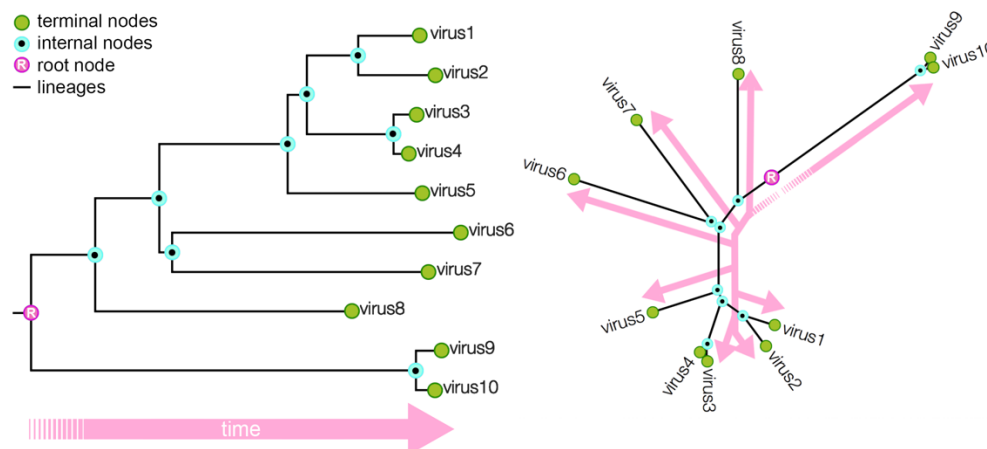


Figure 25. Different ways of displaying the same phylogeny, here a phylogeny of viruses. A: normal display; B: radial display. Pink arrows indicate the time axis starting at the root and running from the past to the present, ending at the branch tips. Figures adapted from http://epidemic.bio.ed.ac.uk/how_to_read_a_phylogeny.

The third thing to note are the internal nodes, each of which represents the last common ancestor (LCA) of the lineages that are derived from it. In the trees in Figure 25, for example, we see that virus1 and virus2 share a LCA, that virus3 and virus4 share a LCA, that all four viruses 1-4 share a LCA, etcetera. A special case of an internal node is the root, which represents the LCA of all the lineages in the tree. As we will learn in Chapter 9, the root is not always present in a tree, but if it is then it defines a time axis along all the branches. By definition, the root is the oldest node in the tree, so time always progresses away from the root, along the branches towards the tips. Here it is important to note that there are different ways of displaying the same phylogenetic tree. In the

"radial" display (Figure 25B) the time axis runs away from the root along all the lineages of the tree, ending in the tips. In the "normal" display (Figure 25A) the time axis only runs horizontally from left to right, but not along the vertical lines. The vertical lines are only drawn to show how the different lineages are connected, i.e. which ancestral lineage connects to which daughter lineages.

Now re-examine Darwin's diagram of connected evolutionary lineages in Figure 23 and look for the time axis. Note that time runs vertically (up), not horizontally. While the drawn lines indicate how the evolutionary lineages are connected by descent, the space between the branches does not mean anything at all. For example, the order of the letters A-L in the diagram, although alphabetical, does not mean anything in terms of the biology of the lineages. Indeed, the branches in phylogenetic trees can always be rotated, as long as no changes are made in the connections between the lineages or in the length of the branches along the time axis. The best visualization of this point is an artwork located in the lobby of the Broad Institute: a phylogenetic mobile of mammals (Figure 26). The mobile is the same as the "normal" display (Figure 25 on the left) rotated 90 degrees clockwise. Here, the root node is located in the string where the mobile attaches to the ceiling, and the time axis runs vertically down the strings towards the species in the tips, but not along the horizontal hangers. Thus, the phylogenetic distance between two organisms can be measured by summing the lengths of all the strings that separate them.

Figure 26.
Phylogenetic mobile
of mammals with a
sequenced genome.
Diameter: 3.5 m.
Location: lobby of
the Broad Institute
in Cambridge,
Massachusetts, USA.
Design: Peter Agoos.

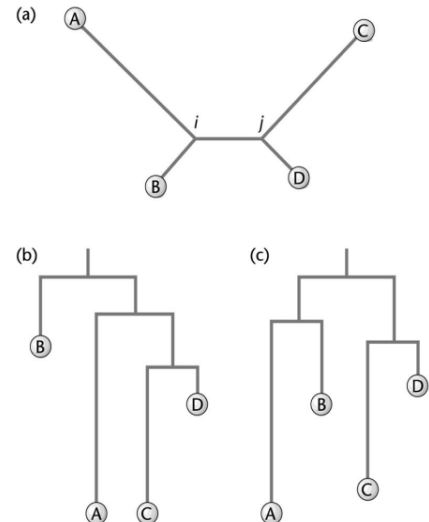


4.3. Rooting: providing a direction to the time axis

In rooted trees, a single node is designated as a common ancestor and a unique path leads from it to all other species studied (see Section 4.2). Unrooted trees only specify the relationship between nodes and give no information about the direction of time and evolution. Figure 27a shows an unrooted tree of four species. Here the internal nodes (represented by *i* and *j*) represent ancestors of which we do not have data. The branch lengths are again scaled with evolutionary distance, e.g. we can say that there had been a lot of changes on the branch leading to species A/C compared to species B/D. However, we cannot say anything about whether or not the branching at *i* occurred earlier or later than in *j*. When a tree is displayed as in Figure 27a, it is clearly an unrooted tree.

Only if a location for the root is chosen, then unrooted trees can be transformed into a rooted tree as in Figure 27b (root chosen in branch leading to B) or in Figure 27c (root chosen in the internal branch between *i* and *j*). Note that sometimes, unrooted trees are erroneously drawn as rooted trees. Thus, you should always read the figure caption carefully to make sure you know if the tree is really rooted or not. Unrooted trees should always be drawn radially branching (as in Figure 27a).

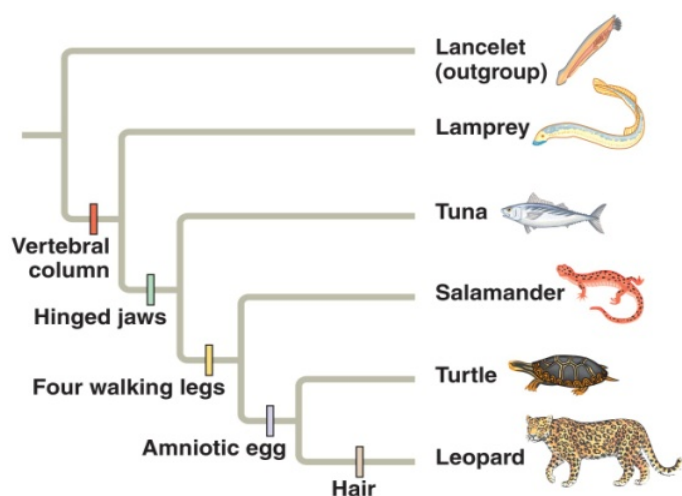
Figure 27. (a) An unrooted tree. This unrooted tree could be converted to a rooted tree, such as the one in (b) if B is the outgroup, or the one in (c) if none of the species is an outgroup. Figure 8.3 in (Higgs and Attwood 2005).



How to choose the correct location of the root for an unrooted tree? Roots can usually be identified in a tree by using an outgroup, i.e. a species for which we know that it separated the earliest from all the other species. For example, if we want to study the relationships between the vertebrates: leopard, turtle, salamander, tuna, and lamprey (a jawless aquatic vertebrate), a proper outgroup to use would be the lancelet, a small animal that lives in mudflats and (like vertebrates) is a member of the *Chordata*, but branched off before the expansion of the vertebrates. Unlike the vertebrates, however, the lancelet does not have a backbone. The root of the tree would then be somewhere along the branch connecting lancelet to the common ancestor of all other vertebrates (see Figure 28).

Figure 28. A phylogenetic tree of vertebrates with Lancelet as an outgroup. Figure 26.11b in (Campbell et al., 2011).

In both rooted and unrooted trees, the leaves are grouped in clusters. This grouping depends heavily on the algorithm used to construct the tree. Some algorithms just give one of potentially many, more or less equally probable, outputs. Other approaches actually calculate many different solutions and give the most probable outcome with some indication of how reliable a particular solution is.



4.4. **Phylogenetic distance and the molecular clock**

Phylogenetic distance is expressed in the number of changes that happened on a given branch: the longer a branch, the more changes occurred. In molecular phylogenetic trees that are based on sequences (i.e. most trees these days), the changes consist of genetic mutations, and the phylogenetic distance might be expressed as the number of mutations per given length of sequence. As explained in Section 4.1, sequences that are very similar have only recently diverged, and the more time has passed since two evolving biological entities diverged, the more their sequences differ. This correlation was first observed by Emile Zuckerkandl and Linus Pauling based on the observation that the number of amino acid differences between hemoglobin sequences in different lineages changes roughly linearly with the fossil age of these lineages, and has become known as the molecular clock (Zuckerkandl and Pauling 1962). In reverse, if the molecular clock is properly calibrated, the number of mutations between two sequences can be used to estimate the time when they diverged. Such molecular dating is an enormously powerful tool that has, for example, been used to discover when the lineage leading to the 2014 Ebola outbreak arose, by comparing the genome sequences of Ebola viruses from many patients (Figure 29).

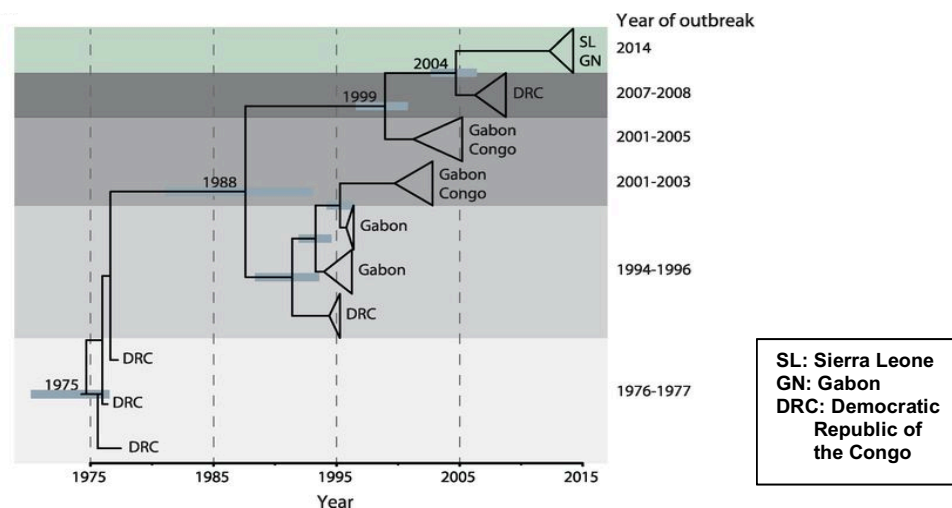


Figure 29. Molecular dating of the 2014 Ebola outbreak. Figure from

<http://scienceintheclassroom.org/research-papers/ebola-outbreak-traced-funeral-traditional-healer/university>.

The molecular clock runs most accurately under neutral evolution, i.e. when there is no selection on the mutations in the sequence. Thus, while the molecular clock is a very useful concept, it does not always hold when the number of mutations in a certain lineage is higher or lower than expected. This is known as positive or negative selection, respectively, where “selection” refers to the third ingredient of evolution (see Section 4.1). Positive selection occurs when the environment of an organism is variable, so that the selected “fit” phenotypic characteristics also change. An example are the proteins in the outer layer of viruses that are under constant attack by the immune system. Negative selection occurs when a certain function is very optimally encoded in the genotype, so that any change in the sequence will have a negative fitness effect. An example are ribosomal proteins that catalyze the translation of RNA into protein and are conserved in all living organisms. Both positive and negative selection are examples of non-neutral evolution, i.e. when the trait affects the fitness of an organism in its environment.

4.5. Types of homology

Homology is defined as the existence of shared ancestry between sequences. This means that all sequences that share a common ancestor in a phylogenetic tree are homologous, and because all lineages in a tree are derived from the root, all sequences in a phylogeny are necessarily homologous. Homology is exactly like family: you are related or you are not related, but there is nothing in between. Homology is a binary property, and although you see it in the scientific literature all the time, statements like “Gene X is highly homologous” or “This gene is more homologous than that gene” are wrong and should be avoided. As we will discuss in Chapter 6, you could instead make statements about the degree of similarity between two sequences. Sequence similarity is a proxy for homology: if the sequences of two genes or proteins are very similar, they probably descend from a common ancestor, and are thus probably homologous.

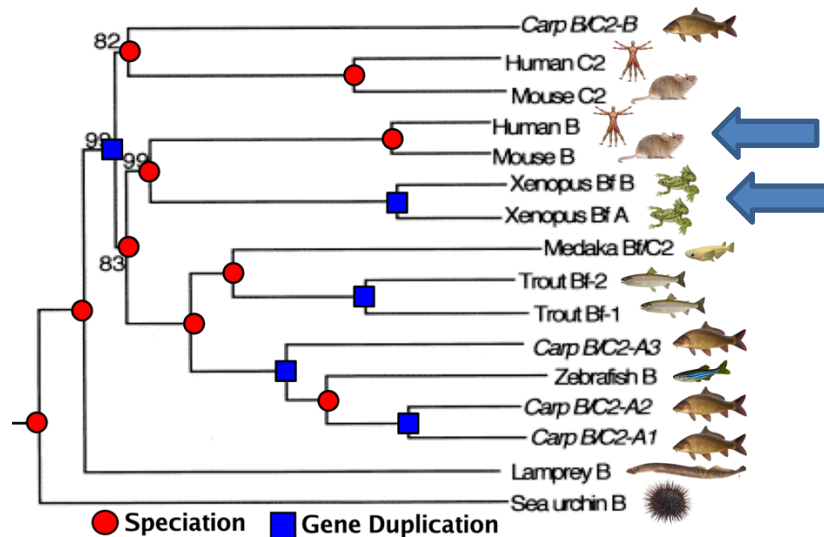


Figure 30. Phylogenetic tree of gene family members that are present in several animals. All genes in the tree are homologous, i.e. they are derived from one gene in the last common ancestor (the root of the tree). However, some animals occur more than once in the tree, which means that somewhere in their ancestral lineage, the gene duplicated in the genome. Based on a comparison of the gene tree with the species tree, likely speciation and gene duplication events are assigned to the internal nodes in the phylogeny.

There are different types of homology, depending on how two genes diverged at their last common ancestor. Observe the phylogeny of a gene family in Figure 30. At every internal node in the tree, the ancestral lineage bifurcated into two daughter lineages, but not all nodes are the same. This becomes clear if we compare the lineage with two branch tips labelled “Human B” and “Mouse B” to the lineage with two branch tips labelled “*Xenopus* Bf B” and “*Xenopus* Bf A” (see arrows). In the first, the ancestor of these two genes probably occurred in the genome of a mammalian ancestor, and the bifurcation we see represents the speciation event where the ancestor of humans diverged from the ancestor of mice. In the second, the ancestor of these two genes probably occurred in *Xenopus* or a recent amphibian ancestor. Since “*Xenopus* Bf B” and “*Xenopus* Bf A” are two genes that both occur in the genome of *Xenopus*, the bifurcation in the tree represents the duplication of this gene within the genome of this amphibian ancestor: it is a gene duplication node. As a rule of thumb, you can recognize a gene duplication node in the phylogenetic tree of a gene family as a bifurcation where the same species is present in both daughter lineages, and a speciation node otherwise. (But there are many exceptions to this rule of thumb! We will address these in the

exercises in Section 4.10.) Homologous genes within one genome that are the result of a gene duplication event are called paralogs. In contrast, orthologs are genes in different species that diverged from a common ancestral sequence as a result of a speciation event.

Figure 31. Fictional example of the evolution of gene family A during in *Bacilli*, illustrating the concepts of orthology and paralogy. The dark shaded tree in the background indicates the evolutionary history of the six *Bacilli* genomes, while the colored lines indicate the evolution of the gene family. The genes A, X, and Y that are found in the present-day genomes are indicated in circles at the branch tips. Figure from (Dutilh et al. 2013).

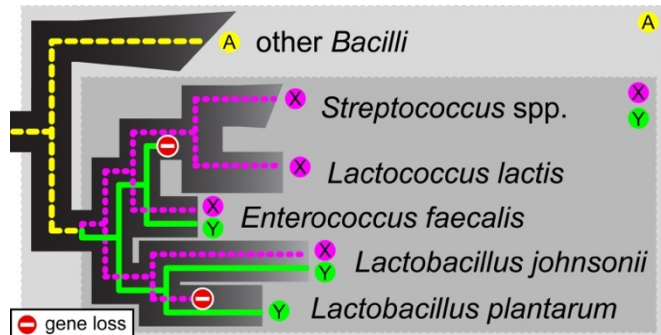


Figure 31 illustrates orthology and paralogy. In this example, before the radiation of the *Lactobacillales* (the cluster of species containing *Streptococcus*, *Lactotoccus*, *Enterococcus*, and *Lactobacillus*), their ancestor contained one gene in this gene family, which was an ortholog of A (yellow dashed line). This ancestral gene duplicated to form the two paralogs X and Y (the pink dashed, and green drawn lines). Thus, A and X/Y are orthologs of each other because they diverged at a speciation event, while X and Y are paralogs of each other because they diverged due to a gene duplication event.

4.6. Gene trees and species trees

The starting point of any phylogenetic work is a collection of sequences that are evolutionarily related, i.e. that are homologous. Such a set could be extracted from public databases using some of the tools described previously (for example BLAST), or it could be data from one's own work. It is important to realize that there are at least two types of trees: gene trees and species trees. If a phylogenetic tree is based on the divergence observed within a single homologous gene, then it represents only the evolutionary history of that gene, i.e. it is a gene tree. Species trees are often made using different sources of information, such as multiple genes, gene order, or, as was traditionally the case, using morphological characters. Discrepancies between gene trees and species trees can arise because genes have their own dynamics within populations and genomes. For example, divergence within genes typically occurs prior to the splitting of populations during speciation. This occurs particularly frequently in genes, for which (allelic) diversity in population is advantageous. A prominent example of this is the major histocompatibility complex (MHC) which has the largest degree of polymorphism among mammalian proteins. If MHC molecules alone were used to determine species trees, many humans would be grouped with gorillas rather than with other humans, because the origin of polymorphism they carry pre-dates the split that gave rise to the two lineages (see Figure 32).

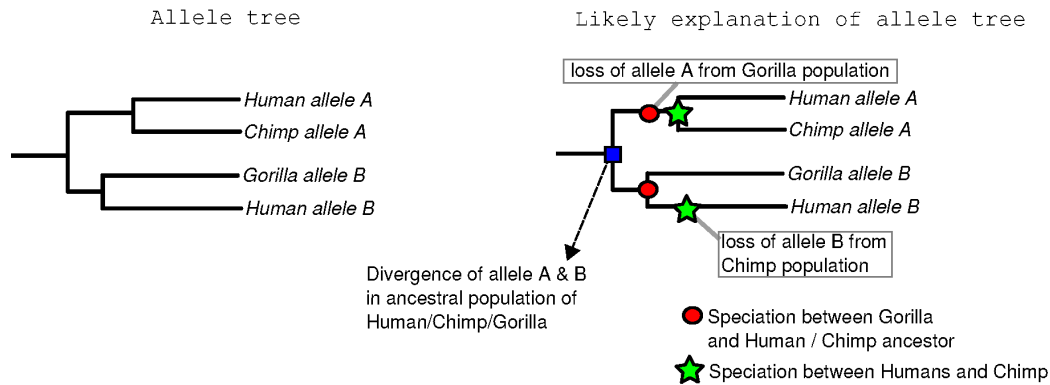
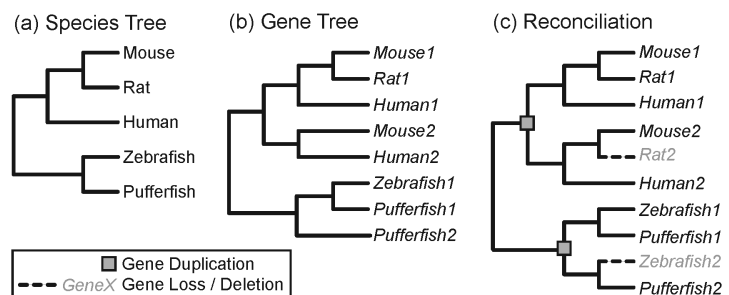


Figure 32. The first tree is a tree of alleles (gene variants) from humans, chimps and gorillas. The second tree is a likely explanation for this tree in terms of allelic divergence and loss of alleles. Allelic divergence means that a mutation generates a new variant for a gene in the population and that the old and new variant (i.e. alleles) remain present in that population. The explanation uses the generally accepted phylogeny of primates wherein human and chimpanzee are more closely related to each other than either is to gorilla.

Gene duplication is another reason that gene trees and species trees may differ. Gene duplication affects trees over much larger time-scales than the allelic divergence, yet they are similar in the sense that one species can harbor multiple variants of a homologous gene and that thus a tree of genes from these species can display multiple genes per species. Gene duplications become especially confusing when they are followed by gene loss, i.e. a gene is completely deleted or deteriorates after a nonsense mutation. We have illustrated the effect on the interpretation of a gene tree in the light of the occurrence of gene duplications and gene losses in Figure 33. The occurrence of multiple copies of the hypothetical gene from the figure in human, mouse and pufferfish already suggests that one or more gene duplications have occurred. We can make a reconciliation of the gene tree with the species trees based on i) the inference of one gene duplication in the ancestor of the mammals followed by the loss of one of the duplicate genes in rat and ii) one gene duplication in the ancestor of the two teleost fish followed by the loss of one of the copies in zebrafish.

Figure 33. Species trees versus gene trees. Panel (a) depicts the species tree and panel (b) depicts a reconstructed gene tree. Panel (c) depicts how we can explain the gene tree by drawing a reconstruction of likely past events on an adjusted version of the gene tree. The gene tree is fictional and was thought up specifically for this example.



4.7. Orthology and the biological function of genes

You may ask: why is it important to know if genes diverged by speciation or by gene duplication? This is not only relevant if we want to know how a gene family evolved (which may be interesting in its own right), but it is also important if we are investigating the function of genes. The function of genes within one family is often somehow similar, but there may be important differences. After a gene duplication event, a genome suddenly contains two paralogous copies of the gene. Most often when this happens, one of the genes accumulates deleterious mutations, becomes non-functional, and is eventually lost from the genome. Occasionally though, both copies are kept, and

one of the two daughter genes acquires mutations that lead to a new function and, if beneficial, the two paralogs become fixed in the lineage. A change in function may be less likely for two gene family members that diverged at a speciation node. In that case, what happened is that two species diverged, but there is still one copy of that same gene within both their genomes. In most cases, the function of the orthologs will be conserved between the two organisms. Thus, orthologs are often viewed as the same gene but in a different species. When analyzing the functions of genes in a new species, finding orthologs of the genes in a well-studied species such as yeast or *Escherichia coli* can provide us with a first clue about their function. This is known as the orthology conjecture.

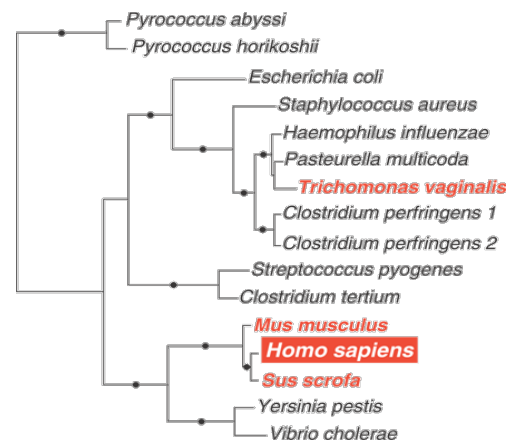
Although the transfer of functional annotation between orthologs is very important in many fields of biology, it is very important to remember that the concepts of paralogy and orthology are defined in terms of the evolutionary events in the history of the gene family (Fitch 1970), and make no statement about the function of the gene or its protein product!

4.8. Horizontal gene transfer (HGT)

The Tree of Life and other tree-like representations of the taxonomic classification of living species have an illustrious history in evolutionary biology (since Darwin, see Figure 23). However, trees cannot provide a completely accurate, nor a complete representation of life's history. For a given group of genomes, some genes within those genomes have very different phylogenies than other genes, which means that not all genes in a genome necessarily have the same evolutionary history. This occurs especially frequently in *Bacteria* and *Archaea* owing to a process known as horizontal gene transfer (HGT) (Doolittle et al. 1999). HGT is most often thought of as a mechanism for the mobilization of chromosomal DNA among bacterial and archaeal cells. Although all types of genes can be susceptible to HGT, different types of genes and groups of organisms vary in their propensity for HGT. Three main mechanisms of HGT have been described:

- Natural transformation, which is the uptake of free DNA from the environment into competent species, exhibited by about 1% of validly described bacterial species.
- Transduction, or the transfer of DNA between an infected cell and its infecting agent (for example a bacteriophage, i.e. a virus that infects bacteria).
- Conjugation, the transfer of mobile genetic elements by specialized structures assembled between two adjacently located cells.

Figure 34. Phylogenetic tree of the gene encoding N-acetylneuraminate lyase. Bacteria donated this gene to the protozoan parasite *Trichomonas vaginalis*. Vertebrates (human, mouse, and pig) together with two bacterial lineages (*Vibrio cholerae* and *Yersinia pestis*) also show a branching pattern indicative of HGT, although it is not possible to infer the direction of the transfer from this tree. Figure adapted from (Andersson, Doolittle, and Nesbø 2001).



Keeping the species tree in mind, it is possible to detect HGT in gene trees. In Figure 34 an example of HGT in the N-acetylneuraminidase lyase gene family is illustrated. The *T. vaginalis* version of this gene clusters with the bacteria *H. influenzae* and *P. multocida*, indicating HGT from bacteria to the protozoan parasite. The vertebrate version of this gene clusters very convincingly with *V. cholerae* and *Y. pestis* genes, indicative of HGT involving bacteria and eukaryotes.

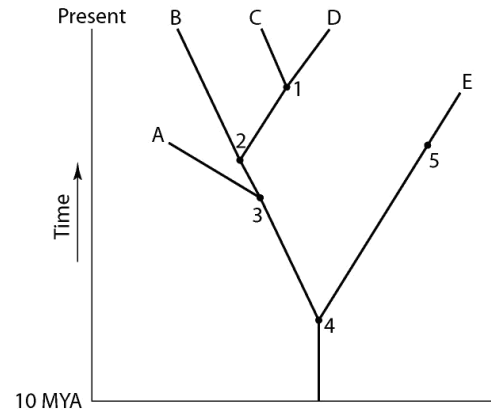
4.9. **Reading and videos**

- Obligatory: http://epidemic.bio.ed.ac.uk/how_to_read_a_phylogeny.
- Optional: from the 9th edition of Campbell (Campbell et al., 2011), (see the info on the course website for the corresponding chapters in 10th edition of Campbell):
 - o Chapter 5 (5.1 Macromolecules, 5.4 Proteins, and 5.5 Nucleic acids)
 - o Chapter 21 (Evolution of genomes)
- Optional: "The never-ending quest to rewrite the Tree of Life" by Carrie Arnold: <http://www.pbs.org/wgbh/nova/next/evolution/microbial-diversity/>
- Optional: "The ancestor tale" by Richard Dawkins.

4.10. **Questions and exercises**

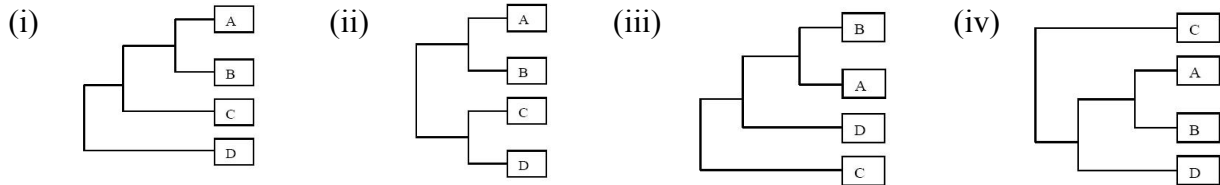
1. The four-chambered hearts of birds and the four-chambered hearts of mammals evolved independently of each other. If one were unaware of this analogy/homoplasy, then one might wrongly conclude that:
 - a. the birds were the first to evolve a 4-chambered heart.
 - b. birds and mammals seem phenotypically more distantly related than is actually the case.
 - c. early mammals possessed feathers.
 - d. the common ancestor of birds and mammals had a four-chambered heart.
 - e. birds and mammals should be placed in the same family.
2. Which statement represents the best explanation for the observation that the nuclear DNA of wolves and domestic dogs has a very high degree of similarity?
 - a. Dogs and wolves have very similar morphologies.
 - b. Dogs and wolves belong to the same order.
 - c. Dogs and wolves are both members of the order Carnivora.
 - d. Dogs and wolves shared a common ancestor very recently.
 - e. Convergent evolution has occurred.
3. Genes that have lost the function of coding for a functional protein product may become "pseudogenes." Which of these is a valid prediction regarding the fate of pseudogenes over evolutionary time?
 - a. They will be preserved by natural selection.
 - b. They will be highly conserved.
 - c. They will ultimately regain their original function.
 - d. They will be transformed into orthologous genes.
 - e. They will have relatively high number of mutations because of not having selection pressure.

Use the tree (right) to answer the following questions. Assume that the tree accurately reflects the evolutionary relationships between species A-E.



4. A common ancestor for both species C and species E might be found at:
 - a. Internal node 1
 - b. Internal node 2
 - c. Internal node 3
 - d. Internal node 4
 - e. Internal node 5
 - f. Internal nodes 1-5.
5. The two currently living species that are most closely related to each other are:
 - a. Species A and species B
 - b. Species B and species C
 - c. Species C and species D
 - d. Species D and species E
 - e. Species E and species A
6. Which of the following statements is true? i) The last common ancestor of B and C occurred more recently than the last common ancestor of D and E. ii) A is the direct ancestor of both B and C. iii) The species at position 3 is ancestral to C, D, and E.
 - a. Statement i.
 - b. Statement ii.
 - c. Statement iii.
 - d. Statements i and ii.
 - e. Statements i and iii.
 - f. Statements ii and iii.
 - g. Statements i, ii, and iii.
7. Which of the following statements is false? i) A is more closely related to B than to E. ii) B is more closely related to C than to D. iii) C is more closely related to B than to A.
 - a. Statement i.
 - b. Statement ii.
 - c. Statement iii.
 - d. Statements i and ii.
 - e. Statements i and iii.
 - f. Statements ii and iii.
 - g. Statements i, ii, and iii.
8. a. Illustrate what the following terms mean by drawing the situations in trees and explaining them in words: i) a monophyletic group; ii) a polyphyletic group; iii) a paraphyletic group.
 - b. For each of the internal nodes 1-5, indicate if it gives rise to a monophyletic group, a polyphyletic group, or a paraphyletic group.

9. Look at the four trees below. The branch lengths are not drawn to scale.
- Assuming the trees were rooted, which trees would be equivalent?
 - Consider them unrooted and redraw them as such (i.e. star representation). Which trees are then equivalent?



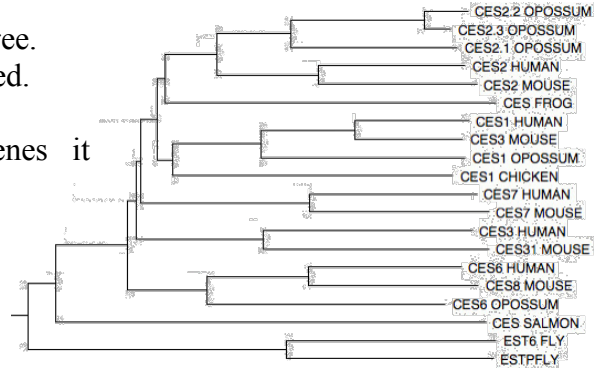
10. A new organism is discovered in the forests of Costa Rica. Scientists there determine that the polypeptide sequence of hemoglobin from the new organism has 72 amino acid differences from humans, 65 differences from a gibbon, 49 differences from a rat, and 5 differences from a frog. These data suggest that the new organism:
- Is more closely related to humans than to frogs.
 - Is more closely related to frogs than to humans.
 - May have evolved from gibbons but not rats.
 - Is more closely related to humans than to rats.
 - May have evolved from rats but not from humans and gibbons.
11. As a rule of thumb, you can recognize a gene duplication node in the phylogenetic tree of a gene family as a bifurcation where the same species is present in both daughter lineages, and a speciation node otherwise (see Section 4.5).
- Sketch the phylogenetic tree of a gene X that underwent the following events throughout its evolutionary history: (i) X invented in a last common ancestor (LCA); (ii) LCA speciated into two daughter species A and B that both retained X; (iii) X_A duplicated, leading to two paralogous copies. Indicate the speciation node with a circle and the gene duplication node with a square. Does the rule of thumb hold?
 - A researcher sequences the genomes of species A and B, and finds that they both contain two genes from gene family X. A rooted phylogenetic tree is shown to the right. Does the rule of thumb hold? What additional evolutionary event(s) could account for this observation?
 - Species A is the researcher's favorite model organism and she has the experimental tools to measure the functions of genes X_{A1} and X_{A2} . She now wants to make a statement about the function of the genes in species B and extrapolates as follows: $X_{A1} \rightarrow X_{B1}$ and $X_{A2} \rightarrow X_{B2}$. Do you agree? Why (not)?
 - Now let's go back to the tree you drew under a. Imagine that the node you indicated as a speciation node actually reflects a gene duplication event in the LCA of species A and B. This means the rule of thumb does not hold. What additional evolutionary event(s) do you need to assume for this to be true? Sketch a new tree indicating the event(s).
 - What if the speciation node in the tree under a was actually a gene duplication event in species A, after it diverged from species B? Sketch a new tree indicating additional evolutionary event(s) do you need to assume.



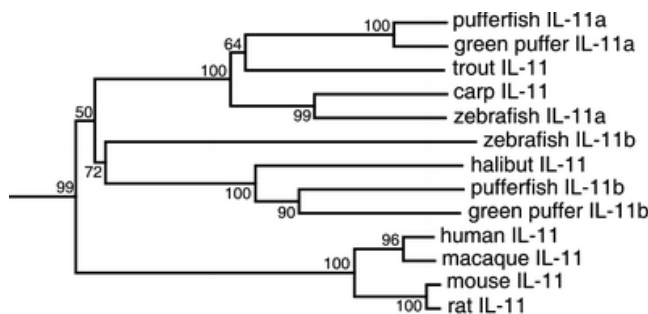
12. A researcher sequences the genome of a newly discovered species, and wants to predict the functions of the encoded genes. For some genes, he can find clear orthologs in a closely related model organism where most gene functions are well-described. Other genes were found to be duplicated in the new genome. Explain in your own words why it is dangerous to transfer functional annotations between paralogous genes.

13. Observe the following rooted phylogenetic tree.

- Explain how you think this tree was rooted.
- List the species that are found in the tree.
- For each species, indicate which genes it contains in its genome.
- For each of the genes, indicate how it ended up in that genome.



14. Interleukin-11 (IL-11) is a key cytokine in the regulation of proliferation and differentiation of hematopoietic progenitors and is also involved in bone formation, and protection of mucosal epithelium from pathogens. (Huisin et al. 2005) studied the evolution of this gene. Below is part of their phylogenetic analysis. Explain from this tree where gene duplication, gene loss and HGT might have taken place.



5. Sequence conservation

As we have seen in Chapter 3, genes and proteins that have only recently diverged from a common ancestor are very similar in their sequence, and the number of differences increases by accumulation of mutations as they diverge down their respective evolutionary lineages. To see what this sequence divergence, and its counterpart sequence conservation look like, let's take a closer look at the sequences themselves.

Figure 35. Multiple protein sequence alignment of the Mad2 interacting motif in plants. On the left a species tree of the plants in the alignment. *Chlorella variabilis* and *Micromonas* have lost the motif.



5.1. Sequence alignments

Figure 35 displays a sequence alignment of a short protein motif* that occurs in some proteins in plants. Sequence motifs can occur in protein or in DNA sequences (see Box 1), and might have some function. For example, the motif in Figure 35 allows the proteins containing this motif to interact with another protein, Mad2. The phylogenetic tree on the left of the sequence alignment shows the evolutionary relationships between these sequences. The hypothesis in a sequence alignment is that the residues in one column are derived from the same residue in the sequence of their common ancestor, i.e. they are homologous residues. Alignment makes it easy to see how the evolutionary process shaped these sequences. For example, you can very quickly see which residues are conserved and which ones are variable in evolution. You can also see when some mutations occurred by comparing the alignment with the phylogenetic tree on the left. For example, the mutation from proline (P) to leucine (L) at position 9 in the alignment happened in the evolution of the *Embryophyta*, and was retained in that lineage after *Physcomitrella patens* branched off from the lineage leading to *Arabidopsis thaliana*, and was conserved in all the species in that lineage. Other amino acids are much less conserved in the Mad2 interacting motif, like positions 6, 7, and 10 that change much more frequently in evolution.

Using a multiple sequence alignment, we can speculate about the importance of specific residues for the function of a protein based on their conservation. Amino acids (or nucleotides) that are very conserved in evolution are probably important, especially if the other residues in the protein are much more variable. Residues that change all the time may be less critical for protein function. A quick look at a sequence alignment can quickly give you an idea which regions in a sequence are important. Check the alignment of the Bzip transcription factor in Figure 36. Two regions in this alignment immediately stand out. While there can be quite some variation in some parts of the Bzip protein, these two regions are conserved, i.e. there are fewer mutations in the protein sequence than expected. Such regions in proteins are separate functional domains, that form semi-independent units in the protein tertiary structure, perform a separate part of the protein function, and can evolve

* A sequence motif is a short piece of sequence that occurs more frequently than expected, so they can be discovered by statistical analysis. Similarly, the word motif can also be used in art for a repeated decorative image forming a pattern, or for a recurrent theme throughout a musical or literary composition (leitmotif). *Motif* is originally a French word so it rhymes with *beef*, not with *motive*.

independently, sometimes being present in different proteins. Domains are longer than motifs. Interestingly, protein domains are a form of modular architecture (see Section 2.2.2) that allows evolution to re-use and recombine simple sub-structures leading to complex function. Examples of domain functions include (i) binding to other molecules such as DNA, another protein, or to a metabolite, (ii) holding a metal ion within the protein that stabilizes it, or (iii) performing a catalytic function like phosphorylating or hydrolyzing other molecules.

Nucleotides and amino acids are usually colored in sequence alignments to facilitate rapid visual inspection of the sequence similarities. In a DNA alignment, adenine is frequently colored green, cytosine is blue, guanine is yellow, and thymine is red. Amino acids are often colored according to their physico-chemical properties (see Figure 37). Several coloring schemes are possible, each highlighting different aspects of the amino acids' physico-chemical properties.

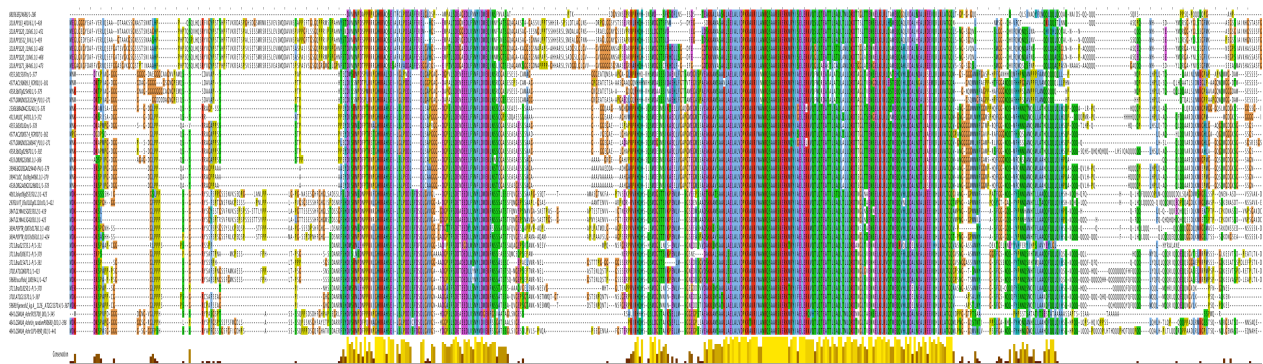


Figure 36. Multiple sequence alignment of 39 proteins in the Bzip transcription factor family. From this alignment, you can very quickly identify the conserved domains that are important for the Bzip function. At the bottom, the conservation is indicated in yellow.

5.2. The amino acids

As we have seen in Figure 38, the genetic code is organized so that mutations in the DNA have a minor effect on the protein sequence. Changes in the wobble base often translate into the same amino acid, and even other nucleotide mutations often translate into amino acids with similar physico-chemical properties. Indeed, some amino acids are more equal than others. Figure 37 shows the physico-chemical characteristics of the twenty most common amino acids, such as their polarity or acidity/basicity. It is also indicated in which secondary protein structure the amino acids are preferentially found. In Chapter 6 we will discuss how to more objectively quantify the similarity between the amino acids.

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

Figure 37. Physico-chemical characteristics of the twenty most common amino acids: size (S/M/L), charge (+/-), hydrophobicity, and secondary structure preference (blue: alpha helix; red: beta strand; green: turns). Figure from http://swift.cmbi.ru.nl/teach/B1B/HTML/PosterA4_nbic_new.pdf.

5.3. Sequence conservation and the genetic code

Evolution depends on variation, which in biological systems is caused by mutations in DNA. While mutations always occur at the level of the genotype, they do not always change the phenotype of the organism. This is good, because biological systems tend to work relatively well and most mutations are detrimental. Still, mutations happen, for example when DNA is inaccurately copied during cell division, or when it is inaccurately repaired after DNA damage. One way biological systems limit the effect of genetic mutations is by including a translation step between the genotype and the phenotype, where the information encoded in genes is translated into proteins that actually do the work. This translation is done by using the genetic code (Figure 38). The genetic code translates $4^3 = 64$ codons into only twenty amino acids, so it is redundant: there are multiple codons that translate into the same amino acid. The number of codons translating into a given amino acid differs per amino acid, and is correlated with the frequency that the amino acids are used in proteins.

Similar codons tend to translate into the same or similar amino acids, which contributes to the conservation of biological function because not all mutations in the DNA directly lead to a change in the protein sequence. Nucleotide substitutions that do not change the protein sequence are called synonymous substitutions, and may be functionally neutral. Nucleotide substitutions that do change the protein sequence are non-synonymous substitutions, but even those may not necessarily affect the organism's fitness. Especially the third nucleotide of a codon, called the “wobble” base, can often be changed without affecting the encoded amino acid (see Figure 38). This means that protein sequences are more conserved than DNA sequences – and that is very important information for

the bioinformatician! First of all, it means that protein sequences allow us to compare much more distantly related organisms than DNA sequences. If we want to detect distant homology, a bioinformatic method that looks at the protein sequence will be more sensitive than a method that only uses the nucleotide sequence (see Chapter 8). Conversely, when we want to differentiate very closely related organisms we should instead look at DNA sequences, because the protein sequences may be completely identical.

1st base	2nd base								3rd base
	U		C		A		G		
U	UUU	(Phe/F) Phenylalanine	UCU	(Ser/S) Serine	UAU	(Tyr/Y) Tyrosine	UGU	(Cys/C) Cysteine	U
	UUC		UCC		UAC		UGC		C
	UUA	(Leu/L) Leucine	UCA		UAA	Stop (Ochre)	UGA	Stop (Opal)	A
	UUG		UCG		UAG	Stop (Amber)	UGG	(Trp/W) Tryptophan	G
C	CUU		CCU	(Pro/P) Proline	CAU	(His/H) Histidine	CGU	(Arg/R) Arginine	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	(Ile/I) Isoleucine	ACU	(Thr/T) Threonine	AAU	(Asn/N) Asparagine	AGU	(Ser/S) Serine	U
	AUC		ACC		AAC		AGC		C
	AUA	(Met/M) Methionine	ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	AUG ^[A]		ACG		AAG		AGG		G
G	GUU	(Val/V) Valine	GCU	(Ala/A) Alanine	GAU	(Asp/D) Aspartic acid	GGU	(Gly/G) Glycine	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GUG		GCG		GAG		GGG		G

Figure 38. The standard genetic code. Yellow: nonpolar amino acids; green: polar; blue: basic; red: acidic; gray: stop codons. Although most organisms translate DNA into protein by the standard genetic code, thirty alternative genetic codes have already been discovered in many branches of the tree of life. See the list at <http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi>.

Image from <https://sbi4u2013.wordpress.com/category/protein-synthesis-genetic-code/page/2/>.

Researchers use the ratio of non-synonymous over synonymous substitutions (known as dN/dS) to investigate whether sequences are more or less conserved than expected. As mentioned in Section 5.1, if a sequence is highly conserved, this probably means that its function is important. If the protein sequence is more conserved than the DNA sequence, the function of the protein is probably where selection happens, i.e. where the function is applied. In such cases, you will find more non-synonymous mutations, and less synonymous mutations in the DNA, so $dN/dS < 1$. This is known as negative selection, and indicates conservation at the protein level. If $dN/dS = 1$, this indicates that selection is neutral: the rate of non-synonymous and synonymous mutations is equal. By looking at the sequence data in this way, researchers have also found situations where $dN/dS > 1$, i.e. the protein sequence is less conserved than expected. This is known as positive selection, and in the exercises in Section 5.8 we will think about this further.

5.4. Consensus sequences and ambiguity characters

Sequence alignments can be visualized as a multiple alignment block, where each sequence is listed on one row and the aligned residues are displayed above and below each other (see Figure 35 and Figure 36). However, these figures can become very large when many sequences are included in the alignment, and it can quickly become unclear what are the most important (conserved) positions

in the alignment. There are several ways to summarize a sequence alignment. The simplest approach is to create a consensus sequence, which at every position contains the most frequent residue in the corresponding alignment column. It is possible to indicate if two or more different nucleotides are (almost) equally abundant, by using ambiguity characters (Figure 39). A special ambiguity character is the N, which basically means we know there is a nucleotide present, but we have no idea which nucleotide it is: it could be an A, C, G, or T. For amino acids, ambiguity characters do not exist, except for the X which indicates an unknown amino acid, i.e. it is equivalent to the N in a nucleotide consensus sequence. Ambiguity characters like N or X mean that a residue is present, we just do not know exactly which one it is. They should not be confused with gaps, which are indicated with a hyphen (-) and are often present in an aligned sequence to indicate the absence of a residue relative to other sequences in the same alignment (see Section 7.2).

A	Meaning: <u>A</u> denine	A	Complement: T/U
C	<u>C</u> ytosine	C	G
G	<u>G</u> uanine	G	C
T/U	<u>T</u> hymine/ <u>U</u> racil	T	A
M	<u>A</u> mino	A or C	K
R	<u>P</u> urine	A or G	Y
W	<u>W</u> eak bond	A or T	W
S	<u>S</u> trong bond	C or G	S
Y	<u>P</u> yrimidine	C or T	R
K	<u>K</u> eto	G or T	M
V	Not T/U	A or C or G	B
H	Not G	A or C or T	D
D	Not C	A or G or T	H
B	Not A	C or G or T	V
N	<u>A</u> ny nucleotide (not a gap)	A or C or G or T	N
-	Gap	Not A, C, G, or T	-

Figure 39. Characters used in nucleotide sequences, including ambiguity characters that may be used in consensus sequences.

5.5. Sequence logos

An alternative way of visualizing multiple sequence alignments is by using sequence logos (Schneider and Stephens 1990). Examples are shown in Figure 40 for an alignment of 20 amino acids, and in Figure 42 for an alignment of 34 nucleotides. Box 1 explains an application of sequence logos. A good website to create them is Weblogo at <http://weblogo.berkeley.edu>.

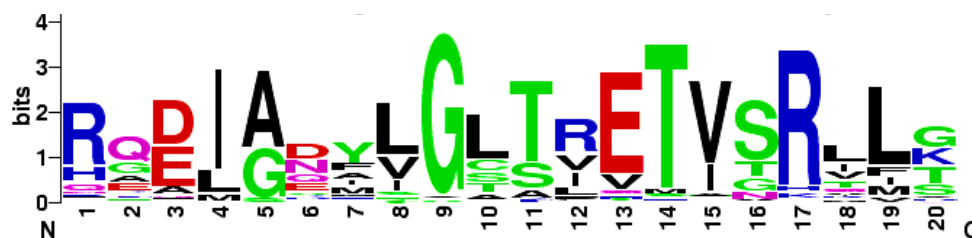


Figure 40. The helix-turn-helix motif from the Catobolite Activator Protein (CAP) family of homodimeric DNA binding proteins. CAP, also known as cAMP Receptor Protein (CRP) is a transcription promoter that binds at more than 100 sites within the *E. coli* genome. Residues 1-7 form the first helix, 8-11 the turn and 12-20 form the DNA

recognition helix. The glycine at position 9 appears to be critical in forming the turn. Positions 4, 8, 10, 15 and 19 are partially or completely buried, and therefore tend to be populated by hydrophobic amino acids, which are colored black. Positions 11-14, 17 and 20 interact directly with bases in the major groove and are critical to the sequence specific binding of the protein. This logo was built based on an alignment of 100 sequences (Pfam accession number PF00325). Figure from <http://weblogo.berkeley.edu/examples.html>.

Box 1: Application of sequence logos: regulatory elements in non-coding DNA

In the 1980s it was suggested that the non-coding DNA that is so abundantly present in many eukaryotic genomes is junk DNA. More recently, detailed bioinformatic analyses showed that non-coding DNA is full of motifs that can even be conserved in different species. Motifs are short pieces of similar sequence that occur repeatedly in different locations on the genome. Eukaryotic genes can be associated with dozens of motifs functioning as transcription factor binding sites (TFBS), regulatory elements, or enhancers. Specific complexes of regulatory proteins called transcription factors can bind to these motifs, causing expression or silencing of the gene. If we could identify all the motifs associated with the genes in a genome, we could build gene expression networks to understand when and where the genes can be expressed.

<i>Haemophilus influenzae</i>	atctc	AAAATATGATCAACTTCACATTTT	ttatt
<i>Pseudomonas aeruginosa</i>	ggttg	AAAAGCCGCGGATCGCCCCCTATAT	ctccc
<i>Shewanella putrefaciens</i>	ctgcc	GAAATAGCGCCAACGGCAGTCTTT	gcgta
<i>Salmonella typhi</i>	---tg	AAACCCTCAAAATCTCCCCCATCT	atact
<i>Escherichia coli</i>	---tg	AAACCCTCAAAATCCCCCCCATCT	ataat
<i>Vibrio cholerae</i>	cccca	AAAGCCTGATGTTGATCAGGCTTT	tttgt
<i>Yersinia pestis</i>	----t	GAAAAGTCTTAATCCTCCCCCAT	tataa

Figure 41. Aligned TFBS of the hslV heat shock protein in seven bacteria. Transcriptional regulation increases expression of heat shock proteins under stresses like high temperature in organisms from bacteria to humans.

Regulatory motifs can be very short and rather noisy, so they are difficult to detect in a genome sequence. The DNA alignment of TFBS of the heat shock protein hslV in Figure 41 shows quite some variation, although some positions are more conserved than others. Which positions are most important becomes immediately clear when we create a sequence logo from this alignment (Figure 42). This sequence logo allows us to perform a more sensitive search for the hslV TFBS in the genome, because we know which positions are most relevant (see Section 8.4).

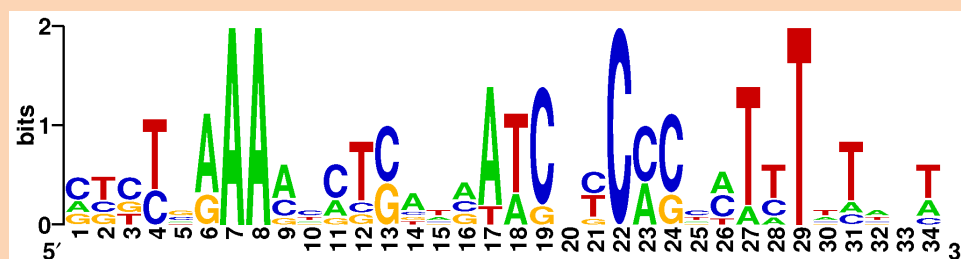


Figure 42. Sequence logo of heat shock protein hslV TFBS created at <http://weblogo.berkeley.edu>.

A sequence logo contains a stack of letters for each aligned position. Within the stack, the letters represent the possible amino acids or nucleotides that are found at that position in the alignment. The height of the letter stack represents the information content of the aligned position. The letters are scaled by their frequency in the alignment, sorted from top (most abundant) to bottom (least abundant). According to Shannon's information theory (Shannon 1948), information is measured in bits, and calculated according to Equation 5 for nucleotides, and Equation 6 for amino acids. Here p_i are the frequencies of the amino acids or nucleotides at that position in the alignment. Because $\log(0)$ is mathematically undefined, the special case for $p_i = 0$ is defined as $\log_2(0) = 0$.

Equation 5. The information content of the nucleotide distribution at a given position in a DNA alignment is equal to the sum of the information content of all four individual nucleotides.

$$I = \sum_{i \in \{A,C,G,T\}} p_i \log_2\left(\frac{p_i}{1/4}\right)$$

Equation 6. The information content of the amino acid distribution at a given position in a protein alignment is equal to the sum of the information content of all twenty individual amino acids.

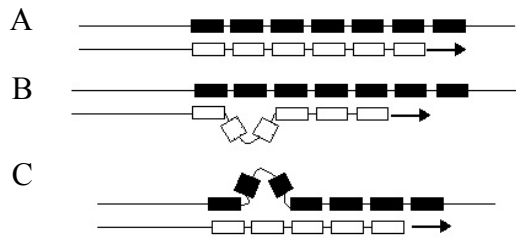
$$I = \sum_{i \in \{A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y\}} p_i \log_2\left(\frac{p_i}{1/20}\right)$$

For example, when a certain column in a DNA alignment contains all four nucleotides at equal frequencies, $p_A = 0.25$, $p_C = 0.25$, $p_G = 0.25$, and $p_T = 0.25$. All $p_i \log_2(1) = 0$, and thus $I = 0$. Conversely, when all the sequences in an alignment contain the same letter at a given site, say adenine (A), then $p_A = 1$, $p_C = 0$, $p_G = 0$, and $p_T = 0$. Thus, the information content for A is $1 \log_2(4) = 2$, while for C, G, and T, $0 \log_2(0) = 0$, and $I = 2$. This is the maximum possible information content for given alignment position. Equation 5 and Equation 6 show that the total information content is calculated as the sum of the information provided by each individual residue (nucleotide or amino acid). If a residue is frequent at the aligned position, its contribution to I will be positive, but if it is rare, it can become negative and decrease I . Thus, the information content is a measure of the degree of conservation of the residues aligned at a certain position.

5.6. Microsatellites

The most rapidly evolving characters on the genome are microsatellites, tracts of repetitive DNA where short motifs (ranging in length from 2-5 bp) are repeated. Microsatellites are also known as short tandem repeats (STRs) because the nucleotide motifs occur in tandem next to one another, or as low-complexity regions. The number of repeated motifs in microsatellites typically ranges from 5-50, and is highly variable in evolution. It is thought that the number of repeated motifs can change because slippage of the DNA polymerase causes problems during DNA replication (Figure 43). Microsatellites are under-represented in protein-coding genes and variation in the number of repeated motifs only rarely influences the phenotype of an organism, so there is less selective pressure on these sequences. Together, this allows them to evolve very rapidly, and makes them ideal genetic characters for analyzing very closely related organisms. For example, microsatellites are used by forensic geneticists as genetic fingerprints to identify individuals by their DNA.

Figure 43. DNA strand slippage during microsatellite replication. Boxes symbolize repeated DNA motifs. Arrows indicate the direction that DNA is replicated (white boxes) from a template strand (black boxes). **A:** Replication of the microsatellite without mutation. **B:** The replicating microsatellite region has gained one unit due to an aberrant loop in the replicated strand that is stabilized by flanking units. **C:** The replicating microsatellite has lost one unit due to a loop in the template strand. Figure from <https://en.wikipedia.org/wiki/Microsatellite>.



5.7. Reading

- Obligatory: Wikipedia page on sequence motifs
http://en.wikipedia.org/wiki/Sequence_motif

5.8. Questions and exercises

1. What is it about short tandem repeat DNA that makes it useful for DNA fingerprinting?
 - a. The number of repeats varies widely from person to person or animal to animal.
 - b. The sequence of DNA that is repeated varies significantly from individual to individual.
 - c. Natural selection responds differently to these variations in different environments.
 - d. Every racial and ethnic group has inherited different short tandem repeats.
2. What would be a consequence of changing one amino acid in a protein consisting of 325 amino acids?
 - a. The primary structure of the protein would be changed.
 - b. The tertiary structure of the protein might be changed.
 - c. The biological activity or function of the protein might be altered.
 - d. Only A and C are correct.
 - e. A, B, and C are correct.
3. Which of the following statements about the genetic code is correct?
 - a. Cysteine and tryptophan have only one codon in the standard genetic code.
 - b. Serine and arginine both have 6 codons in the standard genetic code.
 - c. The Methionine tRNA is a special tRNA that binds to the UGA stop codon.
 - d. None of the above.
 - e. All the above.
4. Which type of mutation should require the minimal change in a nucleotide alignment?
 - a. 3-nucleotide insertion
 - b. 1-nucleotide substitution
 - c. 4-nucleotide insertion
 - d. 1-nucleotide deletion
 - e. 3-nucleotide deletion
5. Which type of mutation should require the minimal change in a protein alignment?

- a. 3-nucleotide insertion
 - b. 4-nucleotide substitution
 - c. 2-nucleotide insertion
 - d. 1-nucleotide deletion
 - e. 3-nucleotide deletion
6. What is the best definition of a motif?
- a. A piece of conserved sequence in an alignment
 - b. A reason for committing a crime
 - c. A pattern that occurs more often than expected
 - d. A binding site for a transcription factor on the genome
7. A researcher wants to determine the genetic relatedness of several breeds of dog (*Canis familiaris*). The researcher should compare homologous sequences of _____ that are known to be _____. Which words can be placed in the gaps? Explain.
- a. carbohydrates; poorly conserved
 - b. fatty acids; highly conserved
 - c. lipids; poorly conserved
 - d. proteins or nucleic acids; poorly conserved
 - e. amino acids; highly conserved
8. Assume that the average rate of change of DNA is approximately 10^{-9} per base pair per year. Consider a genomic region that is approximately 1,000 base pairs long, and two species that diverged from each other 10 million years ago.
- a. Which percentage of sites should approximately differ between these sequences today (keep this calculation as simple as possible)?
 - b. If some sites in this specific region were more difficult to mutate, would the fraction of sites that differ increase or decrease?
 - c. What is the probability that one sequence of 1,000 bp remains unchanged in one year?
 - d. What is the probability that one sequence will stay unchanged in 10^7 years (use your answer to the previous question)?
 - e. What is the probability that both sequences are still the same after 10^7 years?
9. a. Which position(s) is/are most conserved in the alignment below?
 b. Assume the first position is the first position of a codon (i.e. assume that translation starts from the first position). Give a biological reason for some positions being more conserved than others.

```

ATC---TACCGG
ATCTTCTACAGG
ATCAATTACCGC
ATA---TATCGC
ATT---TATCGA

```

10. In bacteria, protein quality control is carried out by a protein network consisting of chaperones and proteases. Central to this network is a protein called ClpB. The alignment below shows predicted transcription factor binding sites for ClpB in *Haemophilus*

influenzae (HI), *Vibrio cholerae* (VC), *Salmonella typhi* (ST), *Thiobacillus ferrooxidans* (TF), *Shewanella putrefaciens* (SP), *Escherichia coli* (EC) and *Yersinia pestis* (YP). The information content is often used to predict and identify transcription binding sites.

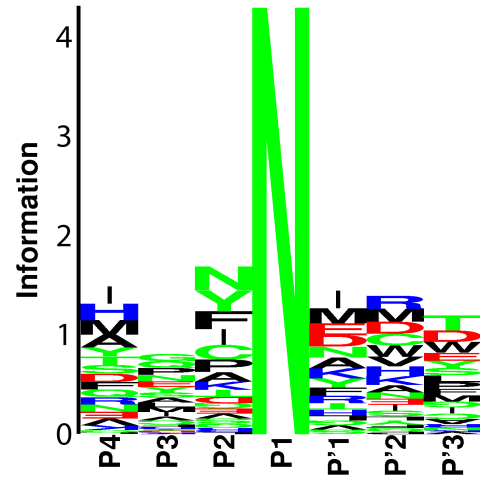
```
YP_1/1-24 TAACCTTGAA
EC_1/1-24 TAACCTTGAA
TF/1-24   TACTCTTGAA
VC/1-24   TTCCCTTGAA
ST/1-24   TCCAGTAGCA
EC_2/1-24 TCCAGTAGCA
YP_2/1-24 TCCAGTAGCA
HI/1-24   GCAAGTAGCA
```

- What is the consensus sequence of the above alignment?
- What is the reverse complement of the consensus sequence?
- Enter the function **infocont** in R and use it to calculate the information content at each position of the alignment (see Equation 5).

```
infocont <- function(A, C, G, T) {
  t <- A + C + G + T
  pA <- A / t
  pC <- C / t
  pG <- G / t
  pT <- T / t
  ic <- 0
  if (pA) {
    ic <- ic + pA * log ((pA / 0.25), 2) }
  if (pC) {
    ic <- ic + pC * log ((pC / 0.25), 2) }
  if (pG) {
    ic <- ic + pG * log ((pG / 0.25), 2) }
  if (pT) {
    ic <- ic + pT * log ((pT / 0.25), 2) }
  return (ic) }
```

- Create a sequence logo of this alignment using Weblogo. Validate the values for the information content you calculated above. (Hint: turn "Small Sample Correction" off under "Advanced Logo Options".)

11. Asparaginyl Endopeptidase (AEP) is an enzyme that cleaves proteins (cell's own proteins, or any bacterial or viral) into smaller fragments. The fragments originating from pathogens are used by the adaptive immune system to detect an infection. The logo above (right) gives the information content of the known cleavage sites by AEP. This logo is based on the experimental data available from AEP cleavages in 18 proteins. The cleavage occurs between P1 and P1' positions. P1' – P3' are the flanking regions on the right (C-terminal) of the cleavage site, while P2-P4 are flanking regions on the left (N-terminal).



- Which amino acid is preferred by AEP at the cleavage site? Are there any exceptions to this?
 - We know that asparagine occurs in proteins with 6% frequency. However, an average protein of 200 amino acids would have one or two cleavage sites by AEP. How can this be?
12. Assume all nucleotides occur independently of each other with the same frequency in DNA sequences.
- What is the probability of finding the nucleotide sequence 5'-GGATATCCGC-3' by chance in a random DNA molecule?
 - How often do you expect to find this same sequence in a given 10 kb DNA molecule?
 - Imagine we have a genome with a total size of 4×10^9 base pairs. What is the probability of finding a specific 20-nucleotide sequence in this genome? What about a specific 3-nucleotide sequence, or a specific 16-nucleotide sequence?

6. Quantifying sequence similarity

Gene and protein sequences contain an enormous amount of information about the function of the proteins, the evolution of the genomes where they are encoded, and much more that we can use for studying biology. Two sequences that are very similar have probably diverged very recently, and probably perform the same function. Conversely, two sequences that diverged long ago are probably very different, and their function is less likely to be retained. In this chapter, we will discuss how we can objectively quantify sequence similarity.

Arabidopsis thaliana.....RVL RMVN TLG
Aquilegia coerulea.....KVL RMVN TLA

Figure 44. Two aligned protein sequences taken from Figure 35.

6.1. Sequence identity and the identity matrix

If we wanted to calculate a similarity score to quantify the similarity between two aligned sequences, we could count one point for all matching residues and zero points for residues that are different, resulting in an alignment score of 8 for the example in Figure 44. This scoring scheme makes no distinction between the types of amino acids that are aligned, and only looks at which positions between two sequences are identical. The alignment score resulting from this analysis is called sequence identity: 8/10 aligned residues are identical, so the sequence identity is 80%. We can place these scores into a matrix called a substitution matrix, a square matrix that shows for every possible substitution between two residues how many points it would contribute to an alignment score. The identity matrix is the simplest example of a substitution matrix (Figure 45).

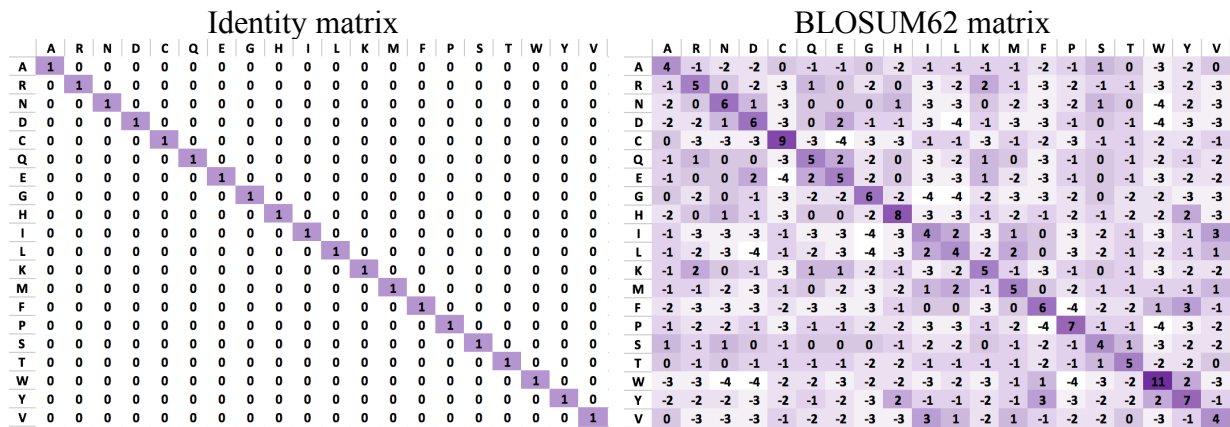


Figure 45. Substitution matrices: the identity matrix (left) and the BLOSUM62 matrix (right).

6.2. Blocks substitution matrix (BLOSUM)

Not all amino acid substitutions are equally likely to occur in evolution. Biases occur at the level of mutations and at the level of selection. First, some amino acids are more likely to mutate into one another because their codons are more similar in the genetic code (see Section 5.3). Second, some substitutions are more likely to get fixed in evolution because the amino acids have similar physico-chemical properties (see Section 5.2). We would like to see these different probabilities reflected in the substitution matrix, so that we can calculate a more meaningful alignment score.

The blocks substitution matrix (BLOSUM) (Henikoff and Henikoff 1992) attempts to capture the true substitution probabilities between amino acids in a very elegant way, by measuring how often amino acids really replaced each other in evolution. Henikoff and Henikoff did this by using a dataset of well-aligned homologous protein sequences where they counted the frequency that each pair of amino acids was observed in the same alignment column. As explained in Section 5.1, amino acids in the same alignment column are homologous residues that replaced each other in evolution. These observed substitution frequencies for each pair of amino acids (q_{ij}) were corrected for the expected substitution frequencies (e_{ij}), which depend on how often the amino acids occur in the data. From these two numbers, they directly calculated the logarithmic BLOSUM substitution scores S_{ij} (see Box 2).

One commonly used BLOSUM matrix, BLOSUM62, is shown in Figure 45. Positive values represent amino acids that were aligned more often than expected, while negative values represent amino acids that were under-represented in aligned positions, i.e. their substitution was avoided in evolution. If two amino acids are aligned just as often as expected, then $q_{ij} = e_{ij}$ and the substitution score is zero. Thus, if $S_{ij} = 0$ then the amino acids i and j occur in the same column just as often as you would expect if the sequences were completely shuffled. Note that self-substitution scores S_{ii} are always positive, reflecting the fact that evolution is generally conservative, not changing the amino acid.

As explained in Box 2, log-odds scores allow us to easily combine probabilities by summing the values. Thus, we can quantify the similarity between two sequences by calculating an alignment score, summing the substitution scores of all aligned amino acids along the length of the whole alignment. If the alignment score is high, it means that many of the aligned amino acids have positive substitution scores, i.e. they tend to preferentially replace each other in evolution. If the alignment score is low, it means that many of the aligned amino acids have negative scores, i.e. they do not often replace each other in evolution. From the alignment score we can calculate q/e for the entire sequence (see Box 2). This number quantifies how much more likely it is that two sequences are well-aligned homologs versus random sequences.

Because Henikoff and Henikoff based the BLOSUM substitution scores on observed/expected counts in real data, you can imagine that they depend on how proteins are considered in the dataset. A key aspect in generating BLOSUM matrices is weighting the sequences to reduce sampling bias in the data. This is necessary because the sequences in biological databases are highly biased toward certain species (see Section 1.4), which means that there are many very similar sequences present. The weighting involves clustering the most similar sequences together, and considering each cluster as a single sequence in the blocks of well-aligned homologs. Sequences are clustered together if they have $\geq \mathcal{G}\%$ identity, and the substitution statistics are calculated only between clusters. Different BLOSUM matrices are produced with different thresholds \mathcal{G} . The BLOSUM62 matrix uses $\mathcal{G} = 62\%$, and is one of the most widely used amino acid substitution matrices. BLOSUM matrices produced with high values of \mathcal{G} mean that only highly similar sequences are clustered, and the blocks of well-aligned homologs thus contain relatively fewer mutations. If \mathcal{G} is low, less similar sequences are merged, and the well-aligned blocks contain relatively more substitutions. So if you compare two distantly related sequences with the BLOSUM45 matrix you will be less likely to say they resemble random unrelated sequences than if you compare them with the BLOSUM80 matrix.

Box 2: From frequencies in well-aligned homologs to log-odds substitution scores and back

The BLOSUM substitution matrices contain statistical substitution scores called log-odds scores that are calculated from reliable, well-aligned protein sequence alignments. First, we count for every pair of amino acids how often they are aligned (observed), q_{ij} for two amino acids i and j . Second, we calculate how often we would expect these amino acids to be aligned, if everything would be completely random: a pair of amino acids (with frequencies p_i and p_j) should occur with the expected frequency $e_{ij} = p_i^2$ if $i = j$, and with frequency $e_{ij} = p_i p_j$ if $i \neq j$. This correction accounts for the fact that some amino acids are more common than others. Third, we divide the observed by the expected frequency, which results in the statistical odds of observing that mutation: q_{ij}/e_{ij} is a value greater than 1 if the amino acids are aligned more frequently than expected and a value between 0 and 1 if the amino acids are aligned less frequently than expected. Finally, we take the log of this ratio to scale these values symmetrically around 0, instead of asymmetrically around 1 (Equation 7). Values in BLOSUM matrices are calculated as $S_{ij} = 2 \log_2 (q_{ij}/e_{ij})$ and are rounded off. If the observed number of substitutions between two amino acids is as expected, then $S_{ij} = 0$. If the observed frequency is lower than expected, then $S_{ij} < 0$. Conversely, if the observed frequency is greater than expected, then $S_{ij} > 0$.

Equation 7. Log-odds scores are commonly used in bioinformatics, for example as the weights in BLOSUM matrices. Base-10 logarithms (\log_{10}), base-2 logarithms (\log_2), and natural logarithms (\ln , base- $e = 2.71828$) are frequently used. Note that the base of the logarithm does not matter, provided that you use it consistently.

$$\text{log-odds score} = \log (\text{observed/expected})$$

So what do the log-odds scores mean? Consider a sequence alignment where the amino acids threonine (T) and phenylalanine (F) are aligned. If we look in Figure 45, we see that the substitution score is -2. Knowing that $S_{ij} = 2 \log_2 (q_{ij}/e_{ij})$ and that $S_{TF} = -2$, we can calculate q_{TF}/e_{TF} as $2^{(-2/2)} = 0.5$. This means that T and F are observed in the same alignment columns (q_{TF}) half as often as expected based on their frequency (e_{TF}). Because these numbers are based on blocks of well-aligned homologs, we could say that based only on the amino acids T and F, two aligned sequences are two times more likely to be random sequences than well-aligned homologs.

T	P	I
F	P	L

Log-odds scores are a commonly used mathematical trick in bioinformatics to work with probabilities. One of the advantages of using log-odds scores is that probabilities can easily be combined by summing the values, whereas normal probabilities must be multiplied. If we consider that the amino acids T and F were part of two longer aligned sequences TPI and FPL, then we could use Figure 45 to calculate the alignment score as $-2 + 7 + 2 = 7$. From there, we can calculate the observed/expected ratio (q/e) of this short alignment as $2^{(7/2)} = 11.3$. This means that TPI and FPL are about eleven times more likely to occur in well-aligned homologs than in random sequences. We will further discuss sequence alignment in Chapter 7.

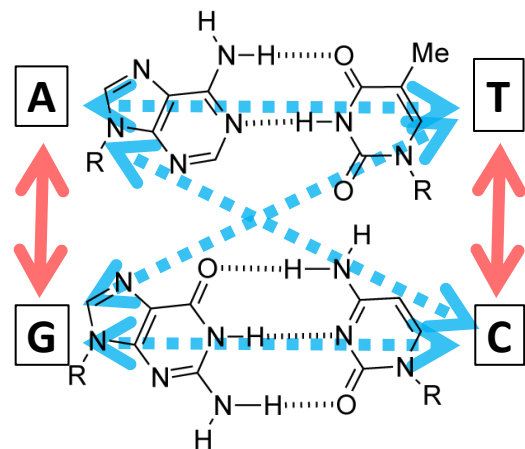
6.3. Point accepted mutations (PAM) matrices

Although BLOSUM matrices are the most frequently used substitution matrices, there are other matrices that you will sometimes find used in bioinformatics. Margaret Dayhoff calculated the point accepted mutations (PAM) matrices using a database of amino acid changes in groups of closely related proteins (protein families) (Dayhoff, Schwartz, and Orcutt 1978). For this matrix, the mutations that occurred during the evolution of these protein families were counted and their frequency scored in a matrix, much as the frequencies in the BLOSUM matrix. A matrix with an evolutionary distance of 1 PAM would correspond to roughly 1% divergence in a protein (one amino acid replacement per hundred amino acids). The PAM1 matrix would thus have numbers very close to 1 in the main diagonal, and small off-diagonal numbers. A matrix with an evolutionary distance of 0 PAMs would have only ones on the main diagonal and zeroes elsewhere. Assuming that proteins diverge as a result of accumulated and uncorrelated mutations, a mutational probability matrix for a protein sequence that has undergone N percent accepted mutations, i.e. a PAM- N matrix, can be derived by multiplying the PAM1 matrix by itself N times. The result is a whole family of scoring matrices. A PAM250 matrix, which corresponds to an evolutionary distance of 250 substitutions per hundred residues (each residue can change more than once), works well for general sequence similarity calculations. Remember that not all the residues of a sequence have to change at this evolutionary distance: if some positions get substitutions very frequently, they will change several times, while other positions, e.g. the ones that are crucial for the function, will not change.

6.4. Transition and transversion mutations in DNA

DNA sequences are more difficult to align since at each position, we can have one of only four different bases as opposed to one of twenty in peptide alignments, so the probability of randomly similar region and spurious alignments becomes higher. The most commonly used substitution matrix for nucleotides is the identity matrix, i.e. a scoring scheme that counts one point for a match, and zero points for a mismatch. However, some nucleotide substitution matrices exist that contain a higher substitution score for transitions than for transversions, because transitions (purine \rightarrow purine or pyrimidine \rightarrow pyrimidine) happen more frequently than transversion (purine \rightarrow pyrimidine or pyrimidine \rightarrow purine).

Figure 46. Transitions are $A \leftrightarrow G$ and $C \leftrightarrow T$ mutations (red, drawn). Transversions are $A \leftrightarrow C$, $A \leftrightarrow T$, $C \leftrightarrow G$, and $G \leftrightarrow T$ mutations (blue, dashed). Transitions are about twice as frequent in evolution as transversions.



6.5. A substitution matrix is a model of evolution

A substitution matrix is a model of evolution that describes what evolutionary events we think may occur, and how likely they are. (Note that this is a conceptual model, see Figure 47.) In this model, the scores in a substitution matrix quantify the likelihood that the residues in question replaced each other in evolution. Things like sequence alignments and phylogenetic trees are our interpretations of observed sequence data, i.e. they are hypotheses about what really happened in evolutionary history. The hypothesis in a sequence alignment is that the aligned residues are homologous, i.e. they are derived from the same residue in the ancestral sequence (see Section 5.1). The hypothesis in a phylogenetic tree is that the organisms descend from a common ancestor, and that the bifurcations and branch lengths in the tree reflect their ancestral relationships (see Section 4.1). A quantitative model of evolution allows us to score to what extent these specific hypotheses are consistent with that model, and by extension to discover which of several possible hypotheses is most likely to reflect the true evolutionary history (given that model of evolution).

If we compare the identity matrix and the BLOSUM62 matrix (Figure 45), the latter is a better model of evolution because it more accurately reflects the real substitution probabilities that happen in evolution. However, even the BLOSUM matrix is still a relatively simple model of evolution because it only takes substitution mutations into account, and not any of the other mutations that can happen, such as insertions, deletions, gene duplications, tandem duplications (Figure 43), rearrangements, inversions, etcetera.

Conceptual model

From Wikipedia, the free encyclopedia

A **conceptual model** is a representation of a system, made of the composition of **concepts** which are used to help people **know**, **understand**, or **simulate** a subject the model represents. It is also a set of concepts. Some **models** are **physical objects**; for example, a toy model which may be assembled, and may be made to work like the object it represents.

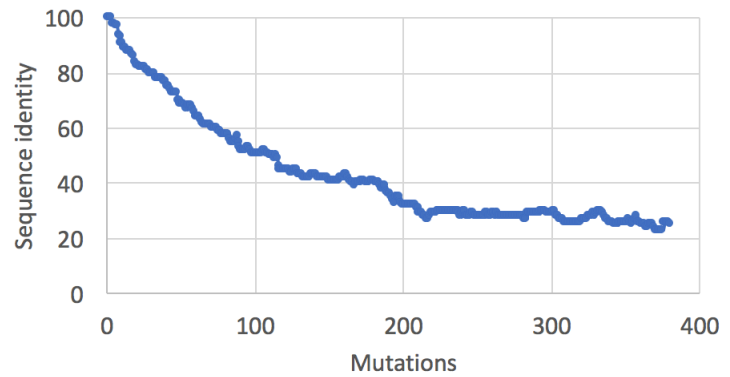


Figure 47. The model of evolution discussed in Section 6.5 is a conceptual model, unlike the “Evolution of Man” model set available for £12.79 at <http://www.everythingdinosaur.com/evolution-of-man-model-set.html>.

6.6. Mutational saturation and evolutionary distance

Consider a recently duplicated gene in a gene family where the molecular clock holds (see Section 4.4). Naturally, the similarity between the two gene sequences will decrease in time – we have already discussed this in Chapter 3 and this is why we can use sequences to estimate evolutionary divergence. What we have not discussed is *how* the sequences diverge. To make it easy, let’s think about a 100 nucleotide piece of DNA that does not code for a protein and has no other selective pressures on it, so that the mutations are truly random. We could make a model of this situation in Excel or Python to investigate what happens (Figure 48).

Figure 48. Sequence identity between two 100 nucleotide DNA sequences that diverge by random mutations.



After every mutation, we calculate the alignment score between the two sequences by using the identity matrix. At first, the identity decreases linearly, because every mutation changes the sequence with exactly 1%. However, after a while we see that the sequence identity reaches a plateau at around 25%, because mutations happen at positions in the sequence that were already mutated before. What happens is called mutational saturation, and this has been mathematically described by Jukes and Cantor (Jukes 1969).

Equation 8. The Jukes-Cantor correction formula to compute the true evolutionary distance between two nucleotide sequences d based on their divergence D .

$$d = -\frac{3}{4} \ln \left(1 - \frac{4}{3} D \right)$$

In the Jukes-Cantor correction formula for nucleotide sequence evolution (Equation 8), d is the actual evolutionary distance between two sequences, and D is the sequence divergence (i.e. one minus the sequence identity). As D approaches $3/4$ (25% identity, see Figure 48) and the two sequences are almost random, the evolutionary distance between the sequences approaches infinity. If D is small, then $\ln(1 - 4/3 D) \approx -4/3 D$ and thus $d \approx D$, whereas if $D > 1/2$, i.e. if half of the positions are non-matching, d is much greater than D . Thus, mutational saturation makes it difficult to estimate the real time of divergence between two sequences, especially if the sequences diverged long ago.

While we can mathematically approximate the true evolutionary distance between nucleotide sequences from their sequence identity by using the Jukes-Cantor formula, the situation is more complicated when we want to do the same for protein sequences. We compare protein sequences at the amino acid level, but the mutations occur at the nucleotide level, and the genetic code forms a complex translation table that has to be taken into account. Several correction formulas have been developed to address this, the most frequently used is probably Kimura correction (Equation 9). When you use bioinformatic programs to compare sequences and infer their true evolutionary distance, you will often be able to select the correction formula of your choice.

Equation 9. The Kimura correction formula to compute the true evolutionary distance between two protein sequences d based on their divergence D .

$$d = -\ln(1 - D - 0.2D^2)$$

6.7. Reading

- Optional: Where did the BLOSUM62 alignment score matrix come from? (Eddy 2004b)

6.8. Questions and exercises

1. Read how the BLOSUM matrix was made in Section 6.2.
 - a. Explain why the BLOSUM matrix is a model of evolution.
 - b. Which amino acids are most conserved in evolution?
 - c. Which amino acids are least conserved in evolution?
 - d. Does the BLOSUM62 matrix agree with the PAM250 matrix about which amino acids are most likely to remain unchanged in evolution? (Hint: find the PAM250 substitution matrix online.)
2.

<i>Sequence1</i>	VWEDNWDDD
<i>Sequence2</i>	VSEDNRDDD
<i>Sequence3</i>	VWDDNWDED

 - a. Calculate pairwise alignment scores between all the sequences using the identity matrix. Is Sequence1 more similar to Sequence2 or to Sequence3? What would happen if you would use the identity matrix with match/mismatch scores of 2/0 instead of 1/0?
 - b. Calculate pairwise alignment scores between all the sequences using the BLOSUM62 matrix. Is Sequence1 more similar to Sequence2 or to Sequence3?
 - c. Which of these substitution matrices gives you the most biologically meaningful similarity score? Explain why.
3. Observe the following sequence alignment. Let's assume a model of evolution represented by the BLOSUM62 matrix. How likely is it that these two sequences are well-aligned homologs? (Hint: use the odds ratio explained in Box 2.)

SKRNITK
NAPNVSA

4.
 - a. Using your graphical calculator, make a graph of the Jukes-Cantor model (i.e. d , distance between two sequences, as a function of D , the fraction of sites that differ between two sequences).
 - b. Make a graph of the Jukes-Cantor model in R. (Hint: use the **seq** command. See Sections 2.3.3 and 2.5.3.)
 - c. Explain how the Jukes-Cantor distance d depends on the observed distance D .
 - d. If two sequences evolve by random drift and they are observed to differ 20% of sites, what is the Jukes-Cantor distance between these sequences?
 - e. Two other sequences differ at 80% of their sites, i.e. they are 20% identical. Explain this observation.
 - f. Explain in your own words why d goes to infinity as D approaches 0.75.

5. Write a script in R that calculates the alignment score between *Sequence1* and *Sequence2* from Exercise 2 using the identity matrix. (Hint: try doing this with a function that takes two vectors as input: `seq1 <- c('V', 'W', 'E', 'D', 'N', 'W', 'D', 'D', 'D')` and `seq2 <- c('V', 'S', 'E', 'D', 'N', 'R', 'D', 'D', 'D')`).

7. Algorithms for sequence alignment

Sequence alignment is defined as the bioinformatic operation of placing the residues in two or more sequences side by side and inserting gaps into the sequences in such a way that their alignment score is maximized. Sequence alignment is probably the single most important operation in bioinformatics. First, it allows us to quantify sequence similarity (see Chapter 6), which is needed to calculate the odds that sequences are homologous, i.e. they descend from a common ancestor (see Section 4.5 and Box 2 in Section 6.2). If we find that sequences are indeed homologous, then alignments are needed to infer their evolutionary relationships and reconstruct phylogenetic trees (see Chapter 3 and Chapter 9). As we have discussed in Chapter 5, sequence alignments are also important for analyzing conserved and variable regions in related proteins, that may provide hints about the importance of these regions for function. These, and many other questions depend on good sequence alignments that can be generated in a reproducible way.

In this chapter, we will discuss how computers create sequence alignments, by looking at some classic sequence alignment algorithms. While short and closely related sequences might be aligned manually (e.g. Figure 49A), it becomes difficult when the sequence similarity is low or when the sequences are very long. Two sequences could be aligned in many ways. In general, alignment algorithms attempt to find the best alignment by searching for the one that is most consistent with a given model of evolution (see Section 6.5 and Section 7.3). They do this by calculating an alignment score that quantifies the agreement of each alignment with the model of evolution, and identifying the alignment with the highest score.

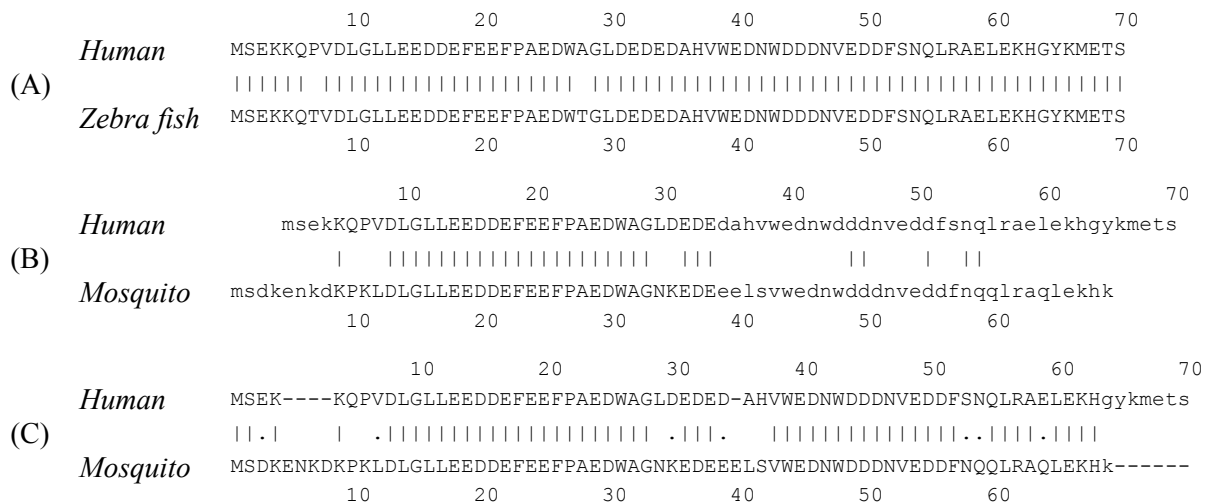


Figure 49. Alignment of the human proteasomal subunit DSS1 with its homolog in zebrafish (A) or mosquito (B) without gaps. A pipe "|" between the sequences indicates a matching residue. (C) shows the optimal alignment with the mosquito homolog with gaps. A dot "." between the sequences indicates a mismatch, but among similar amino acids (positive BLOSUM score). Upper case letters show correctly aligned regions (see Section 7.2).

7.1. Visualizing genome evolution using dot-plots

The dot-plot is one of the simplest methods to compare sequences graphically (Figure 50). The dot-plot is a simple table or matrix, where the rows correspond to the residues of the first sequence and the columns to the residues of the second sequence. The positions are left blank if the two residues are different, and filled with a dot if they match. A dot-plot gives a quick view of a relation between

two sequences. Stretches of similarity are rather easy to catch because they show up as diagonals in the plot. Horizontal or vertical shifts indicate insertions or deletions in the sequence. Look at the word “CROWFOOT” in Figure 50A. If we just compare the two sequences in the dot-plot, it is impossible to say whether these eight letters were inserted in the horizontal sequence, or deleted from the vertical sequence. Thus, we use the word indel to indicate such a region in a sequence alignment, without having to specify if the evolutionary event was an insertion or a deletion. We can further filter out the noise in a dot-plot by only showing dots that are within a region where both sequences are identical along a longer stretch, i.e. if they are part of a longer diagonal (see Figure 50B). Dot-plots can also be used to find repeats within one sequence: by using the same sequence for the rows and the columns, we can observe intra-sequence repeats as multiple diagonals in the dot-plot.

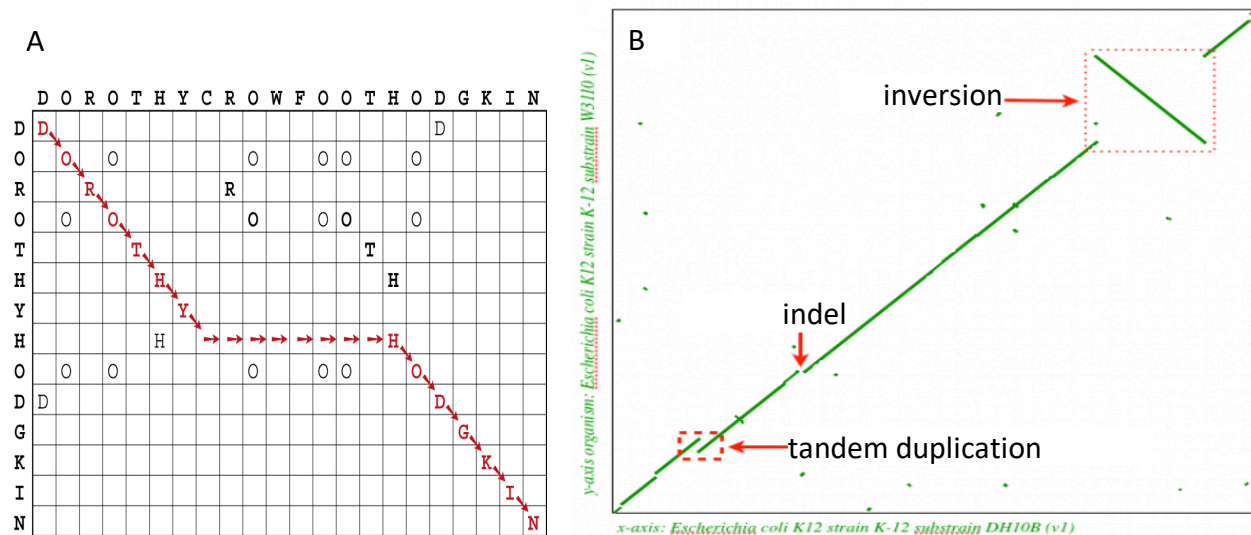


Figure 50. (A) Dot-plot showing similarities between a short name (Dorothy Hodgkin) and a full name (Dorothy Crowfoot Hodgkin). The path shown with arrows is the best alignment. Figure from (Lesk 2002). **(B)** Dot-plot alignment between the genomes of two *Escherichia coli* strains showing various signatures of genomic evolution. Figure from https://genomeevolution.org/wiki/index.php/2011_BSA_Workshop.

7.2. Gaps

We have discussed in Chapter 6 how substitution mutations between sequences can be scored to obtain a meaningful alignment score, but we did not yet take gaps into account. Gaps correspond to indel mutations, i.e. an insertion or deletion in one of the compared sequences. The human and mosquito sequences in Figure 49 contain two highly similar parts, but without inserting a gap into the human sequence between the 35th and the 36th amino acids, these parts cannot be well aligned. In Figure 49B, only the first half of the sequence is aligned correctly (shown with upper case letters), while in Figure 49C, inserting a gap allowed both parts to be aligned. This indel mutation represents an evolutionary event, and as such it needs to be included into our model of evolution. Only then can we calculate a meaningful alignment score between the two sequences.

Sequence alignment algorithms use negative scores to penalize gaps in an alignment. These penalties are subtracted when calculating the alignment score between two aligned sequences. The negative BLOSUM substitution scores of some physico-chemically distinct amino acids are

calculated from blocks of well-aligned homologs (see Section 6.2), but the value of the gap penalty is usually determined empirically because it is very difficult to obtain high quality alignments between homologs with a lot of gaps. (“Empirically” means that the user has to play with the gap penalty value and see when the alignment looks good.)

There are three main ways of implementing gap penalties when calculating alignment scores.

1. A fixed gap penalty subtracts a fixed penalty from the alignment score for every gap, regardless of its size. This gap penalty is consistent with a model of evolution where all gaps are equally likely to occur, regardless of their size. Thus, each evolutionary event (indel mutation) is weighted equally.
2. A linear gap penalty subtracts a fixed penalty for each gap position, so three single gaps at different positions in the alignment will have the same total penalty as having a single stretch of three gaps. This approach reflects a model of evolution where bigger gaps are less likely to be fixed, as those may have a greater effect on the encoded function.
3. An affine gap penalty compromises between the two options above by splitting the gap penalty into two parts: (i) the gap opening or gap existence penalty, and (ii) the gap extension penalty. The first is a penalty for having a gap at all, the second is a smaller penalty for extending already opened gaps. The affine gap penalty is default option in most sequence alignment and homology search algorithms.

Sometimes you might choose to give no penalty for gaps at the end of an aligned gene or protein. The rationale for this would be that those generally have a much smaller effect on the function of a protein than gaps that lie in the middle of the sequence. For a more detailed discussion on how to set gap penalties, see (Vingron and Waterman 1994).

7.3. Needleman-Wunsch global alignment algorithm

Alignment algorithms try to maximize the alignment score by shifting the sequences relative to one another and introducing gaps. Each possibility is scored using a model of evolution. Here, the model of evolution consists of a substitution matrix and gap penalties. The alignment with the highest score is considered the optimal alignment, i.e. the one most consistent with the given model of evolution. You can imagine that introducing gaps into the sequences greatly increases the number of different comparisons between two sequences and it is challenging to calculate alignment scores for all of them.

Dynamic programming is an approach to relatively quickly identify the optimal alignment between two sequences. One of the earliest sequence alignment algorithms was a dynamic programming algorithm developed by Needleman and Wunsch (Needleman and Wunsch 1970). Their algorithm identifies the optimal global alignment between two sequences, i.e. an alignment covering all residues in both sequences from start to end. The general idea of the algorithm is the following. Assume we have two sequences with length N_1 and N_2 that we want to align. For any two positions i and j in either sequence (i.e. $0 < i \leq N_1$ and $0 < j \leq N_2$) we calculate the score for the partial alignment from the start of the sequences until position (i, j) . At position (i, j) there are only three things that can happen: (i) i and j are aligned with each other; (ii) i is aligned with a gap in sequence 2; or (iii) j is aligned with a gap in sequence 1. The highest possible alignment score at that position $H(i, j)$ can be calculated with Equation 10, where S_{ij} is the substitution score between

the amino acids at positions i and j , g is the gap penalty, and the highest possible alignment score at the start of both sequences $H(0,0) = 0$.

Note that gap penalties are always subtracted from the alignment score, whether they are written as a positive or as a negative number.

Equation 10. Calculating the highest alignment score for a given field in the Needleman-Wunsch algorithm.

$$H(i,j) = \max \begin{cases} H(i-1,j-1) + S_{ij} & \text{(diagonal)} \\ H(i,j-1) - g & \text{(horizontal)} \\ H(i-1,j) - g & \text{(vertical)} \end{cases}$$

Let us explain how the Needleman-Wunsch algorithm works with an example, where we align two very short protein sequences **SPEARE** and **SHAKE** using the PAM250 matrix (see Figure 51). We set the gap penalty $g = -6$. The steps we need to follow are as follows (see Figure 52).

1. Draw an alignment matrix H by putting the residues from one sequence in the row heads (it does not matter which) and the residues from the other in the column heads.
2. Set $H(0,0) = 0$ and give all the fields at the edge of the alignment matrix H scores that are multiples of -6 , which is the gap penalty g .
3. Fill in S_{ij} values from the substitution matrix into the matrix (the red numbers in Figure 52). In this example, we used PAM250 substitution scores (Figure 51).

Figure 51. Excerpt from the PAM250 substitution matrix including only the substitution scores that are needed to align the sequences **SHAKE and **SPEARE**.**

	A	E	P	R	S
A	2	0	1	-2	1
E	0	4	-1	-1	0
H	-1	1	0	2	-1
K	-1	0	-1	3	0
S	1	0	1	0	2

4. Fill in the scores in the H matrix using Equation 10 starting from $H(1,1)$. To calculate the score in $H(1,1)$, select the maximum value from three options:
 - i. If we take a diagonal step coming from $H(0,0)$ where the value is 0, the two serine residues (S) are aligned. The PAM250 substitution score $S_{SS} = 2$, so the score would become $0 + S_{SS} = 2$.
 - ii. If we take a horizontal step, we would come from $H(1,0)$ where the value is -6 , and a gap would be added at the start of sequence **SHAKE**. The score would become $-6 + -6 = -12$.
 - iii. If we take a vertical step, we would come from $H(0,1)$ where the value is -6 , and a gap would be added at the start of sequence **SPEARE**. The score would become $-6 + -6 = -12$.

Choose the maximum of these three values (diagonal, 2) and draw a diagonal arrow from the field $H(1,1)$ to indicate the move that led to this score. Then we continue with field $H(1,2)$, where the options are:

- i. Diagonal: $-6 + -1 = -7$
- ii. Horizontal: $2 + -6 = -4$
- iii. Vertical: $-12 + -6 = -18$

We choose for the second possibility, write the score $H(1,2) = -4$ and draw a horizontal arrow to indicate a gap. Using Equation 10 we fill up the whole alignment matrix H (Figure 52). In each field (i,j) the score $H(i,j)$ is the maximal alignment score we could get when

aligning the sequences from (0,0) until position (i, j) . Note that for some fields the diagonal, horizontal, or vertical steps lead to the same highest score. In such cases, there is more than one possibility to obtain the optimal alignment score, and two, or even three arrows should be drawn. Importantly, this could yield multiple equivalent optimal alignments in step 6.

- When the matrix is full with scores and arrows, the final optimal global alignment score is the score in the field at the lower right corner of the matrix H . After all, in every field we entered the maximum possible score $H(i, j)$ that you could get coming from $H(0,0)$ until position (i, j) . In Figure 52 the optimal global alignment score is 6.

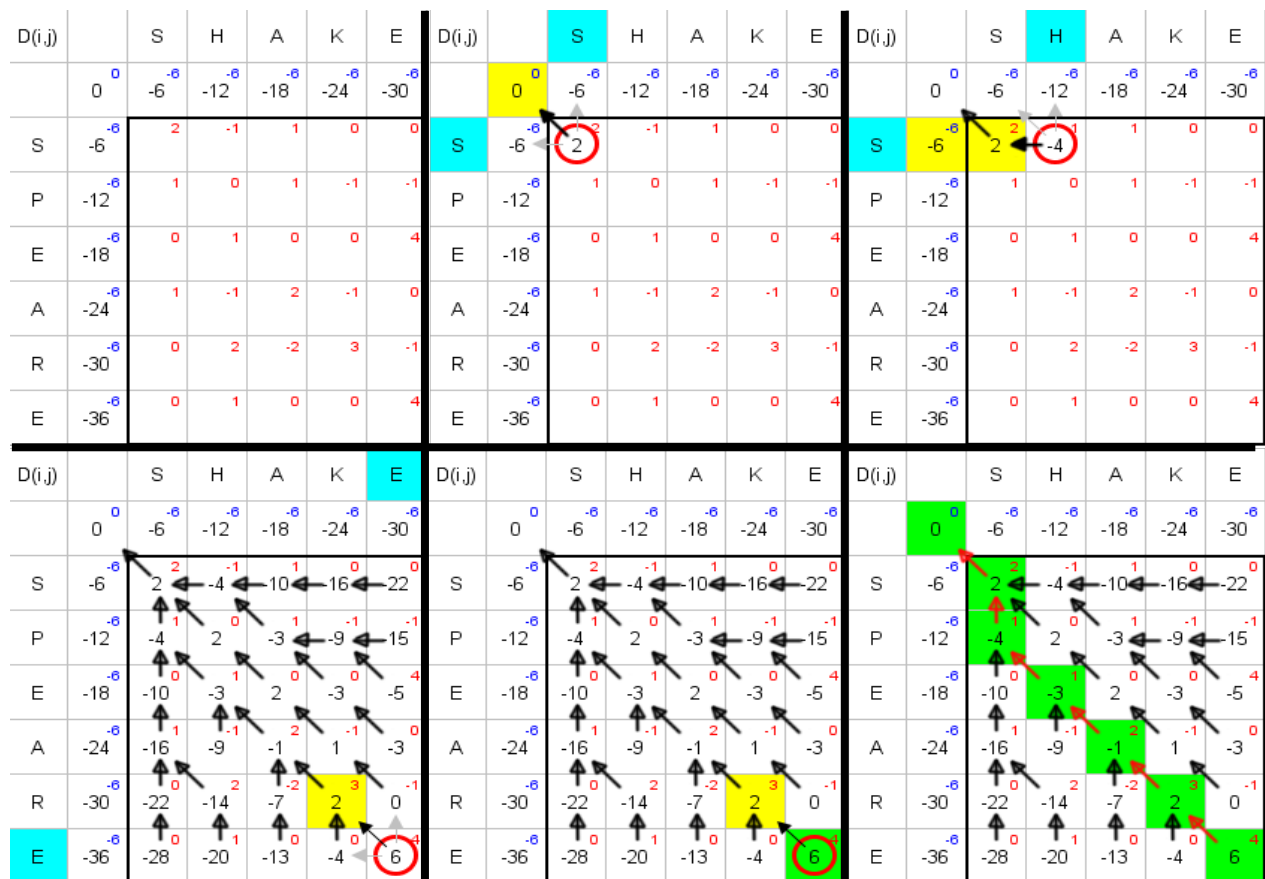


Figure 52. Global alignment by dynamic programming with the Needleman-Wunsch algorithm in six steps. Top row: initializing the H matrix and filling it with scores (Equation 10). Bottom row: reconstructing the global alignment by backtracking from the bottom right hand field. Figures created using <http://baba.sourceforge.net>.

- Finally, we reconstruct the sequence alignment by using the arrows in the matrix. We start in the lower right corner field, and by following the directions of the arrows, we reconstruct the alignment from back to front in a process called backtracking. A diagonal arrow means a match between the residues. For example, the arrow leaving the lower right hand corner is a diagonal, so the last two positions in the alignment become E aligned to E. An arrow pointing up or to the right means introducing a gap in the horizontal or vertical sequence, respectively. For example, the arrow pointing up from field (2, 1) introduces a gap between the serine (S) and histidine (H) residues in the horizontal sequence SHAKE that is aligned to the proline (P) in the vertical sequence. The resulting optimal alignment thus becomes:

7.4. Smith-Waterman local alignment algorithm

As we have seen in Section 5.1, proteins often contain different domains that each perform a certain aspect of the protein function. When we want to identify sequences that are functionally related to a query, it is often valuable to consider local alignments that identify similar sub-regions of the sequences. Smith and Waterman developed a dynamic programming algorithm to make local alignments that resembles the Needleman-Wunsch algorithm (Section 7.3), except that an arrow is only drawn if the alignment score is positive (Smith and Waterman 1981). If the maximum possible score in a given field would be negative, it is set to 0 and no arrow is drawn from that field (see Equation 11 and Figure 53).

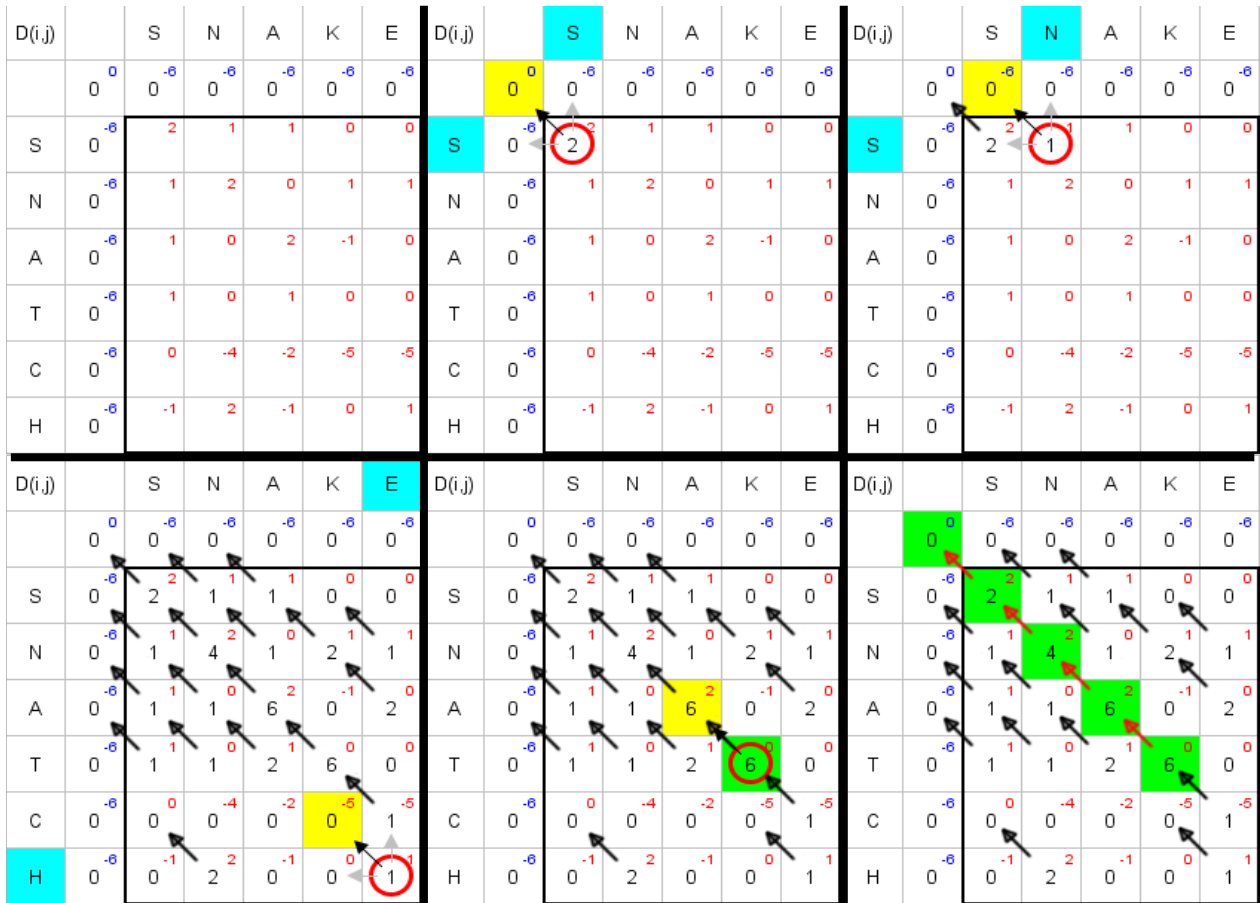


Figure 53. Local alignment by dynamic programming with the Smith-Waterman algorithm in six steps. Top row: initializing the H matrix and filling it with scores (Equation 11). Bottom row: reconstructing the local alignment by backtracking from the highest scoring field in the matrix. Figures created using <http://baba.sourceforge.net>.

Equation 11. Calculating the highest alignment score for a given cell in the Smith-Waterman algorithm.

$$H(i,j) = \max \begin{matrix} H(i-1, j-1) + S(a_i, b_j) & \text{(diagonal)} \\ H(i, j-1) - g & \text{(horizontal)} \end{matrix}$$

$$\left\{ \begin{array}{l} H(i-1, j) - g \\ 0 \end{array} \right. \quad \begin{array}{l} \text{(vertical)} \\ \text{(start again)} \end{array}$$

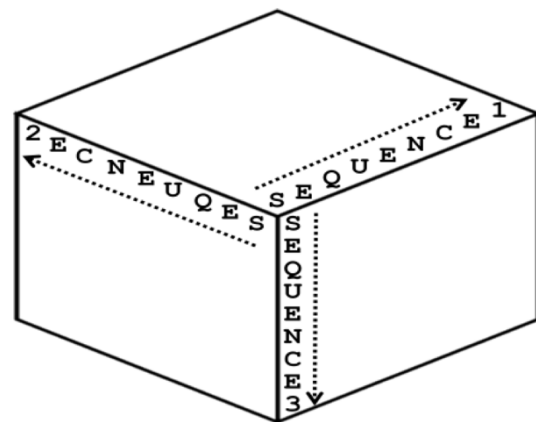
The backtracking procedure for local alignments begins at the highest-scoring field in the alignment matrix H and follows the arrows back until the first field containing a 0 is reached and the arrows stop. The highest scoring cell does not need to be at the bottom right hand corner because we are looking for the optimal sub-alignment, which can be in the middle of either of the two sequences. In Figure 53 we aligned the two short protein sequences SNAKE and SNATCH by using the Smith-Waterman algorithm, the PAM250 substitution matrix, and a gap penalty of -6. After filling in the alignment matrix H there are two fields with the highest score (6), meaning that there are two possible optimal alignments, both with a score of 6 (see below). This is because the PAM250 substitution score between arginine (K) and threonine (T) is zero, and does not add or subtract anything from the alignment score.

SNA	and	SNAK
SNA		SNAT

7.5. Multiple sequence alignment

So far, we have discussed algorithms for creating pairwise sequence alignments (i.e. alignments of two sequences), but what if we want to align multiple sequences? Multiple sequence alignments may be more accurate than pairwise sequence alignments, because more information can be taken into account to identify which residues are homologous, i.e. derived from the same residue in the last common ancestor (see Section 5.1). Conceptually, dynamic programming can easily be extended to multiple sequences. The alignment matrix H simply becomes multi-dimensional and the algorithm would work successively through each dimension to fill all the cells in H with the optimal possible score at each point in the matrix (Figure 54). However, it becomes computationally very expensive when the number of sequences increases, and for more than a few proteins of average length, it may take weeks to compute an alignment.

Figure 54. Alignment matrix H for three sequences. Every additional sequence adds a new dimension to this matrix. Dynamic programming for many sequences (dimensions) quickly becomes computationally prohibitive.



The most widely used approach to solve this problem is progressive multiple sequence alignment (Hogeweg and Hesper 1984). This approach first clusters the sequences by similarity and then progressively builds the sequence alignment as a series of pairwise alignments, the most similar

sequences being aligned first, and the following alignments are performed in order of decreasing similarity (Figure 55). The order of alignment is determined by hierarchical clustering (discussed in Chapter 3), and although the clustering looks like a tree it is important to realize that this is not a genuine phylogenetic tree, because it is not based on a multiple sequence alignment. Perhaps confusingly, this clustering is also known as a “guide tree”.

Initially, the performed alignments are sequence-to-sequence alignments (Figure 55B-C), where a substitution matrix such as BLOSUM62 can be used to obtain a substitution score for two aligned residues. Later, alignments with previously aligned blocks of sequences are made by aligning a sequence to an alignment (Figure 55E), or by aligning two alignments (Figure 55D). When aligning previously aligned blocks of sequences, each of the two alignments are treated as a single sequence, but the substitution score at a given position is calculated as the average substitution scores of all the residues in one vs. the other block. So, if you have two alignments with m and n sequences in each, the substitution score at any position is the average of all the $m \times n$ scores of all the residues compared separately. For example, if one previously aligned block of four sequences contains three valines (V) and one isoleucine (I), and we want to calculate the BLOSUM62 substitution score with another previously aligned block of two sequences that contain one alanine (A) and one serine (S), the substitution score for that position in the alignment of the two blocks becomes $(3 \cdot S_{VA} + S_{IA} + 3 \cdot S_{VS} + S_{IS}) / 8 = (3 \cdot -1 + 0 + 3 \cdot -2 + -2) / 8 = -1.375$ (see Figure 45). Whenever a gap needs to be introduced into a previously aligned block of sequences, the gap is placed in all the sequences of the existing alignment at once, so the average gap penalty is the same as the gap penalty for aligning a single sequence.

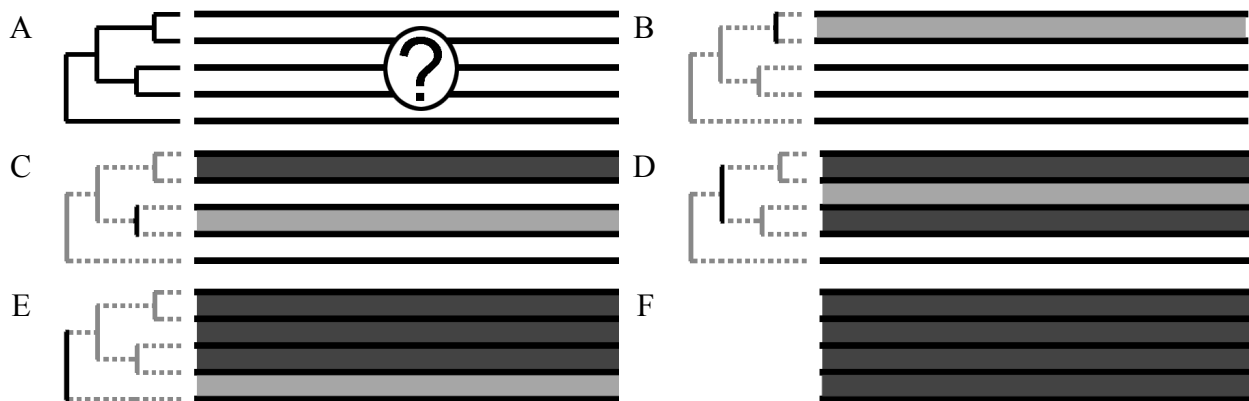


Figure 55. Progressive multiple sequence alignment is facilitated by hierarchical clustering (A). The guide tree to the left is not a phylogenetic tree, but a simple clustering (see Section 3.6). The algorithm aligns sequences and blocks of previously aligned sequences (indicated in dark gray) in pairs, starting with the most similar ones (B), and progressively including more dissimilar sequences/blocks (C-E). The final result is a full multiple sequence alignment (F). Dark gray blocks represent previously aligned sequences; light gray indicates the alignment being performed in the current step.

In progressive multiple sequence alignment, the number of comparisons that the computer needs to perform increases linearly with the number of sequences, rather than exponentially as in multi-dimensional dynamic programming. This means the algorithm is more efficient than dynamic programming in multiple dimensions, and it is fast enough to compute large alignments in reasonable time.

7.6. How to use sequence alignment?

The alignment score quantifies to what extent the alignment is consistent with a given model of evolution. While dynamic programming algorithms like those described in Sections 7.3 and 7.4 guarantee finding the alignment with the highest possible alignment score, a sequence alignment is only ever as good as the model of evolution. Perfect sequence alignment tools do not exist because exceptions are the rule in biology, and no single model of evolution has yet been developed that can accommodate all the possible ways biological sequences might evolve. For example, the model of evolution in the Needleman-Wunsch and Smith-Waterman algorithms contains a substitution matrix and gap penalties, but does not account for variations like inversions or duplications (see Figure 50B). It is good practice to manually inspect an automatically generated sequence alignment, and sometimes improvements need to be made to correct the errors made by alignment algorithms.

The best alignment tool to use also depends on your question, and on the computer power and time you are prepared to invest in answering it. Commonly asked questions include: "Are my sequences evolutionarily related (homologous)?" and "Which sequences in the database look like my query?". These different questions require different approaches to tackle them, and different sequence alignment algorithms have been developed to do this. The first is a question about a limited set of sequences, so a precise and computationally expensive algorithm can be used to answer it. The second is a question about a query sequence and a whole database of possible target sequences. In this case, millions of alignments may be made for the most similar hits in the database to be identified, so the computer algorithm should primarily be fast rather than precise. We will discuss fast sequence similarity searching algorithms in Chapter 8.

To summarize, sequence alignment is used in several ways.

1. Sequence alignment allows us to identify similar sequences. Similarity is used as a proxy for homology, but it might be also caused by non-hereditary mechanisms such as biases in the sequence or microsatellites (see Section 5.6).
2. Sequence alignment is also used to find the optimal alignment between similar sequences, and score them against a model of evolution. This allows us to assess the odds that the sequences are indeed homologous.
3. Only if we believe that sequences are indeed homologous, then we can use sequence alignments to draw evolutionary conclusions, such as inferring a phylogenetic tree or identifying conserved residues that may be important for function.

7.7. Reading

- Obligatory: help page of Clustal Omega: <http://www.ebi.ac.uk/Tools/msa/clustalo/help>
- Optional: What is dynamic programming? (Eddy 2004a)

7.8. Questions and exercises

Use pen and paper or your laptop for the exercises below. If you are in doubt, you can use Google to help you out.

- Align the amino acid sequences DEVD and DEEEVW using the Needleman-Wunsch algorithm, the identity matrix (match = 1 and mismatch = -1), and a gap penalty of -2. Hint: have a good look at the example explained in Figure 52. Start by hand, but once you get the hang of it, you can practice the dynamic programming algorithms by using the Java tool from <http://baba.sourceforge.net>.
- You are given the following two alignments of the same two DNA sequences:


(i) -CGCATG (ii) CGCATG
 ACG-A-G ACGA-G

Using the identity matrix (match = 1 and mismatch = -1), for which values of a linear gap penalty, g , the alignment in (i) is better than the alignment in (ii)? HINT: Try to write it as an equation and solve. Does it make sense to have positive gap penalties?

- Align the short peptide sequences SHAKE and SPEARE using the Needleman-Wunsch algorithm, the BLOSUM62 matrix, and a fixed gap penalty of -6.
 - Is the alignment different if you use the PAM250 matrix (Section 7.3)?
 - What happens if you make the gap penalty higher (-100) or lower (0)?
- The alignment fragment below is part of a multiple alignment of six protein sequences. Sequence 1 is human, sequence 3 is rat, sequence 5 is shark, and the remaining three sequences are chicken, chimpanzee, and mouse, but not necessarily in that order. Amino acids are colored according to their conservation and physical-chemical properties.

	10	20	30	40																																													
L	L	E	V	Q	V	R	E	S	L	A	K	S	S	Q	V	A	I	E	A	L	S	A	M	P	T	V	R	S	F	A	N	E	E	G	E	A	Q	K	F	R	E	K	L	Q	E	I	K	T	L
L	L	E	V	Q	V	R	E	S	L	A	K	S	S	Q	V	A	I	E	A	L	S	A	M	P	T	V	R	S	F	A	N	E	E	G	E	A	Q	K	F	R	E	K	L	Q	E	I	K	T	L
S	L	A	V	K	V	Q	E	S	L	A	K	S	T	Q	V	A	I	E	A	L	S	A	M	P	T	V	R	S	F	A	N	E	E	G	E	A	Q	K	F	R	Q	K	L	E	E	M	K	F	L
S	L	A	V	K	V	Q	E	S	L	A	K	S	T	Q	V	A	I	E	A	L	S	A	M	P	T	V	R	S	F	A	N	E	E	G	E	A	Q	K	F	R	Q	K	L	E	E	M	K	T	L
K	L	S	V	E	V	Q	E	S	L	A	K	A	N	D	V	A	V	E	T	F	L	S	M	K	T	V	R	S	F	A	N	E	D	G	E	S	R	R	Y	G	E	R	L	E	D	T	F	R	L
A	L	A	P	O	M	O	K	A	Q	A	R	A	S	E	V	A	V	E	T	F	Q	A	M	A	T	V	R	S	F	A	N	E	D	G	A	A	A	H	Y	R	O	R	L	O	O	S	H	R	L

- Mark all the completely conserved positions.
 - Are completely conserved positions distributed evenly over the entire length of the alignment? What do you expect regarding the relative importance of the different regions in the aligned proteins?
 - Do the amino acids at conserved positions in this alignment match with most conserved amino acids in PAM250 or BLOSUM62 (see Chapter 6 Exercise 1)? What could this mean?
 - Which sequence is chimpanzee? Which sequence is mouse?
 - Which sequence is closer to human: shark or chicken? Use identity to calculate the distance.
- For the sequences GYYI and DFKYIW, calculate the Smith-Waterman alignment using the BLOSUM62 matrix. Take a gap penalty of -12. What is the optimal local alignment score? You can practice Smith-Waterman algorithm by using the Java tool from <http://baba.sourceforge.net>.

6. Cytochromes are mostly membrane-bound proteins that contain heme groups and carry out electron transport or catalyze reductive/oxidative reactions. In Eukaryotes cytochromes are found in the inner membrane of mitochondria and endoplasmic reticulum.
- 
- Based on the figure, what kinds of secondary structures are present in cytochromes?
 - Download the file <http://tbb.bio.uu.nl/BDA/CytBProt.txt>.
 - The file you downloaded contains the amino acid sequences of cytochrome B proteins from the mitochondrial genome of 16 vertebrate species. The sequences are labeled with the species names. Which file format is used to represent the sequences? Give the file the correct file extension.
 - Look at the sequences. Will very long gaps be necessary to align them? Why (not)?
 - Make a multiple alignment of these protein sequences using the EBI Clustal Omega web server (or Google for other Clustal servers if the EBI server does not work). Select the “Clustal with numbers” output format to make it easier to talk about specific alignment positions. When the alignment is done and you used the EBI web server, you can click the Show Colors button to color the amino acids according to their properties. This will also make it easier to compare the sequences. Can you identify conserved regions that are longer than 10 amino acids?
 - The file <http://tbb.bio.uu.nl/BDA/CytBDNA.txt> contains the nucleotide gene sequences from the same species that encode the cytochrome B proteins. Make alignments of these sequences. Remember to set sequence type to DNA before starting the alignment. Does the DNA alignment look as you expected, given what you saw in the protein alignment?
 - Look at the conservation of the positions in the DNA alignment. Do you see a pattern in the conservation? What does this mean?
 - Would DNA or protein sequence alignments be more suitable for comparing very closely related species?
 - Let us now return to the cytochrome B protein alignment and have a look at a protein sequence from another kingdom, for example from the model plant species *Arabidopsis thaliana*. The protein sequence that you want has the accession number CAA47966.1, find it in the Genbank database and copy-paste the Fasta sequence to the protein fasta file on your computer.
 - Realign your vertebrate cytochrome B protein sequences together with this plant sequence. Are the regions you previously identified as conserved, still conserved in plants?
 - Examine the entry for CAA47966.1 on the Genbank website. Can you conclude anything about the functional properties of the conserved regions?
 - Hexokinases are enzymes that phosphorylate hexose sugars (mainly glucose). After phosphorylation, the sugar can be used in different metabolic processes. Download the following file containing hexokinase protein sequences from human and dog: <http://tbb.bio.uu.nl/BDA/Hexokinases.txt>. Generate a protein sequence alignment. Look at the Percent Identity Matrix on the Result Summary page. What is the percent sequence identity between all the sequences? Which pairs of sequences are most similar to each other?
 - Based on these limited datasets, do hexokinases or cytochrome B evolve faster?

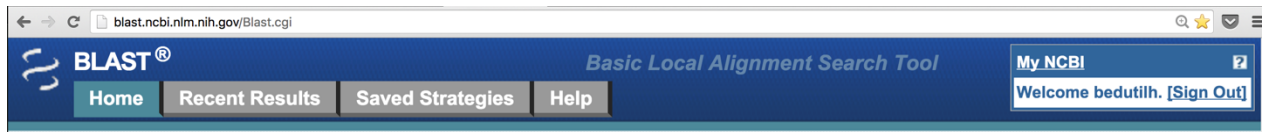
7. The following is a bonus question that may be rather difficult unless you are familiar with Excel (or with a similar program like the free version from Open Office). It is meant to show you that with some practice, you can even write little programs within these spreadsheet programs. So if you feel confident, give it a try! In case you get stuck, the answer is in the back of this Reader.

The idea is to see if you can make a spreadsheet that automatically generates a dot-plot if you fill in the residues of the first sequence in the cells of the first row, and the residues of the second sequence in the cells of the first column (but leaving the very first cell A1 free). This would allow you to quickly visualize stretches of sequence identity as diagonals on the spreadsheet. It is possible to do this by entering a simple function in the cells in the spreadsheet that read the values in the first row and column. If you succeed, the value of the cells will be automatically updated when the residues of either sequence are changed. Use this spreadsheet to generate a dot-plot of the tongue twister “Clean clams crammed in clean cans” versus itself. What feature typical of genome evolution do you observe in the dot-plot?. Hint: read the Excel Help page about switching between relative and absolute references to cells, and use the functions “if” and “exact”.

8. Searching for similar sequences

Sequence analysis can reveal a lot of information, such as the biological function of the sequence, the organism in which the sequence occurs, its evolutionary relationships with other sequences, and by extension the evolutionary relationships between the species they occur in. Much of this information can only be discovered after finding homologs of the gene or protein in a database. If these homologs contain functional or taxonomic annotations in the metadata (see Table 2), this provides the first clues about a sequence of interest.

Dynamic programming guarantees to find the optimal similarity score between two sequences, but it is quite inefficient and becomes computationally prohibitive when many sequences need to be compared (Section 7.6). For example, if we are searching for homologs in a large database like the databases of the INSDC (see Section 1.6), millions of sequences need to be compared to a query. In this chapter, we discuss popular bioinformatic tools to efficiently find homologs of a query sequence in a sequence database. The most efficient tools use heuristics, algorithmic shortcuts that are likely but not guaranteed to find the optimal solution, but gain so much in speed and/or computational memory efficiency that they are the only viable option for some biological questions. Depending on the algorithm, hits may be found several orders of magnitude faster with heuristic methods than with dynamic programming, at just a small loss in sensitivity or accuracy. Then, if we still want to make reliable alignments for evolutionary analyses, we can construct high-quality sequence alignments with only a subset of the sequences that have been retrieved as hits from the database (see Section 7.6).



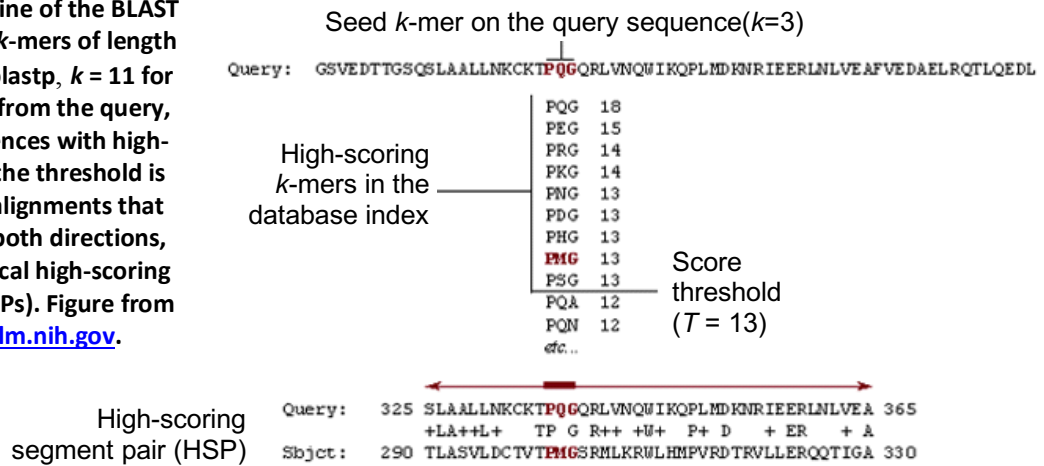
8.1. The Basic Local Alignment Search Tool BLAST

Probably the most popular similarity search tool is the Basic Local Alignment Search Tool BLAST (Altschul et al. 1997; Altschul et al. 1990). Before running BLAST an index is created, consisting of all the words of a given length k that are present in the database (also known as k -mers^{*}). This index is stored in the RAM of the computer, so that BLAST can very quickly access all the database sequences containing a certain word. When you start a sequence similarity search, the BLAST algorithm first scans your query sequence for k -mers that have high-scoring matches with k -mers in the database (see Figure 56). Using the database index stored in the RAM, database sequences containing these high-scoring words are retrieved very quickly. Then, the local alignment around these words is extended in both directions. This alignment process is performed by the CPU, and is a much slower process, because many computational operations need to be performed (see Chapter 7, and see Section 2.6 for a description of RAM and CPU in the computer). BLAST only extends the minority of the database sequences that contain high-scoring k -mers, so relatively few of these computationally costly alignments are performed. Moreover, the alignment extension in either direction stops when the score falls below a threshold. This corresponds to deciding that the

^{*} For $k = 1$, k -mers are called monomers; $k = 2$ are called dimers, dinucleotides (for DNA), or dipeptides (for amino acids); $k = 3$ are called trimers, trinucleotides, or tripeptides, and so on. The k -mers consisting of an undefined small number of nucleotides or amino acids are called oligonucleotides or oligopeptides, respectively. “Oligo” means “few”.

alignment in that direction looks so bad that there is no point in continuing it. To finish the BLAST search, hits are reported if the local alignment around the seed k -mer has a high enough score. Such a locally high-scoring region is called a High-scoring Segment Pair (HSP).

Figure 56. Outline of the BLAST algorithm. Words or k -mers of length k (default: $k = 3$ for blastp, $k = 11$ for blastn) are extracted from the query, and database sequences with high-scoring k -mers (here the threshold is $T \geq 13$) used to seed alignments that are extended in both directions, resulting in local high-scoring alignments (HSPs). Figure from <http://www.ncbi.nlm.nih.gov>.



There are several BLAST flavors that perform sequence similarity searches with different types of sequences. For example, nucleotide sequence similarity searches can be performed by blastn and protein sequence similarity searches can be performed by blastp (Section 8.3). Many sequence databases allow BLAST searches to be performed online. For example, the European Bioinformatics Institute (EBI) allows the nucleotide and protein databases to be searched at <http://www.ebi.ac.uk/Tools/sss>. While you can always choose to use the default settings when running bioinformatic programs, many parameters can be adjusted, and sometimes it is worthwhile to investigate the optimal settings for your question (Kerfeld and Scott 2011).

8.2. Masking low-complexity regions

Both nucleotide and amino acid sequences can be highly repetitive, i.e. they consist of short, frequently repeated sequence motifs. Repetitive regions evolve differently than normal sequences (see Section 5.6), and can lead to sequence similarity between phylogenetically unrelated sequences. In BLAST searches, these regions are known as low complexity regions. To avoid high BLAST scores between potentially non-homologous sequences, BLAST filters out these regions from a query sequence before performing the similarity search. This is done by masking them. The repetitive stretches of nucleotides or amino acids in low complexity regions are replaced by an equivalent number of ambiguity characters (N in nucleotide sequences, X in protein sequences, see Section 5.4). These characters are absent in the substitution matrix, and they are ignored when calculating similarity scores.

8.3. Sequence searches: blastn, (discontiguous) megablast, and blastp

Megablast and blastn are both used to search with a nucleotide query in a nucleotide database. The most important parameter governing the sensitivity of BLAST searches is the length of the seed k -mers that initially identify potential target sequences in the database (see Figure 56). Blastn uses a shorter default word size than megablast, $k = 11$ for blastn, versus $k = 28$ for megablast, so blastn

will find more high-scoring k -mers in the database, and more alignments need to be performed (see Section 8.1). As a result, blastn is slower than megablast, but also more sensitive which means it is better at identifying similarity to distantly related sequences with many mutations. In contrast, megablast is faster because it spends less time extending potentially insignificant alignments than blastn. Megablast is the best choice when searching highly similar sequences, e.g. the same gene in closely related strains, while blastn is better when searching for more distantly related homologs. The word size is adjustable in blastn and can be reduced from the default value to a minimum of 7 to increase search sensitivity.

A search that is both fast and sensitive can be performed with discontinuous megablast. Rather than requiring exactly matching k -mers as seeds for alignment extension, discontinuous megablast uses long, but non-contiguous words according to a certain pattern (Figure 57). This pattern is used both when indexing the database sequences and when extracting them from a query sequence (Section 8.1). Using discontinuous megablast, the number of relevant nucleotides in a pattern could be high like the long k -mers in megablast, so the search is fast. At the same time, the spaced nucleotides ignore potentially mutated positions, so the search is also sensitive. Alternative coding and non-coding patterns can be specified, for example 1101101101101101 for protein coding sequence and 1110010110110111 for non-coding sequence*.

```

TTTCGGAGGGCAAGAAGCTATCGCTGGAAATATTATTGTAAG

TT.CG.AG.GC.AG.AG.TA.CG.TG.A
TT.GG.GG.CA.GA.GC.AT.GC.GG.A
TC.GA.GG.AA.AA.CT.TC.CT.GA.A
CG.AG.GC.AG.AG.TA.CG.TG.AA.T
GG.GG.CA.GA.GC.AT.GC.GG.AA.A
GA.GG.AA.AA.CT.TC.CT.GA.AT.T
AG.GC.AG.AG.TA.CG.TG.AA.TA.T
GG.CA.GA.GC.AT.GC.GG.AA.AT.A
GG.AA.AA.CT.TC.CT.GA.AT.TT.T
GC.AG.AG.TA.CG.TG.AA.TA.TA.T
CA.GA.GC.AT.GC.GG.AA.AT.AT.G
AA.AA.CT.TC.CT.GA.AT.TT.TT.T
AG.AG.TA.CG.TG.AA.TA.TA.TG.A
GA.GC.AT.GC.GG.AA.AT.AT.GT.A
AA.CT.TC.CT.GA.AT.TT.TT.TA.G

```

Figure 57. Example of discontinuous words in a nucleotide sequence according to the pattern 1101101101101101101101101101101. This pattern is strong at identifying protein coding sequence.

Similarity searches with a protein query in a protein database can be performed with blastp. The default k -mer size in blastp is $k = 3$, corresponding to 9 nucleotides which is shorter than the default k -mer size in blastn. By default, high-scoring k -mers are identified by using the BLOSUM62 matrix (as in Figure 56), but other substitution matrices can also be selected. Because of the short k -mer size, and because protein sequences are more conserved than nucleotide sequences (Section 5.3), blastp is much more sensitive than blastn at finding distantly related homologs.

* Non-coding templates attempt to minimize correlation between successive words, when the database sequence is shifted by 4 positions against the query. This means there are more 1s at the ends of the template (at least 3 on each side). See https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp.

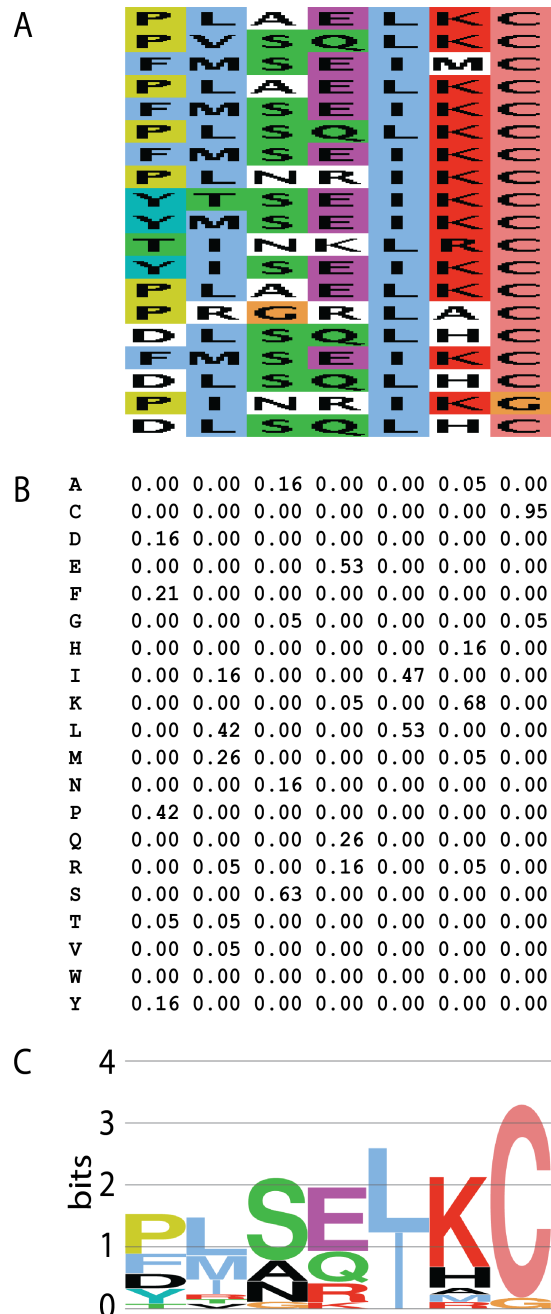
8.4. Sensitive homology searches with sequence profiles

The substitution scores that are used in the sequence similarity searches above are defined at the level of amino acid pairs (see Section 6.2) and they are the same regardless of the position in the sequence. That means that a proline (P) aligned to a proline always contributes 7 points to the alignment score, and a proline aligned to a cysteine (C) always contributes -3 points (Figure 45). Now let's take a look at the multiple sequence alignment in Figure 36. It seems logical that if proline within one of the conserved domains is mutated into a cysteine, this may upset the function of the protein more than if that proline is present in one of the variable regions. This discrepancy in sequence conservation is not taken into account by standard sequence-based alignment scores like those calculated in Section 8.3 or in Chapter 7.

Figure 58. Three ways to view a sequence profile: (A) a multiple sequence alignment, (B) a position-specific scoring matrix (PSSM), and (C) a sequence logo (see Section 5.5).

To calculate a similarity score that gives higher weights to positions in a sequence that are conserved in evolution and lower weights to positions that are variable, we can use sequence profiles. Sequence profiles are similar to sequence logos because they contain information about the amino acids that are found at every position in a sequence alignment, as well as information about the conservation of each position (see Figure 58). Like sequence logos, sequence profiles are based on a multiple sequence alignment, where at each position in the alignment the fraction of amino acid or nucleotide residues is indicated. In the computer, a sequence profile is represented as a position-specific scoring matrix (PSSM), a table that lists the exact frequency of each residue at every position (Figure 58B). When a sequence profile is used to search a database, it can detect much more subtle relationships between proteins than if a sequence is used, and is used to detect distant structural or functional homologs.

Position Specific Iterated-BLAST (PSI-BLAST) is an algorithm based on blastp that exploits sequence profiles to be able to detect distantly related protein sequences. In an initial blastp search, PSI-BLAST first tries to find a set of good homologs in the database. Based on these hits, PSI-BLAST then creates a multiple sequence alignment, and produces a sequence PSSM that indicates the more and less conserved positions in the sequence. This PSSM is



then used to search the database again, often resulting in more hits being found than in the first round (Figure 59). PSI-BLAST can iterate this cycle of identifying similar sequences and building PSSMs as often as you want, but usually you may decide to stop once a hit of interest has been found, or if no new high-scoring sequences are found in the database, and the PSSM does not change anymore. When this happens, we say the algorithm has converged.

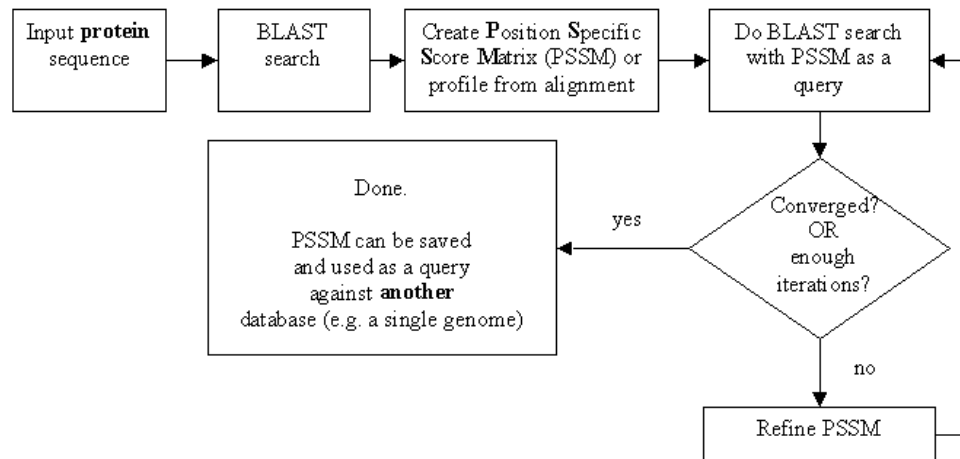


Figure 59. Flow diagram of the steps in Position Specific Iterated-BLAST (PSI-BLAST).

8.5. Translated searches: blastx, tblastn, and tblastx

Sometimes you may want to search with a protein query in a nucleotide database, or with a nucleotide query in a protein database. For these situations, BLAST has implemented several translated search options that use the genetic code (Figure 38) to translate either the database or the query in all six reading frames before performing the similarity search. High-scoring k -mers are identified in protein space (i.e. at the amino acid level), so translated searches are more sensitive than blastn at equivalent values of k . However, because the nucleotide is translated in six reading frames, searches are approximately six times slower than a simple blastp search. Translated BLAST searches are useful when you want to find homologous proteins to a nucleotide coding region, but you do not know *a priori* what the reading frame of the encoded protein is.

Searching with a nucleotide query in a protein database is done with blastx. Blastx translates the query sequence in six frames and finds HSPs at the amino acid level. If multiple HSPs are found between a query sequence and the same database hit, blastx provides combined significance statistics for all those hits (see Section 8.7). This makes the tool particularly useful when the nucleotide sequence contains errors that may lead to frame shifts, or if it contains many mutations that lead to a low sequence similarity. Thus, blastx is often the first analysis performed with a newly determined nucleotide sequence, for example for metagenomes from a relatively unexplored environment, where many stretches of DNA are sequenced that are expected to have only low similarity to the sequences in the database because the microbial species have never been sequenced before (see Section 1.4).

Searching with a protein query in a translated nucleotide database is done with tblastn. For example, tblastn is useful when you have a protein sequence and a bunch of unannotated nucleotide data such as a newly sequenced genome, and you want to know if the genome encodes any homologs

of your protein query. In this case, the database is translated in six frames before identifying high-scoring amino acid k -mers. Like blastx, tblastn provides combined significance statistics when multiple HSPs are found between the same query and target sequences (see Section 8.7).

It is also possible to use a nucleotide query and a nucleotide database, but perform the search in protein space. This is done with tblastx that stores amino acid k -mers from a six-frame translated nucleotide database in the RAM, and also translates the nucleotide query sequence in six frames before identifying high-scoring k -mers between the database and the query. Potential hit sequences in the database that match the high-scoring k -mers are retrieved and the alignments extended at the amino acid level. Thus, tblastx effectively performs $6 \times 6 = 36$ blastp searches between a nucleotide query and a nucleotide database, without having to manually translate the nucleotide sequence into protein. This makes it the slowest to run of all the BLAST flavors. As with the other translated BLAST searches, tblastx accounts for potential frame-shifts and errors in the sequence by calculating hit statistics for the entire query and target sequences at once, not per HSP (see Section 8.7).

8.6. Ultra-fast search tools that skip alignment extension

Because the alignment extension step in the BLAST algorithm requires a lot of operations to be performed by the CPU, this is the step that takes by far the most time in the algorithm (see Sections 2.6 and 8.1). Thus, BLAST is not feasible for high-throughput analyses where tens of thousands, or even up to millions of sequences are compared to a database, such as next-generation sequencing (NGS) datasets (see Chapter 3). One way of implementing faster similarity searching is by only counting high-scoring k -mers that link the query and database sequences, and skipping the CPU-heavy alignment extension step. This approach comes at a definite cost to both the sensitivity and specificity of the hits, meaning that both fewer distantly related homologs will be identified (sensitivity), and that the hits that are identified might not be homologous, i.e. they are false positives (specificity). However, the increase in speed is enormous, making it feasible to search large NGS datasets against a database while translating every nucleotide query sequence in six frames into protein. Depending on the size of the database and the size of the RAM, k -mers of length $k = 4-12$ amino acids can be stored. The most significant hits will be those database sequences that share many high-scoring k -mers with the query sequence.

Several other fast sequence similarity search algorithms exist, and new ones are continually being developed. If you are going to work with large sequencing datasets, check recent literature to discover what is the preferred bioinformatic tool to use in your field.

8.7. Expectation values (E-values)

When doing a sequence similarity search, an important question is whether an identified target sequence is significantly similar to the query. In this case, “significantly similar” means that our alignment score (S) is higher than expected by random. A hit with *some* similarity score will always be found, and the chance of finding a high score increases with a longer query sequence (m) and/or a bigger database (n). To quantify the significance of a hit we calculate the E-value (E), which represents the number of expected hits with a score of at least S , when we use a random query of length m , and search a random database of size n (Equation 12).

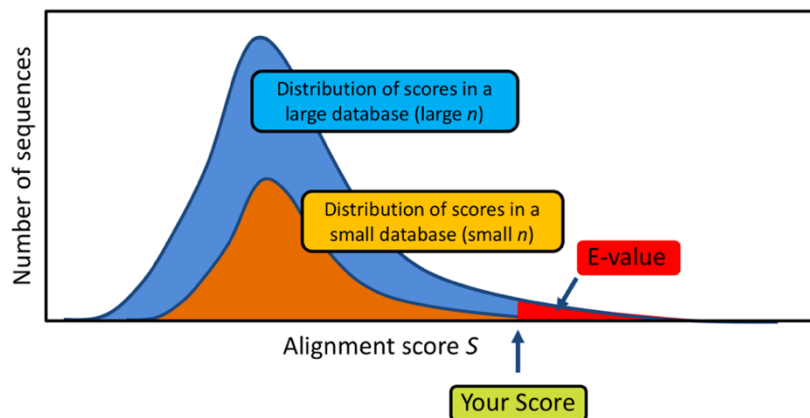
Equation 12. Formula to calculate the E-value of a BLAST hit from the length of the query (m), database size (n), alignment score (S), and scaling factors λ and K . λ is a scaling factor that corrects for the substitution matrix (the substitution matrix directly affects the alignment score S) and K is a scaling factor for the search space size $m \times n$.

$$E = K \times m \times n \times e^{-\lambda S}.$$

Because the E-value represents the number of hits expected by chance, hits with low E-values are more significant than hits with high E-values. To get an intuition for the formula, consider that doubling the length of the query sequence (m) or the database (n) should double the expected number of HSPs E . Also, to achieve an alignment score that is twice as high ($S \Rightarrow 2S$), the HSP must obtain a twice higher alignment score, and according to probability theory such an event is half as likely to happen. So E , the expected number of HSPs with an alignment score of at least S decreases exponentially with the score S .

The cartoon in Figure 60 may further improve your intuition. Imagine that you perform a BLAST search with a given query sequence and find a hit with a certain alignment score in the database. Interestingly, the E-value of this hit will be lower, i.e. more significant if it was found in a small database than if the hit came from a large database, even though the identified sequence and its alignment score with the query remain identical. This can be important to remember when searching for homologs of your gene of interest in the online databases. Because the size of some databases doubles in size every 18 months (see Section 1.2), if you find a border-line hit with an E-value just below your threshold today, you may no longer find that same sequence in the database if you perform the same BLAST search next year.

Figure 60. A cartoon where the red surface area depicts the E-value, the statistic used to assess the significance of BLAST hits. The E-value is defined as the expected number of sequences having an alignment score at least as high as your score S . More high-scoring hits are expected by chance in a large database than in a small database.



It depends on the research question what is a good E-value cutoff, i.e. at which E-value you still want to look at the hits. For example, if you want to identify reliable homologs, only very low E-values should be considered (e.g. $E < 0.00001$ or 10^{-5}). However, if you have absolutely no information about a query sequence and are willing to consider any hints about a potential function, then you might at least want to take a look at a hit with an E-value of 0.1, especially if it is the best hit you can find in the database. If you find a high E-value of say, 16.3 this means that the hit is not very good at all, because you would expect 16.3 hits with that same alignment score or better.

8.8. Reading and videos

- Obligatory: NCBI Blast Tutorial: <http://youtu.be/HXEpBnUbAMo>
- Obligatory: Blast E-values:

- <http://youtu.be/nO0wJgZRZJs>
- <http://youtu.be/Z7ek7UoP7Bg>
- Obligatory: Filion "Once upon a BLAST": <http://blog.thegrandlocus.com/2014/06/once-upon-a-blast>
- Obligatory: Wikipedia page PWMs http://en.wikipedia.org/wiki/Position_weight_matrix
- Optional: (Higgs and Attwood 2005) chapter 7.

8.9. Questions and exercises

For many of the questions below you will need to access the online BLAST tool at EBI or NCBI website. Note that we have listed some hints for answering the questions at the end of this Chapter.

Hints: BLAST

1. On the NCBI BLAST page, look carefully at all the links before you click on something.
2. Choose the "nr" non-redundant protein database for your blast search. In this way you are not limiting yourself to only human or mouse sequences.
3. Make sure you use the correct BLAST flavor; NCBI defaults to Megablast for nucleotide sequences, not blastn!
4. When you want to search for scientific articles relating to a certain topic, there are various specialized search engines you can use. For articles related to biology and medicine you can use NCBI's Pubmed at <http://www.pubmed.gov>. To find scientific articles on all fields of science, you can also try Google Scholar: <http://scholar.google.com>.

Hints: Function Prediction

1. There are several ways you can try to predict the function of a sequence. A much-used method is to search for homologs with a known function, and Blast is a good tool for this.
 2. An easy way to reduce database size is to limit your BLAST search to a subset of a given database through the "Organism" or "Entrez Query" options (found under "Choose Search Set" on the BLAST query page). You can see the actual database size when you click on the Search summary link on top of the results page.
 3. You can turn on the low complexity filter under "Algorithm parameters". If you click on the number of bits following each Blast hit, you will see the alignment of the query sequence (the one you provided) and the hit sequence (the one in the database, it is marked as Sbjct in the alignment). You might notice that some parts of the sequence are printed with greyish low case letters. These are low-complexity regions, which are ignored when computing the alignment score of the hit (see Section 8.2).
 4. Exact amino acid matches in the alignment are marked with the amino acid letter, and amino acids with high values in BLOSUM or another scoring matrix, which probably have comparable properties are marked with +.
1. You have a query sequence that is 40 amino acids long. We BLAST this sequence against a non-redundant protein database with 29,318 sequences, and having total length of 6,548,585 amino acids. BLAST uses BLOSUM62 as the substitution matrix by default. The best hit is in a sequence which is 420 amino acids long. You observe the following alignment of the best hit (for clarity, only the HSP region is shown).

```
Query      27  NFSSSQ  32
```

- a. What is the alignment score?
 - b. How many hits do you expect to find by chance with this alignment score? Use the following parameter values: $K = 0.04$ and $\lambda = 0.27$.
 - c. Why do you think this hit is significant/insignificant?
2. Imagine you discovered a novel fungus and sequenced its genome. You use a gene prediction program to identify 6,500 protein-coding genes and translate them into protein. Using these protein sequences, you perform a blastp search against known fungal genomes.
- a. Of the 6,500 proteins, 5,500 have a match in other species of fungi (*Saccharomyces cerevisiae*, *Schizosaccharomyces pombe*, or *Neurospora crassa*) with a very low E-value. What can you say about these proteins?
 - b. The remaining 1,000 proteins each have a best match with an E-value larger than 10. What can you say about these proteins?
 - c. Of these last 1,000 proteins, can you make another BLAST search to verify your conclusion? What else can you try?
3. a. We have the DNA sequence of a gene and want to identify very distant homologs in a DNA database. Which BLAST flavor would you use to search, and why?
- b. We have the amino acid sequence of a well-studied enzyme from *E. coli*. We have been unable to identify a homologous sequence in *Bacillus subtilis* (another bacterium) using blastp, but experimental evidence shows that *B. subtilis* is capable of doing this enzyme reaction. What type of BLAST could you do to identify the gene responsible for the reaction in *B. subtilis*? Assume that the genome and all predicted proteins of *B. subtilis* are in the databases.
- c. Below you will find part of the output of a blastp search of the protein AP_012432 from zebrafish against a database containing all proteins from yeast. Predict the function of AP_012432.

Sequences producing alignments:		Score (Bits)	E Value
ref YP_00168034	two-component sensor histidine kinase	173	4e-78
ref ZP_02948675	two-component system sensor histidine kinase	173	4e-77
ref LP_03036538	sensory transduction histidine kinase	163	3e-75
ref YP_00128034	unknown extracellular enzyme	25	0.6
ref ZP_01238675	matrix protein homologous to env4	21	1.1
ref LP_02134538	DNA helicase precursor protein	21	1.1

4. We have a short protein segment from chicken:

FGGHNAITYPPGVSLAVGHFFSEWAEKFGDPLYRSSSSSSSSSSSSSSSSSTENKLAFGTHRDRDVGHHFCKAGAAAEKF

We do a BLAST search to predict its function. Top 10 hits are as follows.

Sequences producing significant alignments:		Score (Bits)	E Value
gi 76638832 ref XP_60	similar to SRp25 nuclease [Bos taurus]	40.8	0.017
gi 6649242 gb AAF21439.1	splicing coactivator subunit SRm300 ...	40.0	0.028
gi 66828915 ref XP_647811.1	hypothetical protein DDB0206273 ...	40.0	0.028
gi 66358726 ref XP_626541.1	hypothetical protein cgd2_3540	39.7	0.037

- c. Sketch a graph illustrating how the probability of finding at least one hit depends on the length of k if all the database sequences are random.
- d. From a biological perspective, explain what it means if the length of k is too high.
- e. She compares the speed of her program with two other ways of identifying local sequence homology in a database: BLAST and Smith-Waterman. Explain which of these three approaches is the fastest and which is the slowest.
- f. She compares the sensitivity of her program for detecting distant homologs with blastn and tblastx. Explain which of these three approaches is the most sensitive and which is the least sensitive.

8. A research group identified a gene from patients with disturbed sleeping patterns:

```
>gene_X
gggtgaacag ccgcacggga gtaggtacgc acctgacctc gctggcactg ccgggcaagg
cagaggggtgt ggcgtcgctc accagccagt gcagctacag cagcaccatc gtccatgtgg
gagacaagaa gccgcagccg gagttagaga tggtggaaga tgctgcgagt gggccagaat
```

- a. Go to NCBI BLAST and perform a blastn search with this sequence. (You can copy-paste it from the digital version of the reader.) Use the "Nucleotide Collection" as a database. What is the most likely hit?
 - b. Identify the single nucleotide polymorphism(s) (SNPs) that this patient carries in the gene. Do these mutations cause a difference in the protein sequence if this gene is expressed? Can you find this out using a different type of BLAST?
 - c. What is the difference between blastn and megablast results with this same query? To see the differences, focus in particular on the homologs in distantly related organisms found by both searches (you can get the taxonomy/organism/lineage report by clicking on the "Taxonomy Report" link on the top of the BLAST results page). For example, are you able to find a homolog in a distantly related organism like *Xenopus* in both cases? If not, why? Hint: by default, Blast reports max 100 target sequences, but you can increase this number under "Algorithm parameters" on the Blast input page.
 - d. Search the OMIM database to see if the gene you identified can really cause sleep disorders.
9. The Basic Local Alignment Search Tool (BLAST) identifies sequences in a database that are similar to a query sequence. BLAST does this relatively fast. Explain in your own words why the BLAST algorithm is faster than local alignment by dynamic programming.
10. The genome of the bacteriophage crAssphage was recently discovered by recycling data that was available in publically accessible databases and innovative bioinformatic analysis. The crAssphage genome was discovered in metagenomic DNA sequences isolated from human fecal samples, and contains 80 protein-coding genes. However, most of those proteins do not yet have a reliable functional annotation. At this point in time, we do not even know what bacterial host this phage infects.
- a. Predict the function of the following sequence based on its homologs (using NCBI BLAST).

```
>crAssphage_protein_25
```

MKRNISNTILTKDYIFSQVSKVITIFSTYTGISVEDIQHCIDTGEFISPPFREDTHPSFGFRYDNRNKLKGRDFAGYWWGDC
IDAAATVLSEIVHKQIDISIKSQFLFVLKHIAYTFRNIIYGQDKDENNDYNIARAISNVRNKHPIIELVTRPWNLDAYW
GQFGVNLNLFNLTHFVYPVDQFYINRSTNPIPKYFYDKDKTDLCYGYVLGQDKRGIVNVKLYFPNRNKKTEVKFITNSNTIE
GVINLELDNYDVIIITKSTKDRLSLECYLKSINHSILYGGSTLESKTIGVVNI PHETYKLRQIEYDWLRSKLNRRNGFLISL
MDNDRTGLMEAVILKNDYDIIPIIIPKELGVKDFAE LRSSYSTNVINELTQQVIKYIEENYGEETEFTWDTESNTLPY

- b. In which species did you find the best hits? Is this expected?
- c. Imagine you have the complete DNA sequence of the crAssphage genome. How can you, using BLAST, identify where on the genome the above protein sequence is encoded?

11. Ubiquitin is a regulatory protein that is ubiquitously expressed in eukaryotes. The most prominent function of ubiquitin is labeling proteins for proteasomal degradation. Besides this function, ubiquitination also controls the stability, function, and intracellular localization of many proteins.

- a. What is the HGNC identifier of human ubiquitin? Use the R script from Chapter 2 Exercise **Error! Reference source not found.** to extract the amino acid sequence from the database.
- b. Use BLAST with the human sequence as input to find out if ubiquitin is only present in eukaryotes. Click on “Taxonomy report” link on top of BLAST result page to see the taxonomy of your hits. By default, NCBI only shows the 100 best hits. You can change this by clicking “Algorithm Parameters” on the BLAST query page and selecting a higher number under “Max target sequences”.

9. Phylogenetic inference

The calculation of the grouping and the branch lengths of a tree is done in two major steps. First, using clustering methods based on distance matrix (calculated from a multiple alignment, see Chapter 7) an initial tree is constructed. The clustering methods do not guarantee the optimal solution. Second, the "optimal tree" is found by a search in the tree-space. Here we will explain two optimization methods for optimal tree searches: parsimony and maximum likelihood.

9.1. Clustering using a distance matrix

From a multiple alignment we can compute a matrix of the evolutionary divergence between each pair of sequences in the alignment. Such a matrix is called "distance matrix" and it is symmetric: the distance of sequence i to sequence j is the same as the distance of sequence j to sequence i . The distance between nucleotide sequences, d , can be calculated by several methods. For example, d can be the number of non-matching sites divided by the total number of sites. However this measure has several problems as mentioned in Chapter 6.

A distance matrix is the input for clustering algorithms. These algorithms work as the following:

- Join the closest two clusters to form a single larger cluster.
- Recalculate distances between all clusters and the "new" cluster.
- Repeat the above steps until all sequences are joined within one cluster.

Below, we will discuss two algorithms that are very widely used: UPGMA and neighbor-joining.

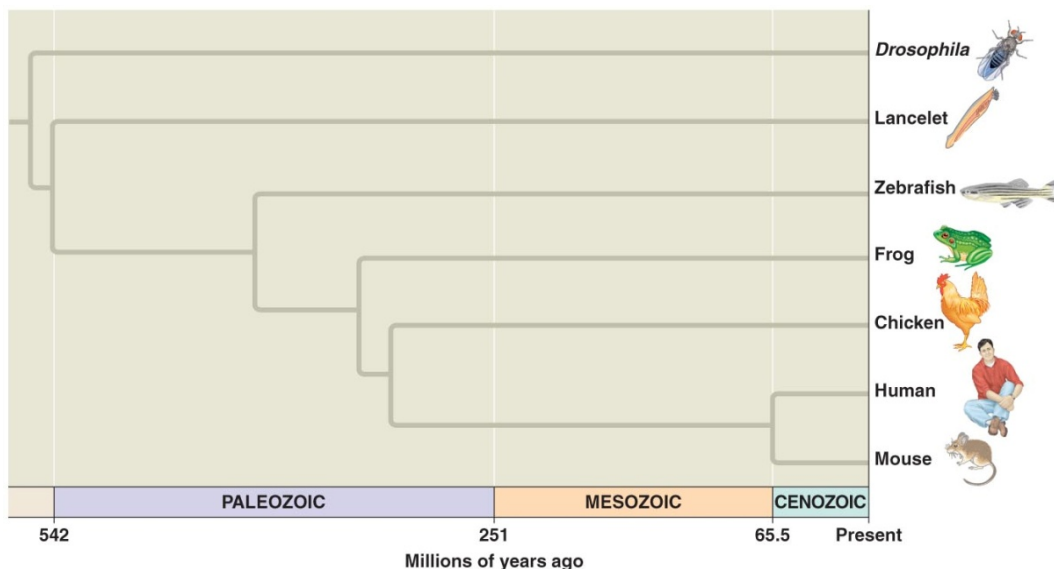


Figure 61. A tree inferred using Unweighted Pair Group Method with Arithmetic mean (UPGMA). Note how all species at the branch tips have equal distances to the common ancestor at the root of the tree. Fig 26.13 in (Campbell et al., 2011).

9.2. Unweighted Pair Group Method with Arithmetic mean (UPGMA)

The UPGMA method (compare with Section 3.6) is one example of a hierarchical clustering method. The distance between two clusters is calculated as the average of the distances from each data point in the first cluster to each other data point in the second cluster. The UPGMA method

assumes that there is a strict molecular clock and that all species evolve at the same rate. Thus, if you use UPGMA to draw a tree, all data points are put on a (horizontal) line and the nodes combining clusters are put at a height equal to half the distance between the clusters, because we are assuming that half of the changes separating the two species, occurred on the branch leading to one species and half occurred on the other.

In an UPGMA tree, the distance from any point in the tree to all descendant leaves is identical. This property is called ultrametricity. An example of an UPGMA tree is given in Figure 61. Take three species: fish, bird and human. Bird and human are more closely related to each other than they are to fish. From fish to bird is the same as the distance from fish to human. The distance between bird and human is smaller than the fish-bird (or fish-human) distance. In reality many species might evolve with different rates, e.g. a line leading to mouse might be slower than one leading to rat. Therefore, the ultrametric trees generated by UPGMA are often incorrect. However, UPGMA is the simplest clustering algorithm and equal amounts of chronological time can be represented by these ultrametric trees. For example, fish and human had a common ancestor 542 million years ago and the evolution in one branch could be faster than the other, but nevertheless there had been 542 million years of evolution in both branches. UPGMA is also widely used for hierarchical clustering of numeric data.

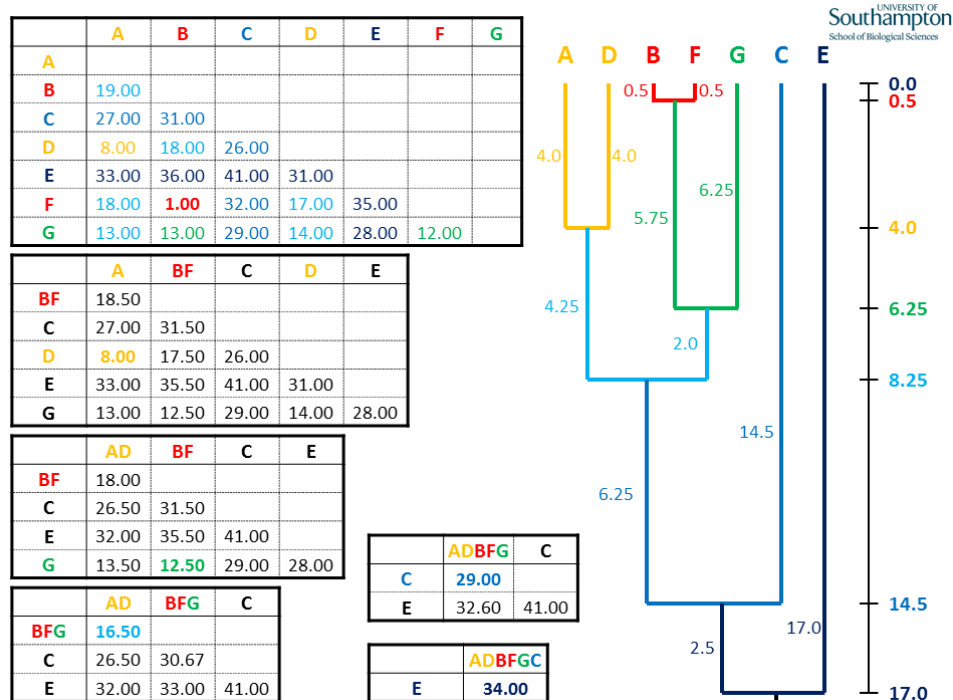


Figure 62. A worked-out example of the Unweighted Pair Group Method with Arithmetic mean (UPGMA) method of phylogenetic construction for seven sequences. Figure from <http://www.soton.ac.uk>.

The distance between two clusters in UPGMA method is defined as the following (see also Figure 17C). Consider you have a cluster X containing N_X sequences and cluster Y containing N_Y sequences. Initially, each cluster would only contain a single sequence. The distance between clusters X and Y is defined as the arithmetic mean of all the pair distances between the sequences in X and Y :

$$d_{XY} = \frac{1}{N_X N_Y} \sum_{i \in X, j \in Y} d_{ij}$$

where i labels all sequences in X and j labels all sequences in Y , and d_{ij} is the distance between sequences i and j . When X and Y are combined into a new cluster Z , the new distances can be defined using existing cluster-to-cluster distances without using sequence pair distances:

$$d_{ZW} = \frac{N_X d_{XW} + N_Y d_{YW}}{N_X + N_Y}$$

Figure 62 demonstrates how this would work.

9.3. Neighbor joining (NJ)

An important disadvantage of UPGMA is that it assumes that the molecular clock holds across all species, i.e. all sequences evolve at the same rate. This assumption is in general false (see Section 4.4). Neighbor Joining (NJ) is similar to UPGMA but uses a modified distance function to account for potential differences in evolutionary rates. Thus, NJ often gives better trees than UPGMA, while it is faster than tree-search algorithms like maximum parsimony and maximum likelihood (these model-based phylogenetic methods will follow in Sections 9.4 and 9.5).

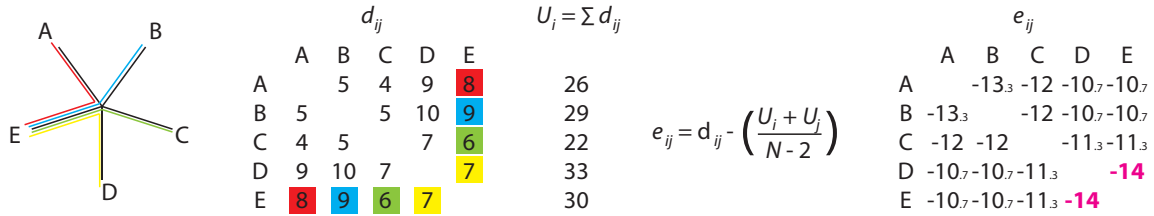
The NJ algorithm proceeds as follows (see Figure 63 for an example). We start out with a regular distance matrix, containing the distances d_{ij} between all pairs of sequences i and j (Step 1). We then calculate for each sequence i (A-E), the “cumulative distance” U_i to all the other sequences j by adding all pairwise distances up: $U_i = \sum d_{ij}$. Using these cumulative distances U_i , we calculate a modified distance matrix e_{ij} (also for all pairs of sequences i and j), according to the formula $e_{ij} = d_{ij} - ((U_i + U_j) / (N - 2))$. The idea is that by adjusting the distance d_{ij} for the cumulative distances U_i and U_j , we can correct for differences in evolutionary rates in that lineage – even without knowing which nodes cluster together, at least at this point. The modified distances e_{ij} in our example are shown in Figure 63 (Step 1, right).

After calculating the modified distances e_{ij} we choose the pair of nodes i and j with the lowest modified distance and replace it by a new “internal” node W that represents their putative last common ancestor. This step is similar to UPGMA (see Section 9.2). In our example, the lowest modified distance value is $e_{DE} = -14$. We now need to calculate the distances between the new node W and the nodes D and E that were replaced by W . In the tree, these distances are represented in the branch lengths $D-W$ and $E-W$, and their sum is d_{DE} . This is done according to the formulas: $d_{DW} = (d_{DE} + (U_D - U_E) / (N - 2)) / 2$ and $d_{EW} = (d_{DE} + (U_E - U_D) / (N - 2)) / 2$ (Figure 63, Step 2). Note that N is the number of elements in the distance matrix, and this number decreases from 5 to 4 to 3 as the iterations progress (Figure 63).

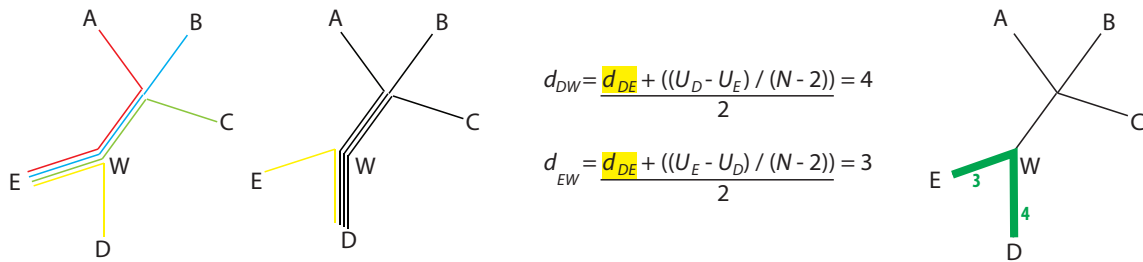
Finally, we determine the distance between W and the remaining nodes, and place them into a new distance matrix (Figure 63, Step 3). Instead of calculating the distance d_{Wi} of W to another node i as the average of all the distances (which is what we do with centroid clustering in Section 3.6, and with UPGMA in Section 9.2), we calculate it as $d_{Wi} = (d_{iE} + d_{iD} - d_{ED}) / 2$. In words, this is half of the sum of the distances from node $D-i$ plus from node $E-i$, minus the distance from node $D-E$. It

turns out this is exactly the distance from W to i . Step 3 in Figure 63 shows this for the example of $i = A$, where $d_{AD} = 9$ is shown with a black line, $d_{AE} = 8$ is shown in red, and $d_{ED} = 7$ is shown in yellow). We thus obtain a new distance matrix where sequences D and E are replaced by node W (Step 3). From there we can repeat the algorithm until we have a fully connected tree (Step 4).

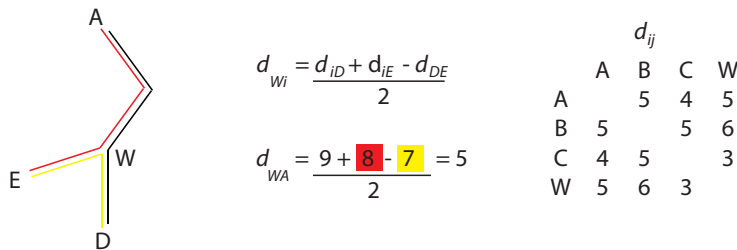
(1) From a regular distance matrix d_{ij} to a modified distance matrix e_{ij} between all nodes ($N = 5$)



(2) Join the two closest neighbors in the modified matrix via a new node W, and calculate their distances to W



(3) Calculate distances between new node W and all remaining nodes i (example for $i = A$)



(4) Iterate steps 1, 2, and 3 until all neighbors are joined into a single tree

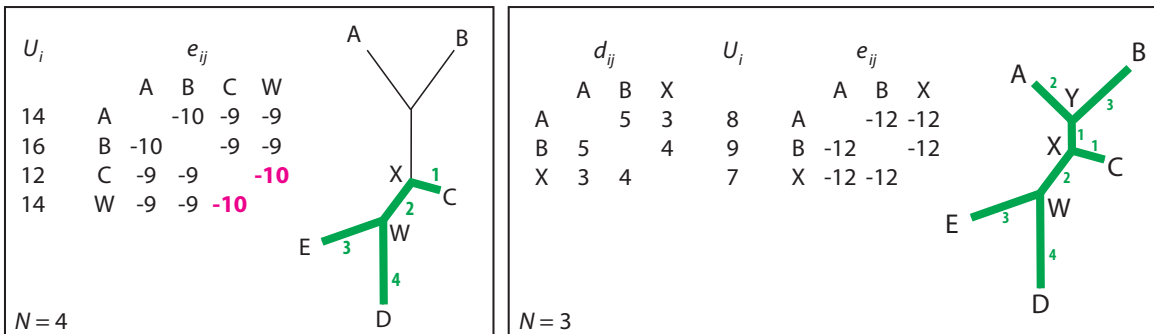


Figure 63. Steps 1-4 in the NJ algorithm. See text for details.

The distances between pairs of sequences are a measure of the evolutionary time that separates them (see Section 4.4). As you can see in the example in Figure 63, the NJ leads to a tree that accurately represents these distances on all the branches. In our example, if we travel along the branches from A to D, we get $2 + 1 + 2 + 4 = 9$, which is precisely the number of mutations that occurred between sequences A and D (see the original matrix d_{ij} in Step 1). In some cases, this can lead to negative branch lengths in a NJ tree when two sequences cluster together that have very different evolutionary rates (average number of mutations per year, since their last common ancestor). Negative branch lengths are an artifact of the NJ algorithm – after all, evolutionary time cannot run backwards. However, NJ does not “stretch” the evolutionary clock on some branches and “compress” it on others as in UPGMA, since the number of mutations is accurately represented in the branch lengths. In an elegant way, NJ preserves the information about the mutational distances between the sequences from the original distance matrix. The trees created with NJ are called non-ultrametric, which means that the molecular clock is not assumed to hold, and evolutionary rates can differ in different branches.

9.4. **Maximum parsimony (MP)**

Maximum parsimony (MP) and Maximum Likelihood (ML, Section 9.5) are completely different ways of creating phylogenetic trees than those explained in the Sections above. Both methods depend on a model of evolution (see Section 6.5), and try to find the best tree for a given multiple sequence alignment in the light of that model. Unlike the clustering algorithms above, MP and ML do not start from the sequence alignment and hierarchically join the sequences in some way. Instead, they search through all the trees that could possibly be created from the sequences, and calculate a score for each tree that represents how well it corresponds with the model of evolution. The tree that has the maximum score is selected as the “true” phylogeny, i.e. the MP or ML tree. (In Section 9.6 we will discuss how to search all possible trees.)

For MP, the model of evolution is very simple: the fewer mutations you find on the branches, the better the tree. This model of evolution is derived from the principle of parsimony, i.e. the idea that the best explanation of an observation is the one with the fewest assumptions. This idea is attributed to William of Ockham (1287-1347), an English Franciscan friar, scholastic philosopher, and theologian who stated that among competing hypotheses, the one with the fewest assumptions should be selected. The principle of parsimony is also known as Ockham’s Razor.

Figure 64 explains MP for one single position in a nucleotide alignment. Assume that the nucleotide at this position is a C in human, chimpanzee, and gorilla, while it is a T in orangutan and gibbon. Now, we are going to search all possible trees containing these five species (Figure 64A-O). In the tree in Figure 64A, the minimum number of substitutions needed to explain the observed nucleotides is one: the most parsimonious explanation would be that the sequence changed from a C to a T at position “*” (or vice versa: note that these trees are not rooted, so there is no time directionality to the branches and different mutations are also possible). Similarly, in Figure 64B we would need at least two substitution events to occur, and so on. Thus, based on this single nucleotide position, the trees in Figure 64A, D, and E are more parsimonious than the other trees in Figure 64. Not all sites are informative, for example if a site is identical in every sequence, then it is completely non-informative for the MP phylogeny. Moreover, if a site is identical in all sequences but differs in only one, then it is also non-informative, because only a single species has the substitution and it does not help us to distinguish between the different possible trees.

Independent of the number of sequences, an “informative position” has at least two different possible characters (nucleotides or amino acids) and if each of these occurs at least twice in the data set. Note that, while “uninformative positions” do not influence the order of the branches in the tree, they are important to calculate the branch lengths. Another example of how parsimony works is given in Figure 26.15 of (Campbell et al., 2011).

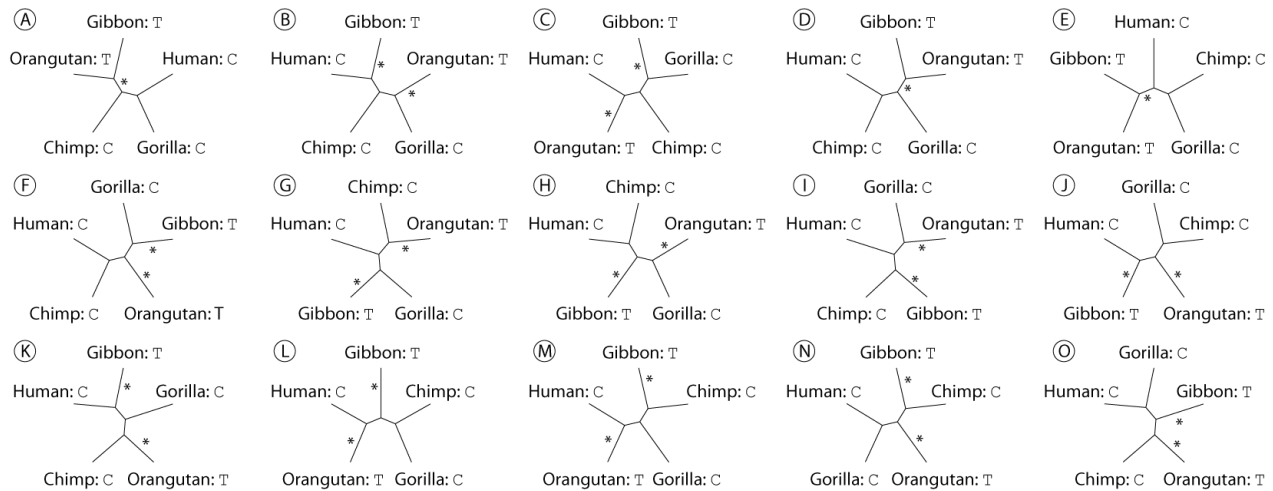


Figure 64. The principle of molecular parsimony for a single nucleotide position. A-O show all possible unrooted trees. Asterisks indicate which mutations are minimally required to explain the observed nucleotides. If this was the only informative nucleotide in the alignment, then A, D, and E would be the maximum parsimony (MP) trees.

9.5. Maximum Likelihood (ML)

The Maximum Likelihood (ML) approach of phylogenetic reconstruction is very similar to MP (Section 9.4), but the model of evolution used in ML is more realistic. Each possible mutation in the sequence is given a specific probability, for example by using the BLOSUM matrix for amino acid substitutions (see Section 6.2). Insertions and deletions also have their own specific probability, for example based on gap penalties as in sequence alignment (see Section 7.2). However, the evolutionary model can be much more advanced than just using a single substitution matrix and gap penalties for the entire alignment. For example, information from the sequence alignment can be taken into account. If the alignment tells us that some positions are more conserved than others, this knowledge can be included into the model so that trees that do require mutations in those positions get a lower likelihood score. As with MP, the evolutionary model in ML allows us to look at a given tree and calculate the likelihood that the sequence alignment could have resulted from it. And like in MP, ML searches all possible trees to find the one that maximizes this likelihood (see Section 9.6). Thus, ML finds the phylogenetic tree that is most likely to have generated the sequence alignment, given an evolutionary model. Because bioinformaticians keep improving the models of sequence evolution, ML is currently one of the best ways to reconstruct a phylogenetic tree.

9.6. How to search all possible trees?

Both MP and ML need to search all possible trees, but how is this done? Once we generate an initial tree using clustering methods, we have to search for the optimal tree in "tree space". The number of tree topologies is enormous: the number of unrooted trees with n species

$N_U = (2n - 5)!! = (2n - 5) \times (2n - 7) \times \dots \times 3 \times 1$ (the notation $!!$ means the factorial of all odd numbers) and the number of rooted trees $N_R = (2n - 3)!! = (2n - 3) \times (2n - 5) \times (2n - 7) \times \dots \times 3 \times 1$. So even if we have only 10 sequences in our data set, the number of possible rooted trees is more than 34 million. Thus, it is not actually possible to test every possible tree exhaustively if we have more than, say, eight species. Instead we have to make a heuristic search in the tree space. In a heuristic search, rather than trying all possible trees, you try and focus on trees that seem to be getting you nearer to the optimal tree. As with all heuristic searches (see Chapter 8), you cannot be *completely* sure that you find the optimal solution, but the enormous improvements in speed of the search algorithm make this a small risk worth taking.

Suppose we have an initial guess at a reasonably good tree. For example, this could be a tree produced by a distance method, which already gives pretty good trees. Then it is wise to start searching for the ML tree from trees that are similar to this initial tree. If, by changing one branch, we find a similar tree (i.e. one that is close in "tree space") that is slightly better, then we accept this change, and again search the neighbors of this tree for slightly better ones. We can keep doing this, slowly meandering through tree space, until we can make no more improvements to the tree, i.e. we have reached a local optimum. Since we have no guarantee that this locally optimal tree is also the true ML tree (global optimum), we can take several different initial trees and repeat the process of finding locally optimal trees. In some cases, it may also help the search for a global optimum if some large changes are included (big jumps in tree space).

9.7. Bootstrapping

Bootstrapping is a widely used statistical technique to measure the reliability of reconstructed trees. This approach creates a number of re-sampled versions of the original sequence alignment in order to establish how firmly the data, i.e. the alignment, supports the conclusion, i.e. the tree. Each re-sampling consists of randomly drawing positions (columns) from the original multiple sequence alignment and placing them in a new "re-sampled multiple sequence alignment" (as exemplified in Figure 65). Note that in bootstrapping, when we draw a random position from the original alignment, we do not remove it and hence it could be drawn again (this is called "sampling with replacement"). The same number of positions is drawn for the bootstrap alignment, as are present in the original alignment. This procedure thus results in a bootstrap alignment that is of equal length as the original alignment, but contains a randomly shuffled, and re-weighted sub-sample of the original alignment.

When we construct a new tree with this re-sampled alignment, we are not guaranteed to get the same tree as from the original alignment. If the phylogenetic signal is strong in the original alignment, there are many positions that support it, so the same tree will be created even with the re-sampled alignment. However, if the phylogenetic signal is weak, re-sampling will be unlikely to generate the same tree. Another way to look at this is to say that if the reconstruction of the initial phylogenetic tree depended on just few "noisy" positions in the alignment (especially if the alignment is short), then the bootstrap re-sampling should reveal this, because it will often reconstruct a different phylogenetic tree based on re-sampling that left the "noisy" positions out.

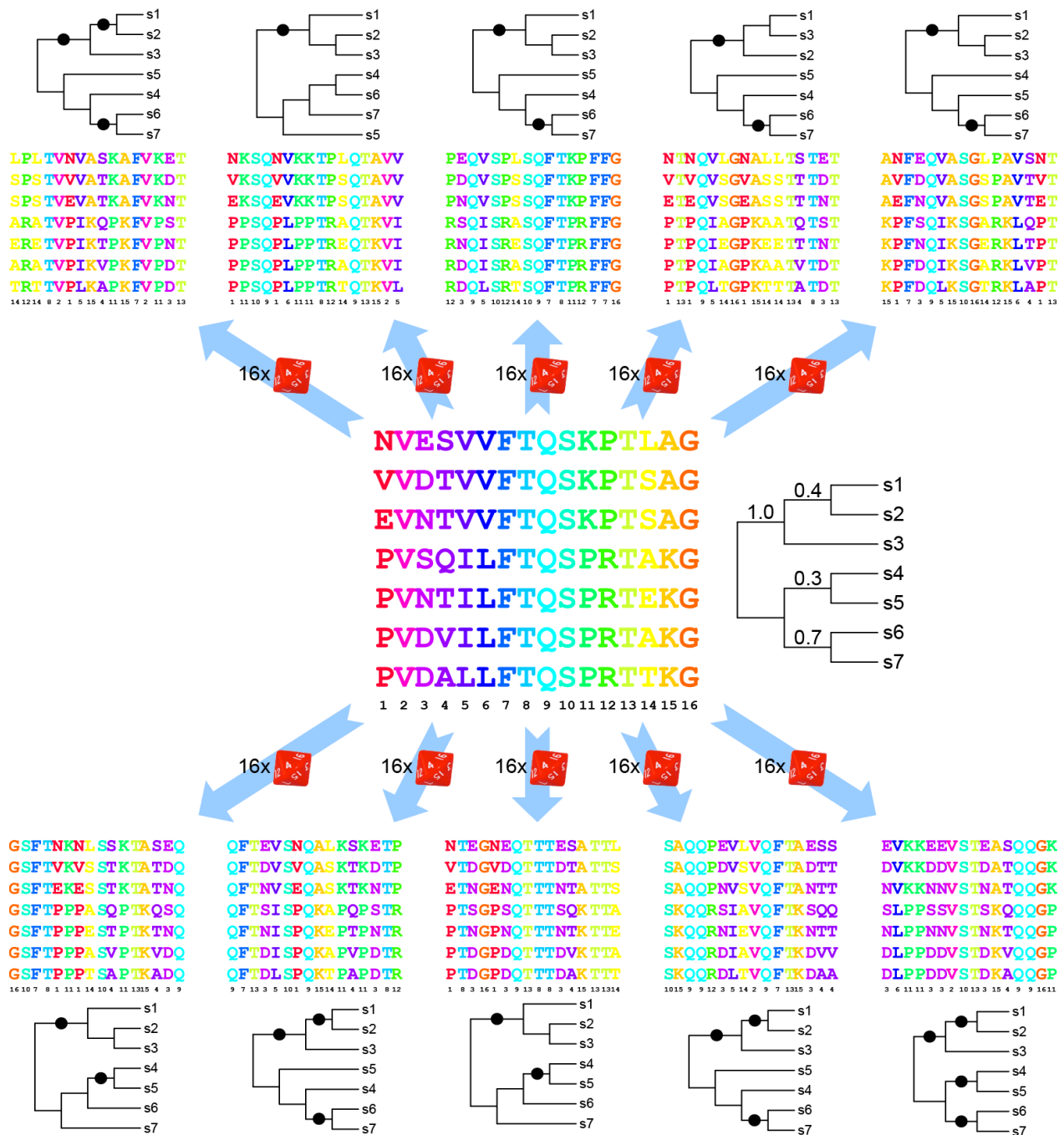


Figure 65. Example of bootstrapping. We start with a 16-amino acid alignment consisting of sequences sq1-sq7, and the unrooted phylogenetic tree that is derived from it (middle). Then, ten sets of columns are re-sampled from this alignment with replacement. This is like rolling a 16-sided dice 16 times. New phylogenies are created from the resampled alignments, called bootstrap trees. Finally, we count how often each branch in the original tree in the middle is found in the bootstrap trees. Branches consistent with the original tree are indicated in the bootstrap trees with black circles. A branch is defined as a split between two groups of sequences. For example, the split between sq4-sq5 and sq1-sq2-sq3-sq6-sq7 is found in 3 of the ten bootstrap trees.

To carry out a bootstrap analysis, the re-sampling process is usually repeated at least 100 or 1,000 times (10 times in Figure 65), and for each randomly re-sampled alignment, a bootstrap tree is generated. Then, we ask for each branch, or group of species in the original tree, in what percentage

of the randomized bootstrap trees do we also find this same group of species? This percentage gives us a measure of statistical confidence that those species really do form a related group. Often, values greater than 70% are thought to be reasonably strong evidence for a "true" clade.

9.8. Reading

- Obligatory: help page of the EBI Phylogeny tool: http://www.ebi.ac.uk/Tools/phylogeny/simple_phylogeny/help
- Obligatory: Chapter 26 in (Campbell et al., 2011).
- Optional: (Higgs and Attwood 2005) Chapter 8.

9.9. Questions and exercises

1. What is the difference between a hierarchical clustering as discussed in Section 3.6 and a phylogenetic tree as discussed in this chapter?
2.
 - a. Construct a phylogenetic tree from the distance matrix below using UPGMA.
 - b. Calculate branch lengths.
 - c. Write the tree in Newick tree format (bracket-notation, see Section 3.7).

	A	B	C	D
A	-	0.2	0.3	0.4
B	0.2	-	0.4	0.4
C	0.3	0.4	-	0.5
D	0.4	0.4	0.5	-

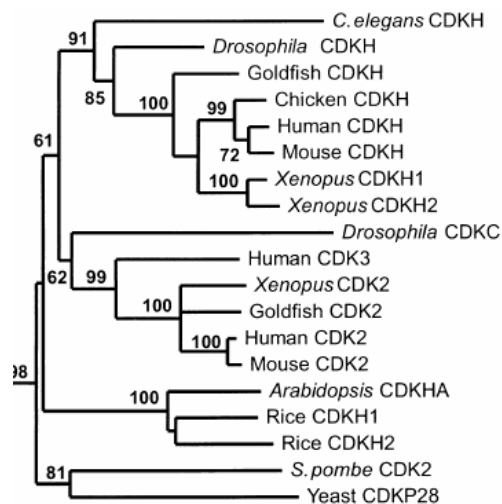
3. Observe the alignment of the Mad2 interacting motif in Figure 35. What was the ancestral residue of the 9th position of the alignment? Give the most parsimonious solution.
4. Four species (A, B, C, and D) are characterized by the respective homologous sequences ATCC, ATGC, TTCG, and TCGG. Use the percentage of substitutions/mismatches as the distance between each pair of sequences, correct the distances using the Jukes-Cantor formula, and derive a phylogenetic tree using UPGMA. Calculate branch lengths and write the final result in Newick tree format.
5. Given the alignment of a protein coding region below (assume that the 1st position in this alignment is also the 1st position in the codon):
 - a. Do synonymous substitutions occur more often than non-synonymous substitutions? What does this mean for selective pressure (see Section 5.3)?
 - b. Do transversions occur more often than transitions in this alignment? (see Section 6.4).
 - c. Where are the informative sites for parsimony method?
 - d. Draw a phylogenetic tree using parsimony method.

Hedgehog	GTGAATGAAT	GGCTTTCCAG	AAGTGAAGTG
Human	GTTAATGAGT	GGTTTTCCAG	AAGTGAAGTG
Hare	GTTAACGAGT	GGTTTTCCAG	AAGTGAAATG
Wombat	GTTAATGAGT	GGTTATCCAG	AAGTGAGATA
Opossum	GTTAATGAGT	GGTTATTTCAG	AAGTGAGATA

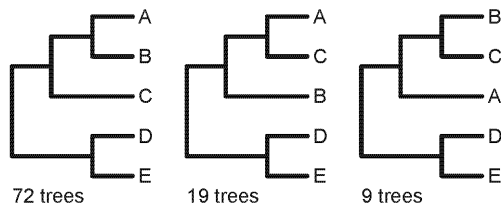
6. Use Maximum Parsimony to derive an unrooted phylogenetic tree for the following alignment.

Fugu	ATGTCGGGAAACACT
ZebraFish	ATGTCGGGAAACTGT
Trout	ATGGCAAGGAAC TCT
Salmon	ATGGCAAGGAAC TCT
Halibut	ATGGCAGGGAAATCT
Shark	ACTGATGCGGGATCA

7. A phylogenetic tree of the cyclin-dependent kinase (CDK) family in eukaryotes was constructed by the neighbor-joining method and is given below (*Schizosaccharomyces pombe* is a fungus, *Caenorhabditis elegans* is a worm, *Xenopus* is a frog, *Arabidopsis* is a plant species). Figure from (Hughes, da Silva, and Friedman 2001).
- Mention two gene loss and two gene duplication events.
 - Use this tree to predict the number of CDK genes in the common ancestor of eukaryotes.
 - Looking at the bootstrap values, how often do you expect a clade with only the Human CDKH gene and the *Xenopus* CDKH1 gene?

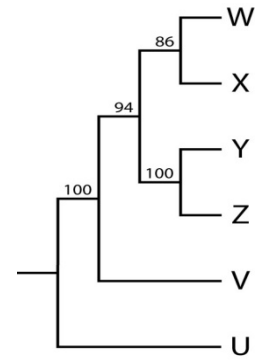


8. In a bootstrap experiment, where you generate in total 100 trees, you observe the trees as listed below. Draw the most likely tree with bootstrap values.

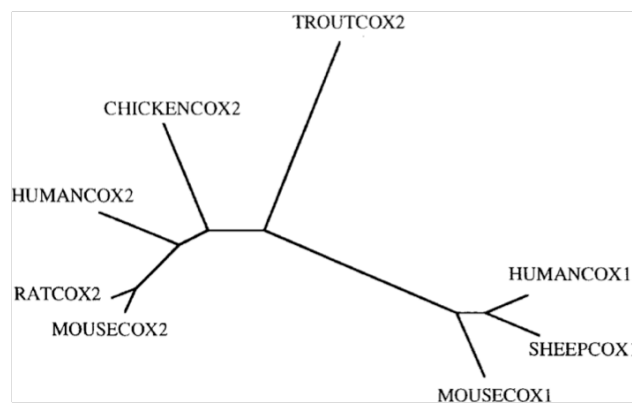


9. Given the tree with bootstrap values below:
- How often are W+X not together as a clade (i.e. grouping of two species) in the 100 bootstrap trees from the re-sampled alignments?

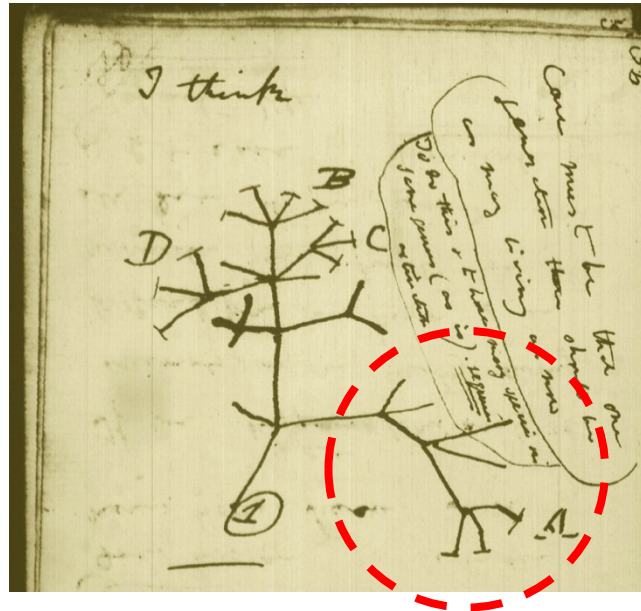
- b. Could the clade W+V be present 14 times in the 100 bootstrap trees?
- c. Could the clade W+Y+Z be present 14 times in the 100 bootstrap trees?



10. An unrooted phylogenetic tree of the COX genes was constructed by the neighbor-joining method and is given below (the tree is adapted from (Zou et al. 1999)).



- a. Write the Newick tree format bracket-notation for this tree.
- b. Draw this unrooted tree in rooted form. Use your biological knowledge to decide where the root can be. Make sure that your tree is the most parsimonious one.
- c. Indicate where possible gene duplications and gene losses have occurred.
- d. We are performing 100 bootstrap experiments, and find that 25 times Human COX1 and Mouse COX1 gene form a clade alone. Based on this add a single bootstrap value to the COX gene tree.
- e. In a more recent study the ortholog of human COX1 gene was found in trout. Does this finding change any of your answers in A, B, and C? If yes, what would be the new answers?



11. In his Notebook B on the Transmutation of Species (1837-1838), Charles Darwin wrote "I think", and then drew a diagram of an evolutionary tree (above).
 - a. What is represented by the letters A, B, C, and D?
 - b. What is represented by the circled number ① ?
 - c. Why would you not be able to obtain this tree by using the UPGMA algorithm?
 - d. Observe the sub-branch within the dashed red circle. In the figure above, label the root of this sub-branch "R" and give arbitrary names to the leaves. Write this sub-branch as a bracket-notation.

12. Let us return to ubiquitin, the widespread gene that we studied in exercise 11 in Section 8.9. Remember what you learned about the evolution of ubiquitin?
 - a. Construct multiple sequence alignments of a set of ubiquitin protein (<http://tbb.bio.uu.nl/BDA/ubiqaa.txt>) and DNA (<http://tbb.bio.uu.nl/BDA/ubiqdna.txt>) sequences at the EBI Clustal Omega web server. (Remember exercise 6 in Section 7.8.) Check out the phylogenetic tree in the tab on the results page.
 - b. Which tree (DNA or protein) gives more evolutionary information? Why?
 - c. Where would you estimate the root of this tree to be? Draw a correctly rooted version of this tree.
 - d. Check the classification of these species in the NCBI Taxonomy at <http://www.ncbi.nlm.nih.gov/Taxonomy/>. Under Taxonomy "Common Tree", you can enter the species names (sometimes you will have to search Google for the Latin name!) and the tool will generate a cladogram showing their shared taxonomic lineages. Compare this species tree with the tree you made using ubiquitin sequences. What is the difference?

10. Glossary of bioinformatic jargon

Adapted from https://mipt.ru/dbmp/student/files/bioinformatics/books/glossary_bioinf.php and <http://home.shirazu.ac.ir/~rmostofi/courses/bioinformatics/bifiles/Glossary%20of%20Bioinformatics.pdf>.

Adenine (A)	Nucleotide, one member of the base pair AT (adenine-thymine) in DNA.
Algorithm	Fixed procedure embodied in a computer program.
Alignment	Process of lining up two or more sequences to achieve maximal levels of identity (and conservation for amino acids). Used to assess the degree of similarity and the possibility of homology. The resulting set of aligned sequences is also called a sequence alignment.
Alignment matrix H	Table used in dynamic programming for comparing all combinations of positions in two or more sequences to identify the highest scoring alignment.
Allele	Alternative form of a genetic locus. At a locus for eye color different alleles might result in blue or brown eyes.
Alternative splicing	Different ways of combining a gene's exons for encoding protein variants.
Ambiguity character	Character in a nucleotide sequence representing one of two or more possible nucleotides, see Figure 39. The only ambiguity character in proteins is the X.
Amino acids	Class of 20 molecules that are combined to form proteins.
Amplification	Increase in the number of copies of a specific DNA fragment.
Annotation	Adding information to a database entry, like the function of a sequence.
Assembly	Merging DNA sequencing reads into a longer contiguous sequence.
Base	One of the molecules that form DNA and RNA molecules (A/C/G/T/U).
Base pair (bp)	Two nucleotides (A-T or G-C) held together by weak bonds.
Best bidirectional hit	Say geneX is the highest-scoring hit from a BLAST search of gene2 in genomeX. If gene2 is also the best hit when reciprocally searching geneX in genome2, they are BBH and probably orthologous.
Bias	Data is biased if the values differ structurally from the random expectation.
Bifurcation	Point in a phylogeny where one evolutionary lineage splits into two.
Bit score	Score in sequence alignment derived from the raw alignment score where the statistical properties of the scoring system are considered. Thus, bit scores can be used to compare alignment scores from different searches.
BLAST	<u>B</u> asic <u>L</u> ocal <u>A</u> lignment <u>S</u> earch <u>T</u> ool. Algorithm to search a sequence database for optimal local alignments to a query.
BLOSUM	<u>B</u> lOcks <u>S</u> ubstitution <u>M</u> atrix. Amino acid substitution matrix where scores are derived from observations of substitution frequencies in protein sequence alignments.
Bootstrapping	Approach to estimate the reliability of a branching order in a phylogenetic tree, based on resampling the positions in a multiple sequence alignment.
Branch length	Length of a branch in a phylogenetic tree corresponding to the number of mutations that occurred. Longer branch lengths mean more mutations.
Chromosome	Linear or circular nucleotide structure containing heritable information.
Cladogram	Tree-like graph indicating relationships between entities and lineages.
Clone	Exact copy of biological material such as a DNA segment, cell, or organism.
Clustering	Grouping objects so that objects in the same cluster are more similar to each other than to objects in other clusters (in some sense or another).

Codon	Triplet of nucleotides coding for an amino acid or protein termination signal (stop codon).
Conjugation	Transfer of mobile genetic elements by specialized structures assembled between two adjacently located cells.
Conserved residues	Positions in a sequence that remain unchanged. In proteins: mutations that preserve the physico-chemical properties of the amino acid.
Contiguous	Uninterrupted. Contigs are assembled after DNA sequencing.
Convergence	Point that the cycles in an iterative algorithm become identical (e.g. PSI-BLAST).
Core (CPU)	<u>C</u> entral <u>P</u> rocessing <u>U</u> nit of a computer. Each core can do a parallel calculation.
Curate, curation	Organizing data collected from various sources, make sure they are consistent.
Cytosine (C)	Nucleotide, one member of the base pair GC (guanine-cytosine) in DNA.
Deletion	Loss of part of a sequence in evolution, e.g. a single residue or segment.
Distance matrix	Symmetrical matrix containing distances between all data points. Distance matrices always contain zeroes on diagonal because distance to self is zero.
Domain	Discrete part of a protein, folds separately and possesses its own function.
Dominant gene	Gene whose phenotype is visible even if present in one copy on a genome.
Draft sequence	Sequence of a genome having lower accuracy than finished sequence.
Dynamic programming	Algorithm to identify the optimal local (Smith-Waterman) or global (Needleman-Wunsch) alignment between two sequences.
E-value	<u>E</u> xpectation <u>v</u> alue. Expected number of equal or better alignments in a database of equal size. A lower E-value means a more significant hit.
Enzyme	A catalyst protein that speeds up a biochemical metabolic reaction.
Exon	Uninterrupted segment of a gene sequence on the DNA.
Expression	Conversion of a gene's information into cellular function, including transcription into mRNA and translation into protein.
FASTA	The first widely used algorithm for database sequence similarity searching. Now mostly a simple file format to store sequences with a description line.
FASTQ	File format to store DNA sequences with a quality score per nucleotide.
File compression	Decreasing the size of a file by removing redundant information.
File format	Standard way to encode information for storage in a computer file.
Gap	Space in an aligned sequence that compensates for an indel. Introduction of a gap in an alignment is associated with a gap penalty in the score.
Gap penalty	Negative score added to an alignment score when aligning a gap. Because an indel mutation is a single evolutionary event, opening a new gap may cost a higher penalty than extending an existing gap in an alignment.
Gene	Physical and functional unit of heredity encoded on a chromosome.
Gene prediction	Prediction of possible genes on a DNA sequence by a computer program.
Genetic code	Table translating 64 nucleotide triplets into amino acids or stop codons.
Genome	All the genetic material of an organism, organized in chromosomes.
Genotype	Genetic identity of an organism, represented by a unique sequence. In biological evolution: the level where variation by mutations occurs.
Global alignment	Alignment of two nucleotide or protein sequences over their entire length.
Guanine (G)	Nucleotide, one member of the base pair GC (guanine-cytosine) in DNA.
Heuristic	Fast method that is likely but not guaranteed to find the correct solution.
High-throughput method	Method to perform a lot of measurements at once, like high-throughput DNA sequencing that reads the nucleotides in many DNA fragments.

Homology	Similarity (e.g. between sequences) by descent from a common ancestor.
Homopolymer	Stretch of identical residues in a sequence, for example AAAAAAA in DNA.
HSP	<u>H</u> igh- <u>s</u> coring <u>S</u> egment <u>P</u> air. Good, gap-less aligned hit in a BLAST search.
Identifier (ID)	Unique name of a piece of information, so that it can be found by computers.
Identity	Extent to which two nucleotide or amino acid sequences are identical.
Indel	Insertion or deletion mutation in one sequence relative to another.
<i>In silico</i>	Studies performed in the computer.
<i>In vitro</i>	Studies performed outside a living organism such as in a laboratory.
<i>In vivo</i>	Studies carried out in living organisms.
Inherit	In genetics, receiving genetic material from parents via biological process.
Insertion	Mutation that inserts a residue or stretch of residues into a sequence.
Intron	Non-coding DNA sequence in between exons of a gene. Introns are transcribed into mRNA but then cut out before it is translated into protein.
Kilobase (kb)	Unit of length for DNA fragments or sequences equal to 1,000 nucleotides.
Knockout	Experimental deactivation of a specific gene, used to study gene function.
Last common ancestor (LCA)	In a group of organisms or sequences, the LCA is a hypothetical ancestor in their phylogeny that existed just before they split into separate lineages.
Local alignment	The alignment of some portion of two nucleotide or protein sequences.
Low complexity region (LCR)	Sequence regions with a biased composition including homopolymeric runs or short-period repeats. LCRs are usually masked in homology searches because they may lead to erroneous alignments.
Mapping	Aligning DNA sequencing reads to their region of origin in a reference sequence, generally at the position with the highest sequence similarity.
Mass spectrometer	Instrument to identify chemicals in a sample by their mass and charge. Can be used in proteomics to identify short oligopeptides.
Megabase (Mb)	Unit of length for DNA fragments or sequences equal to 1 million nucleotides, and roughly equal to 1 centimorgan (cM).
Meiosis	Two consecutive cell divisions in diploid sex cell progenitors. Results in four daughter cells, each with a haploid set of chromosomes.
mRNA	<u>M</u> essenger <u>R</u> NA is transcribed from DNA, is template for protein synthesis.
Metagenomics	Study of combined genome sequences of all organisms in an environment.
Microarray	Slide with miniature reaction areas, e.g. used to identify DNA fragments.
Microsatellites	Fast-evolving genomic regions where 2-5 bp motifs are repeated.
Mitosis	Cell division producing two genetically identical daughter cells.
Modular architecture	Design of a system composed of separate components that can be connected. In this course, we highlighted this for computer scripts and protein domains.
Monophyletic	Properties that are present in a single clade in a phylogenetic tree.
Motif	Short conserved region in a sequence, e.g. a conserved part of a domain.
Multifurcation	Point in a phylogeny where one evolutionary lineage splits into three or more daughter lineages.
Multiple alignment	Alignment of three or more sequences such that residues with common structural positions and/or ancestral residues are aligned in one column.
Mutation	Any heritable change in DNA sequence.
Non-coding	Genomic regions that do not contain genes. Sometimes called junk DNA.
Non-synonymous	Nucleotide substitutions that lead to a change of the encoded amino acid.
Nucleotides	A/C/G/T/U. DNA and RNA are nucleotide, phosphate, and sugar polymers.
Oligonucleotide	DNA molecule composed of ~25 nucleotides. "Ολιγο" means "few".

Oligopeptide	Protein molecule composed of ~2-20 amino acids. "Ολιγο" means "few".
Open reading frame (ORF)	Nucleotide sequence located between a start codon and a stop codon, possibly coding for a gene. Very long ORFs are unlikely to occur by chance.
Operon	Set of functionally related genes being regulated and transcribed together.
Optimal alignment	Alignment of two sequences with the highest possible alignment score. The alignment score depends on
Orthologous	Homologous sequences that arose/diverged by a speciation event.
Ortholog conjecture	The idea that orthologous genes have similar, or closely related functions.
P-value	Probability of obtaining at least the measured value by chance.
PAM	<u>P</u> oint <u>A</u> ccepted <u>M</u> utation. Unit to quantify protein evolutionary change, and an amino acid substitution matrix representing this change.
Paralogous	Homologous sequences that arose/diverged by gene duplication.
PCA	<u>P</u> ri <u>n</u> c <u>i</u> p <u>a</u> l <u>C</u> om <u>p</u> o <u>n</u> e <u>n</u> t <u>A</u> n <u>a</u> ly <u>s</u> i <u>s</u> , visualization method for high-dimensional data.
Peptide	Two or more amino acids joined by a peptide bond.
Phage	Virus infecting bacteria (bacteriophage), archaea, or viruses (virophage).
Phenotype	Physical characteristic of an organism that can be the subject of selection during evolution. Special case: presence of a disease.
Phylogeny	Ancestry trace of related genes or organisms, shown as a tree-like graph containing terminal nodes (leaves), internal nodes, and sometimes a root.
Phylogenetic distance	The distance between two nodes in a phylogenetic tree or between two sequences, often expressed in mutations per alignment position.
Plasmid	Autonomously replicating extra-chromosomal circular DNA molecule.
Point mutation	See substitution.
Polymerase	Enzyme that synthesizes a new DNA/RNA strand based on a template.
PCR	<u>P</u> o <u>l</u> y <u>m</u> e <u>r</u> a <u>s</u> e <u>C</u> h <u>a</u> i <u>n</u> <u>R</u> e <u>a</u> c <u>t</u> i <u>o</u> n amplifies DNA between two primer sequences.
Polymorphism	Difference in DNA sequence among individuals, may cause a phenotype.
Polypeptide	Chain of several amino acids joined by peptide bonds.
Polyphyletic	Properties that are present in several different clades in a phylogeny.
Primer	Oligonucleotide molecule that binds to a specific sequence, to which new nucleotides can be added by DNA polymerase.
Probe	Labeled single-stranded oligonucleotide molecule with a specific sequence, used to detect the complementary sequence in a sample.
Profile	Table listing frequencies, like amino acids at each position in a protein alignment (sequence profile) or species in a metagenome (species profile).
Prokaryotes	Single-cell organisms without a nucleus. Bacteria and Archaea.
Promoter	Genomic site where RNA polymerase binds and transcription is initiated.
Protein	Molecule composed of amino acid chains with specific function in the cell.
Proteome	All proteins expressed by a cell, tissue, or organism.
Proteomics	Study involving isolation and characterization of all expressed proteins.
Pseudogene	Non-functional DNA sequence similar to a gene, maybe a gene remnant.
PSI-BLAST	<u>P</u> o <u>s</u> i <u>t</u> i <u>o</u> n- <u>S</u> pe <u>c</u> i <u>f</u> i <u>c</u> <u>I</u> te <u>r</u> ati <u>v</u> e <u>B</u> LA <u>S</u> T is an iterative search using protein BLAST and sequence profiles. More sensitive than normal BLAST.
PSSM	<u>P</u> o <u>s</u> i <u>t</u> i <u>o</u> n- <u>S</u> pe <u>c</u> i <u>f</u> i <u>c</u> <u>S</u> c <u>o</u> r <u>i</u> ng <u>M</u> a <u>t</u> r <u>i</u> x is a sequence profile used in PSI-BLAST.
Purine	Double-ring basic compounds in nucleic acids: adenine and guanine.
Pyrimidine	Single-ring basic compounds in nucleic acids: cytosine, thymine, and uracil.
Query	Input sequence or other type of search term to identify hits in a database.

RAM	<u>R</u> andom <u>A</u> ccess <u>M</u> emory, used in computers to rapidly store/access data.
Read	Short for DNA sequencing read, stretch of sequence that is read from a single molecule of DNA by a sequencing machine.
Recessive gene	Gene whose phenotype is only visible if not overruled by a dominant gene.
Recombination	Process where progeny gets a set of genes different than either parent.
Repetitive DNA	Sequences of varying lengths that occur in multiple copies in the genome.
Reproducibility	Concept that research can be duplicated; one of the main principles of science.
Residue	Nucleic acid or amino acid in a DNA or protein sequence, respectively.
Restriction enzyme	Protein that recognizes specific, short nucleotide sequences and cuts DNA at those sites. Also called endonuclease.
Retrovirus	Viruses that use their recombinant DNA to insert their genetic material into the host chromosome. The virus is then propagated by the host cell.
Reverse complement	DNA sequence that forms a double-stranded structure with another DNA fragment following A-T/C-G. Reverse complement of GTTACA is TGTAAC.
Reverse transcriptase	Enzyme used by retroviruses to form a complementary DNA sequence (cDNA) from their RNA. Also used to create DNA when sequencing RNA.
rRNA	<u>R</u> ibosomal <u>R</u> NA, class of RNA enzymes found in ribosomes.
Ribosomes	Site of protein synthesis in the cell, composed of rRNA and protein.
RNA	<u>R</u> ibo <u>N</u> ucleic <u>A</u> cid is involved in protein synthesis and cellular chemistry.
Root	Special node in a phylogenetic tree containing the ancestor of all lineages.
Rooting	Adding a root node to an unrooted tree, therewith providing the tree with a time axis. Time always points along the branches away from the root.
Sanger, Fred	One of the earliest, still common method of DNA sequencing. Nobel prize.
Sequence	Order of nucleotides or amino acids in a stretch of DNA or protein, resp.
Sequence similarity	Measure indicating how much aligned sequences are alike.
Sequencing	Determining order of nucleotides in DNA/RNA or amino acids in protein.
Sequence logo	Visualization of a sequence profile, shows amino acid abundances and conservation. Conservation quantified in bits using Shannon information.
Shotgun method	Sequencing method targeting many random genomic pieces in parallel.
SNP	<u>S</u> ingle <u>N</u> ucleotide <u>P</u> olymorphism. DNA mutation of 1 changed nucleotide.
Substitution	Sequence mutation consisting of one changed nucleotide or amino acid.
Substitution matrix	Matrix of values indicating the relative ease that amino acid <i>i</i> mutates into amino acid <i>j</i> for all pairs of amino acids.
Synonymous	Nucleotide substitutions that do not change the protein sequence.
Synteny	Genes occurring in the same order on chromosomes of different species.
Taxonomy	Science of classification, description, and nomenclature of organisms.
Telomere	Specialized structure at the end of chromosomes involved in DNA stability.
Thymine (T)	Nucleotide, one member of the base pair AT (adenine-thymine) in DNA.
Transcription	Synthesis of an RNA copy from a DNA gene; first step in gene expression.
Transcription factor	Protein binding to regulatory regions, helps to control gene expression.
Transduction	DNA transfer between a cell and its infecting agent.
Transformation	DNA uptake from the environment into competent species.
Transcriptome	All transcripts (mRNAs) in a particular tissue at a particular time.
tRNA	<u>T</u> ransfer <u>R</u> NA links mRNA codon to amino acid. Has a cloverleaf structure.
Translation	Ribosomal process where mRNA sequence is converted into protein.

Transposable elements	DNA sequences that can move from one chromosomal site to another.
Ultrametricity	Property of a tree where the distance from any point in the tree to all descendant leaves is identical.
Uracil (U)	Nucleotide, one member of the base pair AU (adenine-uracil) in RNA.
Vector	Variable type in R consisting of a list of numbers or a point in multi-dimensions.
Virus	Noncellular biological entity that can only reproduce in a host cell.
Wild type	The genetic form of an organism that occurs most frequently in nature.

11. References

- Altschul, S F, W Gish, W Miller, E W Myers, and D J Lipman. 1990. "Basic Local Alignment Search Tool." *Journal of Molecular ...* 215 (3): 403–10. doi:10.1016/S0022-2836(05)80360-2.
- Altschul, S F, T L Madden, A A Schaffer, J Zhang, Z Zhang, W Miller, and D J Lipman. 1997. "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs." *Nucleic Acids Res* 25 (17): 3389–3402.
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=9254694.
- Andersson, Jan O., W. Ford Doolittle, and Camilla L. Nesbø. 2001. "Are There Bugs in Our Genome?" *Science* 292 (5523).
- Benson, D A, K Clark, I Karsch-Mizrachi, D J Lipman, J Ostell, and E W Sayers. 2014. "GenBank." *Nucleic Acids Research* 42 (1). National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD 20894, USA.: D32-7. doi:10.1093/nar/gkt1030 gkt1030 [pii].
- Burks, C, J W Fickett, W B Goad, M Kanehisa, F I Lewitter, W P Rindone, C D Swindell, C S Tung, and H S Bilofsky. 1985. "The GenBank Nucleic Acid Sequence Database." *Computer Applications in the Biosciences* 1 (4). Theoretical Biology and Biophysics Group, Los Alamos National Laboratory, University of California, NM 87545.: 225–33.
<http://www.ncbi.nlm.nih.gov/pubmed/3880345>.
- Darwin, C. 1859. *The Origin of Species by Means of Natural Selection*. London: Murray.
- Dayhoff, M.O., R.M. Schwartz, and B.C. Orcutt. 1978. "A Model of Evolutionary Change in Proteins." In *Atlas of Protein Sequence and Structure*, 345–52. Washington DC.
- Dobzhansky, Theodosius. 1973. "Nothing in Biology Makes Sense except in the Light of Evolution." *Source: The American Biology Teacher* 35 (3): 125–29.
<http://www.jstor.org/stable/4444260>.
- Doolittle, W F, M Huynen, Berend Snel, P Bork, R S Gupta, B J Soltys, J W Stiller, and B D Hall. 1999. "Lateral Gene Transfer, Genome Surveys, and the Phylogeny of Prokaryotes." *Science* 286: 0.
- Durbin, Richard, Sean Eddy, Anders Krogh, and Graeme Mitchison. 1998. *Biological Sequence Analysis*. Cambridge University Press. <http://www.cambridge.org/gb/academic/subjects/life-sciences/genomics-bioinformatics-and-systems-biology/biological-sequence-analysis-probabilistic-models-proteins-and-nucleic-acids?format=PB&isbn=9780521629713>.
- Dutilh, Bas E., Lennart Backus, Rob A. Edwards, Michiel Wels, Jumamurat R. Bayjanov, and Sacha A F T van Hijum. 2013. "Explaining Microbial Phenotypes on a Genomic Scale: GWAS for Microbes." *Brief Funct Genomics* 12 (4). CMBI, NCMLS, Radboud University Medical Centre. Geert Grooteplein 28, 6525 GA Nijmegen, The Netherlands. dutilh@cmbi.ru.nl: 366–80. doi:10.1093/bfpg/elt008 elt008 [pii].
- Eddy, Sean R. 2004a. "What Is Dynamic Programming?" *Nat Biotech* 22 (7): 909–10.
- . 2004b. "Where Did the BLOSUM62 Alignment Score Matrix Come From?" *Nature Biotechnology* 22 (8): 1035–36. doi:10.1038/nbt0804-1035.
- Field, D, G Garrity, T Gray, N Morrison, J Selengut, P Sterk, T Tatusova, et al. 2008. "The Minimum Information about a Genome Sequence (MIGS) Specification." *Nat Biotechnol* 26 (5). Natural Environmental Research Council Centre for Ecology and Hydrology, Oxford OX1 3SR, UK. dfield@ceh.ac.uk: 541–47. doi:nbt1360 [pii] 10.1038/nbt1360.
- Fitch, W. M. 1970. "Distinguishing Homologous from Analogous Proteins" 19 (2): 99–113.
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=54147.

ist_uids=5449325.

Henikoff, S, and J G Henikoff. 1992. "Amino-Acid Substitution Matrices from Protein Blocks." *Proceedings of the National Academy of Sciences of the United States of America* 89 (22).

Henikoff, S Fred Hutchinson Canc Res Ctr, Howard Hughes Med Inst, Div Basic Sci, Seattle, Wa 98104: 10915–19.

Hesper, B, and Paulien Hogeweg. 1970. "Bioinformatica: Een Werkconcept." *Kameleon* 1 (6): 28–29.

Hey, Tony, Stewart Tansley, and Kristin Tolle. 2009. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research.

<https://www.amazon.com/Fourth-Paradigm-Data-Intensive-Scientific-Discovery/dp/0982544200>.

Higgs, Paul G, and Teresa K Attwood. 2005. *Bioinformatics and Molecular Evolution*. Blackwell Science.

Hogeweg, Paulien, and B Hesper. 1984. "The Alignment of Sets of Sequences and the Construction of Phyletic Trees: An Integrated Method." *J Mol Evol* 20 (2): 175–86.

http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=6433036.

Hug, Laura A, Brett J Baker, Karthik Anantharaman, Christopher T. Brown, Alexander J Probst, Cindy J Castelle, Cristina N Butterfield, et al. 2016. "A New View of the Tree and Life's Diversity." *Nature Microbiology* 1 (April): 16048. doi:10.1038/nmicrobiol.2016.48.

Hughes, A L, J da Silva, and R Friedman. 2001. "Ancient Genome Duplications Did Not Structure the Human Hox-Bearing Chromosomes." *Genome Research* 11 (5). Cold Spring Harbor Laboratory Press: 771–80. doi:10.1101/gr.160001.

Huising, Mark O., Corine P. Kruiswijk, Jessica E. van Schijndel, Huub F. J. Savelkoul, Gert Flik, and B. M. Lidy Verburg-van Kemenade. 2005. "Multiple and Highly Divergent IL-11 Genes in Teleost Fish." *Immunogenetics* 57 (6). Springer-Verlag: 432–43. doi:10.1007/s00251-005-0012-2.

Huttenhower, Curtis, D Gevers, Rob Knight, Sahar Abubucker, Jonathan H. Badger, Asif T. Chinwalla, Heather H. Creasy, et al. 2012. "Structure, Function and Diversity of the Healthy Human Microbiome." *Nature* 486 (7402). Nature Publishing Group: 207–14. doi:10.1038/nature11234 nature11234 [pii].

Jukes, Thomas H. 1969. "How Many Nucleotide Substitutions Actually Took Place?" In *Mammalian Protein Metabolism*, edited by Hamish N Munro, III, 21–132. New York: Space Sciences Laboratory.

Kerfeld, C A, and K M Scott. 2011. "Using BLAST to teach 'E-Value-Tionary' concepts." *PLoS Biol* 9 (2). Joint Genome Institute, Walnut Creek, California, United States of America. CKerfeld@lbl.gov: e1001014. doi:10.1371/journal.pbio.1001014.

Lesk, A M. 2002. *Introduction to Bioinformatics*. Oxford New York: Oxford University Press.

Needleman, Saul B., and Christian D. Wunsch. 1970. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins." *Journal of Molecular Biology* 48 (3). Academic Press: 443–53. doi:10.1016/0022-2836(70)90057-4.

Quackenbush, John. 2001. "Computational Analysis of Microarray Data." *Nature Reviews Genetics* 2 (6). Nature Publishing Group: 418–27. doi:10.1038/35076576.

Sanger, F, and A R Coulson. 1975. "A Rapid Method for Determining Sequences in DNA by Primed Synthesis with DNA Polymerase." *J Mol Biol* 94 (3): 441–48.

http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=1100841.

- Schneider, Thomas D., and R. Michael Stephens. 1990. "Sequence Logos: A New Way to Display Consensus Sequences." *Nucleic Acids Research* 18 (20). Oxford University Press: 6097–6100. doi:10.1093/nar/18.20.6097.
- Schnell, Ida Bærholm, Philip Francis Thomsen, Nicholas Wilkinson, Morten Rasmussen, Lars R D Jensen, Eske Willerslev, Mads F Bertelsen, et al. 2012. "Screening Mammal Biodiversity Using DNA from Leeches." *Current Biology : CB* 22 (8). Elsevier: R262-3. doi:10.1016/j.cub.2012.02.058.
- Shannon, Claude E. 1948. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27 (July 1928): 379-423-656. doi:10.1145/584091.584093.
- Smith, T F, and M S Waterman. 1981. "Identification of Common Molecular Subsequences." *J Mol Biol* 147: 195–97.
- Snel, Berend, Peer Bork, and Martijn A Huynen. 2002. "Genomes in Flux: The Evolution of Archaeal and Proteobacterial Gene Content." *Genome Res* 12 (1): 17–25. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=11779827.
- Snel, Berend, G Lehmann, P Bork, and M A Huynen. 2000. "STRING: A Web-Server to Retrieve and Display the Repeatedly Occurring Neighbourhood of a Gene." *Nucleic Acids Res* 28 (18): 3442–44. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=10982861.
- Spang, Anja, Jimmy H Saw, Steffen L Jørgensen, Katarzyna Zaremba-Niedzwiedzka, Joran Martijn, Anders E Lind, Roel van Eijk, Christa Schleper, Lionel Guy, and Thijs J G Ettema. 2015. "Complex Archaea That Bridge the Gap between Prokaryotes and Eukaryotes." *Nature* 521 (7551): 173–79. doi:10.1038/nature14447.
- Tatusov, R L, N D Fedorova, J D Jackson, A R Jacobs, B Kiryutin, Eugene V Koonin, D M Krylov, et al. 2003. "The COG Database: An Updated Version Includes Eukaryotes." *BMC Bioinformatics* 4 (1): 41. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12969510.
- van Noort, Vera, Berend Snel, and M A Huynen. 2003. "Predicting Gene Function by Conserved Co-Expression." *Trends Genet* 19 (5): 238–42. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12711213.
- Vingron, Martin, and Michael S. Waterman. 1994. "Sequence Alignment and Penalty Choice: Review of Concepts, Case Studies and Implications." *Journal of Molecular Biology* 235 (1). Academic Press: 1–12. doi:10.1016/S0022-2836(05)80006-3.
- Woese, C R, and G E Fox. 1977. "Phylogenetic Structure of the Prokaryotic Domain: The Primary Kingdoms." *Proc Natl Acad Sci U S A* 74 (11): 5088–90. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=270744.
- Zou, Jun, Norman F. Neumann, Jason W. Holland, Miodrag Belosevic, Charles Cunningham, Christopher J. Secombes, and Andrew F. Rowley. 1999. "Fish Macrophages Express a Cyclo-Oxygenase-2 Homologue after Activation." *Biochemical Journal* 340 (1).
- Zuckerkandl, Emile, and Linus Pauling. 1962. "Molecular Disease, Evolution, and Genic Heterogeneity." *Horizons in Biochemistry*. doi:10.1016/S0140-6736(54)91105-8.
- Zvelebil, Marketa, and Jeremy O. Baum. 2007. *Understanding Bioinformatics*. 1st ed. Garland Science. <http://www.garlandscience.com/product/isbn/9780815340249>.

12. Answers to the questions

12.1. Answers to questions in Section 1.9

1: D. 2: B. 3: B (note that while searching Google might also work, it is not a bioinformatical approach). 4: C. 5: D. 6: B.

7. From short to long: Illumina (50-400 bp), Ion Torrent (up to ~500 bp), Sanger (up to ~1,500 bp), Nanopore (up to ~100,000 bp). Thus, single Nanopore reads could be sufficient to sequence some viral genomes in a single read. Note, however, that currently raw Nanopore reads still have quite high error rates (about 1 in 10 nucleotides). These errors can be corrected by reading the same sequence multiple times and comparing the sequences to each other. Generally, if you sequence a genome with any DNA sequencing platform you will have to use sequence assembly to combine the reads into longer contiguous sequences (contigs).
8. B.
9. D and B (depending on the sequencing technology).
10. No. Sequence identifiers should be unique in a Fasta file, otherwise we cannot distinguish the different sequences. The ID is the part that follows the greater-than sign ">", and before the first whitespace character (space, tab, or newline). In the displayed fragment, all the IDs are "seq". Use underscores "_" or periods "." instead of spaces to make them unique.
11. a. A Google search reveals this answer:
 - RefSeq: GenBank sequences that are manually curated by the NCBI staff. RefSeq records are owned by NCBI and can be updated as needed to maintain current annotation or to incorporate additional information.
 - GenBank/GenPept: unreviewed sequences submitted from individual laboratories and large-scale sequencing projects. Since these sequence records are owned by the original submitters and cannot be altered, GenBank might contain many low-quality sequences.

GenBank includes publicly available DNA sequences submitted from individual laboratories and large-scale sequencing projects. GenBank is part of the International Nucleotide Sequence Database Collaboration (INSDC) along with the European Nucleotide Archive and the DNA Data Bank of Japan (DDBJ). Submitted sequence data is exchanged daily between the three collaborators to achieve comprehensive worldwide coverage. GenBank can be very redundant, meaning that it may contain many copies of the same sequence. GenBank sequence records are owned by the original submitter and cannot be altered by a third party.

RefSeq sequences are not part of the INSDC but are derived from INSDC sequences to provide non-redundant curated data representing our current knowledge of known genes. Some records include sequence information gathered from more than one INSDC record. Records may include sequence, descriptive information, publications, or feature annotation that is not available from any single INSDC record. RefSeq

records are owned by NCBI and therefore can be updated as needed to maintain current annotation or to incorporate additional information.

See: <https://www.ncbi.nlm.nih.gov/books/NBK50679> for more FAQs.

- b. Draft: a good start. It has holes where the sequence isn't known, much of the remainder is low-quality. Good for looking around and seeing how a Neanderthal is like a human, or getting an idea of the genes that are present in a bacterium. Not suitable to determine which genes are absent from a genome! Complete: a good accomplishment, it has relatively few errors. May still have some holes, which are mostly in hard-to-sequence regions. Good for detailed analyses and determining gene absence. See: <https://www.quora.com/What-are-the-differences-between-complete-genome-sequence-and-draft-genome-sequence> for a further discussion of this topic. There are different levels of draftness/completion.

12. See table below for the taxonomy (i). This information is available from the NCBI Taxonomy database at <http://www.ncbi.nlm.nih.gov/Taxonomy>.

Species	Genus	Family	Order	Class	Phylum	Super-kingdom
Human papillomavirus	-	<i>Papillomaviridae</i>	-	-	-	Viruses
<i>F. nucleatum</i>	<i>Fusobacterium</i>	<i>Fusobacteriaceae</i>	<i>Fusobacteriales</i>	<i>Fusibacteriia</i>	<i>Fusobacteria</i>	<i>Bacteria</i>
<i>A. thaliana</i>	<i>Arabidopsis</i>	<i>Brassicaceae</i>	<i>Brassicales</i>		<i>Streptophyta</i>	<i>Eukaryota</i>
<i>D. melanogaster</i>	<i>Drosophila</i>	<i>Drosophilidae</i>	<i>Diptera</i>	<i>Insecta</i>	<i>Arthropoda</i>	<i>Eukaryota</i>
<i>W. pipientis</i>	<i>Wolbachia</i>	<i>Anaplasmataceae</i>	<i>Rickettsiales</i>	<i>Alphaproteobacteria</i>	<i>Proteobacteria</i>	<i>Bacteria</i>
<i>S. cerevisiae</i>	<i>Saccaromyces</i>	<i>Saccharomycetaceae</i>	<i>Saccharomycetales</i>	<i>Saccharomycetes</i>	<i>Ascomycota</i>	<i>Eukaryota</i>
<i>H. sapiens</i>	<i>Homo</i>	<i>Hominidae</i>	<i>Primates</i>	<i>Mammalia</i>	<i>Chordata</i>	<i>Eukaryota</i>

See table below for the other questions (ii-vi). This information is available from different sources, for example from the NCBI Genome database at <http://www.ncbi.nlm.nih.gov/genomes>, but sometimes also from specialized genome database websites. Note that the exact genome size in base pairs depends on the build version of the genome sequence, and this can differ as new versions are released. This also holds for the number of genes: this is a number that shifts in time as new discoveries are made about the genes on a given genome. Thus, in the last column (Missing genes?): it is always possible that genes were missed during gene annotation, but this is of course less likely with completely sequenced genomes than with draft genomes. Shorter genes are more likely to be missed than longer genes, and it is more likely that genes are missed in larger genomes than in shorter genomes.

Species	Genome length	Chromosomes	Complete?	Genes	Missing?
Human papillomavirus	7,320 bp	1	Yes	6	No
<i>F. nucleatum</i>	2,268,272 bp	1	Yes	2,182	Maybe
<i>A. thaliana</i>	119,146,348 bp	5 + chloroplast + mitochondrion	Yes	38,125	Probably
<i>D. melanogaster</i>	137,547,960 bp	4 pairs + mito	No	17,682	Probably
<i>W. pipientis</i>	1,201,350 bp	2	Yes	1,115	Maybe
<i>S. cerevisiae</i>	12,156,677 bp	16 pairs + mito	Yes	6,353	Probably
<i>H. sapiens</i>	3,088,269,832 bp	23 pairs + mito	No	59,911	Probably

- 13.a. Let's assume that I do not want to go all the way to a pristine coral reef to collect samples myself (or I simply do not have the money). I could perform this whole study just sitting at my desk in Utrecht by using datasets that others have already collected (data recycling). I would use metagenomics to answer this question. A metagenomic dataset contains nucleotide sequence information from a whole microbial environment at once, so you could answer this question by looking for antibiotics resistance genes in metagenomes from a pristine coral reef. First, I would look for metagenomic datasets from pristine coral reefs. MG-RAST (<http://metagenomics.anl.gov>) or EBI metagenomics (<https://www.ebi.ac.uk/metagenomics>) are large metagenomics databases, the first contains 180 coral reef metagenomes, the second 42 metagenomes from three projects. Next, I would find a database of antibiotics resistance genes, e.g. http://www.broadinstitute.org/annotation/genome/escherichia_antibiotic_resistance, <https://card.mcmaster.ca>, or <http://ardb.cbcb.umd.edu>, and you could probably find more if you looked hard enough. Then I would search the metagenomic DNA sequencing reads for cases that are similar to the antibiotics resistance genes in the database (see Chapter 8). This would allow me to determine if any resistance genes are convincingly present in the pristine coral reef samples.
- b. Browse the websites of the different databases and try to find relevant information. The number of times a paper is cited can be found via Google Scholar. In February 2017, I could find the information in the table below.

	<i>E. coli</i> abx database	CARD	ARDB
# genes	Limited to <i>E. coli</i>	2374 seqs, 902 SNPs	23137 genes
Updated in	Data from 2010?	December 2016	July 2009
Times cited	No publication?	2017 paper: 2x; 2015 paper: 15x; 2010 paper: 259x cited	2009 paper: 412x cited

Thus, the *E. coli* database seems less useful for our purpose. It seems that ARDB contains more genes than CARD, and it is also cited more often, so it is probably the best database to use. What does worry me a bit though, is that the CARD database is newer than ARDB: the paper was published later, and the database was updated more recently. So it will be worth reading the CARD paper to see why they published a newer antibiotics database after CARD was already there, that is smaller. Maybe they explain why ARDB is not as good or something. We will skip that for this exercise though.

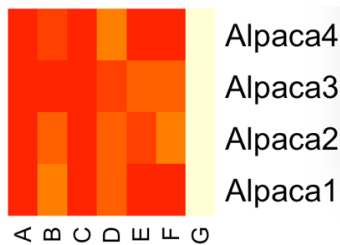
- c. Let's take a look if other people already did this study, by searching some relevant keywords in a literature database like Pubmed or Google Scholar. A search like "pristine coral reef antibiotics resistance metagenomics" gets relatively few relevant hits, perhaps because the "coral reef" in the search terms is too specific. If we remove that and search for "antibiotics resistance metagenomics", we can find quite a few studies that suggest antibiotics resistance genes are present in a range of environments, including soils, waters, and even corals, as found by Wegley et al. 2007 "Metagenomic analysis of the microbial community associated with the coral *Porites astreoides*" (Environ. Microbiol. 9, 2707-2719): "Several genes for the resistance of antibiotics and toxic compounds were observed (n = 34), specifically resistance to fluoroquinolones (n = 13) and toxic compounds cobalt, zinc and cadmium (n = 19)." Based on this abundance of antibiotics

resistance genes in many environments, I would expect that we would probably also find some in pristine coral reefs.

12.2. Answers to questions in Section 3.9

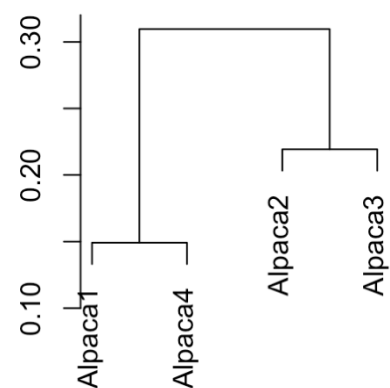
1.
 - a. The data in Figure 13 has three dimensions. The right panel shows the first two principal components.
 - b. Figure 14 shows the first two principal components of the microbial community profiles of hundreds of human body site samples from different individuals. The profiles are similar to the stacked bars in Figure 12a (but at a lower taxonomic level, e.g. species or genera instead of phyla). If the abundances of N types of microbes were measured in each sample, the complete dataset had N dimensions. The first two PCs reflect two of these dimensions, i.e. the two that give the maximal separation of the data.
 - c. The samples from the nose and skin appear mixed in the graph. From the data perspective, this suggests that the skin microbiomes in some samples are more similar to nose samples than to other skin samples, and *vice versa*. A biological interpretation could be that the skin and nose samples are both taken from directly air-exposed surfaces, and perhaps the microbiomes of these surfaces are more influenced by microbes that land on us from the environment than the gastrointestinal, urogenital, and oral microbiomes.
 - d. The fact that some skin samples are more similar to nose samples than to other skin samples in the first two PCs does not mean that they could not be distinguished in any of the following dimensions (PC 3- N). The first two principal components show the overall maximal separation of the data, but they might hide subtle differences. You would have to go through all the PCs (dimensions) to check if nose and skin separate out in any of them.
2.
 - a. Manhattan distance or Euclidean distance is suitable to cluster vectors with a similar magnitude.
 - b. If two vectors have a very different magnitude, you could normalize the values by dividing each value by the sum of all values. This scales the values such, that the sum of all values in each sample becomes equal to 1. Alternative ways of scaling include dividing each value by the maximum value (this scales the maximum value to 1 and the remaining values linearly between 0 and 1), or taking the logarithm of each value (this normalizes values with an exponentially skewed distribution).
 - c. Correlation, e.g. Pearson or Spearman correlation is suitable to cluster vectors with a similar profile. Note that correlation yields a similarity measure between 1 (same profile) and -1 (inverse profile), so that needs to be converted to a distance between 0 (identical) to 1 (different).
 - d. To convert correlation r to distance D , you can use the formula $D = 0.5 - r / 2$.
3.
 - c. After loading the data from the file into the data matrix **alp**, you can view it by simply typing **alp**. There are four rows and seven columns (excluding the headers).
 - d. Type **rowSums(alp)**. Alpaca1 has the most reads sequenced (1,428,177 reads). This tells you nothing about the alpaca – it just means that for this sample, more DNA was used for sequencing than for the others.

- e. Alpaca1 and Alpaca4 seem to have similar microbiomes because they are both high in B, D, and G, but low in A, C, E, and F. However, this is just based on eyeballing the heatmap, so we need to quantify these distances to know for sure.



- f. This command scales the values in the matrix by dividing each value by the sum of the values in that row. In effect, it normalizes the values to an equal amount of data for each alpaca: typing `rowSums(alp_norm)` shows that the values for each alpaca now add up to 1.
- g. The new heatmap is almost identical to the one in e, because the `rowSums(alp)` values were already quite similar, so similar that you cannot observe the difference in the color range between red (lowest value; `alp`: 876; `alp_norm`: 0.0006133694) and very light (highest value; `alp`: 936,890; `alp_norm`: 0.6727784).
- h. The smallest value in the distance matrix `min(deuc)` is 0.1491943 between Alpaca1 and Alpaca4. The Euclidean distance between the normalized microbiomes of Alpaca1 and Alpaca3 is 0.2500360. Using Equation 2, we can confirm this: $D_{A1A3} = \sqrt{((0.008601175 - 0.013566745)^2 + (0.21986420 - 0.04331009)^2 + (0.0009102513 - 0.0429988582)^2 + (0.10910412 - 0.06530074)^2 + (0.004902754 - 0.122372568)^2 + (0.0006133694 - 0.1068690554)^2 + (0.6560041 - 0.6055819)^2)} = \sqrt{0.06251802} = 0.250036$.
- i. A meaningful name might be `dman`. Type:
- ```
dman <- dist(alp_norm, method='manhattan')
```
- $$D_{A1A3} = |0.013566745 - 0.008601175| + |0.21986420 - 0.04331009| + |0.0009102513 - 0.0429988582| + |0.10910412 - 0.06530074| + |0.004902754 - 0.122372568| + |0.0006133694 - 0.1068690554| + |0.6560041 - 0.6055819| = 0.5415594.$$
- j. Yes, they are. You can easily compare the entire matrices at once with the command:
- ```
deuc <- dman
```

- k. `dpear` contains the distances between microbiome profiles as calculated from the Pearson correlation values, using the formula $D = 0.5 - r/2$ (see exercise 2d).
- l. Typing: `calp <- hclust(deuc)` followed by `plot(calp)` displays the clustering displayed to the right. As a bracket-notation, this can be written as: `((Alpaca1, Alpaca4), (Alpaca2, Alpaca3))`;
- m. Alpaca1 and Alpaca4 have most similar microbiomes. This is consistent with our answer under e, and with the distance matrices under h. A biological hypothesis may



be that Alpaca1 and Alpaca4 have a similar diet, live close together at the same zoo (cross-inoculation), or belong to the same alpaca subspecies.

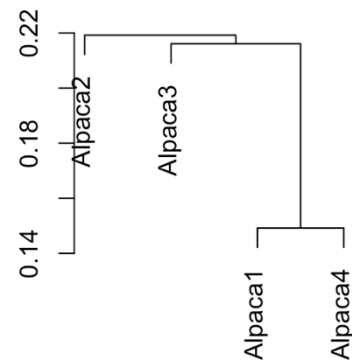
- n. Typing **str(calp)** reveals that variable **calp** contains 7 items. Typing **calp\$height** lists the branch lengths of the clustering: 0.1491943, 0.2192003, and 0.3096554. We can confirm the first value 0.1491943, by looking at the distance matrix **deuc** – it is the smallest distance, i.e. between Alpaca1 and Alpaca4. Next, we have to create a new distance matrix where Alpaca1 and Alpaca4 are now in one cluster. The distance of the cluster to other nodes is determined as follows (by default, **hclust** uses complete linkage, see Figure 17):

	(Alpaca1,Alpaca4)	Alpaca2
Alpaca2	$\max(0.3025191, 0.3096554) = 0.3096554$	
Alpaca3	$\max(0.2500360, 0.2160898) = 0.2500360$	0.2192003

The next lowest value is 0.2192003, which joins Alpaca2 and Alpaca3 in a cluster.

- o. Typing **?hclust** shows the help page, where you can see that the command for single linkage clustering should be: **slalp <- hclust(deuc, method="single")**. You can plot this new clustering with **plot(slalp)**. The clustering has changed (right). Note that in this example, single linkage leads to more chained clustering that complete linkage (compare to Figure 17E and D).

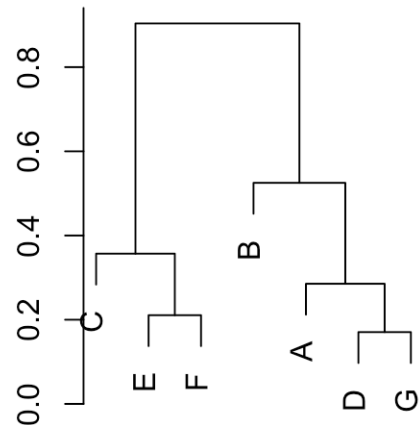
We can see the new branch lengths by typing **slalp\$height**. The first branch length is still 0.1491943, i.e. the smallest distance in the distance matrix **deuc**. However, the new distance matrix should now be calculated using single linkage, so the next lowest value is now 0.2160898.



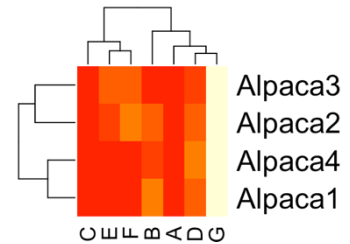
	(Alpaca1,Alpaca4)	Alpaca2
Alpaca2	$\min(0.3025191, 0.3096554) = 0.3025191$	
Alpaca3	$\min(0.2500360, 0.2160898) = 0.2160898$	0.2192003

- p. **alpt <- t (alp)** – transposes the matrix so that rows are columns and *vice versa*, and places the new matrix in **alpt**. Verify this by typing **alpt**.
alpt_norm <- alpt / rowSums (alpt) – this is the same normalization as under f above. It scales the values in **alpt** by dividing every value by the sum of the values in that row and places the resulting data matrix in **alpt_norm**. Verify this by typing **alpt_norm** and **rowSums(alpt_norm)**.

`dalpt_norm <- dist (alpt_norm, method='euclidean')` – calculates the Euclidean distance matrix between all rows (now: bacteria) and places it in the data matrix `dalpt_norm`. Verify this by typing `dalpt_norm`.
`cbac <- hclust (dalpt_norm, method='complete')` – clusters the distance matrix in `dalpt_norm` using complete linkage and places it in the data matrix `dalpt_norm.hclust` uses complete linkage by default, so “, `method='complete'`” could also be left out.
`plot (cbac)` – plots the clustering of bacteria according to their occurrence profiles across alpacas. See the output clustering figure to the right.



- q. If two species of bacteria D and G cluster together, this means that the Euclidean distances between their normalized abundance profiles across alpacas is low. This means that their relative abundances in alpaca poo samples are always very similar (see Equation 2). Note that although the relative abundance profiles are similar, the absolute abundances are very different: G is very abundant while D is much rarer (see the heatmap under e), but these differences were eliminated by normalizing. The Dutch botanist and microbiologist Lourens Baas Becking (1895-1963) said “Alles is overal, maar het milieu selecteert” (everything is everywhere, but the environment selects). Whether or not it is true that everything is everywhere (this seems pretty unlikely), the availability of nutrients certainly determines which bacteria can grow in a given environment. So, if two species of bacteria D and G always occur together across different alpaca poos, this could mean that those bacteria use the same nutrients, and those nutrients are only available in some alpacas. Another possible explanation is that these bacteria depend on nutrients supplied by each other, instead of on the same external nutrient source. This is also known as a cross-feeding dependency.
- r. The rows of the bi-clustered heatmap are sorted according to the alpaca clustering as in exercise l, the columns are sorted according to the bacterial clustering as in exercise p.



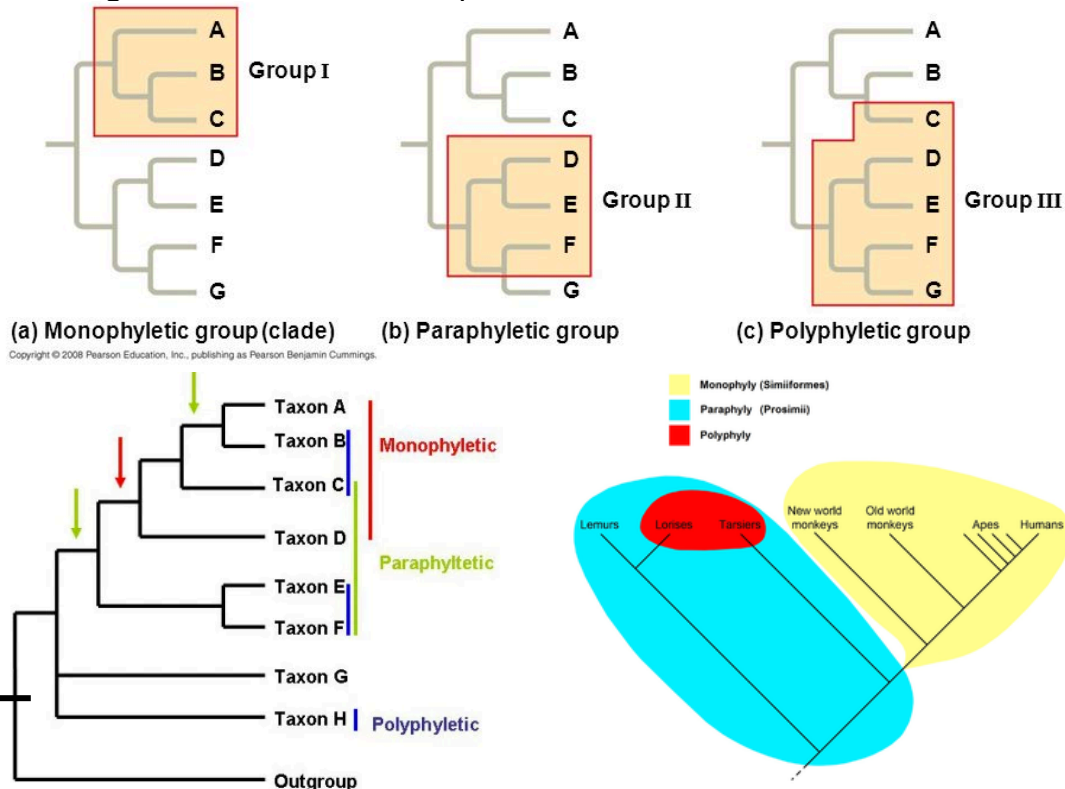
4. a. Sequencing of the total transcribed RNA in a sample.
- b. There are 25 cancer samples included in this study (columns). Note that it is unlikely that the rows are the cancer samples and the columns are the genes, because transcriptomics will certainly identify more than 25 genes.
- c. Tumors were sampled from patients and total transcribed RNA was extracted. RNA was reverse-transcribed to DNA. DNA was sequenced using second generation sequencing (Illumina, Ion Torrent). Sequencing reads were compared to the human genome to identify which genes were expressed (see Chapter 8). Tumors and genes were both clustered after normalizing their abundances, and the results displayed as a bi-clustering heatmap.
- d. The three that are most to the left are the most similar because the clustering branch lengths are the shortest for these samples.

- e. If two genes (rows) cluster together, this means that they tend to be co-expressed, i.e. their expression is always high or low together. Perhaps those genes are part of the same pathway, or otherwise functionally related in some way.

12.3. Answers to questions in Section 4.10

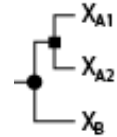
1: D. 2: D. 3: E. 4: D. 5: C. 6: A. 7: B.

8. a. See figures below for some examples.



- (i) A monophyletic group consists of all the lineages that descend from a given ancestor.
- (ii) A paraphyletic group consists of an incomplete subset of the lineages that descend from a given ancestor, who was also part of the group.
- (iii) A polyphyletic group consists two or more lineages whose last common ancestor was not part of the group.
- b. All internal nodes give rise to a monophyletic group. A monophyletic group is defined as all the lineages that descend from a given internal node.
9. a. Rooted: (iii) and (iv) would be identical.
- a. Unrooted: all trees would be identical. You can see this by drawing the trees in a star shape.
10. B.

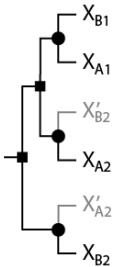
11. a. See the tree to the right. Yes, the rule of thumb holds.



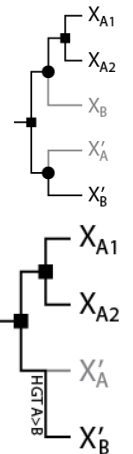
- b. The rule of thumb could hold. To explain this tree, let's compare it to the tree you drew under a. Gene X_{B2} corresponds to gene X_B , but to explain gene X_{B1} we could assume a HGT event happened where X_{A1} was transferred to species B, leading to gene X_{B1} .

Alternatively, we could assume that X_{A2} and X_{B2} are actually orthologs of two additional genes X'_{B2} and X'_{A2} that were independently lost (see figure).

It requires additional evidence or knowledge of the biological system to decide which of these scenarios is more likely. (Note that more complicated evolutionary scenarios are possible, but we tend to believe the simplest explanation according to Occam's razor, see Section 9.4.)



- c. The function transfer $X_{A1} \rightarrow X_{B1}$ seems reasonable, since these genes have recently diverged and occur in different genomes. But the $X_{A2} \rightarrow X_{B2}$ prediction is dangerous. X_{A1} and X_{A2} diverged by gene duplication, and their functions may have diverged. Based on the phylogenetic tree, it is not possible to say which of the genes X_{A1} or X_{A2} is functionally equivalent to X_{B2} .
- d. If the oldest node indicates a gene duplication event in the LCA of species A and B, this means that the LCA contained two genes X and X' when it speciated into species A and B. If the present-day observation is the tree you drew in exercise a, i.e. there are two copies X_{A1} and X_{A2} in species A and one copy X_B in species B, the simplest explanation is that species B lost ancestral gene X, and species A lost ancestral gene X'. See the tree to the right.
- e. If the oldest node in the tree was a gene duplication event in species A, this means that there were two gene duplications in species A, so you would expect three copies of gene X in species A. We observe only X_{A1} and X_{A2} , so at least one gene, X'_A was lost from species A. Since X could have been invented by species A after it diverged from species B, the gene must have made it into the genome of species B via a HGT event before X'_A was lost, as shown in the tree to the right.

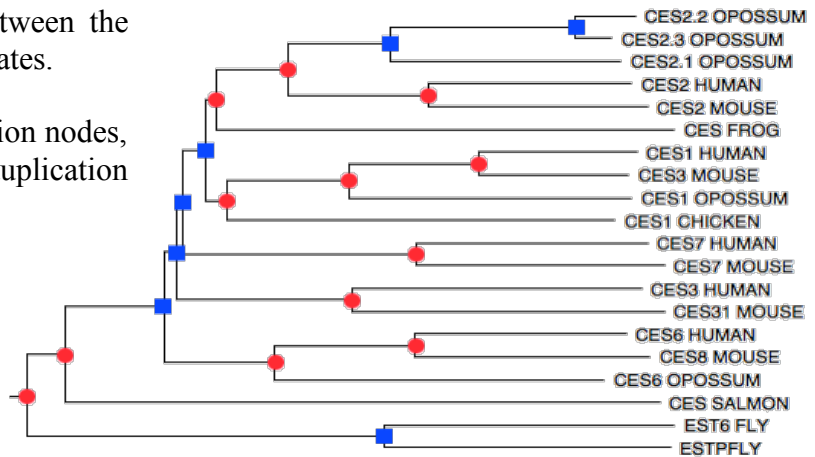


12. Paralogs arose due to a gene duplication event, and thus the idea is that two very similar genes co-existed within the same genome for some time. During that time, it is likely that some aspect of their function diverged, like the tissue where the gene is expressed, its metabolic specificity, or perhaps the catalytic function. After all, why would an organism keep two genes with 100% identical functions in its genome?* Thus, it is dangerous to transfer a gene function between paralogs, or even from an ortholog in another species to one of two co-paralogs (this is the situation as described in this question and in exercise 11c). In contrast, orthologs arose due to a speciation event, and although the two orthologous genes are now present in different genomes, it is likely that their function remained much the same.

* In some cases, multiple identical genes are indeed kept in the same genome. This is thought to be for dosage effects: more genes lead to more transcription and translation, and thus more protein to perform an important function.

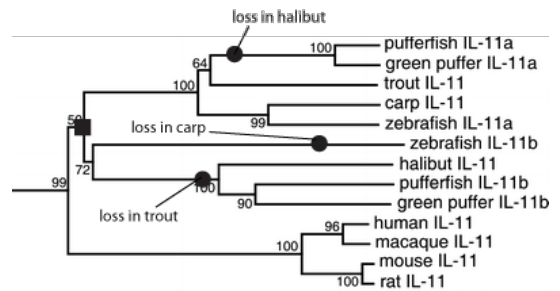
13. a. The root is placed between the insects and the vertebrates.

Red circles are speciation nodes, blue squares are gene duplication nodes.



b. Species	c. Gene	d. How did it get there?
Chicken	CES1 CHICKEN	Duplicated as paralog in <i>Bilateria</i> ancestor, all paralogs lost.
Fly	EST6 FLY	Duplicated as paralog of ESTPFLY in fly ancestor.
	ESTPFLY	Duplicated as paralog of EST6 FLY in fly ancestor.
Frog	CES FROG	Duplicated as paralog in <i>Bilateria</i> ancestor, all paralogs lost.
Human	CES1 HUMAN	Duplicated as paralog of CES2 in <i>Tetrapoda</i> ancestor.
	CES2 HUMAN	Duplicated as paralog of CES1 in <i>Tetrapoda</i> ancestor.
	CES3 HUMAN	Duplicated as paralog of CES1/2/7 in <i>Tetrapoda</i> ancestor.
	CES6 HUMAN	Duplicated as paralog of CES1/2/7/3 in <i>Tetrapoda</i> ancestor.
	CES7 HUMAN	Duplicated as paralog of CES1/2 in <i>Tetrapoda</i> ancestor.
Mouse	CES2 MOUSE	Duplicated as paralog of CES1 in <i>Tetrapoda</i> ancestor.
	CES3 MOUSE	Duplicated as paralog of CES2 in <i>Tetrapoda</i> ancestor.
	CES7 MOUSE	Duplicated as paralog of CES1/2 in <i>Tetrapoda</i> ancestor.
	CES8 MOUSE	Duplicated as paralog of CES1/2/3/7 in <i>Tetrapoda</i> ancestor.
	CES31 MOUSE	Duplicated as paralog of CES1/2/7 in <i>Tetrapoda</i> ancestor.
Opossum	CES1 OPOSSUM	Duplicated as paralog of CES2 in <i>Tetrapoda</i> ancestor.
	CES2.1 OPOSSUM	Duplicated as paralog of CES2.2/2.3 in <i>Opossum</i> ancestor.
	CES2.2 OPOSSUM	Duplicated as paralog of CES2.3 in <i>Opossum</i> ancestor.
	CES2.3 OPOSSUM	Duplicated as paralog of CES2.2 in <i>Opossum</i> ancestor.
	CES6 OPOSSUM	Duplicated as paralog of CES1/2/7/3 in <i>Tetrapoda</i> ancestor.
Salmon	CES SALMON	Has been in salmon since <i>Bilateria</i> ancestor.

14. The mammals have only one IL-11, but almost all fish species have both IL11a and IL11b. Thus, the common ancestor of all fish probably obtained two copies of this gene as a result of a gene duplication. However, trout, carp (both missing IL-11b), and halibut (missing IL-11a) have only one IL-11. The simplest explanation is that those fish have lost one of the IL-11 genes. This gene duplication did not occur in mammals. In the figure above, the gene duplication is indicated with a square, and the deletions are indicated with circles.



12.4. Answers to questions in Section 5.8

1. A (see Section 5.6).
2. E.
3. B (see Figure 38).
4. B.
5. A or E.
6. C.
7. D. Different breeds of dogs are very closely related, so you need poorly conserved sequences to distinguish them.
8. a. Rate = 10^{-9} per base pair per year. Fraction = rate \times time = $10^{-9} \times 10^7 = 10^{-2} = 1\%$. However, because we are analyzing the difference between two sequences, we can consider the time of divergence to be the "double" (both sequences are evolving), so we multiply the last answer by two, and thus, expect a difference of 2%.
b. If some sites in this specific region were more difficult to mutate, their mutation rate would be $<10^{-9}$ per base pair per year for those sites, while it stays the same for the other sites. Altogether, the fraction of sites that differ would thus decrease.
c. For each residue, there is a chance of 10^{-9} of a mutation happening in one year, so the probability that one site will not change is $1 - 10^{-9}$. Because the sequence is 1,000 bp long, we raise these probabilities to the exponent of 1,000: $(1 - 10^{-9})^{1,000} = 0.999999$. This means that in one year, there is a 99.9999% chance that the sequence does not change.
d. We have to multiply the above answer by itself the number of years which pass: (chance of staying the same in one year)^{number of years} $\rightarrow (0.999999)^{10,000,000} = 4.539 \times 10^{-5}$.
e. The probability that both sequences are the same after 10^7 years is: $4.539 \times 10^{-5} \times 4.539 \times 10^{-5} = 2.06 \times 10^{-9}$ which is even much more unlikely than finding just one of the sequences unchanged!
9. a. Positions 1, 2, 7, 8, 11.
b. The 3rd base pair in a codon is often redundant, i.e., different nucleotides in the 3rd base pair would still code for the same amino acid. Therefore, the conservation is low in the 3rd position. Variants of a codon on the 1st and 2nd position mostly translate for a different amino acid, and thus these positions are highly conserved in an alignment of related sequences. Position 10 has a variant nucleotide at the 1st base of a codon that does not lead to a change in the protein sequence, as CGG, AGG, CGC, and CGA all translate into arginine (see Figure 38).
10. a. Use Figure 39: TCCASTWGMA.
b. TKCWASTGGA (see Figure 39).
c. After entering the function **infocont** in R, you can call it as follows:

infocont (0, 0, 1, 7)

The four numbers represent the number of As, Cs, Gs, and Ts at position 1. This tells you that the information content of the first position is 1.456436. You can also validate this manually according to Equation 5: we observe 7/8 times a T, so $p_T = 7/8 = 0.875$; we observe 1/8 times a G, so $p_G = 1/8 = 0.125$. For A and C, $p_A = p_C = 0$. Remember

that we assumed $0 \times \log_2(0) = 0$, so the information content of the first site is:
 $I = (p_A \times \log_2(p_A / 0.25)) + (p_C \times \log_2(p_C / 0.25)) + (p_G \times \log_2(p_G / 0.25)) + (p_T \times \log_2(p_T / 0.25))$

$= 0 + 0 + (0.125 \times \log_2(0.125 / 0.25)) + (0.875 \times \log_2(0.875 / 0.25)) \approx 1.46$. The information content for positions 1-10: 1.456436, 0.5943609, 1.045566, 0.5943609, 1, 2, 1, 2, 1, and 2.

- d. Go to <http://weblogo.berkeley.edu/> to create the figure below. Note that the heights of the letter stacks correspond to the information scores calculated above. When making a sequence logo of few sequences, Weblogo uses "Small Sample Correction" by default. If the heights of the stacks are not exactly the same as calculated above, this is because you did not turn off "Small Sample Correction" before clicking "Create Logo". You can turn "Small Sample Correction" off under the "Advanced Logo Options".

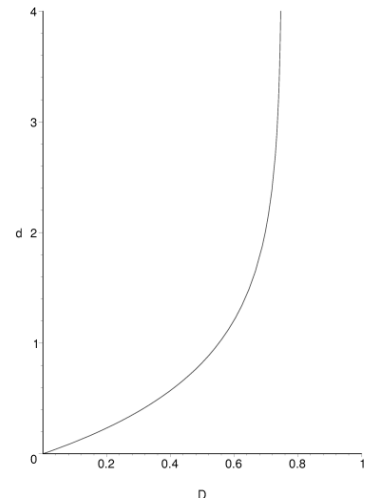


11. a. AEP prefers asparagine (N) at the cleavage site. There are no exceptions.
 b. This means that AEP does not cleave at all N's in a protein. Several things might be going on. One possibility is that AEP may rely on other signals to make a cleavage, but it is not clear from the sequence logo what these signals are. It might also be that some asparagine residues are not exposed on the protein surface because they lie within the tertiary (or quaternary) structure of the protein. This could make them inaccessible to AEP.
12. a. Given the assumption that each nucleotide is independent, the chance of any one to appear at a given point in the sequence is $1/4$. The sequence 5'-GGATATCCGC-3' is 10 nucleotides long, meaning that the probability of it occurring is: $(1/4)^{10} = 0.000000953674316$.
 b. In a 10 kb molecule, you have 19,982 chances to find this sequence (9,991 times on each strand). Each time you have a probability of 0.000000953674316 to find it, so in total you expect to find it 0.01905632019 times, which after rounding off to a whole number is equal to zero. So, if we are given one 10 kb sequence, we do not expect to find this specific 10-nucleotide at all.
 c. To calculate the expected frequency, we multiply the probability of finding a specific 20-nucleotide sequence with the size of the whole genome. Thus, we expect to find the 20-mer $(1/4)^{20} \times (4 \times 10^9) = 0.0036$ times on the genome. However, the question was not how often we will find it, but what is the probability of finding it at least once? In this case, 0.0036 happens to be a good approximation of this probability, because $(1/4)^{20} \times (4 \times 10^9) \ll 1$. However, if we want to calculate the probability of finding a specific 3-nucleotide sequence in a genome of 4×10^9 nucleotides, the probability should be calculated in the correct way, i.e. $1 - (1 - (1/4)^3)^{4000000000} \approx 1$. The probability of finding a specific 16-nucleotide sequence in a genome of 4×10^9 nucleotides is ~60%.

12.5. Answers to questions in Section 6.8

1. a. The BLOSUM matrix contains log-odds substitution scores that represent the probability that one amino acid is replaced by another in evolution (Box 2). These probabilities are a representation of what happened in evolutionary time, and in that sense, they are a model (see Figure 47).
 b. Most conserved: W: tryptophan. Second most conserved: C: cysteine
 c. Least conserved: A: alanine, I: isoleucine, L: leucine, S: serine, and V: valine.
 d. Often, the non-diagonal values agree between the BLOSUM62 and PAM250 matrices (at least in sign). Thus, the two matrices agree for the majority of the substitutions.
2. a. Using the identity matrix, seq1 is equally similar to seq2 as it is to seq3, because both seq2 and seq3 have 2 mismatches compared to seq1. This does not depend on the exact scores in the identity matrix, e.g. if you assume match = 2, mismatch = -2, then seq1 is still equally similar to seq2 as it is to seq3.
 b. With BLOSUM62:
 Score of similarity between 1 and 2 = $4 - 3 + 5 + 6 + 6 - 3 + 6 + 6 + 6 = 33$
 Score of similarity between 1 and 3 = $4 + 11 + 2 + 6 + 6 + 11 + 6 + 2 + 6 = 54$
 Score of similarity between 2 and 3 = $4 - 3 + 2 + 6 + 6 - 3 + 6 + 2 + 6 = 26$
 Sequence 1 is more similar to sequence 3 than to sequence 2.
 c. The BLOSUM62 matrix contains substitution scores derived from observed substitution rates in blocks of well-aligned homologs. These sequences were shaped by evolution itself, so is a biologically meaningful measure of protein sequence similarity (see Section 6.2).
3. The alignment score is $1 + -1 + -2 + 6 + 3 + 1 + -1 = 7$. Thus, the observed/expected ratio is $2^{(14/2)} = 128$. Thus, these two sequences are 128 more likely to be well-aligned homologs than unaligned sequences.

4. a. The Jukes-Cantor model is given in Equation 8. The graph is displayed to the right.



- b. First check out how `seq()` works by typing `?seq`. You will find out that it allows you to print a range of numbers by specifying a start and end value, and the increment size. Try it by typing `seq(100)`, which will display a vector of whole numbers between 0 and 100. Similarly, `seq(0,1,.01)` will display a vector of numbers between 0 and 1, with an increment size of 0.01. These will be our X-axis values, and we can quickly plot them by typing `plot(seq(0,1,.01))`. Now let's move on to the Jukes-Cantor formula (see Equation 8). Something else we will need is the natural logarithm, which is implemented in R as `log()`. Type: `log(seq(0,1,0.01))` and `plot(log(seq(0,1,0.01)))`. Now we have enough ingredients, and we can proceed to implement the full Jukes-Cantor formula in R. Type the following:

```
plot(-(3/4)*log(1-(4/3)*seq(0,1,0.01)))
```

R will give a warning message: “NaNs produced”. NaN stands for “Not a Number”. You can probably guess which ones they are, but you can also see them by removing the “`plot()`” part, and R will just list all the Y-axis values in order.

- c. To see the behavior of this function, we can infer its main features from either the graph or from the equation. First, we observe that when $D \rightarrow 3/4$, $d \rightarrow \infty$. This singularity can be seen in the graph at the value $D = 0.75$. Second, when the fraction of sites that differ between the sequences is zero ($D = 0$), we notice that $d = 0$ as well. Third, when difference between the sequences is small, the distance d increases linearly with D , because initially, there is only a low probability of several mutations per site. We can see this by the slope of the curve at small values of D , where the slope is almost equal to 1.
 - d. If $D = 0.20$, then $d = 0.23$.
 - e. If two sequences are only 20% identical, $D = 0.8$ and $d = \infty$. What does it mean if d goes to infinity? Basically, Jukes and Cantor suggest that two compared sequences diverged an infinitely long time ago if $D \geq 0.75$. In other words, these sequences are not recognizably similar anymore, and based on our observation of the data alone, these two sequences might as well not be homologous at all! So the simplest explanation of the observation is that these are two completely different genes with no shared common ancestor, i.e. they are not related.
 - f. This threshold lies at 0.75 because two completely random, unaligned sequences consisting of 25% A's, 25% C's, 25% G's, and 25% T's should be exactly 25% identical (or 75% different). Thus, $d = 0.75$ is the random baseline, or null-hypothesis for unrelated DNA sequences with a 50% GC content.
5. The following function `id_score` takes two vectors as input: `seq1` and `seq2`.

```
id_score <- function(seq1, seq2) {
  score <- 0
  for (position in 1:length(seq1)) {
    if (seq1[position] == seq2[position]) {
      score <- score + 1 } }
  return(score) }
```

Next, we define the two input vectors as follows:

```
seq1 <- c('V', 'W', 'E', 'D', 'N', 'W', 'D', 'D', 'D')
seq2 <- c('V', 'S', 'E', 'D', 'N', 'R', 'D', 'D', 'D')
```

And finally, we can call the function as follows:

```
id_score(seq1, seq2)
```

This gives the resulting number of matching positions as output: 7.

12.6. Answers to questions in Section 7.8

- Three different alignments are possible, each with the same score. All three should be reported.

DE--VD	D-E-VD	D--EVD
DEEEVW	DEEEVW	DEEEVW

- The score for alignment in (i) is $4 + 3g$ and in (ii) it is $g - 1$. For (i) to be a better alignment: $4 + 3g > g - 1$ or $g > -2.5$. So, $g = -2$ or $g = -1$ are the values of g that makes the alignment of (i) better than (ii). It does not make sense to have positive gap penalties.
- Use the Java tool from <http://baba.sourceforge.net> to make your life easier. Remember that you need to set the score table (substitution matrix) every time you roll back. The resulting alignment matrices for questions a, b, and c are shown in the table below. When using the BLOSUM62 matrix with a fixed gap penalty of -6 or -100 the alignments are the same as when using the PAM250 matrix with a gap penalty of -6. With a gap penalty of 0, free gaps can be added to avoid aligning amino acids that would otherwise lead to a penalty. The substitution score between H-E is zero (see Figure 45) so there are four options with equal score.

a. BLOSUM62 fixed gap -6	b. PAM250 fixed gap -6	c. BLOSUM62 fixed gap -100																																																																																																																																																	
<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>-6</td><td>-12</td><td>-6</td><td>-30</td></tr><tr><th>H</th><td>-6</td><td>4</td><td>-2</td><td>-8</td><td>-20</td></tr><tr><th>A</th><td>-12</td><td>-2</td><td>2</td><td>-3</td><td>-15</td></tr><tr><th>K</th><td>-6</td><td>-8</td><td>-2</td><td>1</td><td>-4</td></tr><tr><th>E</th><td>-30</td><td>-20</td><td>-14</td><td>-4</td><td>0</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	-6	-12	-6	-30	H	-6	4	-2	-8	-20	A	-12	-2	2	-3	-15	K	-6	-8	-2	1	-4	E	-30	-20	-14	-4	0	<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>-6</td><td>-12</td><td>-6</td><td>-30</td></tr><tr><th>H</th><td>-6</td><td>2</td><td>-4</td><td>-10</td><td>-22</td></tr><tr><th>A</th><td>-12</td><td>-4</td><td>2</td><td>-3</td><td>-15</td></tr><tr><th>K</th><td>-6</td><td>-10</td><td>-3</td><td>2</td><td>-5</td></tr><tr><th>E</th><td>-30</td><td>-22</td><td>-14</td><td>-7</td><td>0</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	-6	-12	-6	-30	H	-6	2	-4	-10	-22	A	-12	-4	2	-3	-15	K	-6	-10	-3	2	-5	E	-30	-22	-14	-7	0	<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>-100</td><td>-100</td><td>-100</td><td>-100</td></tr><tr><th>H</th><td>-100</td><td>4</td><td>-96</td><td>-196</td><td>-396</td></tr><tr><th>A</th><td>-100</td><td>-96</td><td>2</td><td>-97</td><td>-297</td></tr><tr><th>K</th><td>-100</td><td>-196</td><td>-97</td><td>1</td><td>-192</td></tr><tr><th>E</th><td>-100</td><td>-396</td><td>-297</td><td>-192</td><td>0</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	-100	-100	-100	-100	H	-100	4	-96	-196	-396	A	-100	-96	2	-97	-297	K	-100	-196	-97	1	-192	E	-100	-396	-297	-192	0																																					
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	-6	-12	-6	-30																																																																																																																																														
H	-6	4	-2	-8	-20																																																																																																																																														
A	-12	-2	2	-3	-15																																																																																																																																														
K	-6	-8	-2	1	-4																																																																																																																																														
E	-30	-20	-14	-4	0																																																																																																																																														
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	-6	-12	-6	-30																																																																																																																																														
H	-6	2	-4	-10	-22																																																																																																																																														
A	-12	-4	2	-3	-15																																																																																																																																														
K	-6	-10	-3	2	-5																																																																																																																																														
E	-30	-22	-14	-7	0																																																																																																																																														
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	-100	-100	-100	-100																																																																																																																																														
H	-100	4	-96	-196	-396																																																																																																																																														
A	-100	-96	2	-97	-297																																																																																																																																														
K	-100	-196	-97	1	-192																																																																																																																																														
E	-100	-396	-297	-192	0																																																																																																																																														
<div>SPEARE</div> <div> </div> <div>S-HAKE</div>	<div>SPEARE</div> <div> </div> <div>S-HAKE</div>	<div>SPEARE</div> <div> </div> <div>S-HAKE</div>																																																																																																																																																	
c. BLOSUM62 fixed gap 0, has four solutions (more gaps can be added for “free”)																																																																																																																																																			
<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>H</th><td>0</td><td>4</td><td>-4</td><td>-4</td><td>-4</td></tr><tr><th>A</th><td>0</td><td>-4</td><td>4</td><td>-4</td><td>-4</td></tr><tr><th>K</th><td>0</td><td>-4</td><td>-4</td><td>4</td><td>-4</td></tr><tr><th>E</th><td>0</td><td>-4</td><td>-4</td><td>-4</td><td>4</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	0	0	0	0	H	0	4	-4	-4	-4	A	0	-4	4	-4	-4	K	0	-4	-4	4	-4	E	0	-4	-4	-4	4	<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>H</th><td>0</td><td>4</td><td>-4</td><td>-4</td><td>-4</td></tr><tr><th>A</th><td>0</td><td>-4</td><td>4</td><td>-4</td><td>-4</td></tr><tr><th>K</th><td>0</td><td>-4</td><td>-4</td><td>4</td><td>-4</td></tr><tr><th>E</th><td>0</td><td>-4</td><td>-4</td><td>-4</td><td>4</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	0	0	0	0	H	0	4	-4	-4	-4	A	0	-4	4	-4	-4	K	0	-4	-4	4	-4	E	0	-4	-4	-4	4	<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>H</th><td>0</td><td>4</td><td>-4</td><td>-4</td><td>-4</td></tr><tr><th>A</th><td>0</td><td>-4</td><td>4</td><td>-4</td><td>-4</td></tr><tr><th>K</th><td>0</td><td>-4</td><td>-4</td><td>4</td><td>-4</td></tr><tr><th>E</th><td>0</td><td>-4</td><td>-4</td><td>-4</td><td>4</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	0	0	0	0	H	0	4	-4	-4	-4	A	0	-4	4	-4	-4	K	0	-4	-4	4	-4	E	0	-4	-4	-4	4	<table><tr><th>D(i,j)</th><th>S</th><th>H</th><th>A</th><th>K</th><th>E</th></tr><tr><th>S</th><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>H</th><td>0</td><td>4</td><td>-4</td><td>-4</td><td>-4</td></tr><tr><th>A</th><td>0</td><td>-4</td><td>4</td><td>-4</td><td>-4</td></tr><tr><th>K</th><td>0</td><td>-4</td><td>-4</td><td>4</td><td>-4</td></tr><tr><th>E</th><td>0</td><td>-4</td><td>-4</td><td>-4</td><td>4</td></tr></table>	D(i,j)	S	H	A	K	E	S	0	0	0	0	0	H	0	4	-4	-4	-4	A	0	-4	4	-4	-4	K	0	-4	-4	4	-4	E	0	-4	-4	-4	4
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	0	0	0	0																																																																																																																																														
H	0	4	-4	-4	-4																																																																																																																																														
A	0	-4	4	-4	-4																																																																																																																																														
K	0	-4	-4	4	-4																																																																																																																																														
E	0	-4	-4	-4	4																																																																																																																																														
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	0	0	0	0																																																																																																																																														
H	0	4	-4	-4	-4																																																																																																																																														
A	0	-4	4	-4	-4																																																																																																																																														
K	0	-4	-4	4	-4																																																																																																																																														
E	0	-4	-4	-4	4																																																																																																																																														
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	0	0	0	0																																																																																																																																														
H	0	4	-4	-4	-4																																																																																																																																														
A	0	-4	4	-4	-4																																																																																																																																														
K	0	-4	-4	4	-4																																																																																																																																														
E	0	-4	-4	-4	4																																																																																																																																														
D(i,j)	S	H	A	K	E																																																																																																																																														
S	0	0	0	0	0																																																																																																																																														
H	0	4	-4	-4	-4																																																																																																																																														
A	0	-4	4	-4	-4																																																																																																																																														
K	0	-4	-4	4	-4																																																																																																																																														
E	0	-4	-4	-4	4																																																																																																																																														
<div>SPE-ARE</div> <div> </div> <div>S--HAKE</div>	<div>SPEARE</div> <div> </div> <div>S-HAKE</div>	<div>SP-EARE</div> <div> </div> <div>S-H-AKE</div>	<div>S-PEARE</div> <div> </div> <div>SH--AKE</div>																																																																																																																																																

- Positions 2, 11, 16, 17, 19, 24, 26, 27, 28, 29, 30, 31, 32, 33, 35, 44, 50.
 - No, they are grouped. This implies that certain parts (i.e. the well-conserved parts) of the sequence are functionally more important than others.

- c. Not always. It seems that what is more important is where an amino acid is. If an amino acid is within a conserved region, it is more likely that it will be conserved independent of which amino acid it is.
- d. Second (chimp) and fourth (mouse).
- e. We use match=1; mismatch=0 (alternative is match=1; mismatch=-1):
 Score(shark,human)= 25 ; Score(human,chicken) =23. So, shark sequence is closer to human than chicken is to human if we use identity.
5. The Smith-Waterman matrix H is shown below to the left. Following the arrows starting from the cell with the highest score, we find the best local alignment with alignment score 11 (right).

D(i,j)		D	F	K	Y	I	W
	0	-12	-12	-12	-12	-12	-12
	0	0	0	0	0	0	0
G	-12	-1	-3	-2	-3	-4	-2
	0	0	0	0	0	0	0
Y	-12	-3	3	0	7	-1	2
	0	0	3	0	7	0	2
Y	-12	-3	3	-2	7	-1	2
	0	0	3	1	7	6	2
I	-12	-3	0	-3	-1	11	-3
	0	0	0	0	0	11	3

YI
 ||
 YI

6. a. Alpha helices (red) and random coil (unordered, green).
- c. The sequences are stored in Fasta format. Note that we use the .txt extension to make it easy for you to open the file in a text editor – actually the preferred extension is .faa for fasta amino acids. We recommend configuring your computer so that fasta files (.faa, .fna, and .fasta) are automatically opened in a text editor such as Notepad++.
- d. We expect that the sequences are homologs, since the question states that they are all vertebrate mitochondrial cytochrome B proteins. This means we can align them sensibly (aligning non-homologous proteins is nonsense). If sequences have different lengths, then we expect gaps after aligning them. Here all the sequences have more or less the same length, so we do not expect large gaps (still, it could be the case if there were many insertions and deletions).
- e. To align sequences using the Clustal webserver, go to <http://www.ebi.ac.uk/Tools/msa/clustalo/> and copy paste the sequences into the box. Always use the slow or full alignment algorithm, and not the fast one, since it produces better alignments. There is a single gap in the frog sequence around position 280-290 of the alignment. The asterisks (*) below the alignment show completely conserved positions between the sequences, so stretches of >10 asterisks are what we are looking for: they occur at positions 130-150 and, to a lesser extent, at positions 270-290.
- f. The DNA alignment is not very different from the protein alignment, it is just three times longer. Also, the gap in the frog sequence is 3 nucleotides long (around position 860), which is to be expected because one amino acid is encoded by a codon of three nucleotides.
- g. With a bit of wishful thinking/searching, you find regions in the alignment where the conserved positions (asterisks) follow a “** ** ** ** ” pattern (repetitions of two conserved, one variable). This is what might be expected from aligned codons, where the

nucleotide at the third codon position can be changed without changing the encoded amino acid (see Figure 38).

- h. Protein sequences are more conserved than DNA sequences, and closely related species have more similar sequences than distantly related species. For very closely related species, it may be possible that the protein sequences are still identical, while the DNA sequences have already diverged. In those cases, the DNA sequences will better reflect the divergence times of the species than the protein sequence, because the latter is too conserved to have a signal. This is also known as “bad phylogenetic resolution” at short evolutionary distance.
- i.

```
>gi|402962|emb|CAA47966.1| cytochrome b (mitochondrion) [Arabidopsis]
MTIRNQRFSLLKQPISSSTLNQHLVDYPTPSNLSYWWGFGSLAGICLVIQIVTGVFLAMHYTPHVDLAFNSVE
HIMRDVEGGWLLRYMHANGASMFLIVVYLHIFRGLYHASYSPPREFVWCLGVVIFLLMIVTAFIGYVLPWGQ
MSFWGATVITSLASAIIPVVGDTIVTWLWGGFSDNATLNRFFSLHLLPFILVGASLLHLAALHQYGSNNPL
GVHSEMDKIAFYFYPYVKDLVGWVAFIAFFSIWIFYAPNVLGHPDNYIPANPMSTPPHIVPEWYFLPIHAIL
RSIPDKAGGVAAIAPVFICLLALPFFKSMYVRSSSRPIHQGMFWLLADCLLLGWIGCQPVEAPFVTIGQI
SPLVFFLFFAITPILGRVGRGIPNSYTDHT
```
- j. Although the plant sequence is, as expected, more divergent than the vertebrate sequences, the conserved stretches mentioned above remain unchanged.
- k. Inspecting the additional information that is given in the CAA47966.1 GenBank entry, one realizes that the first conserved stretch corresponds to the heme binding site, and the second conserved stretch corresponds to the Qq binding site. Indeed, the C-terminal domain of cytochrome B is involved in forming the ubiquinol/ubiquinone binding sites, but not the heme binding sites. The N-terminal portion of cytochrome b contains heme binding sites. This conservation supports the idea that both these binding sites are crucial for the function of cytochrome b.
- l. The sequences are highly similar in pairs: DOG_3 and HUMAN_3; Dog_1 and HUMAN_1; and DOG_2 and HUMAN_2 are >89% identical to one another, but less similar to the other hexokinases. A likely explanation is that in an ancestral organism (the vertebrate ancestor of both dogs and humans) there occurred two gene duplications in the hexokinase protein family, followed by functional differentiation of the daughter proteins (hexokinases 1, 2, and 3). Now (in the present-day genomes of human and dog) we observe that HUMAN_1 is closest to Dog_1, instead of being closer to HUMAN_2. This suggests that the gene duplication and functional diversification occurred before the speciation of humans and dogs.

HUMAN_P	100.00	52.63	52.09	52.81	52.81	54.76	54.55
DOG_3	52.63	100.00	89.71	52.65	52.98	56.46	56.18
HUMAN_3	52.09	89.71	100.00	52.71	52.49	55.85	55.69
Dog_1	52.81	52.65	52.71	100.00	95.42	71.84	72.74
HUMAN_1	52.81	52.98	52.49	95.42	100.00	72.06	73.06
DOG_2	54.76	56.46	55.85	71.84	72.06	100.00	94.06
HUMAN_2	54.55	56.18	55.69	72.74	73.06	94.06	100.00

- m. The highest pairwise identity scores of human and dog hexokinase sequences are around 90-95%, whereas human and dog cytochrome B sequences are about 75% identical. Thus, the data shows that hexokinase sequences are more conserved, after the gene duplication, than the cytochrome B sequences.

	C	L	E	A	N	C	L	A	M	S	C	R	A	M	M	E	D	I	N	C	L	E	A	N	C	A	N	S
C	C					C					C									C					C			
L		L					L														L							
E			E													E						E						
A				A				A					A										A			A		
N					N														N					N			N	
C	C					C					C									C					C			
L		L					L														L							
A				A				A					A										A			A		
M									M					M	M													
S										S																		S
C	C					C					C										C					C		
R											R																	
A				A				A					A										A			A		
M									M					M	M													
M										M					M	M												
E			E														E						E					
D																		D										
I																			I									
N					N															N					N			N
C	C					C					C										C					C		
L		L					L															L						
E			E													E							E					
A				A				A					A											A			A	
N					N																N					N		N
C	C					C					C										C					C		
A				A				A					A											A			A	
N					N															N						N		N
S											S																	S

7. This is a difficult exercise that is meant to illustrate that you could generate results (like dot-plots) faster with a computer than by hand. To solve it, the following information is important (from the help page of Excel): “By default, a cell reference is relative. For example, when you refer to cell A2 from cell C2, you are actually referring to a cell that is two columns to the left (C minus A), and in the same row (2). A formula that contains a relative cell reference changes as you copy it from one cell to another. For example, if you copy the formula **=A2+B2** from cell C2 to C3, the formula references in C3 adjust downward by one row and become **=A3+B3**. If you want to maintain the original cell reference when you copy it, you “lock” it by putting a dollar sign (\$) before the cell and column references. For example, when you copy the formula **=A\$2+B\$2** from C2 to D2, the formula stays exactly the same. This is an absolute reference.”

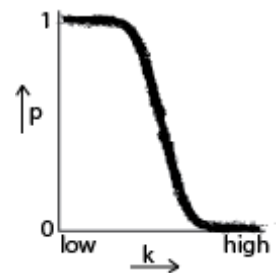
A dot-plot can be generated in Excel by writing the characters in the first row and the first column manually (the gray cells), and using a formula in all other cells (white). For example, in cell B2 the formula was **=IF(EXACT(B\$1,\$A2),B\$1,"")** and after writing this formula in one cell it was “drag-copied” to all other cells. Note that the commas in this Excel formula might be semicolons (;) depending on the version of Excel. In this case, the cell reference is “mixed”, either the column or the row value is preceded with a dollar sign to “lock” it (for example, **B\$1** or **\$A2**).

The result is the dot-plot above. You can observe that the sequence **CLEANC** is present in two diagonals besides the main one, reminiscent of a duplicated region (see Figure 50B).

12.7. Answers to questions in Section 8.9

1.
 - a. 26 (see Figure 45).
 - b. $E = 0.04 \times 40 \times 6,548,585 \times e^{(-0.27 \times 26)} = 9365.3$ (see Equation 12).
 - c. This hit is not significant because its E-value is very high: we expect more than 9,000 hits with at least this alignment score, given the length of the query and the size of the database. We may conclude that we have probably encountered this hit by chance in this relatively large database. Indeed, the HSP is very short.
2.
 - a. If you have 5,500 proteins that each have significant hits (low E-values) with single proteins from other fungi it is likely that many of these proteins are orthologs (i.e. these genes were present in the common ancestor of the fungi, see Section 4.5 – but actually you would have to make phylogenetic trees to be certain they are orthologs). Since these proteins are still present in many fungi, they might perform important functions like housekeeping, DNA replication, etc.
 - b. If the E-values of found protein matches are larger than 10 (a very bad match) it is most likely that the protein from your predicted gene is specific to your novel fungus. Such genes are called "lineage specific genes".
 - c. You have several options: (1) You can BLAST against other species besides the fungi you already tried. (2) Because the gene prediction method you have used will not always be correct, you could try other gene prediction methods to test if your predicted gene set is robust. (3) You could also use the DNA sequence of your fungus in a blastx search to rule out any possibility that you might have missed protein-coding genes. Note that in this case, you have to check if the putative genes you discovered are not degenerating pseudogenes. (4) Since we do not have any significant blastp hits for these proteins, we cannot try PSI-BLAST here, because to build a sequence profile in PSI-BLAST, we need to have at least a few reasonable hits from an initial blastp search.
3.
 - a. We are searching with a DNA query in a DNA database, so we have the following options: (1) The worst choice is megablast, because it has a big word length to seed the alignment, and thus it can only find very similar sequences with few mutations. (2) A better option would be blastn, that searches for shorter and less perfect hits rather than megablast. (3) Discontiguous megablast combines the speed of megablast with the sensitivity of blastn, because it has a big word length and thus finds fewer hits than blastn, but the words are discontiguous so that mutations are possible, for example at the third nucleotide of the codon (see Section 8.3). (4) tblastx is the most sensitive option, because it searches for homology at the protein level by translating both the database (t-) and the query (-x) into protein. Thus, tblastx will give the most sensitive similarity search, and it is the best answer to this question.
 - b. First we can try tblastn, i.e. search with amino acid sequence of *E. coli* enzyme in a database containing the six-frame translated version of the *B. subtilis* genome sequence. Next, if we can identify homologs of this enzyme in other bacteria, we can use PSI-BLAST to build a sequence profile with those hits, and use that to very sensitively search for homologs in *B. subtilis*.
 - c. Histidine kinase (possibly important for the sensory system).

4. a. We cannot predict the function of this protein from this BLAST search, because i) hits seem equally likely (i.e. they have similar E-values) and ii) each hit suggests a different function (like enolase, nuclease, splicing coactivator subunit, etc) or no function (hypothetical protein).
b. We can set the low complexity filter on if it is not yet set, because there is a repeat in this sequence. Also, we can try using PSI-BLAST, which would build a sequence profile (PSSM) and might be able to identify more distant homologs.
5. a. To determine the significance, we look at the E-values. Since the match from site 2 has a lower E-value (5×10^{-11} or 5×10^{-11} , versus 1×10^{-4} or 1×10^{-4}), the second match (with AF9_HUMAN protein) is more significant.
b. In the first search the low complexity filter is on, and in the second it is off. The main match in the second search is at a poly-serine region (SSSSSSSSSSSSSSSS). Such repeats can occur in genomes without necessarily being homologous (see Section 5.6), so based on this match you cannot say if the hit protein (AF9_HUMAN) is a homolog of our query sequence or not. However, in the first search the match is outside the repeat region, and this implies "real" sequence similarity. The E-value is still quite low (although higher than the AF9_HUMAN hit), so this match with a chicken ubiquitin protein is more likely to reflect homology. Note that homologous proteins can easily be present in very different organisms, i.e., the organism itself is rarely a good criterion to evaluate the quality of a hit.
6. a. We can use blastx: search with a translated nucleotide query against protein database. The program will translate our nucleotide sequence in six different reading frames and search against the protein sequence databases.
b. No, because in order to use PSI-BLAST, we need to build a PSSM. PSSMs are built from initial BLAST hits. If we do not have any BLAST hits, then we do not have enough information to build a position specific substitution matrix either.
7. a. To find a hit using $k=99$, there has to be a stretch of 99 nucleotides in the database that is identical to that sequence in the raindrop. Such long stretches are expected if the microorganisms in rain have close relatives in the database. This seems unlikely, so maybe we would not expect very many hits.
b. She looks for subsequences of length k , so it is a local homology search algorithm.
c. See rough sketch to the right.
d. If k is very high, this means that you are looking for a sequence in the database with a very long identical match. Long identical matches are increasingly less likely as sequences diverge in evolution, because mutations break these long matches, leading to different k -mers. This means that the sequence is not very conserved.
e. Smith-Waterman is the slowest because it needs to create, fill, and backtrack alignment matrices for the query and all sequences in the database. Her k -mer program is the fastest, because it only needs to store read through all the database sequences once, and store all the k -mers in RAM memory, and then she can search that database very rapidly (see Section 2.6). BLAST is in the middle, because it also uses k -mers to quickly find potential hits, but that step is followed by a slower step extending those hits.



- f. The most sensitive for finding distant homologs is tblastx, because it searches for homology in the protein sequence, which is more conserved than DNA (see Section 5.3). Her program is the least sensitive because it does not allow any mutations between query and target. Blastn is in the middle because it searches for homology at the DNA level but allows some mutations.
8.
 - a. To do a blastn nucleotide search at <http://www.ncbi.nlm.nih.gov/BLAST/> make sure that you select “blastn” under Program Selection. The default is megablast which returns only highly similar matches. Here, we also want to find more divergent sequences, so we choose blastn (blastn has a shorter word size than megablast and can thus find more divergent sequences). Also, you want to make the search database as broad as possible, not only the human sequence database, so you want to search for homologs in the non-redundant database (nr). The best nucleotide hit is the *Homo sapiens* period circadian clock 2 gene (PER2). This hit has a very low E-value ($7 \cdot 10^{-84}$ at the time of writing this Reader).
 - b. The SNP in the patient sequence (a mutation from A→G at position 66 of the alignment) is only visible by looking at the alignment. If you use a blastx instead of blastn for the search, you can figure out that this SNP will cause a non-synonymous change in the protein sequence (the amino acid changes from serine S to glycine G).
 - c. With blastn you can find homologs in more distant species. This is because the *k*-mer length is lower than in megablast ($k = 11$ vs. $k = 28$ in megablast). Thus, megablast is used for finding highly similar nucleotide sequences. For this specific sequence, we can find homologs in *Xenopus* frogs with blastn, but not with megablast.
 - d. To search OMIM, go to the NCBI website <http://www.ncbi.nlm.nih.gov/> and select OMIM from the pulldown menu on the top left. Enter the gene or protein name you want to search in the text field (PER2) and hit “Search”. In the results, the effect of SNPs can be investigated by following the “OMIM dbSNP” link at the top of the results page. In dbSNP, the first hit links to a lot of information about this gene, with information on SNPs causing sleep disorders at the end of this page. For PER2 you find that the SNP SER662GLY causes sleeping disorders, which means a mutation at position 662 of the protein sequence from serine to glycine. This SNP is linked to a Familial Advanced Sleep Phase Syndrome.
 9. Several features contribute to the speed of BLAST. First, it creates an index of the *k*-mers in the database sequences and stores those in the Random-Access Memory (RAM) of the computer, allowing database sequences with high-scoring words matching the query to be rapidly identified. Second, it only compares those database sequences with matching high-scoring words to the query sequence because those are likely potential good hits, while dynamic programming would have to compare all the sequences in the database. Third, it starts the detailed sequence comparison at the high-scoring word, and extends the alignment in both ways until the score drops below a cutoff value. Thus, it does not need to create a complete dynamic programming matrix between the two sequences, which also saves a lot of computer time.
 10.
 - a. A blastp search with default parameters identifies the crAssphage protein as the top hit, which is an identical sequence and has no annotation besides “hypothetical protein”. If you look at the lower scoring hits, they suggest that the query protein might be a DNA

primase or a topoisomerase. The E-value is well below the rule-of-thumb cutoff of 10^{-3} . For phages, you will only very rarely find good hits, because they mutate very fast.

- b. The hits are in bacteria (*Alkaliphylus*, *Chlamydia*, *Veilonella*, *Riemerella*), which makes sense since the query protein is from a bacteriophage, and these homologs might have been picked up by the phage during an infection event of bacteria. Bacteria and phages show a lot of horizontal gene transfer (HGT, see Section 4.8), so hits to other bacteria are quite common in their genomes.
- c. To identify the location of the protein on the crAssphage genome, you should perform a tblastn search against the genomic DNA sequence.

11. a. The following amino acid sequence can be found in Genbank or EBI:

```
>human_ubiquitin
MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLLIFAGKQL
EDGRTLSDYNIQKESTLHLVLRGG
```

- b. A blastp search with the given human sequence as a query returns several hits, not only in eukaryotes, but also in viruses (you can see this by inspecting the taxonomy report). (In case you were in doubt, viruses are not eukaryotes.) Looking at the alignments you see how conserved this protein is. For example, zebra fish ubiquitin is 100% identical to human ubiquitin. Even for the viral ubiquitin the identity is 99% with the human sequence.

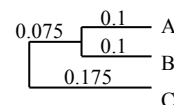
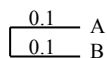
12.8. Answers to questions in Section 9.9

1. A hierarchical clustering is a representation of the similarity between data points or vectors (see Chapter 3), while a phylogenetic tree is a hypothesis about the evolutionary history of nucleotide or amino acid sequences. Thus, the branch lengths in a phylogeny are supposed to accurately reflect evolutionary time (see Section 4.4), while the branch lengths in a hierarchical clustering “just” reflect the similarity between the data points (see Sections 3.4 and 3.6). For example, while the algorithms for centroid clustering and UPGMA are very similar, an important difference is that distances between sequences in UPGMA are converted to evolutionary distances using Jukes-Cantor or Kimura correction (Section 6.6).
2. a. From the first distance matrix, we identify the pair of sequences with the shortest distance: A and B, and join them into a cluster (A, B) with distance 0.2 between them. Then we re-calculate the average distances between the sequences in new cluster (A, B) and the other sequences, obtaining the second matrix. Then we once again identify the pair of sequences with the shortest distance: (A, B) and C, join them into a cluster ((A, B), C), and re-calculate the distances for the last matrix, that we need to know the final distances in the phylogenetic tree: (((A, B), C), D). Remember that it is necessary to calculate the average distance based on the values from the original matrix: $(A, D) + (B, D) + (C, D) / 3 = 0.43$.

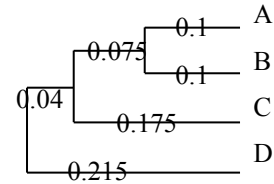
	A	B	C	D
A	0	0.2	0.3	0.4
B	0.2	0	0.4	0.4
C	0.3	0.4	0	0.5
D	0.4	0.4	0.5	0

	A, B	C	D
A, B	0	0.35	0.4
C	0.35	0	0.5
D	0.4	0.5	0

	(A, B), C	D
(A, B), C	0	0.43
D	0.43	0



- b. Taking all clusters and branch lengths into account, the final tree becomes:
- c. $((A:0.1, B:0.1):0.075, C:0.175):0.04, D:2.15);$

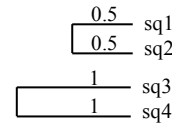
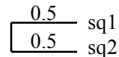


3. P (phenylalanine). This solution only requires one mutation at that position to L (leucine) since the ancestor of the *Embryophyta* to explain the observed pattern.
4. First, we calculate an initial distance matrix with the dissimilarities between all sequences.

	sq1	sq2	sq3	sq4
sq1	0	1	2	4
sq2	1	0	3	3
sq3	2	3	0	2
sq4	4	3	2	0

	sq1, sq2	sq3	sq4
sq1, sq2	0	2.5	3.5
sq3	2.5	0	2
sq4	3.5	2	0

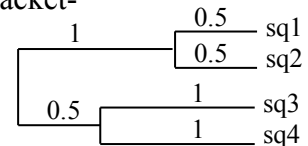
	sq1, sq2	sq3, sq4
sq1, sq2	0	3
sq3, sq4	3	0



The smallest distance cluster is (sq1, sq2) with a distance of 1 between them. We re-calculate the new distance matrix by averaging the distances between (sq1, sq2) and the other sequences and again identify the smallest distance between sq3 and sq4, and group them into a cluster (sq3, sq4).

Then we re-calculate the third distance matrix and draw the final bracket-notation and phylogenetic tree:

$((sq1:0.5, sq2:0.5):1, (sq3:1, sq4:1):0.5);$

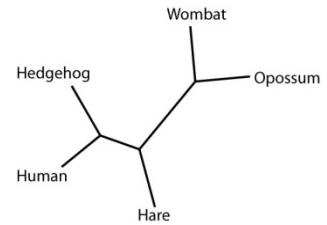


5. a. In total, we count 13 substitutions. First, we notice that 9/13 substitutions occur at the third position of the codons. Mutations at the third position are often synonymous (i.e. they do not lead to a change of the amino acid, see Figure 38, and see Section 5.3 for a definition of synonymous and non-synonymous mutations). Therefore, at first glance, the tendency seems to be that there are more synonymous than non-synonymous substitutions. However, if we carefully analyze the substitutions, we obtain the following list.

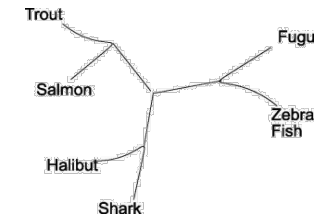
- 3x GAG \leftrightarrow GAA, which translates to: E \leftrightarrow E (synonymous)
- 2x TTT \leftrightarrow TTA, which translates to: F \leftrightarrow L (non-synonymous)
- 2x ATG \leftrightarrow ATA, which translates to: M \leftrightarrow I (non-synonymous)
- 2x ATG \leftrightarrow CTG, which translates to: M \leftrightarrow L (non-synonymous)
- 1x AAT \leftrightarrow AAC, which translates to: N \leftrightarrow N (synonymous)
- 1x GTG \leftrightarrow GTT, which translates to: V \leftrightarrow V (synonymous)
- 1x TCC \leftrightarrow TTC, which translates to: S \leftrightarrow F (non-synonymous)
- 1x TTT \leftrightarrow CTT, which translates to: F \leftrightarrow L (non-synonymous)

Observe that the mutations under c and d occur in the last codon of the sequences, and that we have considered here mutations of ATG. This means that only 5 mutations are synonymous, so the answer to this question is no.

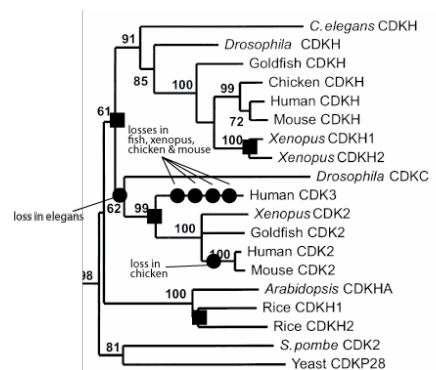
- c. No, there are 8 transitions and 5 transversions.
- d. The most informative sites are those in which at least two substitutions to the same nucleotide occur; i.e. sites 15, 27, 28, 30 (see Section 9.4).
- e. The MP tree is displayed to the right.



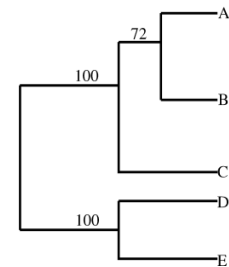
6. The informative sites are: 4, 6, 7, 9, and 12. The MP tree is displayed to the right.



7. a. In the figure to the right, gene duplications are indicated with squares, and gene deletions are indicated with circles.
- b. The common ancestor of eukaryotes had probably 1 CDK gene, because yeast, *Schizosaccharomyces pombe* and *Arabidopsis thaliana* have still one CDK gene.
- c. Never, because bootstrap value of *Xenopus* CDKH1 and CDKH2 clade is 100.



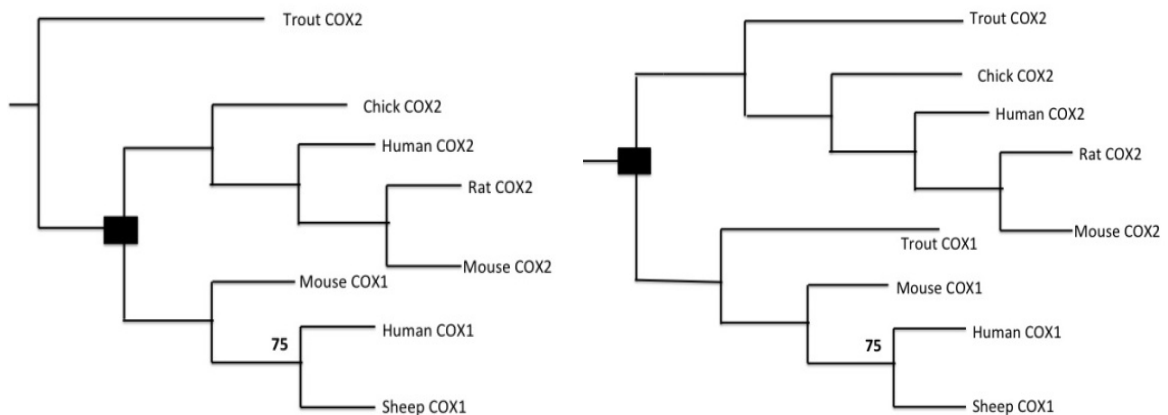
8. In all trees, D and E are found to form a clade. Also, in all cases, A, B, and C come together in clade (though in different ways). From the three possibilities of clustering A, B and C, we choose to draw the most probable one, as indicated below in the figure to the right. Note, that by drawing this tree with the probabilities observed, we lose some information, namely, that 19 times B was left alone while A and C formed a clade, and that 9 times A was on its own, while B and C formed a clade. The only information we can see now, is that in 72 cases A and B remained together.



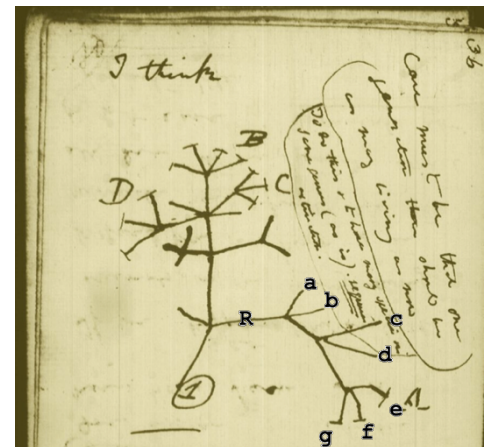
9. a. W and X formed a clade 86/100 times, so they did not form a clade 14/100 times.
- b. No. We know from the tree that W, X, Y, and Z formed a branch 94/100 times, so there is a maximal number of 6 times that W could form a clade with V.
- c. Yes. The only information that is guaranteed is that W and X formed a clade 86/100 times and that W, X, Y, and Z formed a clade 94/100 times, so it is possible that in the other 14 cases, W, Y, and Z formed a clade.
10. a. For brevity, we abbreviate the gene names to three letters and a number (for example, SHEEPCOX1 becomes SHE-1, etc). The bracket-notation for this tree is as follows:
 $(((((\text{RAT-2}, \text{MOU-2}), \text{HUM-2}), \text{CHI-2}), \text{TRO-2}), ((\text{HUM-1}, \text{SHE-1}), \text{MOU-1}));$
 Note that there are many ways of writing this tree as a bracket-notation. Another possible solution is:
 $(((((\text{SHE-1}, \text{HUM-1}), \text{MOU-1}), \text{TRO-2}), \text{CHI-2}), \text{HUM-2}), \text{MOU-2}), \text{RAT-2});$

Here is a nice explanation of the different options to come from an unrooted tree to newick format in different ways:
<http://evolution.genetics.washington.edu/phylip/newicktree.html>

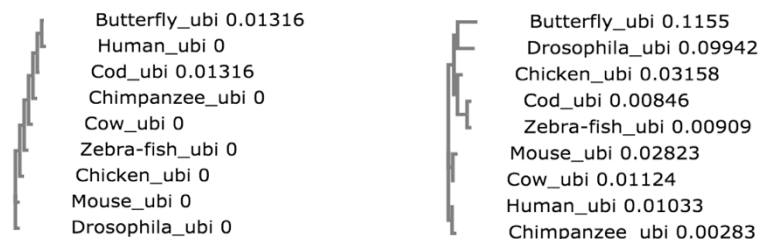
- b. The rooted tree is displayed below to the left (under e).
- c. See tree image below to the left for the gene duplication, indicated with a square. Deletions occurred in sheep (COX2), rat (COX1), and chicken (COX1).
- d. See tree image below to the left for the bootstrap value.
- e. This finding suggests that gene duplication occurred earlier, but the deletions do not change. Most likely the bootstrap value would not change either. The most parsimonious tree is displayed below to the right.



11. a. A, B, C, and D are species (or groups of species).
- b. The number ① represents the root or common ancestor of the taxa in the tree.
- c. The distances from the branch tips to the root differ, whereas in an UPGMA tree all distances to the root are equal. Moreover, there are multifurcating branches (split into more than two daughter lineages) whereas in an UPGMA tree all branches are always bifurcating (split in two daughter lineages).
- d. Several correct answers are possible, for example:
(a, b, ((c, d), (e, f, g))) ;



12. a. The phylogenetic tree based on the protein sequence alignment is the left one, the one based on the DNA alignment is the one to the right.



- b. The aim of this exercise is to show that one cannot create good phylogenies with highly conserved protein sequences: the branch lengths in the protein tree are zero and thus the topology does not make any sense. If, instead, we use the nucleotide sequence (that is less conserved than protein because synonymous mutations would accumulate, such as many of those in the third nucleotide of the codon) it is possible to find the evolutionary relatedness between the species.
- c. Based on our knowledge of the groups of organisms in the tree, we can place the correct position of the root of the tree between the major groups insects and vertebrates:



- d. If we enter all the species into the NCBI Taxonomy Common Tree tool, the result looks something like this:



One species that has an unexpected position in the ubiquitin tree, when compared to NCBI taxonomy, is chicken, which is not supposed to be connected to the fish. We would expect the chicken gene to be closer to the mammals, as they are both *Amniota*. Also, in the ubiquitin tree mouse and cow cluster together, while mouse belongs in the *Euarchontoglires* at the root of the human/chimp clade according to NCBI taxonomy.

Course Guide for *Basic R* as a part of Systems Biology

Rutger Hermesen & Can Kesmir

Getting Started with R

Getting R To work on the assignments you will need a computer with R installed. During the computer sessions, campus computers are available on which R is installed by default (available in My Work Space). But we highly recommend that you also install it to your own laptop or desktop computer, so that you can continue working with R where and whenever you please. R and RStudio are free and available for Windows, Mac OS, and Linux computers.

To install R, visit the “Comprehensive R Archive Network” (CRAN) website¹. At the top of the main CRAN page there are links to download instructions for each of the above operating systems .

Windows users should select the base package and then download a file with a name that looks something like this: “R-2.x.1-win32.exe”. After downloading is complete, running this file should install R. Mac users want a file named “R-2.x.1.dmg” or similar. Linux users will have to find a similar file under the appropriate flavor of Linux, or install the system through a package manager of their choice. Please note, the instructions below are written for a Windows system. While you are absolutely allowed use your Mac or Linux PC, there may be some small differences that you will be responsible for handling.

Getting RStudio We recommend that you use RStudio as your integrated development environment (IDE). Again, campus computers come with RStudio pre-installed, but you can easily install it to any other computer (Windows, Mac OS X, Linux). To do so, visit <https://rstudio.org/>. Click *Download RStudio*, click *Download RStudio Desktop*, click *Recommended For Your System*, download the appropriate file and run it (choose default answers for all questions).

Once RStudio is installed, open it from the desktop icon or the Start-Programs menu; on campus computers, you may have to look under “Physics” in the Programs menu or search the Start Menu.

Console versus scripts R works by typing *commands* into a *console*. The computer executes your commands when you press *enter*; you’ll practice with this below. A particular statistical analysis typically requires the execution of a long list of commands. It is important to store all these commands in a *text file*, so that you can easily redo or correct your analysis. A text file that contains all commands required for an analysis is called a *script*. You can open a new, empty text file in RStudio by clicking *file, new file, R Script*.

During each session, you will create a script (or perhaps multiple scripts) that contains all analyses and calculations of that session. Save your scripts for use in future weeks. Spend some time carefully annotating them with the help of the hash sign (#) as explained below. This is your way to learn and to remember *what* you did and *why* you did it. Your scripts will become your personal “cheat sheets”.

Tip 1 Write your name and the date of creation at the top of the script.

¹<http://cran.us.r-project.org/>

Tip 2 In RStudio, you can send a line or multiple lines of your script to the console by selecting the lines and subsequently pressing [Ctrl] + [Enter].

Getting help

There are many ways you can get help.

- You can access the documentation (help files) for any function by typing a question mark followed by the name of the function, or by using the function `help()`. For example:

```
> ?library
```

or

```
> help(library)
```

will both open the documentation for the `library()` function.

- You can also get online help from the CRAN website², in particular by clicking on the *Manuals* link on the left-hand side of the web page. There, you can download the manual named “An Introduction to R” by Venables and Smith; this manual is also available on Blackboard, under *Course Content, Documentation R*.
- Check out Appendix ?? of this guide, which is a Reference Card listing commonly used R functions.
- Honors student Wendy Lichtenauer wrote a manual with very clear explanations in Dutch of most R functions used in this course. This “Handleiding voor R” is included as Appendix ?? in this course guide. We highly recommend that you use it throughout the course.
- Google it. There is a huge community of R users and most of the problems that you may encounter have been discussed and solved in online forums—so use Google to find them. For example: if you want to compute the square root of two, but you do not know the command for that, type “square root function R” in Google.
- Ask questions during the computer session. We’re there to help.
- Carefully study the lecture slides.

Typesetting conventions

In this course guide, we use certain typesetting conventions to distinguish various text elements.

Normal text Normal text (*e.g.*, descriptions and notes) is displayed in a Roman font.

R code Code is set in typewriter font, with a gray background. Lines of code are preceded by a `>` (“greater than”) sign, like this:

²<https://cran.r-project.org/>

```
> install.packages("binom", dependencies = TRUE) # install package  
      binom
```

Note that function names and other keywords are set in bolt face (as in **install.packages**) and character strings are dark gray (as in "binom").

Annotations On any line of code, any text appearing *after* a hash sign (#) is ignored by R and can therefore be used to annotate the code. In this book, annotations are set in a *gray, Roman, italics* font.

R output Output produced by R is set in a slightly smaller typewriter font with a normal white background.

Basics of R: Tutorial

Below starts a tutorial that teaches you your first steps in R. Learn by doing! Type the lines of code below one by one in an empty R script, send them in the R console (using `[Ctrl] + [Enter]`), and study what happens. Do not hesitate to play around with the code! Make sure you completely understand what the commands do. Ask questions about every detail you do not understand.

1. *Assignment and basics*

- There are three ways to assign a value to an object:
 1. an equals sign `=`
 2. a “left arrow” `<-` (less than, hyphen)
 3. a “right arrow” `->` (hyphen, greater than).

You can type the name of any object to retrieve the value of that object. Try it:

```
> n <- 15
> n
```

```
[1] 15
```

The object (variable) `n` now stores the value 15.

Now try the following:

```
> n<-15
```

```
[1] 15
```

Note that we did not insert spaces around the “arrow”. Yet, `n` is again assigned the value 15. The point is: you can use spacing to make your commands look “pretty” (and readable), but R usually does not care. It’s (generally) up to you. Don’t, however, be so silly as to put a space in the middle of the name of an object or operator.

Here is one place where R insists on the correct spacing: The “arrow” assignment operator is actually two symbols, a less than sign and a hyphen or minus. There cannot be a space between them!

We recommend that you leave spaces on each side of the arrow operator (and operators in general). That way you’re less likely to make critical spacing errors. For example, suppose your fingers get all crossed up, and you type this:

```
> n < -15
```

Try it and see what happens. Usually not a problem—just retype it correctly:

```
> n <- 15
```

```
[1] 15
```

Instead of the arrow, you can also use an equality sign:

```
> a = 12
> a
```

```
[1] 12
```

You can even do this:

```
> 24 -> z # the arrow always points towards the variable name
> z
```

```
[1] 24
```

- As in any good programming language, you can insert comments into R code. Any line or partial line in R beginning with the hash symbol (`#`) is a comment, or note, and R will ignore it.

```
> # This is just a note.
> ### And so is this.
> n <- 15 # And so is this.
```

```
[1] 15
```

In the last line, R will assign the value 15 to the object `n` and ignore all the comments after the hash symbol.

- If you surround an assignment with parentheses, R prints the value to the screen. Compare the output of this command

```
> n <- 8
```

with the output of this command

```
> (n <- 8)
```

```
[1] 8
```

Note, by the way, that `n` previously had the value 15; we have now overwritten it with the value 8.

- Variable names must start with a letter, but may also contain numbers and periods:

```
> (var.s13 <- 4)
```

```
[1] 4
```

- R is case sensitive. That is, `n` and `N` are different variables.

```
> (N <- 26.42)
```

```
[1] 26.42
```

```
> n # The value of n is still 8.
```

```
[1] 8
```

- To see a list of your objects, use the function `ls()`. The parentheses `()` are required, even though there are no arguments.

```
> ls()
```

```
[1] "a" "n" "N" "var.s13" "z"
```

- Use `rm()` to delete objects you no longer need.

```
> rm(n)
> ls()

[1] "a"  "N"  "var.s13" "z"
```

- You may obtain online help about a function using either the `help()` command or a question mark.

```
> ?ls
> help(rm)
```

- Several commands are available to help find a command whose name you do not know.

```
> apropos("help") # Find commands with "help" in their name

[1] ".helpForCall"  "help"          "help.search"   "help.start"
[5] "link.html.help"
```

2. Simple calculations

- R may be used for simple calculations, using the standard arithmetic symbols `+`, `-`, `*`, `/`, and `^` (exponentiation), as well as parentheses.

```
> a <- 12 + 14
> a
```

```
[1] 26
```

```
> 3*5
```

```
[1] 15
```

```
> (20 - 4)/2
```

```
[1] 8
```

```
> 7^2
```

```
[1] 49
```

- Standard mathematical functions and constants are available.

```
> exp(2) # Exponential function
```

```
[1] 7.389056
```

```
> log(10) # Natural log
```

```
[1] 2.302585
```

```
> log10(10) # Log with base 10
```

```
[1] 1
```

```
> log2(64) # Log with base 2
```

```
[1] 6
```

```
> pi # The mathematical constant pi
```

```
[1] 3.141593
```

```
> cos(pi) # The cosine of pi
```

```
[1] -1
```

```
> sqrt(100) # The square root of 100
```

```
[1] 10
```

Data types

You have seen that a variable can store a *number*. However, several other data types are available. Usually, you do not need to declare these; they will be assigned automatically.

Character data A character object is represented by a collection of characters between double (") or single (') quotes. For example: "x", 'test character' and "Mike". One way to create character objects is as follows:

```
> name <- "Mike" # Character data (strings) need quotes like "this"
> name
[1] "Mike"
```

Note that double quotes in the output indicate that we are dealing with an object of type “character”.

Logical data An object of data type “logical” can have the value TRUE or FALSE and is used to indicate if a condition is true or false. Such objects are usually the result of logical expressions.

```
> q1 <- TRUE # Logical data; possible values are TRUE or FALSE
> q1
[1] TRUE
```

Note that TRUE and FALSE can be abbreviated as T and F:

```
> q2 <- F
> q2
[1] FALSE
```

However, to avoid confusion with possible variables called F or T, we recommend that you stick with TRUE and FALSE.

We can assign 9 to a variable x and ask whether x is greater or less than 10:

```
> x <- 9
> x > 10
[1] FALSE
```

The result is the logical value FALSE because 9 is *not* greater than 10.

```
> x < 10
[1] TRUE
```

The result is TRUE because 9 is less than 10.

```
> x == 9
[1] TRUE
```

Note that we used a double equality sign == to compare two objects. If we had used a single equality sign, we would have assigned the number 9 to the variable x instead. The result is TRUE because x equals 9.

We can also ask whether x is *unequal* to some number or expression.

```
> x != 10 #Is x unequal to 10?
```



```
[1] TRUE
```

If you wish to know whether two logical expressions are both true, you use logical operator `&`, called the AND operator:

```
> x == 10 & x > 2 #Is x equal to 10 and larger than 2?
```

```
[1] FALSE
```

This combined expression is false because the first expression (`x == 10`) is false.

If you wish to know whether at least one of the logical expressions is true, you use logical operator `|`, called the OR operator:

```
> x == 10 | x > 2 #Is x equal to 10 or larger than 2?
```

```
[1] TRUE
```

The combined expression is true because the second expression (`x > 2`) is true.

Factor data To represent categorical data (*i.e.*, data of which the value range is a collection of code names), R uses the *factor* data type. Factor data will be treated in more detail later on after you learned about vectors.

Vectors

- Vectors may be created using the `c()` function. Separate vector elements by commas.

```
> a <- c(1, 7, 32, 16) # Again, spaces have no effect, but improve
                        readability.
> a
```

```
[1] 1 7 32 16
```

- Vectors do not need to consist of numbers; vectors of character data or logicals are allowed, too:

```
# A vector of character strings.
> wind <- c("north", "west", "south", "east")
> wind
```

```
[1] "north" "west"  "south" "east"
```

```
> logic <- c(TRUE, FALSE, TRUE) # A vector of truth values.
> logic
```

```
[1] TRUE FALSE TRUE
```

- Sequences of integers may be created using a colon (`:`).

```
> b <- 1:10
> b
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> c <- 20:15
> c
```

```
[1] 20 19 18 17 16 15
```

- Other regular vectors may be created using the `seq()` (sequence) and `rep()` (repeat) commands.

```
> d <- seq(1, 5, by = 0.5)
> d
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
> e <- seq(0, 10, length = 5)
> e
```

```
[1] 0.0 2.5 5.0 7.5 10.0
```

```
> f <- rep(0, 5)
> f
```

```
[1] 0 0 0 0 0
```

```
> g <- rep(1:3, 4)
> g
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> h <- rep(4:6, 1:3) # Study what this does!
> h
```

```
[1] 4 5 5 6 6 6
```

- Vectors of random numbers can be created with a set of functions that start with **r**, such as **rnorm()** (draw numbers from a normal distribution) or **runif()** (draw numbers from a uniform distribution in the interval (0, 1)).

```
> x <- rnorm(5) # Standard-normal random numbers
> x
```

```
[1] -1.4086632  0.3085322  0.3081487  0.2317044 -0.6424644
```

```
> y <- rnorm(7, 10, 3) # Normal random numbers with mu = 10, sigma = 3
> y
```

```
[1] 10.407509 13.000935  8.438786  8.892890 12.022136  9.817101  9.330355
```

```
> z <- runif(10) # Uniform random variables in the interval (0, 1)
> z
```

```
[1] 0.925665659 0.786650785 0.417698083 0.619715904 0.768478685 0.676038428
[7] 0.050055548 0.727041628 0.008758944 0.956625536
```

- If a vector is passed to an arithmetic calculation, it will be applied element-by-element.

```
> c(1, 2, 3) + c(4, 5, 6)
```

```
[1] 5 7 9
```

If the vectors involved are of different lengths, the shorter one will be repeated until it has the same length as the longer one.

```
> c(1, 2, 3, 4) + c(10, 20)
```

```
[1] 11 22 13 24
```

```
> c(1, 2, 3) + c(10, 20)
```

```
[1] 11 22 13
```

Warning message:

```
longer object length is not a multiple of shorter object length in: c(1, 2,
3) + c(10, 20)
```

- Basic mathematical functions will apply element-by-element as well.

```
> sqrt(c(100, 225, 400))
```

```
[1] 10 15 20
```

- To select subsets of a vector, use square brackets (`[...]`). Inside the square brackets, you write (or construct) a vector that specifies which elements you wish to select.

```
> d
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
> d[3] # Third element of vector d
```

```
[1] 2
```

```
> d[5:7] # Elements 5 to 7
```

```
[1] 3.0 3.5 4.0
```

```
> my.selection = 5:7 # Create a vector
> d[my.selection] # Use it to select elements
```

```
[1] 3.0 3.5 4.0
```

- If you put a logical vector inside the brackets, R will return only those elements of `d` where the logical vector has value `TRUE`. For instance:

```
> d[c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE, TRUE)]
```

```
[1] 3.0 3.5 4.0 4.5 5.0
```

So, the first four values of `d` are not selected, because the first four values of the logical vector are `FALSE`. The last five values *are* selected, because the last five values of the logical vector are `TRUE`.

This technique can be very useful because it is very easy to generate useful logical vectors. Here's an example:

```
> d > 2.8 # Which elements of d are larger than 2.8?
```

```
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

Note that this creates a logical vector that specifies which elements of `d` are larger than 2.8.

Here's another example:

```
> d == 3 # Which elements of d are equal to 3?
```

```
[1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
```

Using this trick, we can easily select all elements of `d` that are greater than 2.8:

```
> d[d > 2.8]
```

```
[1] 3.0 3.5 4.0 4.5 5.0
```

Similarly, you can use this trick to select a specific range of the data:

```
> d[d > 2 & d < 4] # Select numbers between 2 and 4
```

```
[1] 2.5 3.0 3.5
```

This (not so useful) code select the value 3:

```
> d[d == 3]
```

```
[1] 3.0
```

This (very useful) code selects the elements of `d` that are *not* equal to 3:

```
> d[d != 3]
```

```
[1] 1.0 1.5 2.0 2.5 3.5 4.0 4.5 5.0
```

(In other words, this removes all elements with the value 3.0 from the vector!)

Make sure you understand this technique; it is very powerful!

- The number of elements in a vector can be found with the `length()` function.

```
> length(d)
```

```
[1] 9
```

What is being calculated in the following command?

```
> length(d[d > 2.8])
```

```
[1] 5
```

~ **PROBLEM 1.** Create the following vectors in R.

$$a = (5, 10, 15, 20, \dots, 160)$$

$$b = (87, 86, 85, \dots, 56)$$

Use vector arithmetic to multiply these vectors (element by element) and call the result d . Select subsets of d to identify the following.

- What are the 19th, 20th, and 21st elements of d ?
- List all the elements of d which are less than 2000?
- How many elements of d are greater than 6000?

~ **PROBLEM 2.** Create a character vector named `char` that has capital letters "A", "B", "C", "D" repeated four times (Hint: use function `LETTERS`)

~ **PROBLEM 3.** Create a numerical vector named `numbers` using the `runif()` function consisting of 30 values between -2 and 10. Read help function on `runif` first.

- Select all values greater than 3. How many do you get? (Hint: use function `length`)
- What is the sum of the values greater than 3?
- Re-create another vector using the same code. How many values are now greater than 3?

Matrices

- Matrices can be created with the `matrix()` function, specifying all elements (column-by-column) as well as the number of rows and number of columns.

```
> A <- matrix(1:12, nr = 3, nc = 4)
> A
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	4	7	10
[2,]	2	5	8	11
[3,]	3	6	9	12

You may also specify the rows (or columns) as vectors, and then combine them into a matrix using the `rbind()` or `cbind()` functions.

```
> a <- c(1,2,3)
> a
```

```
[1] 1 2 3
```

```
> b <- c(10, 20, 30)
> b
```

```
[1] 10 20 30
```

```
> c <- c(100, 200, 300)
> c
```

```
[1] 100 200 300
```

```
> d <- c(1000, 2000, 3000)
> d
```

```
[1] 1000 2000 3000
```

```
> B <- rbind(a, b, c, d) # Create a matrix with vectors a, b, c, d as rows.
> B
```

	[,1]	[,2]	[,3]
a	1	2	3
b	10	20	30
c	100	200	300
d	1000	2000	3000

The name of the function `rbind()` stands for “row bind”; indeed, it binds rows together.

```
> C <- cbind(a, b, c, d) # Create a matrix with vectors a, b, c, d as
  columns.
> C
```

	a	b	c	d
[1,]	1	10	100	1000
[2,]	2	20	200	2000
[3,]	3	30	300	3000

The name of the function `cbind()` stands for “column bind”; indeed, it binds columns together.

- Use the function `t()` to transpose a matrix:

```
> TC <- t(C) # Transpose matrix C
> TC == B # The resulting matrix should be precisely B.
```

```
      [,1] [,2] [,3]
a TRUE TRUE TRUE
b TRUE TRUE TRUE
c TRUE TRUE TRUE
d TRUE TRUE TRUE
```

- To select a subset of a matrix, use the square brackets and specify rows before the comma, and columns after the comma.

```
> C[1:2,] # Select rows 1 and 2.
```

```
      a b c d
[1,] 1 10 100 1000
[2,] 2 20 200 2000
```

```
> C[,c(1,3)] # Select columns 1 and 3.
```

```
      a c
[1,] 1 100
[2,] 2 200
[3,] 3 300
```

```
> C[1:2,c(1,3)] # Select column 1 and 3 from rows 1 and 2.
```

```
      a c
[1,] 1 100
[2,] 2 200
```

- Note what happens if you multiply two matrices:

```
> B*t(C)
```

```
      [,1] [,2] [,3]
a 1e+00 4e+00 9e+00
b 1e+02 4e+02 9e+02
c 1e+04 4e+04 9e+04
d 1e+06 4e+06 9e+06
```

The matrices are multiplied element by element.

- Maybe you've learned about *matrix multiplication* in high school. In R matrix multiplication is performed with the operator `%*%`. Remember that, in matrix multiplication, the order matters!

```
> B%*%C
```

```
      a      b      c      d
a   14    140    1400 1.4e+04
b   140   1400   14000 1.4e+05
c   1400  14000  140000 1.4e+06
d  14000 140000 1400000 1.4e+07
```

```
> C%*%B
```

```
      [,1] [,2] [,3]
[1,] 1010101 2020202 3030303
[2,] 2020202 4040404 6060606
[3,] 3030303 6060606 9090909
```

- You may apply a function to the rows or columns of a matrix using the **apply()** function.

```
> C
      a  b   c   d
[1,] 1 10 100 1000
[2,] 2 20 200 2000
[3,] 3 30 300 3000
```

```
> sum(C)
[1] 6666
```

```
> apply(C, 1, sum) # Sums of the rows.
[1] 1111 2222 3333
```

```
> apply(C, 2, sum) # Sums of the columns.
      a  b   c   d
6    60  600 6000
```

If it is not clear to you what the “1” and “2” stand for in the above example, simply type “**?apply**” to look it up in the documentation.

Factors

We noted above that R uses the *factor* data type to represent categorical data. The possible values of a factor are called *levels*. For example: suppose a variable `sex` can assume values `male` or `female`. Then this variable should be encoded in R as a factor `sex` with two levels, `male` and `female`.

- Sometimes people confuse the factor type with the character type. Characters are often used for labels in graphs, column names or row names. Factors must be used when you want to represent a categorical variable and want to analyze it. Factor objects can be created from character objects or from numeric objects, using the function `factor()`. For example, to create a vector of length five of type factor, do the following:

```
> sex <- c("male","male","female","male","female") # Character type
> sex

[1] "male"   "male"   "female" "male"   "female"
```

The object `sex` is now a character object. You need to transform it to factor:

```
> sex <- factor(sex)
> sex

[1] male   male   female male   female
Levels: female male
```

Use the function `levels` to see the different levels of a factor variable:

```
> levels(sex)

[1] "female" "male"
```

Note that the result of the `levels` function is of type character.

- Another way to generate the `sex` variable is as follows:

```
> sex <- c(1,1,2,1,2)
> sex

[1] 1 1 2 1 2
```

Now, the object `sex` is an integer variable. To use it as a categorical variable, you need to transform it to a factor:

```
> sex <- factor(sex)
> sex

[1] 1 1 2 1 2
Levels: 1 2
```

Note that now the R output states that the vector is a factor with two levels, 1 and 2. You could rename the levels, such that level “1” becomes “male” and level “2” becomes “female”:

```
> levels(sex) <- c("male","female")
> sex

[1] male   male   female male   female
Levels: male female
```

- A *very* useful R function that takes some time to get used to is `tapply()`. In many cases, your data consist of measurement on individuals/cases that belong to different groups/treatments/levels. Technically, you'll have a vector containing all measurements, and a vector of type factor that specifies to which group the case belongs. The function `tapply()` allows you to apply a function, such as `mean()` or `sd()`, to your data, similarly to the function `apply()`. But the advantage of `tapply()` is that it will apply the function separately to each group/level.

An example may help. Suppose we have obtained the age of 5 students:

```
> age <- c(18, 25, 23, 29, 37)
```

Their respective genders are given by the `sex` variable you defined above:

```
> sex
[1] male  male  female male  female
Levels: male female
```

Now, we would like to know the mean age of the males and the mean age of the women. This can be obtained in one go:

```
> (meanAge <- tapply(age, sex, mean))
male female
 24      30
```

The mean age of the males is 24, the mean age of the females is 30.

Lists

- Another basic structure in R is a list. The main advantage of lists is that the “columns” (they’re not really ordered in columns any more, but are more a collection of vectors) don’t have to be of the same length, unlike matrices or data frames (see below). Actually a list is a matrix where two rules are over-written: i) In every column you can have a different type of data, e.g. in column one integers, and in column two characters. ii) not all the columns need to have same length. The following lines construct a list by giving names and values. This list should appear in your work space as well.

```
> L <- list(first="A", second=c("male", "female"), third=seq(0, 1,
length=5))
```

```
> L
$first
[1] "A"
$second
[1] "male" "female"
$third
[1] 0.00 0.25 0.50 0.75 1.00
```

```
> names(L)
```

```
[1] "first" "second" "third"
```

- Now let’s see an example of how to work with lists. If we want to know the values of the numbers plus 1:

```
> L$third + 1
[1] 1.00 1.25 1.5 1.75 2
```

Notice how we access the elements in a list using the "\$" symbol. However, doing so we did not really change the values in L. To do that we need to assign the new values to L:

```
> L$third <- L$third + 1
```

Data frames

- All elements of a matrix must be of the same mode (numeric, character, logical, etc.). If you try to put different modes in a matrix, all elements will be coerced to the most general—usually the character mode.

```
> Person <- c("Bob", "Bill", "Betty")
> TestA <- c(80, 95, 92)
> TestB <- c(40, 87, 90)
> grades <- cbind(Person, TestA, TestB)
> grades
```

```
      Person TestA TestB
[1,] "Bob"   "80"  "40"
[2,] "Bill"  "95"  "87"
[3,] "Betty" "92"  "90"
```

Note that all numbers are now interpreted as character strings. That is usually not what you want.

- The solution to this problem is another complex object called a *data frame*. The data frame views rows as cases and columns as variables. All elements in a column must be of the same mode, but different columns may be of different modes.

```
# Create a data frame from vectors:
> grades.df <- data.frame(Person, TestA, TestB)
> grades.df
```

```
  Person TestA TestB
1  Bob     80     40
2  Bill     95     87
3 Betty     92     90
```

- Note that the columns of data frames have *names*. These can be used to extract values from the data frame.

```
> grades.df$TestB
```

```
[1] 40 87 90
```

```
> grades.df$Person[2]
```

```
[1] Bill
```

```
Levels: Betty Bill Bob
```

You can change the names of the columns:

```
> colnames(grades.df) <- c("Name", "Test1", "Test2")
> grades.df
```

```
  Name Test1 Test2
1  Bob     80     40
2  Bill     95     87
3 Betty     92     90
```

- The function `str()` returns the structure of any R object.

```
# Name is a factor with 3 levels, Test1 and Test2 are numerical vectors.
> str(grades.df)
```

```
'data.frame':  3 obs. of  3 variables:
 $ Name : Factor w/ 3 levels "Betty","Bill",...: 3 2 1
 $ Test1: num  80 95 92
 $ Test2: num  40 87 90
```

The matrix created with `cbind()` has a different structure: all data are converted to character type.

```
> str(grades) # structure of the matrix created with cbind.
```

```
chr [1:3, 1:3] "Bob" "Bill" "Betty" "80" ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:3] "Person" "TestA" "TestB"
```

- Within RStudio, you can also view data frames as follows:

```
> View(grades.df)
```

Try it!

- Summary functions applied to a data frame will be applied to each column. This may not always work:

```
> mean(grades.df)

[1] NA
Warning message:
In mean.default(grades.df) :
  argument is not numeric or logical: returning NA
```

Note that there is a problem: the mean of the column `Name` is obviously undefined. R returns the value `NA` for “not available”. You may want to specify the column of interest:

```
> mean(grades.df$Test1)

[1] 89
```

- `NA`s have to be carefully handled. For example, let’s include one `NA` in the previous data frame.

```
> grades.df[2,3] <- NA
> grades.df

  Name Test1 Test2
1  Bob    80    40
2  Bill   95     NA
3 Betty   92    90
```

Now, R cannot compute the mean of the `Test2` column:

```
> mean(grades.df$Test2)

[1] NA
```

Therefore, the result is another `NA`. You can tell R to remove the `NA` values:

```
> mean(grades.df$Test2, na.rm = TRUE)

[1] 65
```

Note that the result is the mean of the remaining values.

- Continuous variables can be converted into factors as follows:

```
> grades.df$Test1 <- as.factor(grades.df$Test1)
> grades.df$Test1 # Note the change in the output from R: now Test1 has 3
                  levels

[1] 80 95 92
Levels: 80 92 95

# Calling the structure reveals this change: Test1 is now a factor with 3 levels.
> str(grades.df)

'data.frame':   3 obs. of  3 variables:
 $ Name : Factor w/ 3 levels "Betty","Bill",...: 3 2 1
 $ Test1: Factor w/ 3 levels "80","92","95": 1 3 2
 $ Test2: num  40 NA 90
```

Data Import

- Data files should ideally be set up as text or CSV (comma-separated values) files with rows as cases and columns as variables. Data sets for this course will be found in these formats on Blackboard. Save them to your desktop and use the functions `read.table()` or `read.csv()` to import them into R as data frames. This will require a complete path to the file's location on your computer.
- Finding the path to your file can be annoying. Here's a convenient alternative: the *Import Dataset* button in the "Environment" window of RStudio. (Look it up!) This starts a wizard that guides you through the process. In the end, the wizard sends a command to your console that contains the path and any options that you specified. Don't forget to copy this command back to your script! This way, next time you run the analysis from your script, the file is imported automatically.
- Try to import a text file. First download the data file "lynxhare.txt" from Blackboard. This file contains the Canadian lynx and snowshoe hare pelt-trading records of the Hudson Bay Company, starting in 1852. (This data is downloaded from <https://github.com/bblais/Systems-Modeling-Spring-2015-Notebooks/blob/master/data/>). How to do this depends on the browser that you're using. Possibly you have to click on the file to open it in the browser, and then right-click to choose *Save As . . .*. Then either use the data import wizard or use the following code:

```
> lynx_hare <- read.table("C:/.../lynxhare.txt",
  header = TRUE, sep = ";")
> head(lynx_hare) # The function head() returns the first 7 rows of the
  dataset.
```

```
  year hare lynx
1 1852 80000 2174
2 1853 80000 2106
3 1854 90000 3021
4 1855 69000 4754
5 1856 81000 7324
6 1857 95000 8197
```

```
> str(lynx_hare)
```

```
'data.frame':  88 obs. of  3 variables:
 $ year: int  1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 ...
 $ hare: int  80000 80000 90000 69000 81000 95000 71000 28000 18000 19000 ...

 $ lynx: int  2174 2106 3021 4754 7324 8197 6913 4772 2383 1540 ...
```

- Import a CSV file (you can find the data file "lynxhare.csv" on Blackboard):

```
> lynx_hare <- read.csv("lynxhare.csv",
  sep = ";")
```

- If you need to import data from an Excel file, you can simply save the Excel sheet as a CSV file. In Excel, use *Save As* and select CSV or comma-separated values. Then use the steps outlined above to import the CSV file. Now let us

explore our data a little bit. For example, what are the mean and standard deviation of hare population? We can easily find out about this by:

```
>mean(lynx_hare$hare)
```

```
[1] 47452.38
```

```
>sd(lynx_hare$hare)
```

```
[1] 36885.5
```

Let us now repeat it for lynx population:

```
>mean(lynx_hare$lynx)
```

```
[1] NA
```

NA is a special character in R and stands for not applicable/not available. Because in some years the data is missing for lynx population, we get the NA result. To ignore the years when the data was missing:

```
>mean(lynx_hare$lynx, na.rm = TRUE)
```

```
>sd(lynx_hare$lynx, na.rm = TRUE)
```

```
[1] 2478.327
```

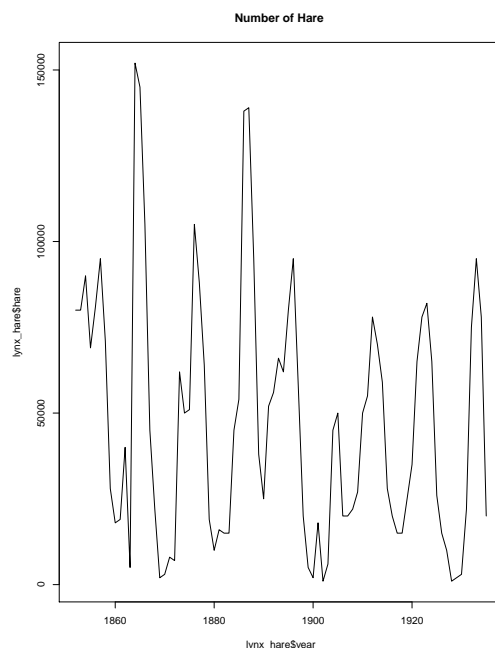
```
[1] 1910.491
```

Graphics

- R has built-in functions to automatically plot many standard statistical graphics. Histograms and box plots may be generated with `hist()` and `boxplot()`, respectively.
- Here's an example. To start with let's make a simple plot of the number of hare that you used before. `plot()` function takes to vectors and uses them for the x and y values:

```
> plot(x=lynx_hare$year, y=lynx_hare$hare, main = "Number of Hare ",
      type=="l")
# main = "." sets the title of the plot.
# type="l" allows for plotting with lines. If you want to plot with points, use type
  = "p"
```

Hopefully, you should be able to see this output:



- Once you've created a graphic that you're happy with, you can copy the entire thing. In the graphic window, click on export. You have three choices: *save as image*, *save as PDF*, and *copy to clipboard*. Image and PDF allow you to save the graph in a portable file. Copy to clipboard allows you to copy and paste the graph, *e.g.* into Ms Word. Simply click on *copy to clipboard* and on *copy plot*. Then paste it into Word and adjust its size.

Try it: save the plot you created above to a PDF file!

- Often, your data set contains various groups, and you would like to make a box plot for each group. Using `boxplot`, that is easy. Here's an example.

Let us load a new data set from Black Board on the CO₂ levels measured in different years and per month. This data set is in the file `CO2_scripps.csv`. Read the file and inspect what kind of data you have:


```
>co2<-read.table("CO2_scripps.csv", header=TRUE, sep=";", check.names
  = FALSE)
head(co2)
```

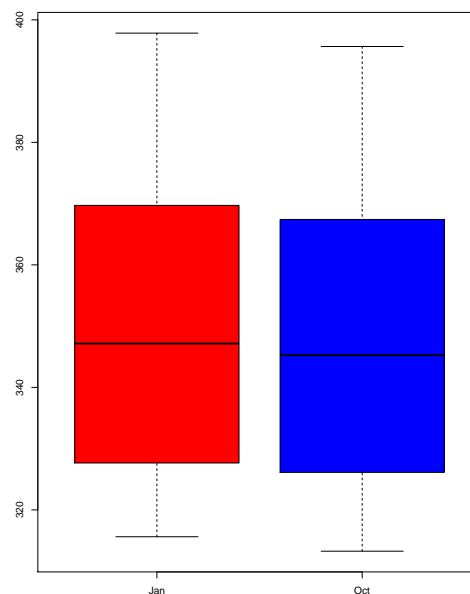
```
Year Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1 1959 315.62 316.38 316.71 317.72 318.29 318.15 316.54 314.80 313.84 313.26
  314.80 315.58
2 1960 316.43 316.97 317.58 319.02 320.03 319.59 318.18 315.91 314.16 313.83
  315.00 316.19
3 1961 316.93 317.70 318.54 319.48 320.58 319.77 318.57 316.79 314.81 315.38
  316.10 317.01
4 1962 317.94 318.55 319.68 320.63 321.01 320.55 319.57 317.40 316.25 315.42
  316.69 317.70
5 1963 318.74 319.07 319.86 321.39 322.25 321.48 319.74 317.77 316.21 315.99
  317.12 318.31
6 1964 319.57 320.00 320.75 321.83 322.25 321.89 320.44 318.70 316.70 316.79
  317.79 318.71
```

```
Annual Average 1 315.97 2 316.91 3 317.64 4 318.45 5 318.99 6 319.62
```

Let us make a boxplot to compare the CO₂ levels in four seasons:

```
> boxplot(co2$Jan, co2$Oct, names=c("Jan", "Oct"), col = c("red", "
  blue"))
#names assigns the values to use on the x axis, and the colors give the colors to
  use.
```

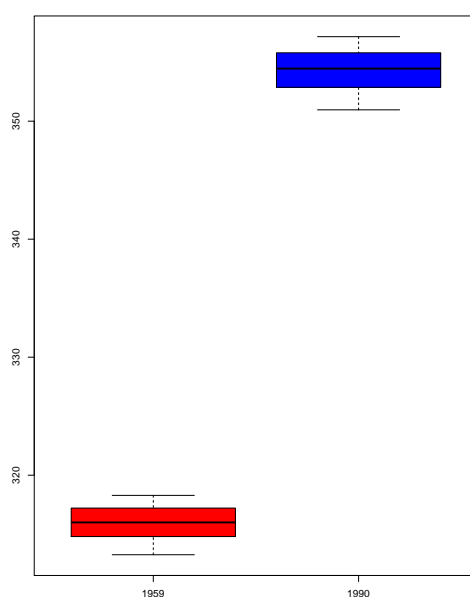
Suprisingly, there is not much of a difference between the two:



Maybe the amount of CO₂ is increasing in time. Let's plot three different years:

```
> boxplot(unlist(co2[co2$Year=="1959",2:13]), unlist(co2[co2$Year=="
  1990",2:13]), names=c("1959", "1990"), col = c("red", "blue"))
#here unlist makes a vector from one row of co2 dataframe
```

Indeed, there is a large increase in the amount of CO₂ in the years between 1959 and 1990:

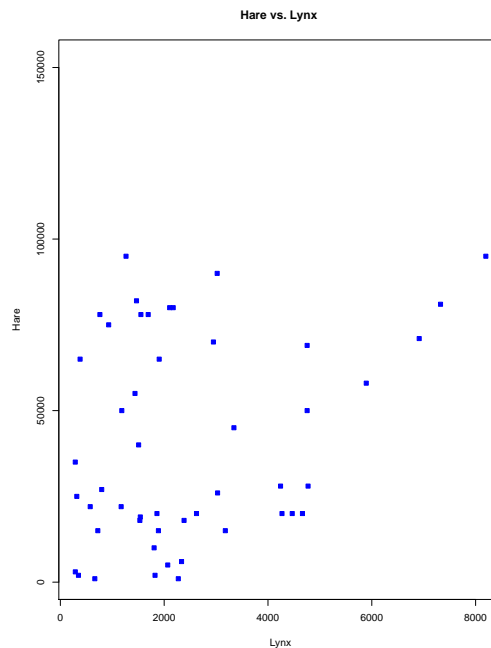


- Scatter plots are plots of two numeric variables. For example, using the `lynx_hare` data set, we can make a scatter plot of the number of hare versus number of lynx:

```
> plot(x=lynx_hare$lynx, y=lynx_hare$hare, main = "Hare vs. Lynx ",
  col="blue",
  xlab = "Lynx", ylab = "Hare", bg="blue", pch=22)
#I use pch=22 and bg="blue" to get filled symbols in color blue.
```

R Packages

One of the strengths of R is that the system can easily be extended. The system allows you to write new functions and share those functions with other users through a so-called “R package”. Other users can then download the package and install the collection of functions as a “library”. The R package may also contain other R objects, for example data sets or documentation. There is a lively R user community and many R packages have been written and made available on CRAN for other users. Just a few examples: there are packages for drawing maps, drawing graphics, exporting objects to HTML, and the list goes on and on. When you download R, a number of packages (around 30) are downloaded by default.



- To use a function from an R package, that package has to be attached to your system. When you start R, not all of the downloaded packages are attached; only seven are by default. To attach (load) another package, use the **library()** function. For instance, to load the package "ggplot2", run:

```
> library(ggplot2) # Attach or "load" a library.
```

- Use the function **search()** to see a list of packages that are currently attached to the system:

```
> search()

[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:datasets" "package:utils"
[7] "package:methods" "Autoloads" "package:base"
```

This list is also called the *search path*. The first element of the output is ".GlobalEnv", which is the current workspace of the user.

- You will occasionally need a package that is not yet installed on your computer. If you have a connection to the internet then a package that is available on CRAN can be installed very easily using the function **install.packages()**. For instance, to install the "binom" package, run the following command:

```
# Install the binom package; the quotes '...' or "..." are required.
> install.packages("binom", dependencies = TRUE)
```

Because we specified **dependencies = TRUE**, if the package **binom** uses ("depends on") yet other packages, these are installed as well. After the installation, you still need to attach the package to your current R session to start using it:

```
> library(binom) # Attach or "load" the binom library.
```

- Another example: The library “MASS” provides an object called `shoes`. To access this object, run the following commands:

```
> install.packages("MASS", dependencies = TRUE) # Install the package, if
  you don't have it yet
> library(MASS) # Attach/load the library.
> data(shoes) # this makes available a data frame called "shoes"
# on some systems, you have to add quotes, like this:
> data("shoes")
# Print the object "shoes":
> shoes
```

```
$A
[1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
$B
[1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
```

We stress that, as with all software, you need to download and install packages only once on a given computer (using `install.packages()`). But you need to load it (using `library()`) in every R session in which you wish to use it.

- The function `library()` can also be used to list all the available libraries on your system with a short description. Run the function without any arguments.