Lecture Notes

Institute of Bioinformatics, Johannes Kepler University Linz

Bioinformatics I

Sequence Analysis and Phylogenetics

Winter Semester 2016/2017

by Sepp Hochreiter

Institute of Bioinformatics
Johannes Kepler University Linz
A-4040 Linz, Austria

Tel. +43 732 2468 4520
Fax +43 732 2468 4539
http://www.bioinf.jku.at

Legend

- (→): explained later in the text, forward reference
- *italic*: important term (in most cases explained)

Literature

- D. W. Mount, Bioinformatics: Sequences and Genome analysis, CSHL Press, 2001.
- D. Gusfield, Algorithms on strings, trees and sequences: computer science and cmomputational biology, Cambridge Univ. Press, 1999.
- R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological sequence analysis, Cambridge Univ. Press, 1998.
- M. Waterman, Introduction to Computational Biology, Chapmann & Hall, 1995.
- Setubal and Meidanis, Introduction to Computational Molecular Biology, PWS Publishing, 1997.
- Pevzner, Computational Molecular Biology, MIT Press, 2000.
- J. Felsenstein: Inferring phylogenies, Sinauer, 2004.
- W. Ewens, G. Grant, Statistical Methods in Bioinformatics, Springer, 2001.
- M. Nei, S. Kumar, Molecular Evolution and Phylogenetics, Oxford 2000.
- Blast: `http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html`

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Biological Basics

This chapter gives an overview over the biological basics needed in bioinformatics. Students with a background in biology or life sciences may skip this chapter if they are familiar with cell biology or molecular biology.

The chapter starts with the structure of the eukaryotic cell, then states the "central dogma of molecular biology", explains the DNA, explains the RNA, discusses transcription, explains splicing, introduces amino acids, describes the genetic code, explains translation, and finally summarizes the protein folding process.

1.1 The Cell

Each human consists of 10 to 100 trillions (10^{13} to 10^{14}) of cells which have quite different functions. Muscle cells are needed to transform chemical energy into mechanical energy, nerve cells transport information via electrical potential, liver cells produce enzymes, sensory cells must respond to external conditions, blood cells must transport oxygen, sperm and egg cell are needed for reproduction, connective tissue cells are needed for bone, fat, fibers, etc.

We focus on the *eukaryotic* cells, i.e. complex cells with a nucleus as in mammals, in contrast to *prokaryotic* cells (no nucleus) found in bacteria and archaea (organisms similar to bacteria which live in extreme conditions). Each cell is a very complex organization like a whole country with power plants, export and import products, library, production machines, highly developed organization to keep the property, delivery systems, defense mechanism, information network, control mechanism, repair mechanism, regulation mechanism, etc.

A cell's diameter is between 10 and 30 μm and consists mostly of water inside a membrane "bag". The membrane is a phospholipid bilayer with pores which allow things to go out of and into the cell.

The fluid within a cell is called *"the cytoplasm"* consisting besides the water of free amino acids ($\rightarrow$), proteins ($\rightarrow$), nucleic acids ($\rightarrow$), RNA ($\rightarrow$), DNA ($\rightarrow$), glucose (energy supply medium), and more. The molecules of the cytoplasm are 50% proteins, 15% nucleic acids, 15% carbohydrates (storage devices or building blocks for structures), 10% lipids (structures with water hating tails; needed to build membranes), and 10% other. Inside the cytoplasm there are various structures called *organelles* (with membranes) whereas the remaining fluid is called *"cytosol"* (mostly water).

Organelles:

- *Nucleus*: location of the DNA, transcription and many "housekeeping" proteins (→); center is nucleolus where ribosomal RNA is produced.
- *Endoplasmic Reticulum* (ER): protein construction and transport machinery; smooth ER also participates in the synthesis of various lipids, fatty acids and steroids (e.g., hormones), carbohydrate metabolism.
- Ribosomes (→): either located on the ER or free in the cytosol; machinery for translation (→), i.e. mRNA (→) is transformed into amino acid sequences which fold (→) and become the proteins.
- Golgi Apparatus: glycosylation, secretion; processes proteins which are transported in vesicles (chemical changes or adding of molecules).
- Lysosomes: digestion; contain digestive enzymes (acid hydrolases) to digest macromolecules including lipases, which digest lipids, carbohydrases for the digestion of carbohydrates (e.g., sugars), proteases for proteins, and nucleases, which digest nucleic acids.
- Centrosome: important for cell cycle
- Peroxisomes: catabolic reactions through oxygen; they rid the cell of toxic substances.
- Microtubules: built from tubulin, cell structure elements (size of the cell) and transport ways for transport proteins
- Cytoskeleton: Microtubules, actin and intermediate filaments. These are structure building components.
- *Mitochondria*: energy (ATP (→)) production from food, has its on genetic material and ribosomes (37 genes (→) in humans variants are called "haplotypes" (→)), only maternal inheritance

The only difference between cells is the different proteins they produce. Protein production not only determines the cell type but also body functions, thinking, immune response, healing, hormone production and more. The cells are built of proteins and everything which occurs in the human body is realized by proteins. Proteins are the substances of life. In detail they are

- enzymes catalyzing chemical reactions,
- sensors (pH value, chemical concentration),
- storage containers (fat),
- transporters of molecules (hemoglobin transports O_2),
- structural components of the tissue (tubulin, actin collagen),
- mechanical devices (muscle contraction, transport),
- communication machines in the cell (decoding information, transcription, translation),

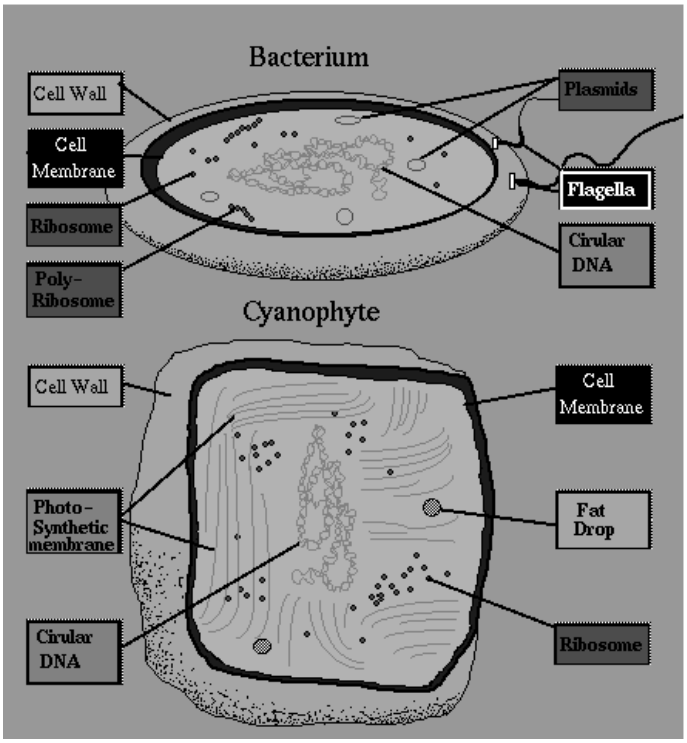


Figure 1.1: Prokaryotic cells of bacterium and cynaophyte (photosynthetic bacteria). Figure from `http://www.zipworld.com.au/~ataraxy/CellBiology/chapter1/cell_chapter1.html`.

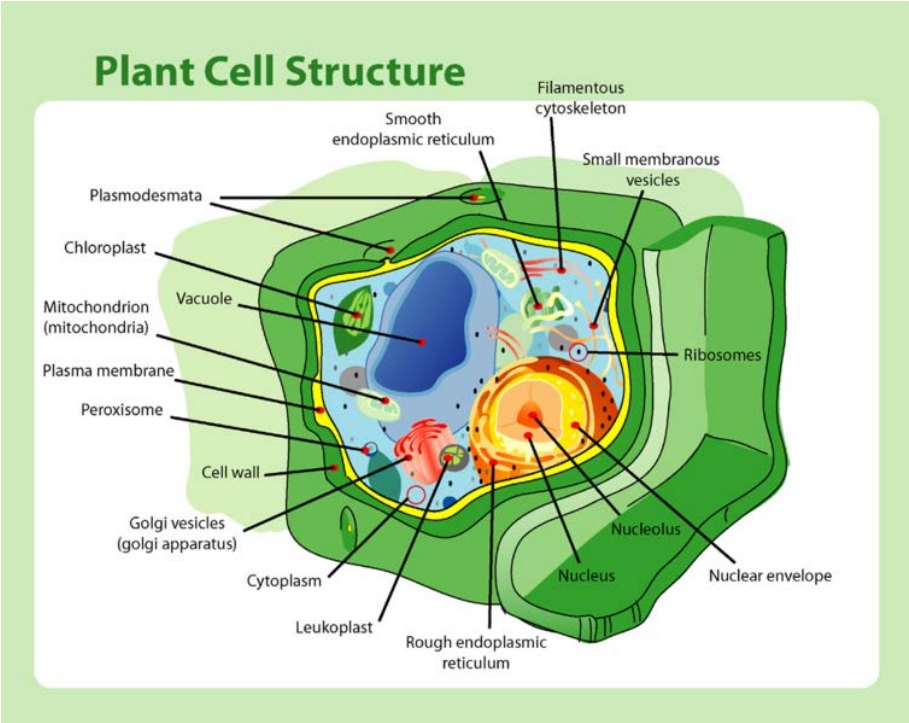


Figure 1.2: Eukaryotic cell of a plant.

- markers
- gene regulation parts (binding to nucleic acids),
- hormones and their receptors (regulation of target cells),
- components of the defense and immune system (antibodies),
- neurotransmitter and their receptors,
- nano-machines for building, reconfiguring, and reassembling proteins, and more.

All information about the proteins and, therefore, about the organism is coded in the DNA (→). The DNA decoding is famous under the term “human genome project” – as all information about an organism is called genome (see Fig. 1.3 for a cartoon of this project).

1.2 Central Dogma of Molecular Biology

The central dogma of molecular biology says "DNA makes RNA makes protein". Therefore, all knowledge about life and its building blocks, the proteins, is coded in the DNA. RNA is the blueprint from parts of the DNA which is read out to be supplied to the protein construction site. The making of RNA from DNA is called *“transcription”* and the making of protein from RNA is called *“translation”*. In eukaryotic cells the DNA is located in the nucleus, but also chloroplasts (in plants) and mitochondria contain DNA.

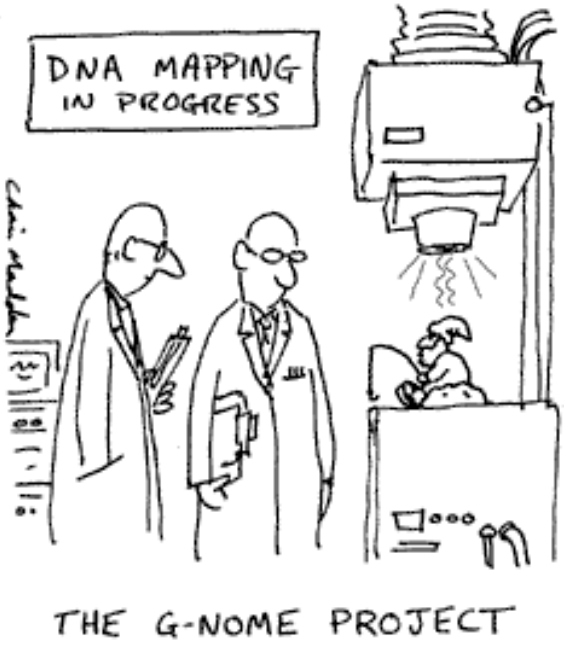


Figure 1.3: Cartoon of the “human genome project”.

The part of the DNA which codes a single protein is called *“gene”*. However scientist were forced to modify the statement "one gene makes one protein" in two ways. First, some proteins consist of substructures each of which is coded by a separate gene. Secondly, through alternative splicing ($\rightarrow$) one gene can code for different proteins.

1.3 DNA

The *deoxyribonucleic acid* (*DNA*) codes all information of life (with some viral exceptions where information is coded in RNA) and represents the human genome. It is a double helix where one helix is a sequence of nucleotides with a deoxyribose (see Fig. 1.5). The single strand DNA ends are called 5’ and 3’ ("five prime" and "three prime"), which refers to the sides of the sugar molecule with 5’ at the phosphates side and 3’ at the hydroxyl group. The DNA is written from 5’ to 3’ and *upstream* means towards the 5’ end and *downstream* towards the 3’ end.

There exist 5 *nucleotides* (see Fig. 1.6): *adenine* (A), *thymine* (T), *cytosine* (C), *guanine* (G), and *uracil* (U). The first 4 are found in the DNA whereas uracil is used in RNA instead of thymine. They form two classes: the *purines* (A, G) and the *pyrimidines* (C, U, T). The nucleotides are often called *nucleobases*.

In the double helix there exist hydrogen bonds between a purine and a pyrimidine where the pairing is A–T and C–G (see Fig. 1.7 and Fig. 1.8). These pairings are called *base pairs*. Therefore each of the two helices of the DNA is complementary to the other (i.e. the code is redundant). The DNA uses a 4-digit alphabet similar to computer science where a binary alphabet is used.

The DNA is condensed in the nucleus through various processes and many proteins resulting in *chromosomes* (humans have 23). The DNA wraps around histones (special proteins) resulting

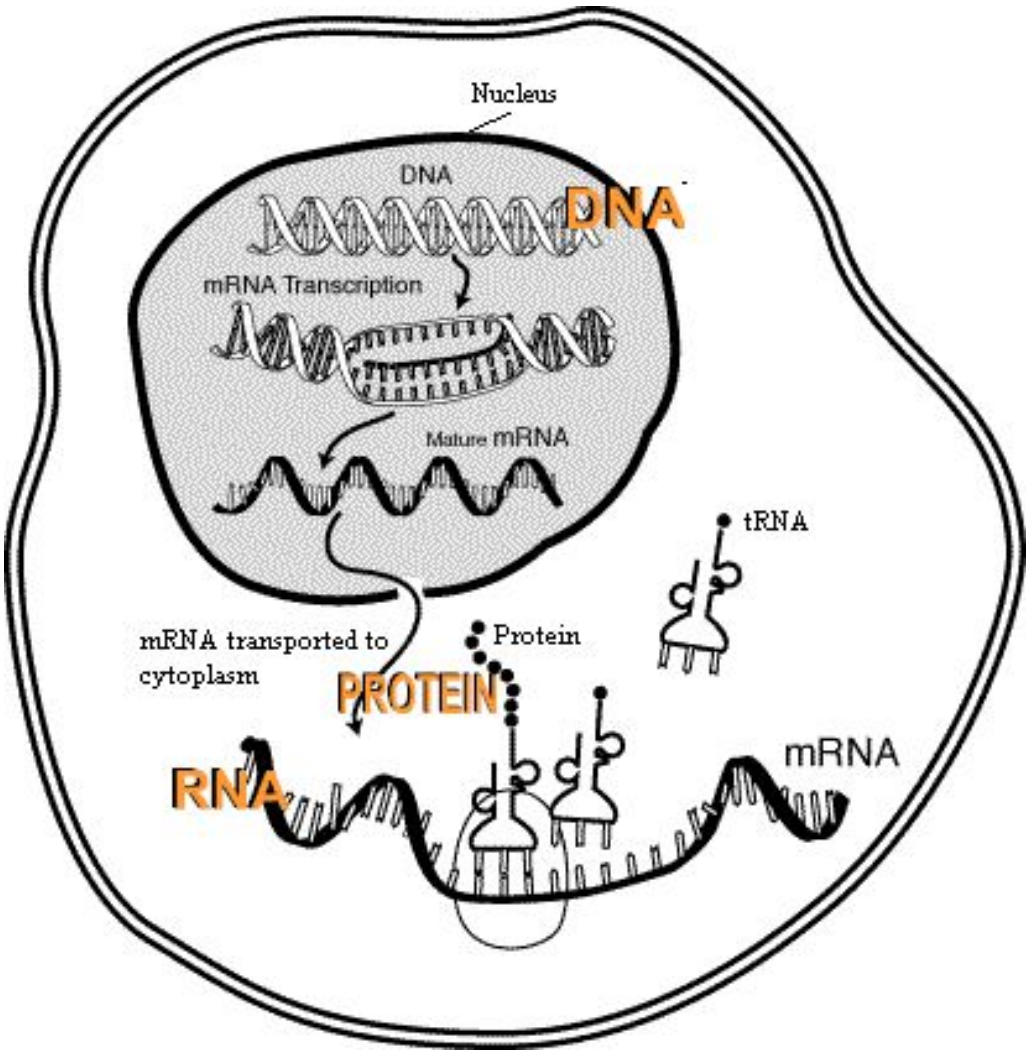


Figure 1.4: Central dogma is depicted.

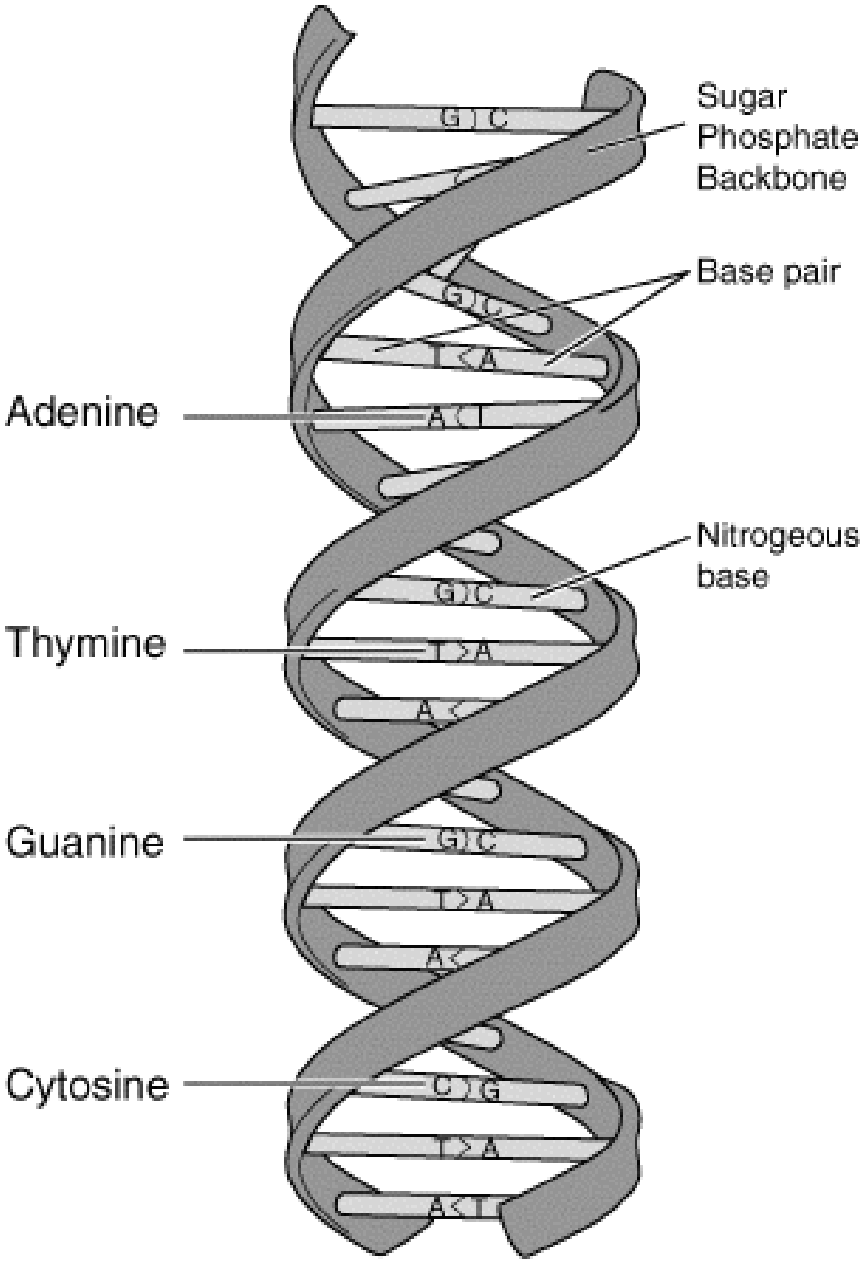


Figure 1.5: The deoxyribonucleic acid (DNA) is depicted.

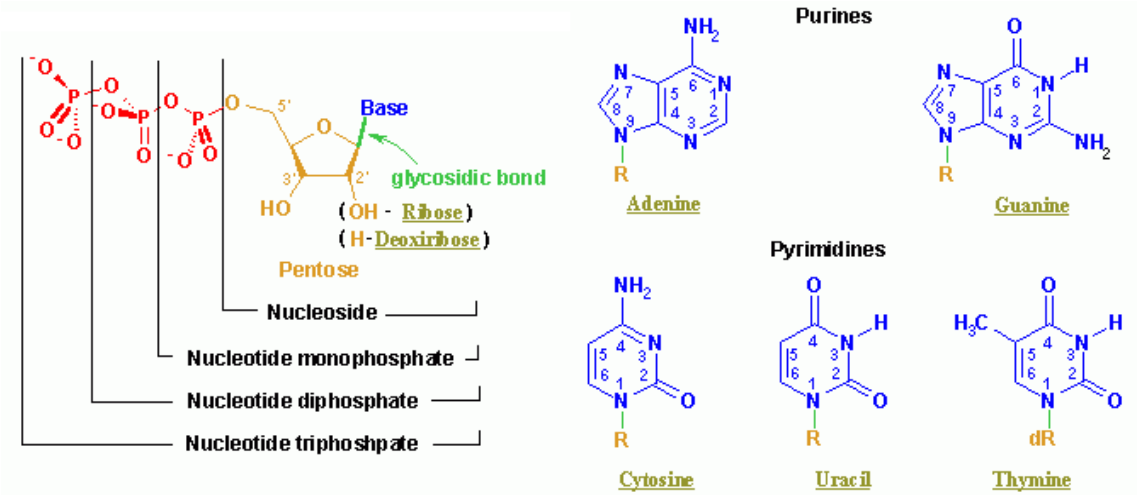


Figure 1.6: The 5 nucleotides.

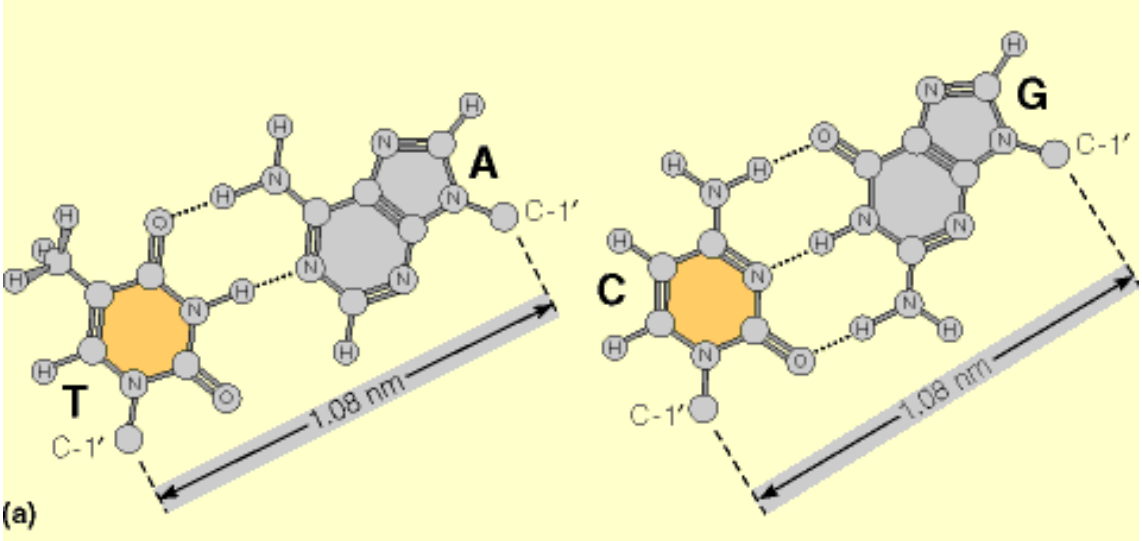


Figure 1.7: The hydrogen bonds between base pairs.

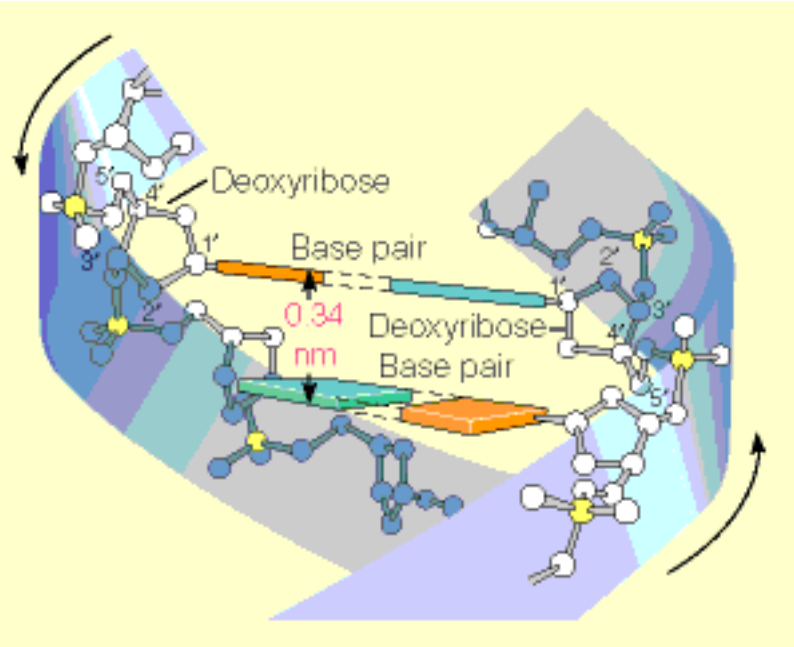


Figure 1.8: The base pairs in the double helix.

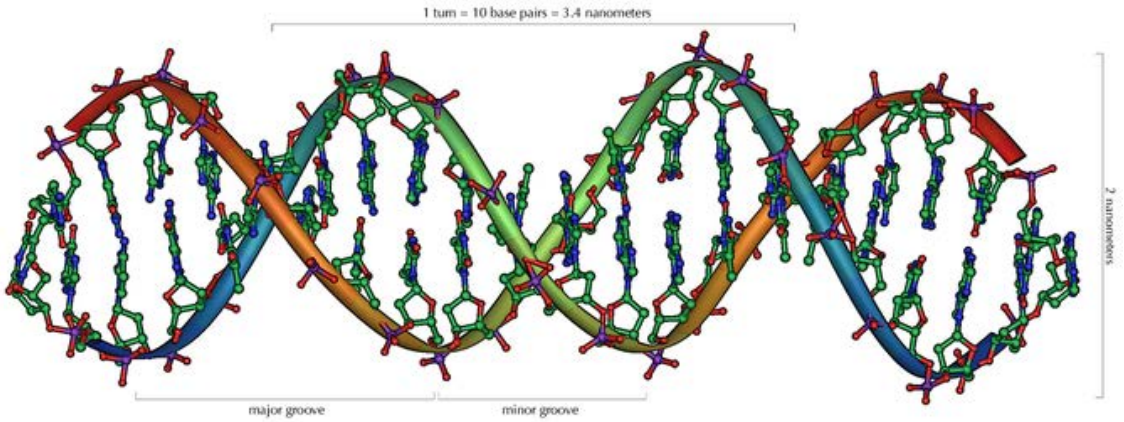


Figure 1.9: The DNA is depicted in detail.

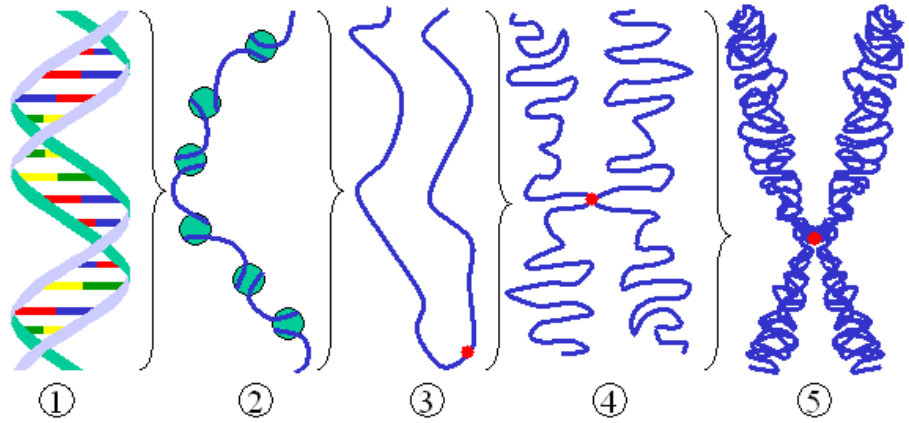


Figure 1.10: The storage of the DNA in the nucleus. (1) DNA, (2) chromatin (DNA with histones), (3) chromatin strand, (4) chromatin (2 copies of the DNA linked at the centromere), (5) chromosome.

in a structure called chromatin. Two strands of chromatin linked together at the centromere give a chromosome. See Fig. 1.10 and Fig. 1.11.

However, the DNA of humans differs from person to person as single nucleotides differ which makes us individual. Our characteristics as eye or hair color, tall or not, ear or nose form, skills, etc is determined by small differences in our DNA. The DNA and also its small differences to other persons is inherited from both parents by 23 chromosomes. An exception is the mitochondrial DNA, which is inherited only from the mother.

If a variation in the DNA at the same position occurs in at least 1% of the population then it is called a *single nucleotide polymorphism* (SNP – pronounced snip). SNPs occur all 100 to 300 base pairs. Currently many research groups try to relate preferences for special diseases to SNPs (schizophrenia or alcohol dependence).

Note, the DNA double helix is righthanded, i.e. twists as a "right-hand screw" (see Fig. 1.12 for an error).

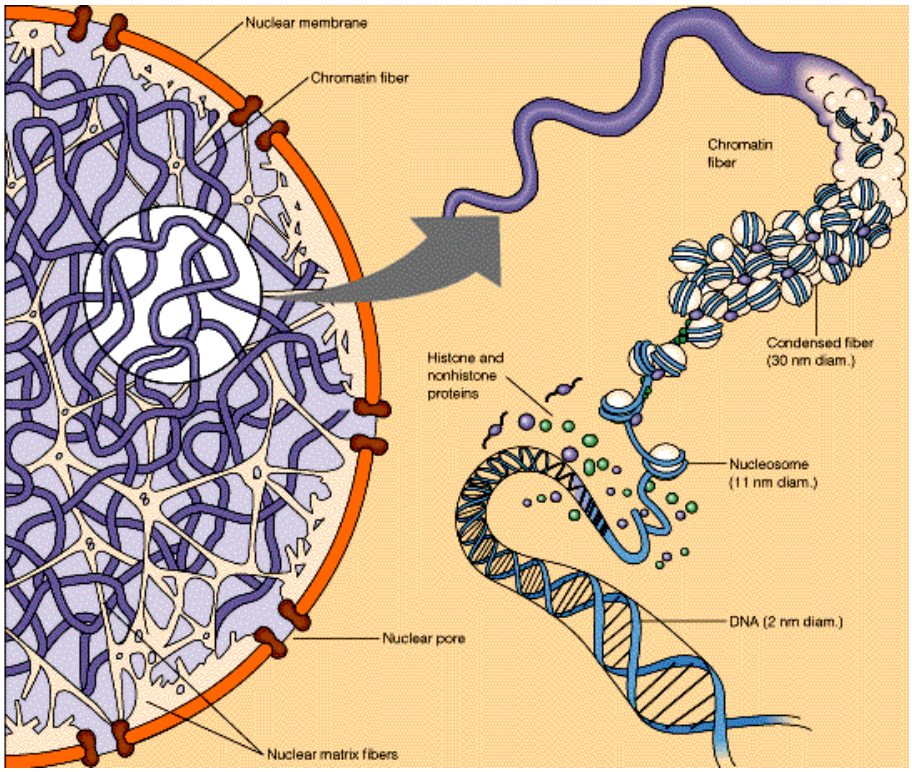


Figure 1.11: The storage of the DNA in the nucleus as cartoon.

Figure 1.12: The DNA is right-handed.

1.4 RNA

Like the DNA the *ribonucleic acid* (*RNA*) is a sequence of nucleotides. However in contrast to DNA, RNA nucleotides contain ribose rings instead of deoxyribose and uracil instead of thymine (see Fig. 1.13). RNA is transcribed from DNA through RNA polymerases (enzymes) and further processed by other proteins.

Very different kinds of RNA exist:

- *Messenger RNA* (*mRNA*): first it is translated from the DNA (eukaryotic pre-mRNA), after maturation (eukaryote) it is transported to the protein production site, then it is transcribed to a protein by the ribosome; It is a "blueprint" or template in order to translate genes into proteins which occurs at a huge nano-machine called ribosome.
- *Transfer RNA* (*tRNA*): non-coding small RNA (74-93 nucleotides) needed by the ribosome to translate the mRNA into a protein (see Fig. 1.14); each tRNA has at the one end complementary bases of a codon (three nucleotides which code for a certain amino acid) and on the other end an amino acid is attached; it is the basic tool to translate nucleotide triplets (the codons) into amino acids.
- *Double-stranded RNA* (*dsRNA*): two complementary strands, similar to the DNA (sometimes found in viruses)
- *Micro-RNA* (*miRNA*): two approximately complementary single-stranded RNAs of 20-25 nucleotides transcribed from the DNA; they are not translated, but build a dsRNA shaped as hairpin loop which is called *primary miRNA* (*pri-miRNA*); miRNA regulates the expression of other genes as it is complementary to parts of mRNAs;
- *RNA interference* (*RNAi*): fragments of dsRNA interfere with the expression of genes which are at some locations similar to the dsRNA
- *Small/short interfering RNA* (*siRNA*): 20-25 nucleotide-long RNA which regulates expression of genes; produced in RNAi pathway by the enzyme Dicer (cuts dsRNA into siRNAs).
- *Non-coding RNA* (*ncRNA*), *small RNA* (*sRNA*), *non-messenger RNA* (*nmRNA*), *functional RNA* (*fRNA*): RNA which is not translated
- *Ribosomal RNA* (*rRNA*): non-coding RNAs which form the ribosome together with various proteins
- *Small nuclear RNA* (*snRNA*): non-coding, within the nucleus (eukaryotic cells); used for RNA splicing
- *Small nucleolar RNA* (*snoRNA*): non-coding, small RNA molecules for modifications of rRNAs
- *Guide RNA* (*gRNA*): non-coding, only in few organism for RNA editing
- *Efference RNA* (*eRNA*): non-coding, intron sequences or from non-coding DNA; function is assumed to be regulation of translation

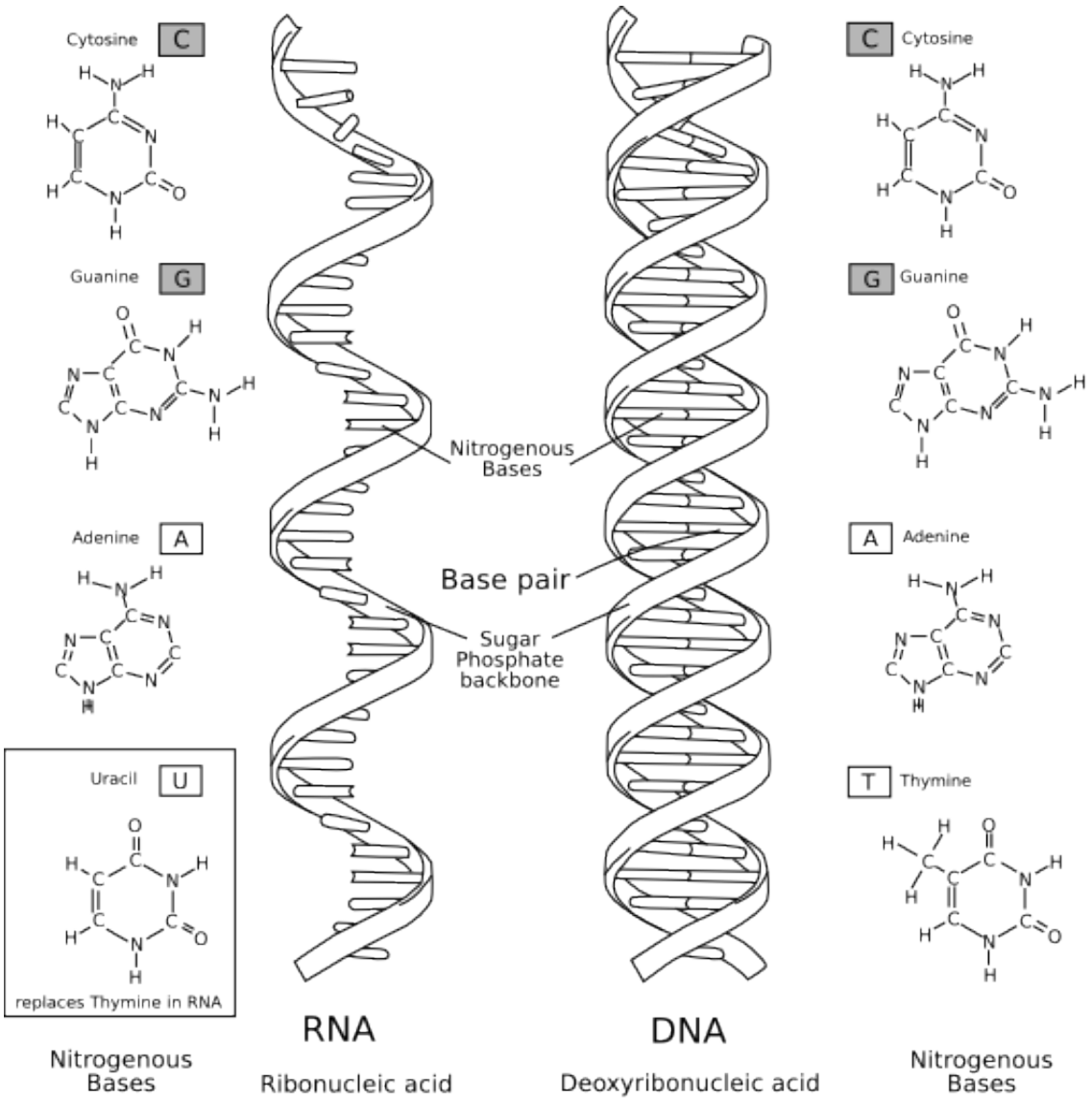


Figure 1.13: The difference between RNA and DNA is depicted.

Figure 1.14: Detailed image of a tRNA.

- *Signal recognition particle* (*SRP*): non-coding, RNA-protein complex; attaches to the mRNA of proteins which leave the cell
- *pRNA*: non-coding, observed in phages as mechanical machines
- *tmRNA*: found in bacteria with tRNA- and mRNA-like regions

1.5 Transcription

Transcription enzymatically copies parts of the DNA sequence by RNA polymerase to a complementary RNA. There are 3 types of RNA polymerase denoted by I, II, and III responsible for rRNA, mRNA, and tRNA, respectively. Transcription reads the DNA from the 3' to 5' direction, therefore the complementary RNA is produced in the 5' to 3' direction (see Fig. 1.15).

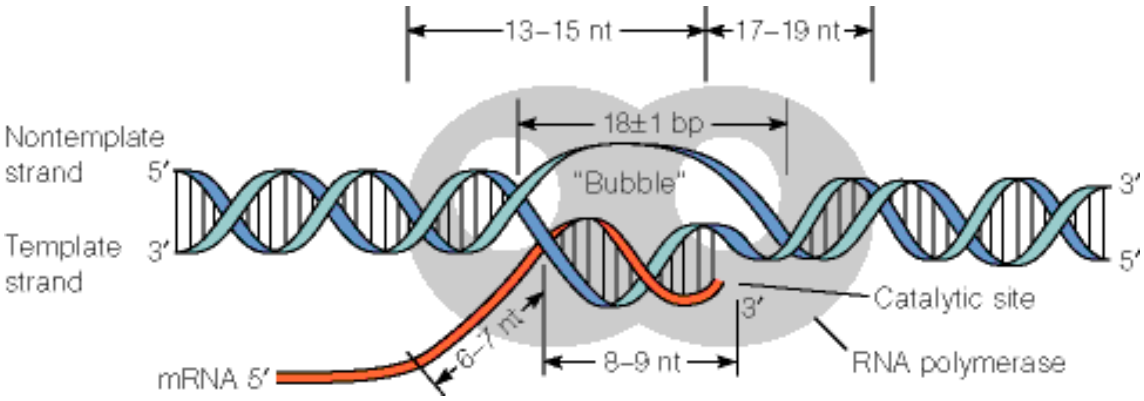


Figure 1.15: The transcription from DNA to RNA is depicted.

Transcription consists of 3 phases: *initiation*, *elongation* and *termination*. We will focus on the eukaryotic transcription (the prokaryotic transcription is different, but easier)

1.5.1 Initiation

The start is marked by a so-called *promoter* region, where specific proteins can bind to. The *core promoter* of a gene contains binding sites for the basal transcription complex and RNA polymerase II and is within 50 bases upstream of the transcription initiation site. It is normally marked through a `TATA` pattern to which a TATA binding protein (TBP) binds. Subsequently different proteins (transcription factors) attach to this TBP which is then recognized by the polymerase and the polymerase starts the transcription. The transcription factors together with polymerase II are the basal transcriptional complex (BTC).

Some promoters are not associated with the `TATA` pattern. Some genes share promoter regions and are transcribed simultaneously. The `TATA` pattern is more conservative as `TATAAA` or `TATATA` which means it is observed more often than the others.

For polymerase II the order of the TBP associated factors is as follows:

- TFIID (Transcription Factor for polymerase II D) binds at the TATA box
- TFIIA holds TFIID and DNA together and enforces the interactions between them
- TFIIB binds downstream of TFIID
- TFIIF and polymerase II come into the game; the σ-subunit of the polymerase is important for finding the promoter as the DNA is scanned, but will be removed later (see Fig. 1.16)
- TFIIE enters and makes polymerase II mobile
- TFIIH binds and identifies the correct template strand, initiates the separation of the two DNA strands through a helicase which obtains energy via ATP, phosphorylates one end of the polymerase II which acts as a starting signal, and even repairs damaged DNA

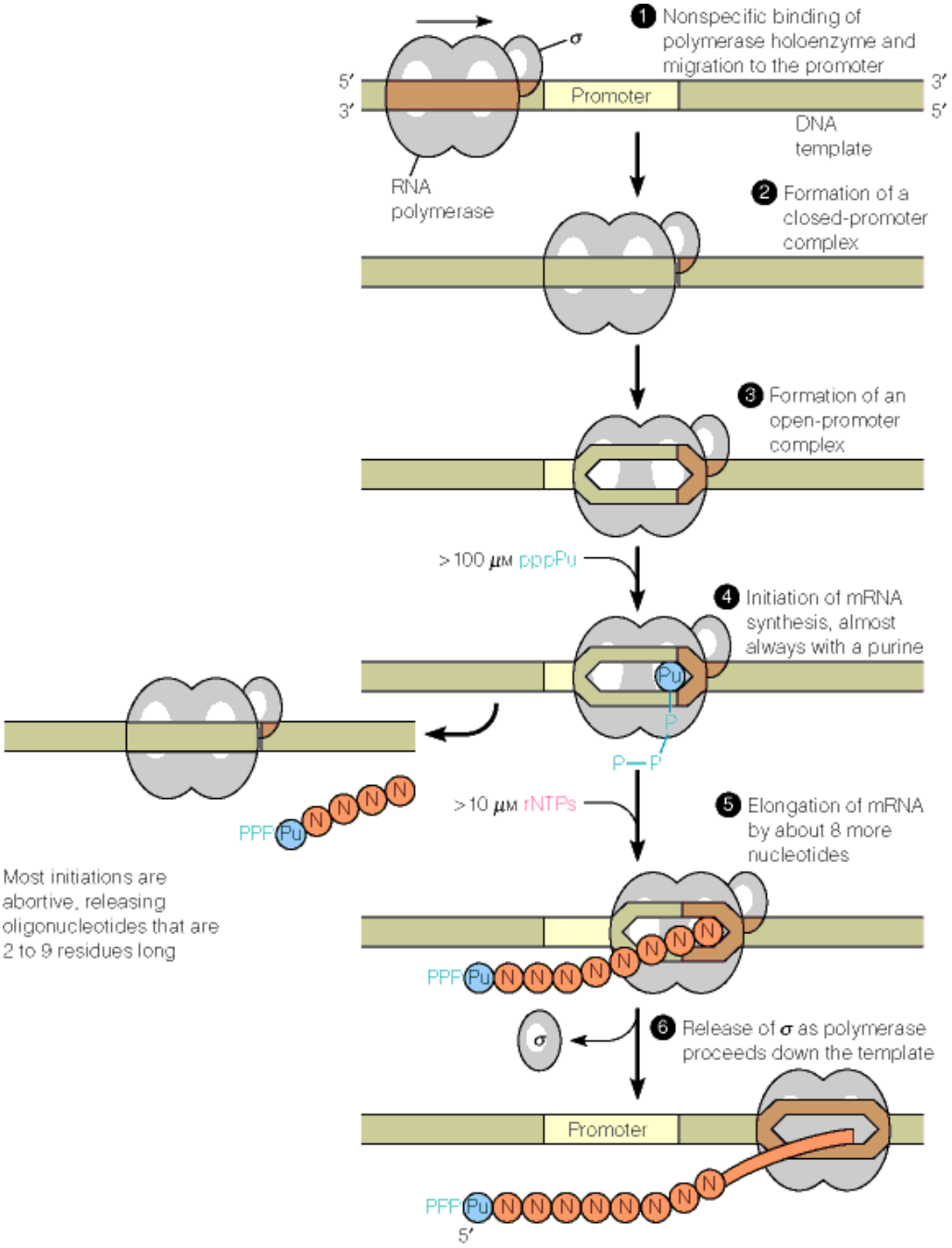


Figure 1.16: The interaction of RNA polymerase and promoter for transcription is shown. (1) The polymerase binds at the DNA and scans it until (2) the promoter is found. (3) polymerase/promoter complex is built. (4) Initiation of the transcription. (5) and (6) elongation with release of the polymerase σ-subunit.

TFIIH and TFIIE strongly interact with one another as TFIIH requires TFIIE to unwind the promoter.

Also the initiation is regulated by interfering proteins and inhibition of the chromatin structure. Proteins act as signals and interact with the promoter or the transcription complex and prevent transcription or delay it (see Fig. 1.17). The chromatin structure is able to stop the initiation of the transcription by hiding the promoter and can be altered by changing the histones.

1.5.2 Elongation

After initiation the RNA is actually written. After the generation of about 8 nucleotides the σ-subunit is dissociated from polymerase.

There are differnent kinds of elongation promoters like sequence-dependent arrest affected factors, chromatin structure oriented factors influencing the histone (phosphorylation, acetylation, methylation and ubiquination), or RNA polymerase II catalysis improving factors.

The transcription can be stimulated e.g. through a `CAAT` pattern to which other transcription factors bind. Further transcription is regulated via upstream control elements (UCEs, 200 bases upstream of initiation). But also far away enhancer elements exist which can be thousands of bases upstream or downstream of the transcription initiation site. Combinations of all these control elements regulate transcription.

1.5.3 Termination

Termination disassembles the polymerase complex and ends the RNA strand. It is a comparably simple process which can be done automatically (see Fig. 1.18). The automatic termination occurs because the RNA forms a 3D structure which is very stable (the stem-loop structure) through the G–C pairs (3 hydrogen bonds) and the weakly bounded A–U regions dissociate.

1.6 Introns, Exons, and Splicing

Splicing modifies pre-mRNA, which is released after transcription. Non-coding sequences called *introns* (intragenic regions) are removed and coding sequences called *exons* are glued together. The exon sequence codes for a certain protein (see Fig. 1.19).

A snRNA complex, the *spliceosome*, performs the splicing, but some RNA sequences can perform autonomous splicing. Fig. 1.20 shows the process of splicing, where nucleotide patterns result in stabilizing a 3D conformation needed for splicing.

However pre-mRNA corresponding to a gene can be spliced in different ways (called *alternative splicing*), therefore a gene can code for different proteins. This is a dense coding because proteins which share the same genetic subsequence (and, therefore, the same 3D substructure) can be coded by a single gene (see Fig. 1.21). Alternative splicing is controlled by various signaling molecules. Interestingly introns can convey old genetic code corresponding to proteins which are no longer needed.

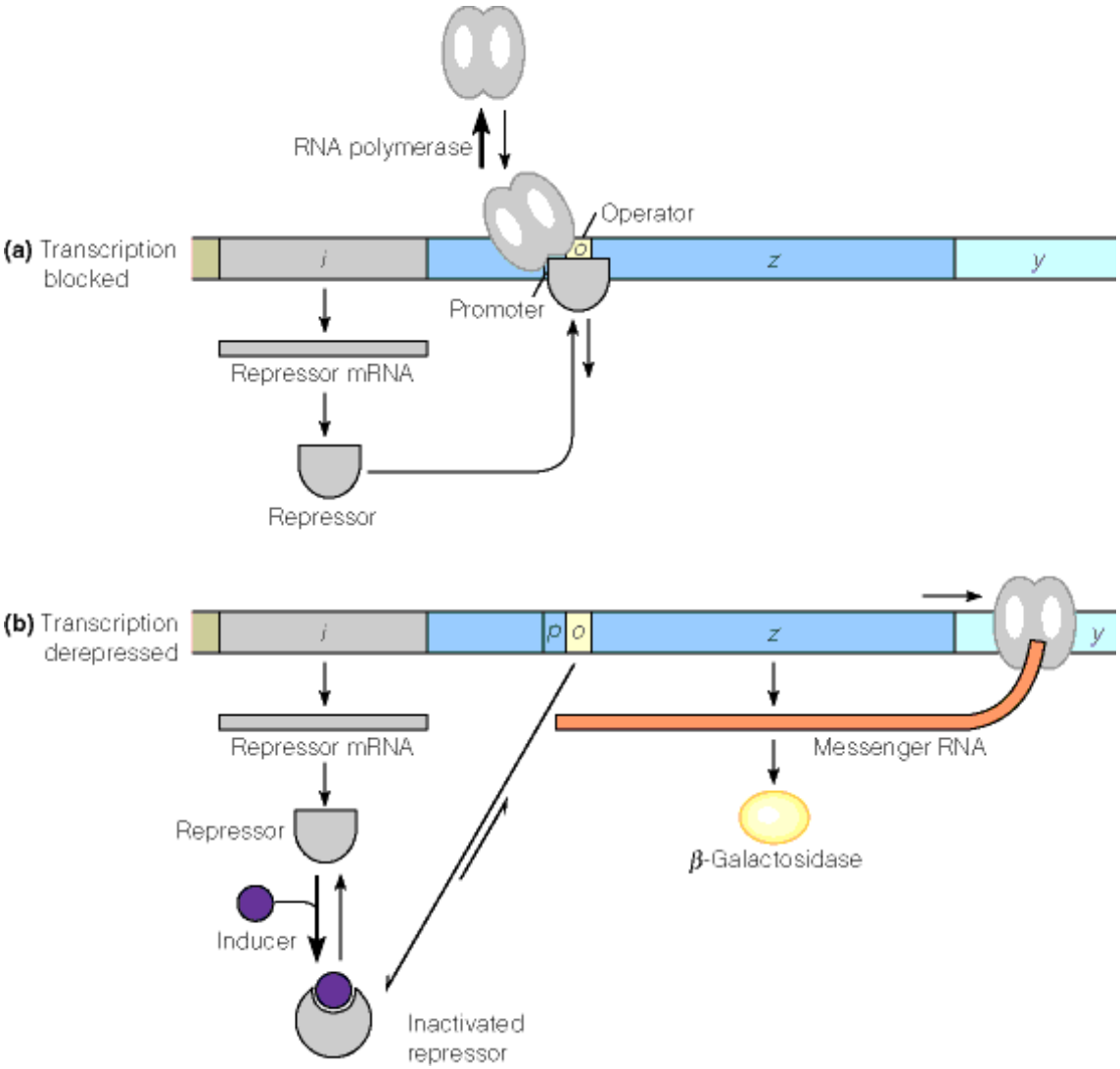


Figure 1.17: Mechanism to regulate the initiation of transcription. Top (a): Repressor mRNA binds to operator immediately downstream the promoter and stops transcription. Bottom (b): Repressor mRNA is inactivate through a inducer and transcription can start.

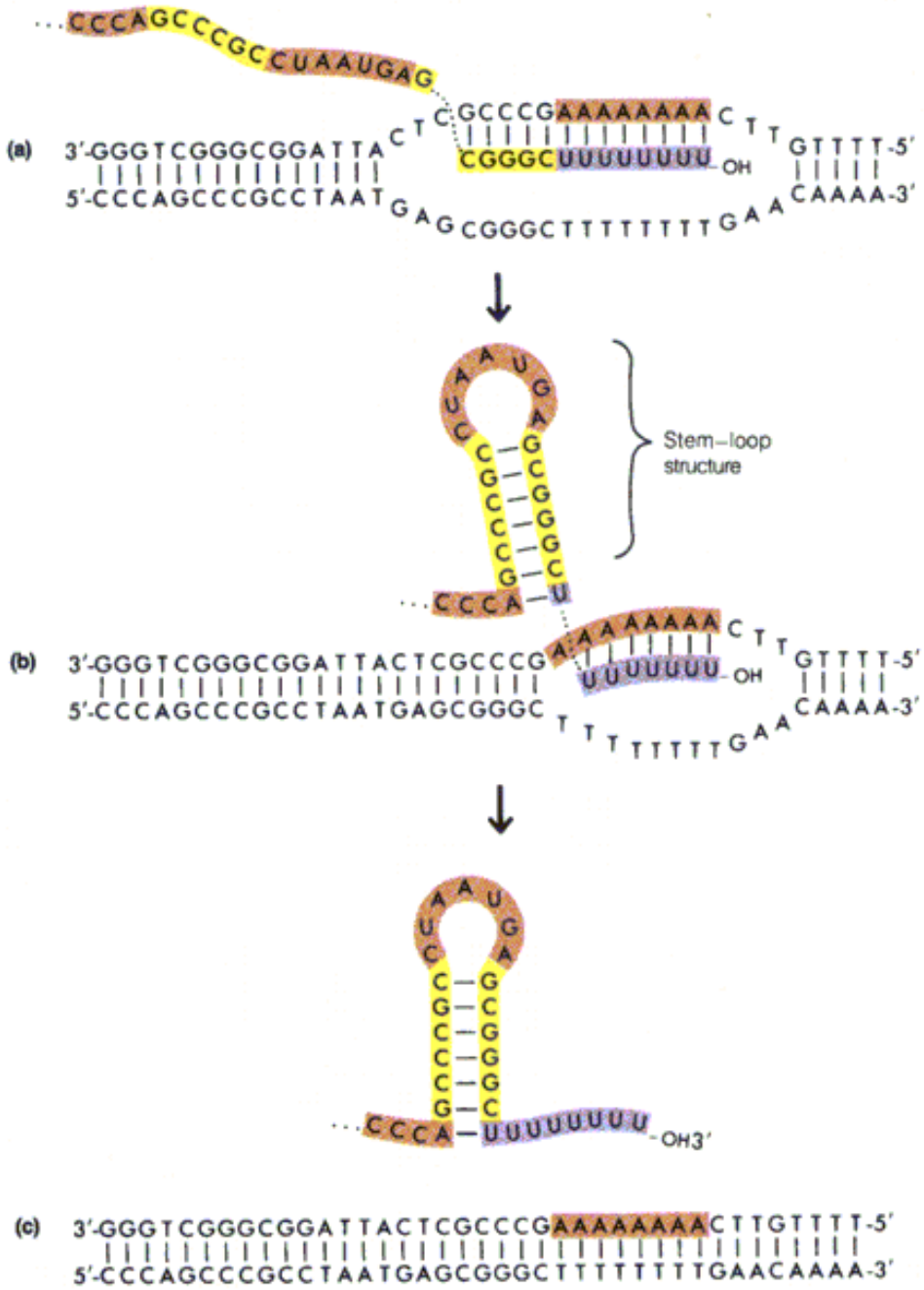


Figure 1.18: Automatic termination of transcription. (a) Region with Us is actual transcribed. (b) The G–C base pairs form a RNA structure which is very stable through the G–C region (the stem-loop structure). (c) the stable structure breaks up the unstable A–U region which is dissociated. Transcription stops.

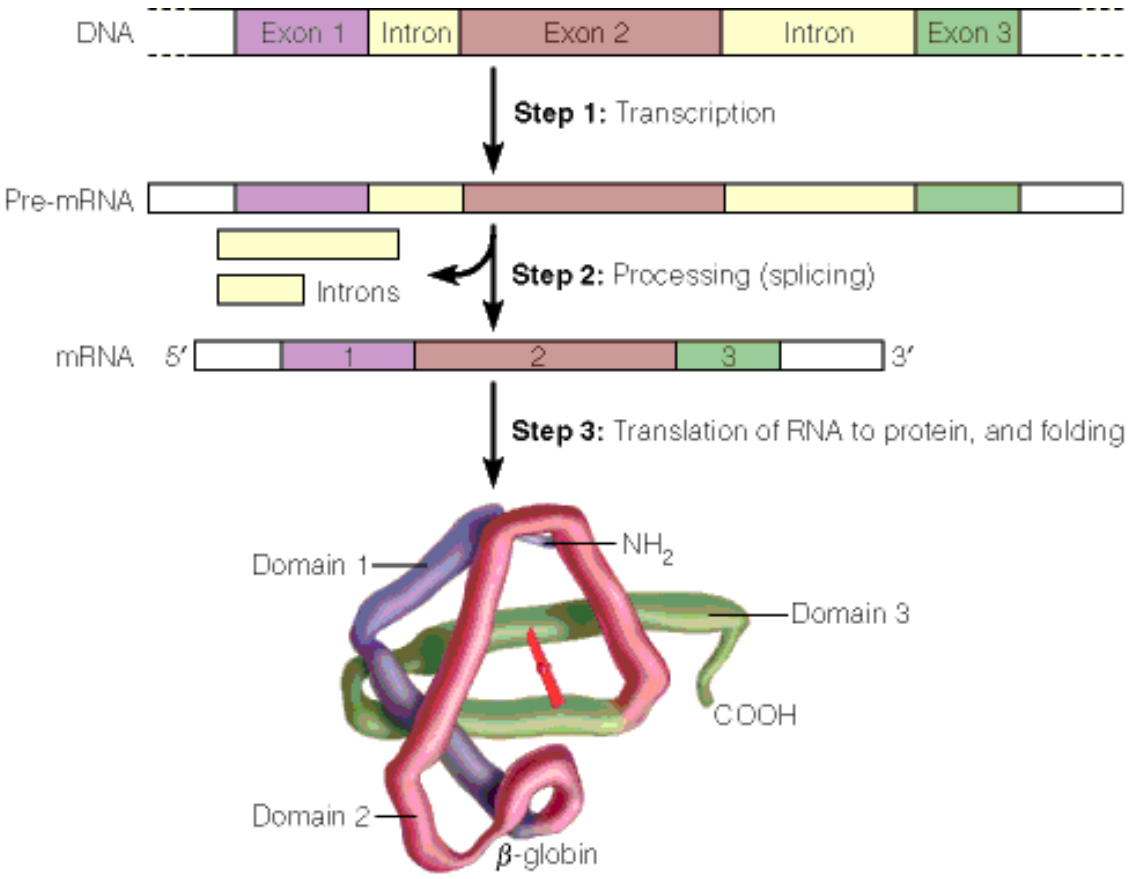


Figure 1.19: Example for splicing: hemoglobin.

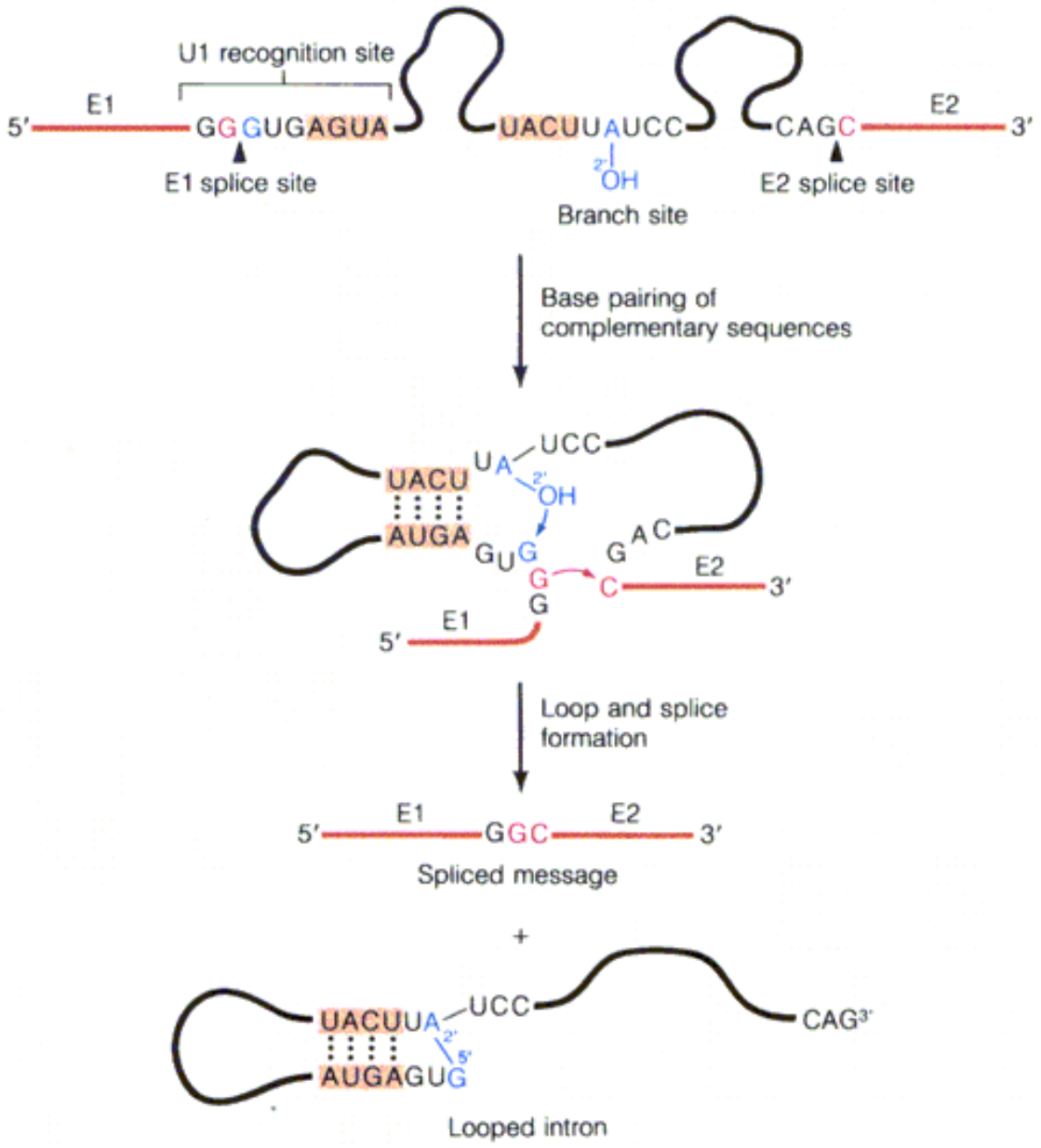


Figure 1.20: Splicing event. Nucleotide pattern stabilize a 3D RNA complex which results in splicing out the intron.

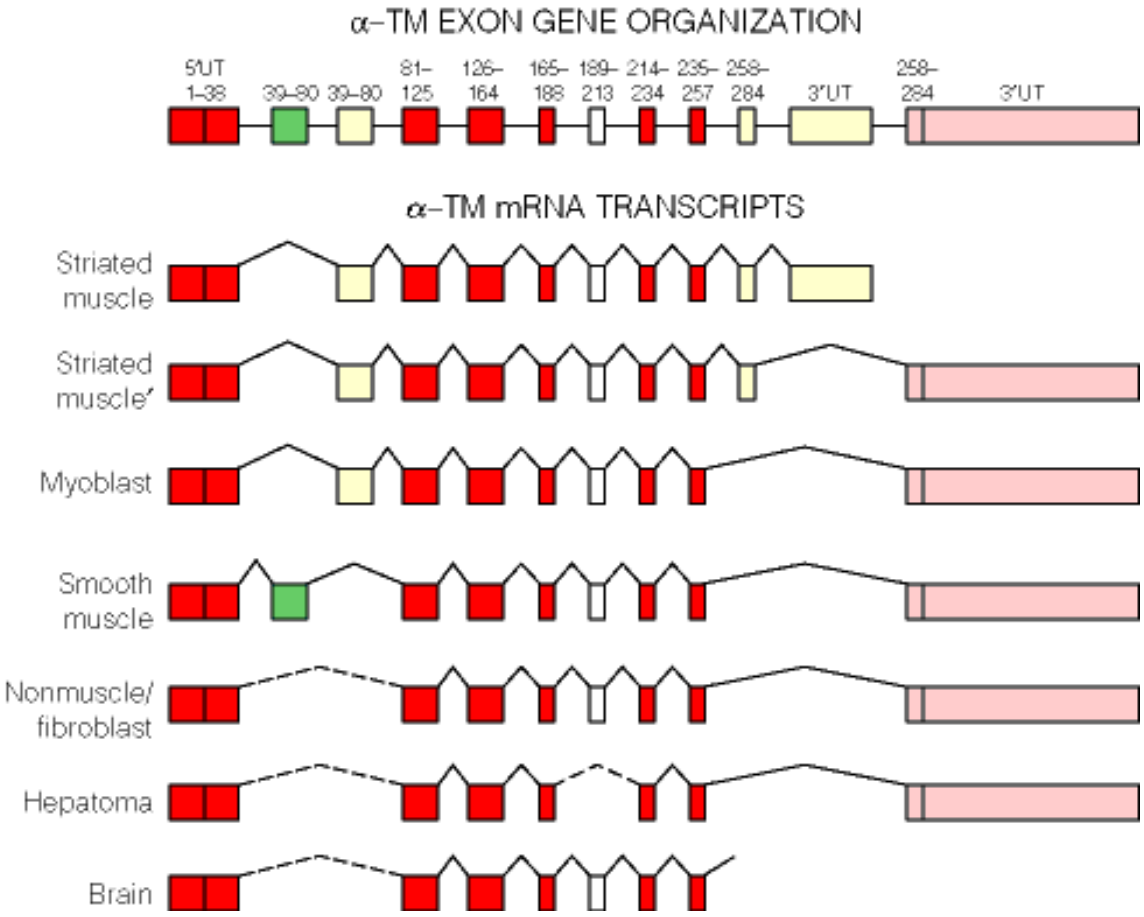


Figure 1.21: Example of alternative splicing. Different proteins are built from one gene through splicing.

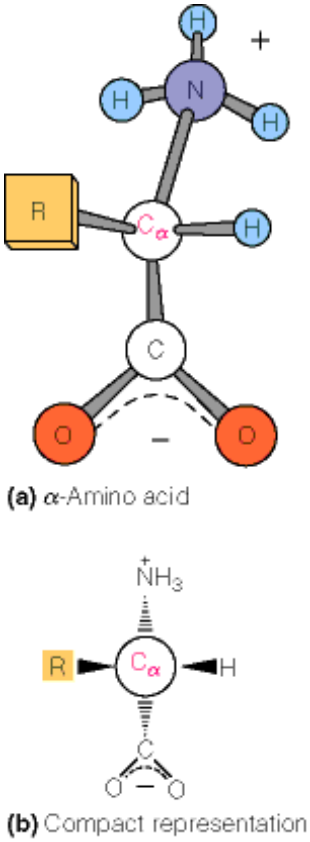


Figure 1.22: A generic cartoon for an amino acid. "R" denotes the side chain which is different for different amino acids – all other atoms are identical for all amino acids except for proline.

1.7 Amino Acids

An *amino acid* is a molecule with amino and carboxylic acid groups (see Fig. 1.22).

There exist 20 standard amino acids (see Fig. 1.23).

In the following properties of amino acids are given like water hating (hydrophobic) or water loving (hydrophilic) (see Tab. 1.2 and Tab. 1.1), electrically charged (acidic = negative, basic = positive) (see Tab. 1.1). The main properties are depicted in Fig. 1.24. Hydrophobic amino acids are in the inside of the protein because it is energetically favorable. Only charged or polar amino acids can build hydrogen bonds with water molecules (which are polar). If all molecules which cannot form these hydrogen bonds with water are put together then more molecules can form hydrogen bonds leading to an energy minimum. Think of fat on a water surface (soup) which also forms clusters. During folding of the protein the main force is the hydrophobic effect which also stabilizes the protein in its 3D structure. Other protein 3D-structure stabilizing forces are salt-bridges which can exist between a positively and negatively charged amino acid. Further disulfide bridges (Cys and Met) are important both for folding and 3D-structure stability. The remaining 3D-structure forming forces are mainly hydrogen bonds between two backbones or two side-chains as well as between backbone and side-chain.

A sequence of amino acids, i.e. residues, folds to a 3D-structure and is called protein. The

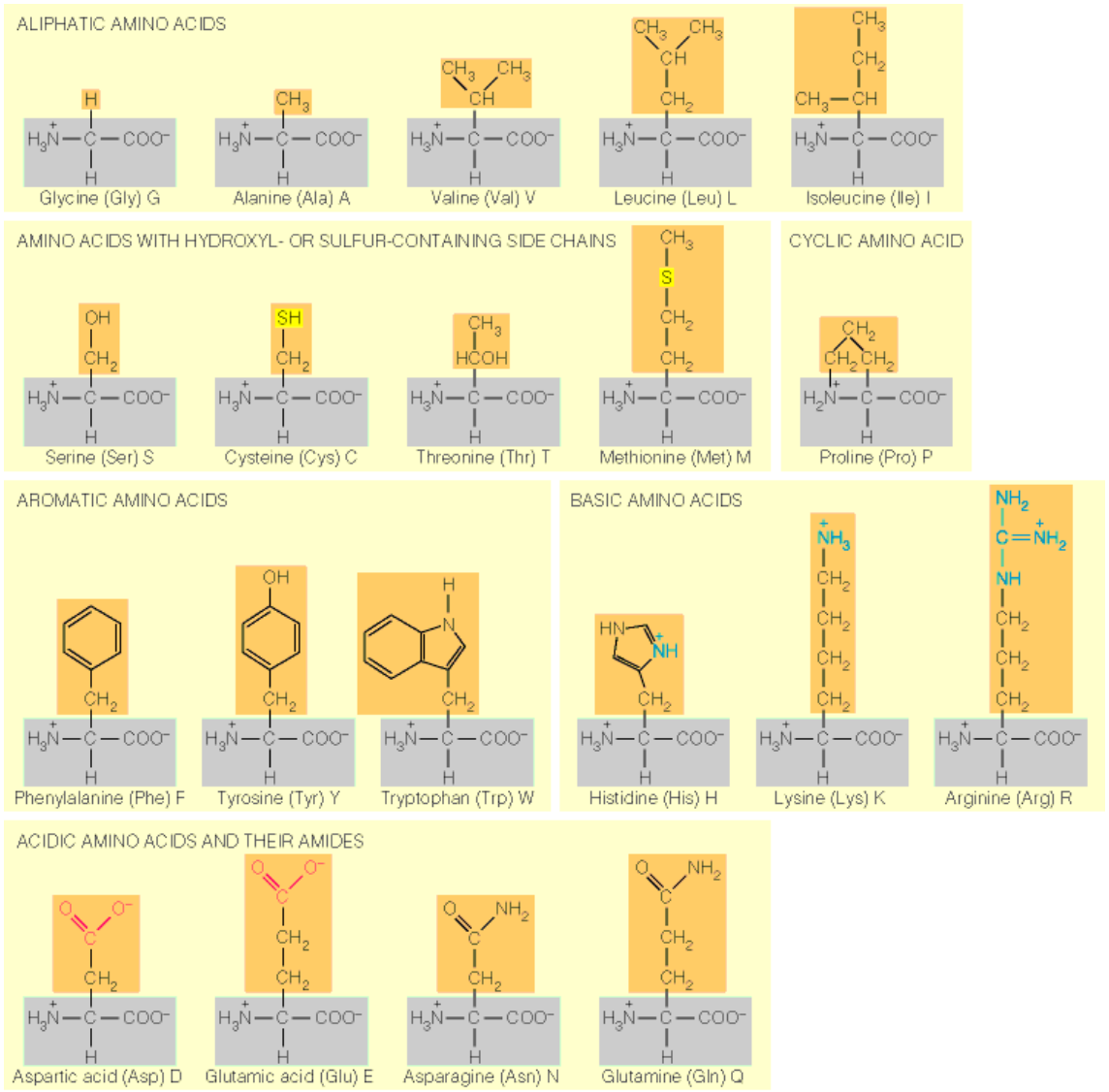


Figure 1.23: All amino acids with their name, three and one letter code. The amino acids are arranged according to their chemical properties.

non-polar (hydrophobic)		
glycine	Gly	G
alanine	Ala	A
valine	Val	V
leucine	Leu	L
isoleucine	Ile	I
methionine	Met	M
phenylalanine	Phe	F
tryptophan	Trp	W
proline	Pro	P
polar (hydrophilic)		
serine	Ser	S
threonine	Thr	T
cysteine	Cys	C
tyrosine	Tyr	Y
asparagine	Asn	N
glutamine	Gln	Q
acidic (-,hydrophilic)		
aspartic acid	Asp	D
glutamic acid	Glu	E
basic (+,hydrophilic)		
lysine	Lys	K
arginine	Arg	R
histidine	His	H

Table 1.1: Main properties of amino acids. Cysteine and methionine are able to form disulfide bonds through their sulfur atoms.

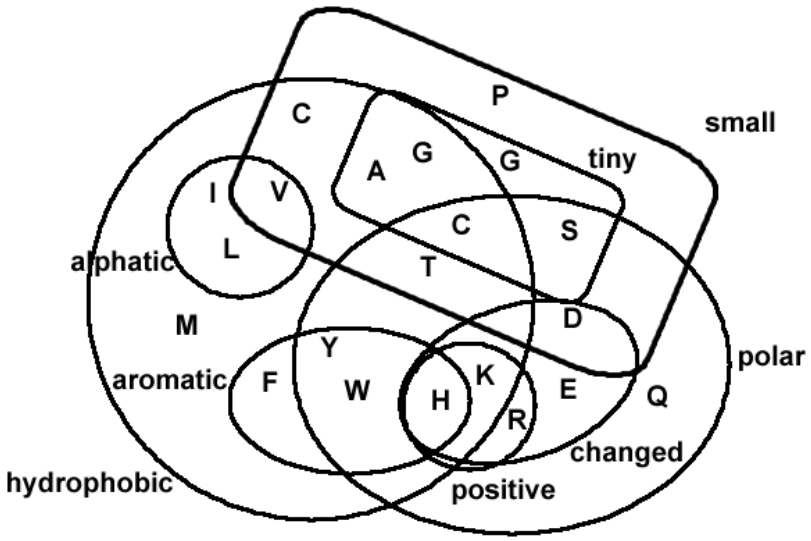


Figure 1.24: Classification of amino acids.

	SA	Hyd Res	Hyd side
Gly	47	1.18	0.0
Ala	86	2.15	1.0
Val	135	3.38	2.2
Ile	155	3.88	2.7
Leu	164	4.10	2.9
Pro	124	3.10	1.9
Cys	48	1.20	0.0
Met	137	3.43	2.3
Phe	39+155	3.46	2.3
Trp	37+199	4.11	2.9
Tyr	38+116	2.81	1.6
His	43+86	2.45	1.3
Thr	90	2.25	1.1
Ser	56	1.40	0.2
Gln	66	1.65	0.5
Asn	42	1.05	-0.1
Glu	69	1.73	0.5
Asp	45	1.13	-0.1
Lys	122	3.05	1.9
Arg	89	2.23	1.1

Table 1.2: Hydrophobicity scales (P.A.Karplus, Protein Science 6(1997)1302-1307)). "SA": Residue non-polar surface area [A2] (All surfaces associated with main- and side-chain carbon atoms were included except for amide, carboxylate and guanidino carbons. For aromatic side chains, the aliphatic and aromatic surface areas are reported separately.); "Hyd Res": Estimated hydrophobic effect for residue burial [kcal/mol]; "Hyd side": Estimated hydrophobic effect for side chain burial [kcal/mol] (The values are obtained from the previous column by subtracting the value for Gly (1.18 kcal/mol) from each residue).

First (5' end)	Second Position				Third (3' end)
	U	C	A	G	
U	UUU **Phe**	UCU **Ser**	UAU **Tyr**	UGU **Cys**	U
	UUC **Phe**	UCC **Ser**	UAC **Tyr**	UGC **Cys**	C
	UUA **Leu**	UCA **Ser**	UAA **Stop**	UGA **Stop**	A
	UUG **Leu**	UCG **Ser**	UAG **Stop**	UGG **Trp**	G
C	CUU **Leu**	CCU **Pro**	CAU **His**	CGU **Arg**	U
	CUC **Leu**	CCC **Pro**	CAC **His**	CGC **Arg**	C
	CUA **Leu**	CCA **Pro**	CAA **Gln**	CGA **Arg**	A
	CUG **Leu**	CCG **Pro**	CAG **Gln**	CGG **Arg**	G
A	AUU **Ile**	ACU **Thr**	AAU **Asn**	AGU **Ser**	U
	AUC **Ile**	ACC **Thr**	AAC **Asn**	AGC **Ser**	C
	AUA **Ile**	ACA **Thr**	AAA **Lys**	AGA **Arg**	A
	AUG **Met**	ACG **Thr**	AAG **Lys**	AGG **Arg**	G
G	GUU **Val**	GCU **Ala**	GAU **Asp**	GGU **Gly**	U
	GUC **Val**	GCC **Ala**	GAC **Asp**	GGC **Gly**	C
	GUA **Val**	GCA **Ala**	GAA **Glu**	GGA **Gly**	A
	GUG **Val**	GCG **Ala**	GAG **Glu**	GGG **Gly**	G

Table 1.3: The genetic code. `AUG` not only codes for methionine but serves also as a start codon.

property of amino acids to form chains is essential for building proteins. The chains are formed through the peptide bonds. An *amino acid residue* results from peptide bonds of more amino acids where a water molecule is set free (see Fig. 1.25). The peptide bonds are formed during translation ($\rightarrow$).

All proteins consist of these 20 amino acids. The specific 3D structure of the proteins and the position and interaction of the amino acids results in various chemical and mechanical properties of the proteins. All nano-machines are built from the amino acids and these nano-machines configure them-selves if the correct sequence of amino acids is provided.

1.8 Genetic Code

The genetic code are instructions for producing proteins out of the DNA information. A protein is coded in the DNA through a gene which is a DNA subsequence with start and end makers. A gene is first transcribed into mRNA which is subsequently translated into an amino acid sequence which folds to the protein. The genetic code gives the rules for translating a nucleotide sequence into an amino acid sequence. These rules are quite simple because 3 nucleotides correspond to one amino acid, where the nucleotide triplet is called *codon*. The genetic code is given in Tab. 1.3. `AUG` and `CUG` serve as a start codon, however for prokaryotes the start codons are `AUG`, `AUU` and `GUG`.

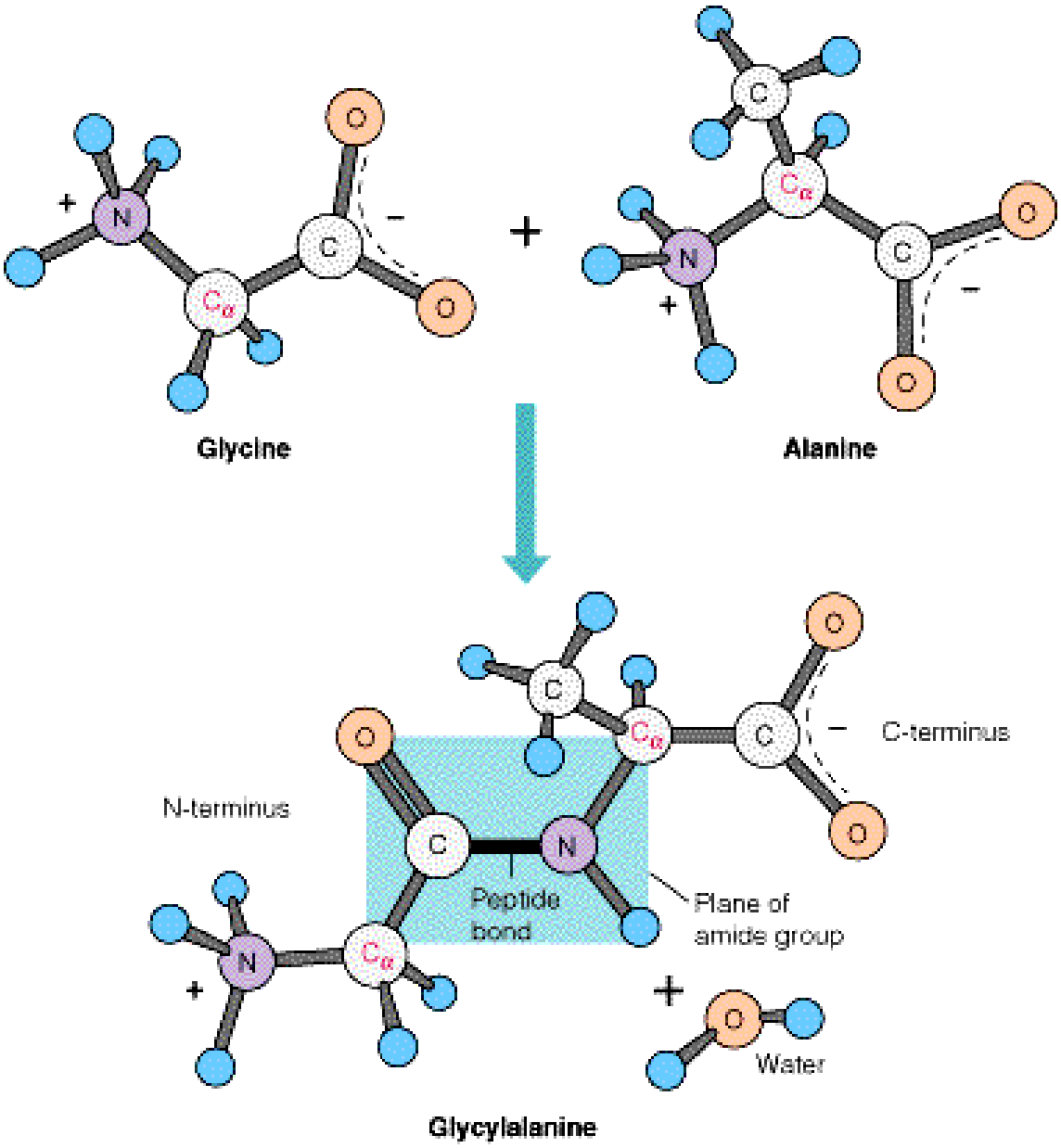


Figure 1.25: Peptide bond between glycine and alanine. The COO side of glycine (the carboxyl group) and the NH^3 side (the amino group) of alanine form a C-NO bond which is called a peptide bond. A water molecule is set free during forming the peptide bond.

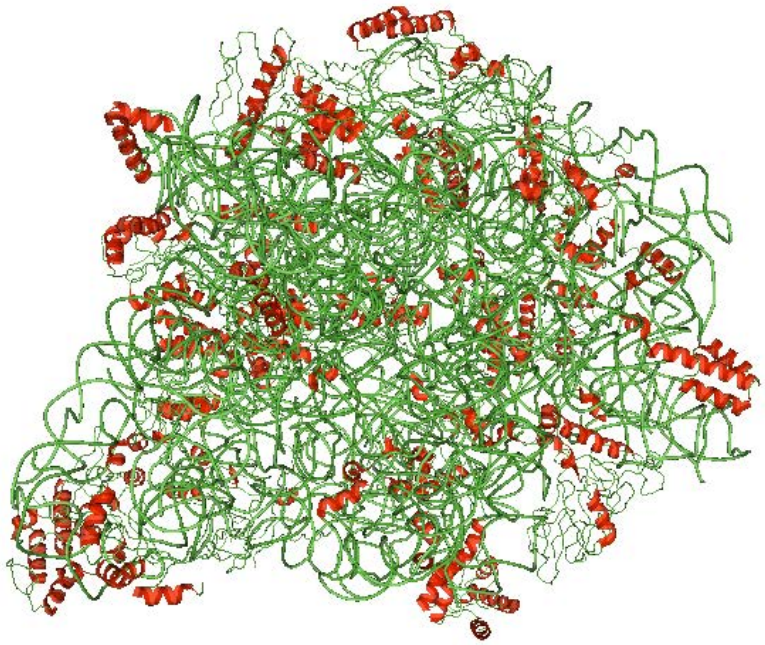


Figure 1.26: Large ribosomal subunit 50S from x-ray diffraction at 2.40 Å. Helices indicate positions of proteins and strands are the RNA.

1.9 Translation

After transcription the pre-mRNA is spliced and edited and the mature mRNA is transported out of the nucleus into the cytosol (eukaryotes). The protein production machinery, the ribosome, is located in the cytosol. The *ribosome* assembles the amino acid sequences out of the code written on the mRNA. See Fig. 1.26 for a detailed image of the ribosome. It consists of two subunits 60S and 40S in eukaryotes and 50S and 30S in bacteria.

As transcription also translation consists of 3 phases: *initiation*, *elongation* and *termination*. The main difference between prokaryotic translation and eukaryotic translation is the initiation (prokaryotic initiation has 3 factors whereas eukaryotic has 11 factors). In prokaryotes the translation initiation complex is built directly at the initiation site whereas in eukaryotes the initiation site is searched for by a complex. We will focus on the prokaryotic transcription.

1.9.1 Initiation

The ribosomes have dissociated subunits if they are not active. On the mRNA the ribosome binding site is marked by the pattern `AGGAGGU` which is called Shine-Dalgarno sequence. At this site the initiation factors IF1, IF2 and IF3 as well as the 30S ribosomal subunit bind. The initiator tRNA binds to the start codon. Then the 50S subunit binds to the complex and translation can start. See Fig. 1.27 for a possible initiation process.

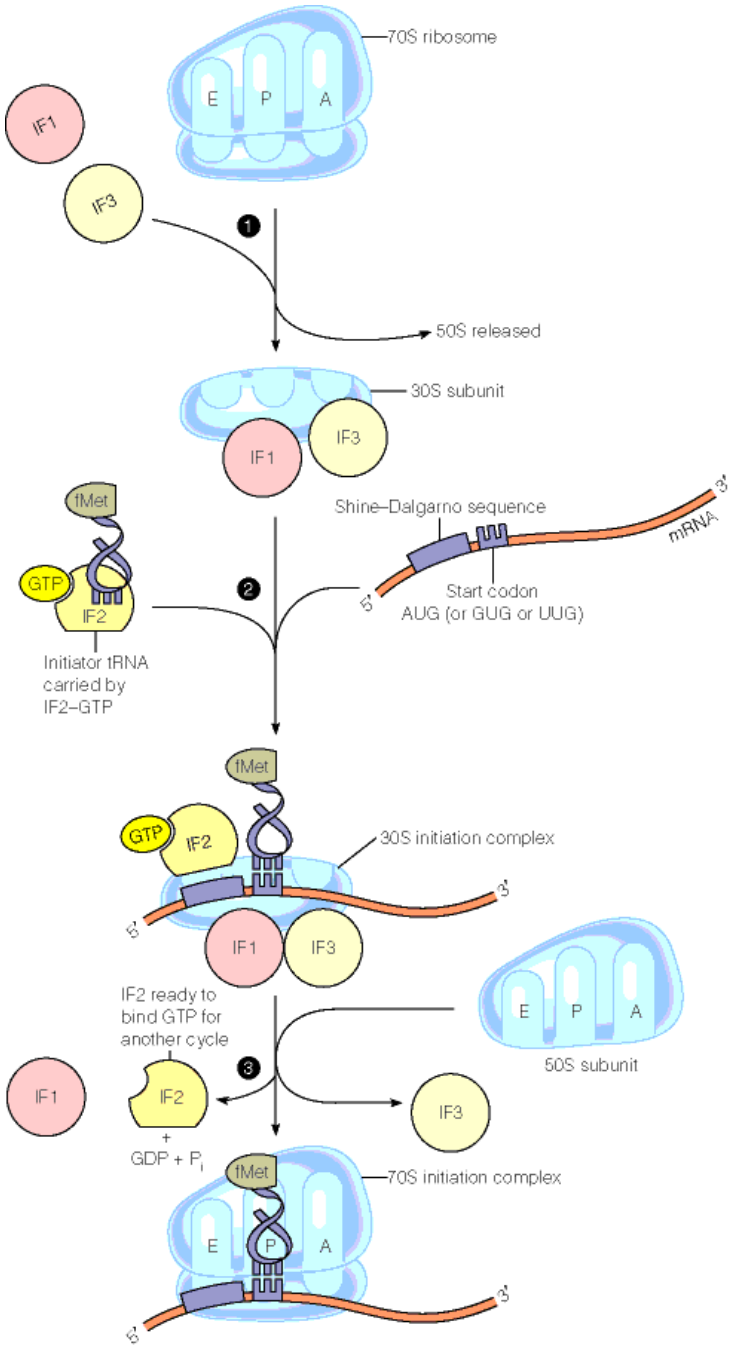


Figure 1.27: Possible initiation of translation (prokaryotes). "E","P","A" denote exit, pepidyl, aminoacyl binding sites, respectively. (1) initiation factors IF1 and IF3 bind to the 30S ribosome subunit, (2) initiation factor IF2, mRNA, and the 30S subunit form a complex at the Shine-Dalgarno sequence before the start codon (mostly `AUG`). The initiator tRNA containing N-formyl methionine (fMet) binds to the start codon, (3) the 50S subunit binds to the complex and IF1, IF2, and IF3 are released.

1.9.2 Elongation

Translation proceeds from the 5' end to the 3' end. Each tRNA which enters the ribosomal-mRNA complex binds at the A-site at its specific codon. Then a peptide bond of the new amino acid attached to the tRNA with the last amino acid of the existing polypeptide chain is built. The tRNA is moved forward to the P-side waiting for the next tRNA to come in. If the tRNA's amino acid forms a peptide bond with the next amino acid then it moves to the E-site where it is released. Figures 1.28 and 1.29 depict how the amino acid sequence is extended.

1.9.3 Termination

Termination is indicated by a stop codon (`UAA`, `UAG`, `UGA`) which enters the A-site. tRNAs cannot bind to this codon however release factors bind at or near the A-site. Either the release factors or the stop codon itself lead to the termination of translation. The amino acid chain is released and the 70S ribosome becomes unstable and dissociates into its parts. See Fig. 1.30 for the translation termination process. The 30S subunit may still be attached to the mRNA and searching for the next Shine-Dalgarno pattern.

Translation occurs at the rate of transcription. *E. coli* ribosomes can synthesize a 300-residue polypeptide in 20 seconds. A speed up of the translation occurs through multiple ribosomes attached to the same mRNA (see Fig. 1.31 for an example).

1.10 Folding

The last stage of protein production is the folding of the polypeptide chain into the protein. Only the correct folded protein can do its job and function correctly. Wrongly folded proteins lead to Creutzfeld-Jacob disease, Alzheimer disease, Bovine spongiform encephalopathy (BSE or "mad cow disease") and even the Parkinson disease may be caused by accumulations of misfolded proteins where degradation is not possible.

Even large proteins always fold in their specific 3D structure, therefore folding is not a random process but a complicated procedure with lot of interactions between the amino acids and water. The folding pathways are sometimes not unique and possess intermediate states of folding.

The folding is sometimes assisted by special molecules called *chaperones*. There are different types of chaperones some hide the hydrophobic regions of the protein to ensure correct folding and avoid interference with other regions or proteins. Other chaperones act as containers where proteins are correctly folded.

The folding of a protein takes from milliseconds up to minutes or hours.

One of the major tasks in bioinformatics is the prediction of the 3D structure from the amino acid sequence. From the 3D structure the function of a protein can be guessed. More interesting is the construction of new proteins and nano-machines based on the predicted 3D structure.

The main forces for stabilizing proteins and for correct folding were given previously at the amino acid characteristics (hydrophobic effects, salt bridges, disulfide bridges, hydrogen bonds).

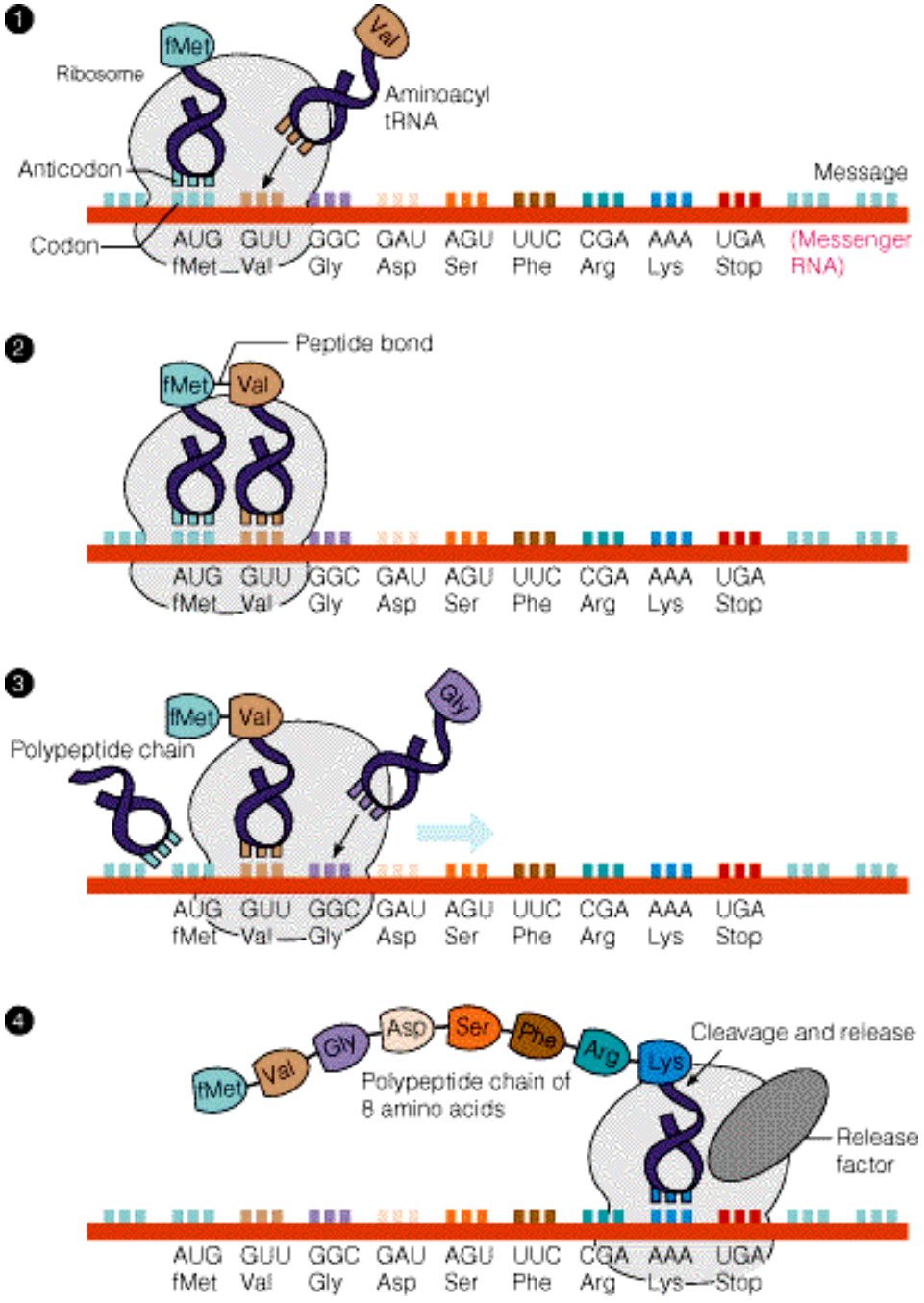


Figure 1.28: The translation elongation is depicted. (1) Val-tRNA binds to the ribosome-mRNA complex at the Val-coding region `GUU`, (2) the initial fMet forms a peptide bound with Val, (3) the next codon codes Gly and Gly-tRNA enters the complex, (4) the stop codon `UGA` lead to a release of the polypeptide.

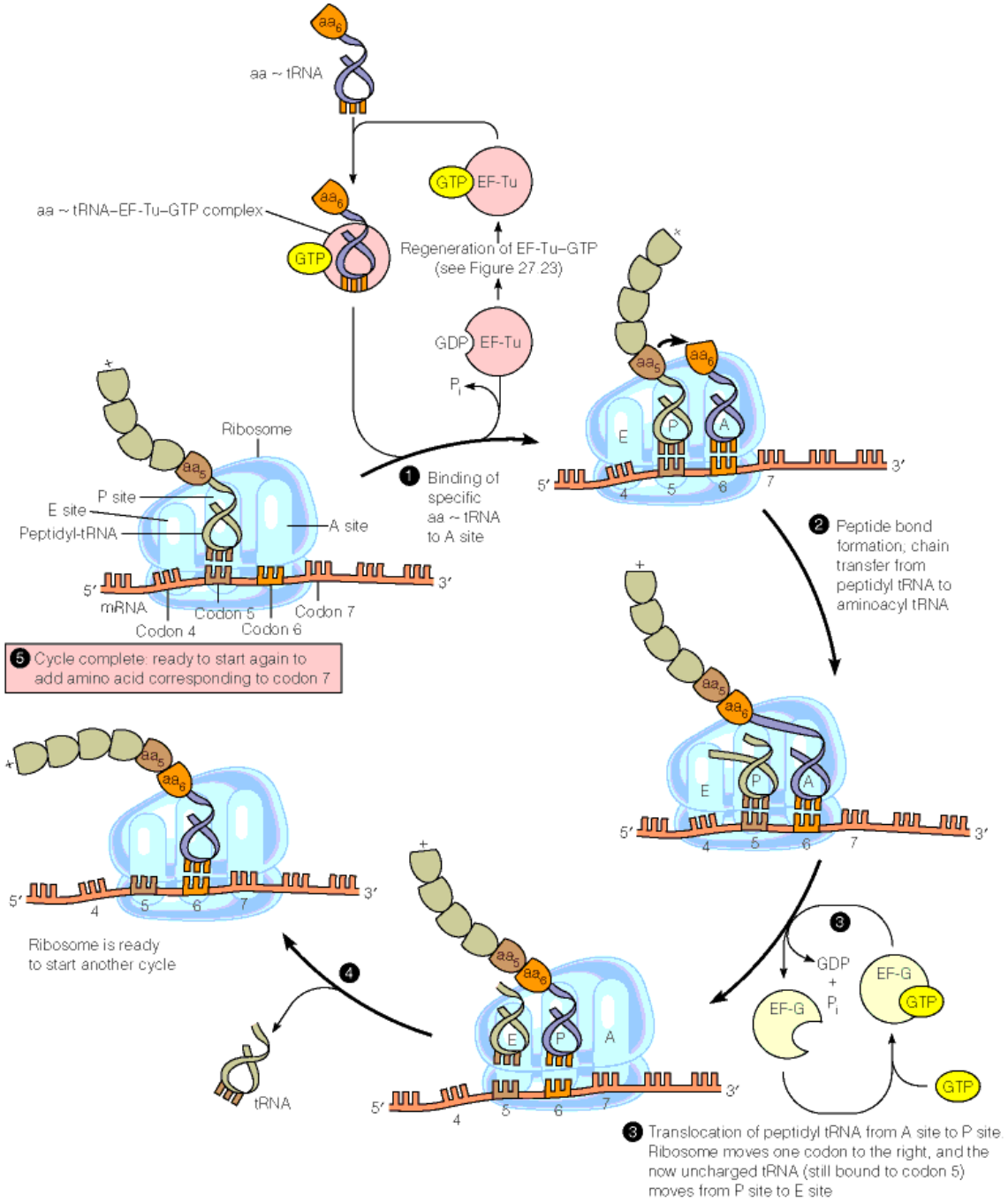


Figure 1.29: Translation elongation. (1) A specific tRNA with amino acid (aa_6) binds at the A-site, (2) amino acids aa_5 and aa_6 form a peptide bond, (3) the aa_5 tRNA moves to the E-site and the aa_6 tRNA to the P-site, (4) the tRNA from the E-site is released and another cycle begins.

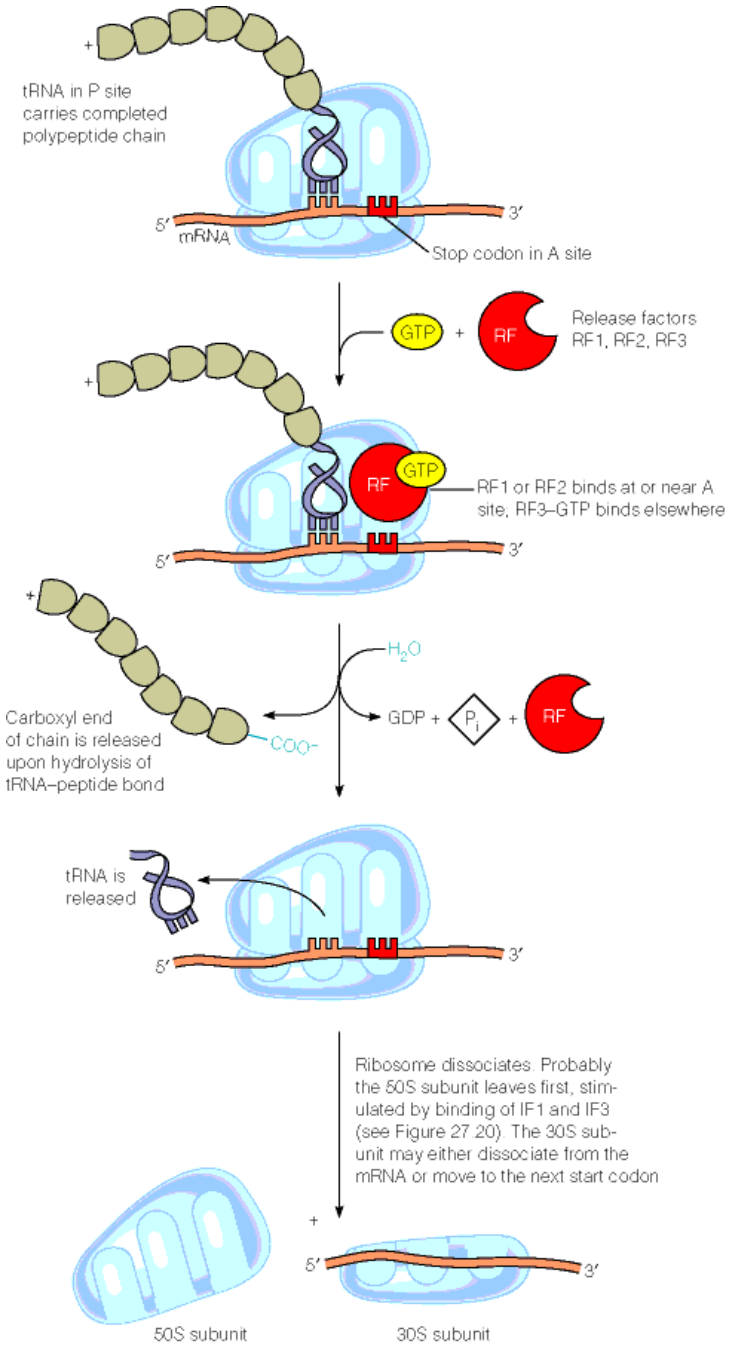


Figure 1.30: Termination of the translation. First a stop codon appears in the A-site, then release factors bind at the A-site, the polypeptide chain is released and the ribosome dissociates.

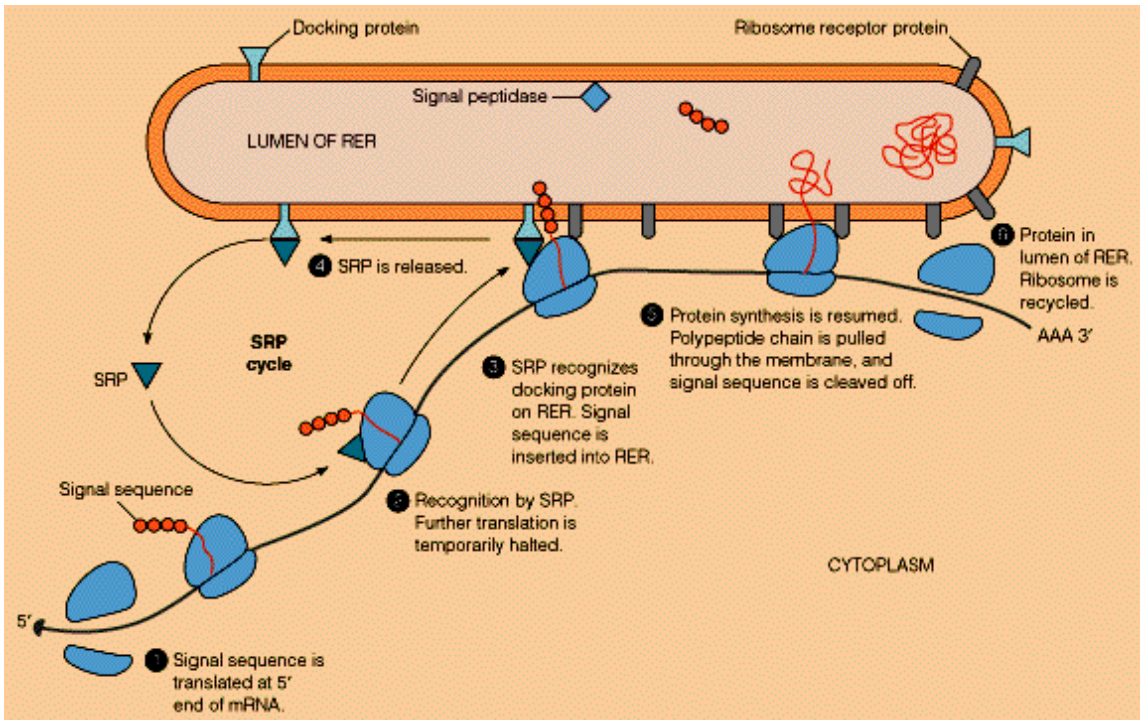


Figure 1.31: Translation with multiple ribosomes is depicted.

Chapter 2

Bioinformatics Resources

This chapter describes resources on the WWW and data bases needed for bioinformatics research.

The European Molecular Biology Laboratory (EMBL – `http://www.embl-heidelberg.de`) maintains a nucleotide data base which is daily updated but supplies many other sources for bioinformatics, too. A spin-off is the European Bioinformatics Institute (EBI – `http://www.ebi.ac.uk/ebi_home.html`) which maintains the SwissProt protein sequence data base and the Sequence Retrieval System (SRS – `http://srs.ebi.ac.uk/`). The ExPASy site (`http://www.expasy.org/`) integrates SwissProt & TrEMBL, PROSITE and some other resources (software, education etc.).

At the University College London the Biomolecular Structure and Modeling (BSM) maintains the PRINTS (protein fingerprints, i.e. multiple motifs) data base and the CATH protein structure data base.

The National Center for Biotechnology Information (NCBI – `http://www.ncbi.nlm.nih.gov/`) hosts the GenBank, the National Institutes of Health (NIH) DNA sequence data base and is famous through its BLAST software including data bases like the NR (non-redundant sequences) data base. NCBI also maintains the ENTREZ (`http://www.ncbi.nlm.nih.gov/Entrez/`) system which gives access to molecular biological data and articles. ENTREZ gives access to nucleotide sequences from GenBank, EMBL, DDBJ (DNA data base of Japan) as well as to protein sequences from SWISS-PROT, PIR, PRF, SEQDB, PDB.

Other important sites are the European EMBnet (`http://www.embnet.org`) and the Sanger Centre founded by the Wellcome Trust (`http://www.sanger.ac.uk/Info/`).

2.1 Data Bases

Some of the important data bases are listed in Tab. 2.1. The most important DNA sequence data bases are GenBank (USA – `http://www.ncbi.nlm.nih.gov/genbank/`), EMBL (Europe – `http://www.embl-heidelberg.de/`), and DDBJ (Japan – `http://www.ddbj.nig.ac.jp/`).

GeneCards is a searchable, integrated, database of human genes that provides concise genomic related information, on all known and predicted human genes.

NR is a data base mainly used with BLAST search and comprises all non-redundant (non-identical) sequences. It contains more than 3 mio. sequences and for a BLAST or PSI-BLAST run a new sequence is compared with all sequences in the NR data base giving the best hits with their statistics.

Name	T	U	URL
EMBL	N	D	http://www.embl-heidelberg.de/
GeneCards	N	?	http://www.genecards.org/
PDB	P	D	http://www.rcsb.org/pdb/Welcome.do
SCOP	P	?	http://scop.berkeley.edu/
CATH	P	?	http://www.cathdb.info/
PIR	P	W	http://pir.georgetown.edu/
SWISS-PROT	P	W	http://www.expasy.org/sprot/
TrEMBL	P	W	http://www.expasy.org/sprot/
Homstrad	P	W	http://tardis.nibio.go.jp/homstrad/
InterPro	P	?	http://www.ebi.ac.uk/interpro/
NR	P	W	ftp://ftp.ncbi.nih.gov/blast/db
Pfam	P	?	http://pfam.sanger.ac.uk/
UniProt	P	?	http://www.expasy.uniprot.org/
PROSITE	P	W	http://www.expasy.org/prosite/
PRINTS	P	?	http://umber.sbs.man.ac.uk/dbbrowser/PRINTS/
BLOCKS	P	?	http://blocks.fhcrc.org/
STRING	P	?	http://string-db.org
DAVID	O	?	http://david.abcc.ncifcrf.gov/
ChEMBL	O	?	https://www.ebi.ac.uk/chembl/
PubChem	O	?	http://pubchem.ncbi.nlm.nih.gov/

Table 2.1: Selected data bases. The column “T” stands for type and gives whether is nucleotide (“N”) or protein (“P”) related or of other interest (“O”). “U” gives the update (“D” = daily, “W”=weekly, “?” = unknown). The last column gives the URL.

Often used if instead of a sequence an average sequences should be processed (average of all sequences which are very similar to the sequences at hand). Processing the average of sequences has given large improvements in protein secondary structure prediction and for protein classification.

PIR (Protein Information Resource) supplies protein sequences which are classified according to the knowledge about the certain sequence and whether sequences are really translated. Another protein sequence data base is SWISS-PROT with much information about the sequences. TrEMBL gives sequences of all coding sequences in EMBL and is an add on to SWISS-PROT, where many sequences will eventually go into SWISS-PROT.

PROSITE is a protein classification data base where proteins are classified according to motifs (special amino acid patterns for the classes). Some classes in PROSITE do not possess a pattern and a profile (a weighted pattern) is supplied. Many protein classes possess patterns like the 2FE-2SE class were a cystine pattern is necessary to keep a ferro-sulfur structure (for electron transfer) through disulfide bonds in place.

PRINTS is also a motif data base (fingerprints) where more than one motif is combined to identify a protein class. The motifs are mostly found by multiple alignment.

BLOCKS is a data base of highly conserved regions and is related to PROSITE and PRINTS.

PFAM is a data base where alignments are coded in hidden Markov models (HMMs).

SCOP is a 3D protein structure data base where domains (separate substructures) are manually classified into structural classes. SCOP is an important data base (besides CATH) for protein 3D structure prediction. The hierarchy of the classification is “class”, “fold”, “superfamily”, “family”. “Class” only separates helical, beta-sheet, or mixed structures, but contains special proteins like membrane proteins, short proteins, or artificially constructed protein. “Fold” classes contain domains with similar 3D structure (same secondary structure in the same arrangement). “Superfamily” contains proteins where a common evolutionary origin is probable based on sequence similarities (remote homologous). “Families” contain proteins which are sufficiently similar (in sequence or structure) to one another, in order to be sure that they are evolutionary related and have in most cases the same function. The sequence data for SCOP can be obtained from the ASTRAL data base.

CATH is like SCOP a 3D protein structure data base of domains. Main difference to SCOP is that the classification is made automatically (even if manual inspections are done). Another difference is the classification scheme, where the hierarchy is “class”, “architecture”, “topology”, “homology”, “sequence”. “Class” is as in SCOP. “Architecture” classes contain proteins which have similar 3D shape even if the secondary structure connection is different. “Topology” also considers in contrast to “architecture” the connectivity of secondary elements and is similar to the “fold” class of SCOP. “Homology” is similar to “family” of SCOP because an evolutionary connection is highly probable. “Sequence” contains evolutionary closely related proteins with the same function.

HOMSTRAD (Homologous STRucture Alignment Database) is a data base of structure-based alignments for homologous protein families. Structures are classified into homologous families and the sequences of each family are aligned on the basis of their 3D structures.

InterPro is a data base of protein families, domains and functional sites. It integrates information from PROSITE, PRINTS, SMART, Pfam, ProDom, etc.

UniProt (Universal Protein Resource) joins the information contained in Swiss-Prot, TrEMBL, and PIR.

STRING is a database of known and predicted protein interactions. The interactions include direct (physical) and indirect (functional) associations.

DAVID (Database for Annotation, Visualization and Integrated Discovery) provides a comprehensive set of functional annotation tools for investigators to understand biological meaning behind large list of genes.

ChEMBL is a manually curated chemical database of bioactive molecules with drug-like properties maintained by the EBI.

PubChem is a database of chemical molecules and their activities against biological assays maintained by the NCBI.

2.2 Software

Tab. 2.2 lists some software which is useful in bioinformatics research. These software is basic bioinformatics software. Important machine learning software can be found at `http://www.kernel-machines.org/` under "software" where the libSVM and torch package is recommended. For feature selection the "spider" software can be used. For feature selection and classification a special software, the PSVM software can be found under `http://www.bioinf.jku.at/software/psvm/`.

EMBOSS is a toolbox with many useful bioinformatics programs (e.g. standard alignment programs) in source code.

Domainatrix is a toolbox based on EMBOSS for protein domain processing (SCOP) with many useful programs.

BLAST is the standard local alignment program. Probably the most used bioinformatics program. For averaging sequences PSI-BLAST is comfortable as it makes multiple runs through a data base (e.g. NR) and provides a multiple alignment of the best hits.

PHRAP is a program for assembling shotgun DNA sequence data.

Babel is a cross-platform program and library which interconverts between many file formats used in molecular modeling and computational chemistry.

BioPerl provides parsers, wrappers for other programs, GUI packages for other programs, a microarray package, etc. written in Perl.

ClustalW is the standard multiple alignment tool (also used by PSI-BLAST).

Modeller produces a 3D model of a sequence given template structures and a multiple alignment of the sequence with the sequences of the template structures. To obtain the 3D model modeler optimizes the structure and satisfies spatial restraints. It is often used after threading spatial restraints or protein classification to build the final model of the structure, where templates are identified by threading or by protein classification.

Phylip is an (old) package for performing phylogenetic research.

Pymol is a very nice molecular viewer which allows to produce images, and movies. It can display the sequence and if the user clicks on an element the according side chains appear in the 3D model.

Software	application	URL
EMBOSS	toolbox	http://emboss.sourceforge.net
Domainatrix	tools domains	http://emboss.sourceforge.net/apps/cvs/embassy/domainatrix/
BLAST	homology search	http://www.ncbi.nlm.nih.gov/BLAST/
PHRAP	shotgun DNA	http://www.phrap.org/
Babel	converts formats	http://openbabel.sourceforge.net/wiki/Main_Page
BioPerl	toolbox perl	http://www.bioperl.org/
ClustalW	multiple alig.	ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalW/
modeller	building model	http://salilab.org/modeller/download_installation.html
phylip	phylogenetics	http://evolution.gs.washington.edu/phylip.html
pymol	good viewer	http://www.pymol.org/
rasmol	fast viewer	http://www.umass.edu/microbio/rasmol/
molscript	nice images	http://www.avatar.se/molscript/obtain_info.html
strap	java toolbox	http://www.charite.de/bioinf/strap/
tinker	mol. dyn., fortran	http://www.es.embnet.org/Services/MolBio/tinker/
biodesigner	mol. dynamics	http://www.pirx.com/biodesigner/download.html
threader	threading	http://bioinf.cs.ucl.ac.uk/threader/
loopp	treading	http://folding.chmcc.org/loopp/loopp.html
prospect	threading	http://compbio.ornl.gov/structure/prospect/
sspro4	sec. struc.	http://contact.ics.uci.edu/download.html
psipred	sec. struc..	http://bioinf.cs.ucl.ac.uk/psipred/
prof	sec. struc.	http://www.aber.ac.uk/~phiwww/prof/
jnet	sec. struc.	http://www.compbio.dundee.ac.uk/www-jpred/legacy/jnet/
PHD	sec. struc.	https://www.rostlab.org/papers/1996_phd/paper.html
DSSP	sec. struc. f. 3D	http://swift.cmbi.ru.nl/gv/dssp/
whatif	mol. modelling	http://swift.cmbi.kun.nl/whatif/
hmmer	alignment HMM	http://hmmer.janelia.org/
ProsaII	struc. verf.	https://prosa.services.came.sbg.ac.at/prosa.php
CE	struc. alig.	ftp://ftp.sdsc.edu/pub/sdsc/biology/CE/src/
DALI	struc. alig.	http://www.ebi.ac.uk/dali/

Table 2.2: Selection of software.

Rasmol is a molecular viewer which is simpler but faster than pymol and does not access the graphic card directly.

Molscript is used to produce nice molecular images for printed papers.

Strap is a java written GUI interface to many programs like different viewers, alignment programs, structural alignment programs.

Tinker is a molecular dynamics software written in fortran where the source code is available. Many optimization tools are implemented to optimize the energy and to compute forces.

Biodesigner is a molecular modeling and visualization program. It is capable of creating homologous models of proteins, evaluate, and refine the models.

Threader (GenThreader) is a threading program which performed well in many tests.

LOOPP is a threading program where the source code is provided.

Prospect is a well known threading program.

SSpro4 is a secondary structure prediction program based on recursive neural networks from Pierre Baldi. Source code is available.

PsiPred is a secondary structure prediction program where the source code is available. It is wildly used and performed good in different competitions.

Prof is a secondary structure prediction program where the source code is available.

Jnet is a secondary structure prediction program where the source code is available.

PHD is a secondary structure prediction program.

DSSP is a program to compute secondary structure out of a 3D structure by determining the hydrogen bonds.

Whatif is a molecular modeling package for proteins in water, ligands, nucleic acids, etc.

Hmmer is a hidden Markov model software package which transforms an alignment into an HMM model. Advantage is that alignments can be coded in a probabilistic framework where the likelihood of a new sequence to belong to the aligned sequences can be computed. The transformation of alignments into HMMs is done via the HMMER software.

ProsaII allows to verify 3D structures of proteins and can pick out parts of the structure which seem to be unlikely to be observed in nature.

CE is a widely used structural alignment program. Given two 3D protein structures, it superimposes them.

DALI is also a structural alignment program with a data base of alignments.

2.3 Articles

To find articles "PubMed" `http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed` is recommended, for machine learning and computer science articles `http://www.researchindex.org/` and for other articles `http://scholar.google.com/`.

organism	size [mio bases]	number genes	av. gene dens. [bases]	chromo-somes
Homo sapiens (human)	2900	30,000	1 / 100,000	46
Rattus norvegicus (rat)	2,750	30,000	1 / 100,000	42
Mus musculus (mouse)	2500	30,000	1 / 100,000	40
Drosophila melanogaster (fruit fly)	180	13,600	1 / 9,000	8
Arabidopsis thaliana (plant)	125	25,500	1 / 4000	10
Caenorhabditis elegans (roundworm)	97	19,100	1 / 5000	12
Saccharomyces cerevisiae (yeast)	12	6300	1 / 2000	32
Escherichia coli (bacteria)	4.7	3200	1 / 1400	1
H. influenzae (bacteria)	1.8	1700	1 / 1000	1

Table 2.3: Overview over some genomes.

Tab. 2.3 lists important steps in genome sequencing, where the size of the genome (number of genes), the average number of genes per 100,000 bases and the number of chromosomes is given. In the following the corresponding genome publication articles are listed.

Human
International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature* . **409** : 860-921. (15 February 2001)

Rat
Rat Genome Sequencing Project Consortium. Genome Sequence of the Brown Norway Rat Yields Insights into Mammalian Evolution. *Nature* . **428** : 493-521. (1 April 2004)

Mouse
Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature* . **420** : 520 -562. (5 December 2002)

Fruit Fly
M. D. Adams, et al. The genome sequence of *Drosophila melanogaster* . *Science* . **287** : 2185-95. (24 March 2000)

Arabidopsis - First Plant Sequenced
The Arabidopsis Genome Initiative. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana* . *Nature* **408** : 796-815. (14 December 2000)

Roundworm - First Multicellular Eukaryote Sequenced
The *C. elegans* Sequencing Consortium.Genome sequence of the nematode *C. elegans* : A platform for investigating biology. *Science* . **282** : 2012-8. (11 December 1998)

Yeast

A. Goffeau, et al. Life with 6000 genes. *Science* . **274** : 546, 563-7. (25 October 1996)

Bacteria - *E. coli*

F. R. Blattner, et al. The complete genome sequence of Escherichia coli K-12. *Science* . **277** : 1453-1474. (5 September 1997)

Bacteria - *H. influenzae* - First Free-living Organism to be Sequenced

R. D. Fleischmann, et al. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* . **269** : 496-512. (28 July 1995)

Chapter 3

Pairwise Alignment

This chapter introduces and discusses pairwise alignment methods. We consider sequences of amino acids but everything can be transferred to sequences of nucleotides.

3.1 Motivation

The cells of most organisms function in a similar way. The proteins produced in cells of different species are very similar to one another because they must perform the same tasks like keeping up the energy supply by transforming and transporting energy (glucose cycles, anaerobic respiration, tricarboxylic acid cycle – the TCA, oxidative phosphorylation – see Fig. 3.1 for an overview of pathways).

Other pathways in living organisms include fatty acid oxidation, thin acid oxidation, gluconeogenesis, HMG-CoA reductase, pentose phosphate, porphyrin synthesis, or urea cycle. Many proteins have the same task in different organism like detecting of damage of and repairing the DNA (housekeeping proteins), carrying substances, membrane proteins, chromosomal proteins, collagens (tissue making), GTP binding proteins, gatekeeper proteins (ER entrance and exit control), molecular chaperones, ribosomal proteins, nucleoproteins, RNA binding proteins, receptor proteins, regulatory proteins, zinc finger proteins (a zinc ion is kept), etc.

If a new sequence is obtained from genome sequencing then the first step is to look for similarities to known sequences found in other organisms. If the function/structure of similar sequences/proteins is known then it is highly likely that the new sequence corresponds to a protein with the same function/structure. It was found that only about 1% of the human genes do not have a counterpart in the mouse genome and that the average similarity between mouse and human genes is 85%. Such similarities exist because all cells possess a common ancestor cell (a mother cell). Therefore, in different organisms there may be mutations of amino acids in certain proteins because not all amino acids are important for the function and can be replaced by amino acids which have similar chemical characteristics without changing the function. Sometimes the mutations are so numerous that it is difficult to find similarities. In some cases the relationship is only at the structural basis but mutations changed the function of the protein (e.g. TIM barrel proteins). However, even the structure is essential to infer the function.

The method to figure out functions of genes by similarities is called *comparative genomics* or *homology search*. A homologous sequence is similar to another sequence where the similarity stems from common ancestry.

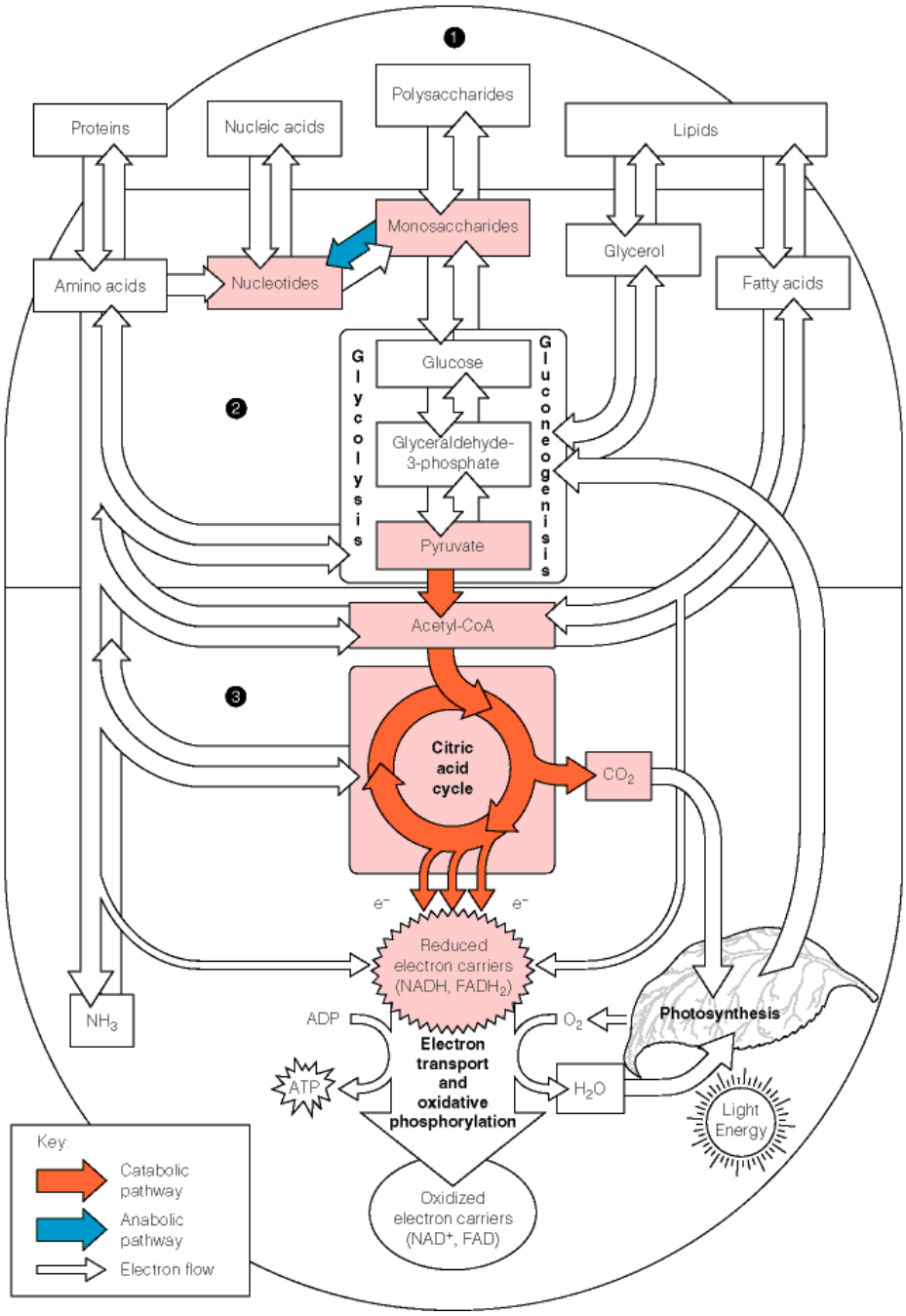


Figure 3.1: The main energetic pathways in the cell are depicted.

The next sections will introduce similarity scoring schemes and alignment algorithms. In general scoring schemes (error functions, cost functions, energy functions, penalty functions) should be separated from optimization algorithms. Many optimization algorithms can be applied to different scoring schemes but there exist also optimization algorithms which are designed for a special scoring scheme. On the other hand scoring schemes can be optimized in different ways. Some general optimization methods for discrete (non-differentiable) problems are random guessing (select a candidate solution, evaluate it, store it if it is the best up to now), exhaustive search (test all candidates), genetic algorithms (better solutions survive and are mutated) or simulated annealing (by introducing a temperature discrete problems are made continuous).

3.2 Sequence Similarities and Scoring

Given two sequences: how similar are they? This questions cannot be answered because it depends on the context. Perhaps the sequences must have the same trend (stock market), contain the same pattern (text), or have the same frequencies (speech) etc. to be similar to one another.

3.2.1 Identity Matrix

For biological sequences it is known how one sequence can mutate into another one. First there are *point mutations* i.e. one nucleotide or amino acid is changed into another one. Secondly, there are *deletions*, i.e. one element (nucleotide or amino acid) or a whole subsequence of element is deleted from the sequence. Thirdly, there are *insertions*, i.e. one element or a subsequence is inserted into the sequence. For our first approach the similarity of two biological sequences can be expressed through the minimal number of mutations to transform one sequence into another one. Are all mutations equally likely? No. Point mutations are more likely because an amino acid can be replaced by an amino acid with similar chemical properties without changing the function. Deletions and insertions are more prone to destroying the function of the protein, where the length of deletions and insertions must be taken into account. For simplicity we can count the length of insertions and deletions. Finally, we are left with simply counting the number of amino acids which match in the two sequences (it is the length of both sequences added together and insertions, deletions and two times the mismatches subtracted, finally divided by two).

Here an example:

```
BIOINFORMATICS            BIOI-N-FORMATICS
                     ⟶
BOILING FOR MANICS        B-OILINGFORMANICS
```

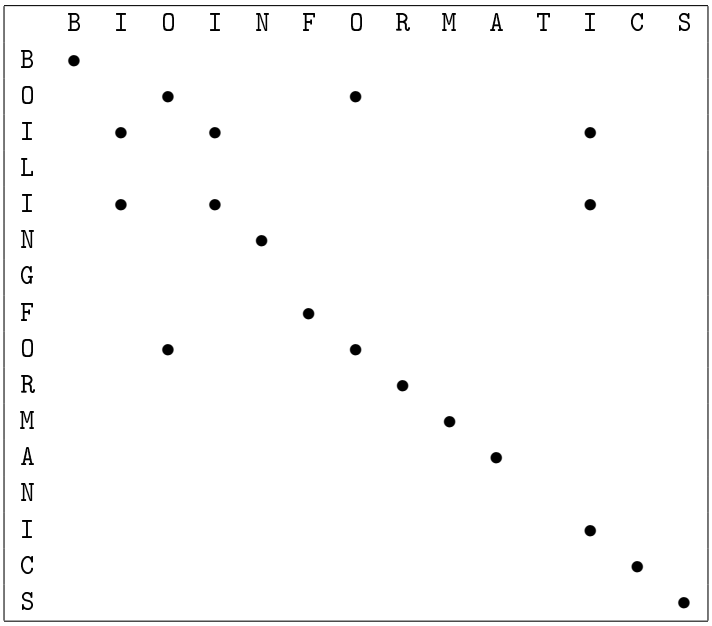
The hit count gives 12 identical letters out of the 14 letters of `BIOINFORMATICS`. The mutations would be:

(1) delete `I`	`BOINFORMATICS`
(2) insert `LI`	`BOILINFORMATICS`
(3) insert `G`	`BOILINGFORMATICS`
(4) change `T` into `N`	`BOILINGFORMANICS`

These two texts seem to be very similar. Note that insertions or deletions cannot be distinguished if two sequences are presented (is the `I` deleted form the first string or inserted in the second?). Therefore both are denoted by a "-" (note, two "-" are not matched to one another).

The task for bioinformatics algorithms is to find from the two strings (left hand side in above example) the optimal *alignment* (right hand side in above example). The optimal alignment is the arrangement of the two strings in a way that the number of mutations is minimal. The optimality criterion scores matches (the same amino acid) with 1 and mismatches (different amino acids) with 0. If these scores for pairs of amino acids are written in matrix form, then the **identity matrix** is obtained. The number of mutations is one criterion for optimality but there exist more (as seen later). **In general, an alignment algorithm searches for the arrangement of two sequences such that a criterion is optimized.** The sequences can be arranged by inserting "-" into the strings and moving them horizontally against each other. For long sequences the search for an optimal alignment can be very difficult.

One tool for representing alignments is the dot matrix, where one sequence is written horizontally on the top and the other one vertically on the left. This gives a matrix where each letter of the first sequence is paired with each letter of the second sequence. For each matching of letters a dot is written in the according position in the matrix. Which pairs appear in the optimal alignment? We will see later, that each path through the dot matrix corresponds to an alignment.



A simple game:
Rules: you can move horizontally "→", vertically "↓", and you can only move diagonal "↘" if you at the position of a dot.
Task: make as many diagonal movements as possible if you run from the upper left corner to the lower right corner.

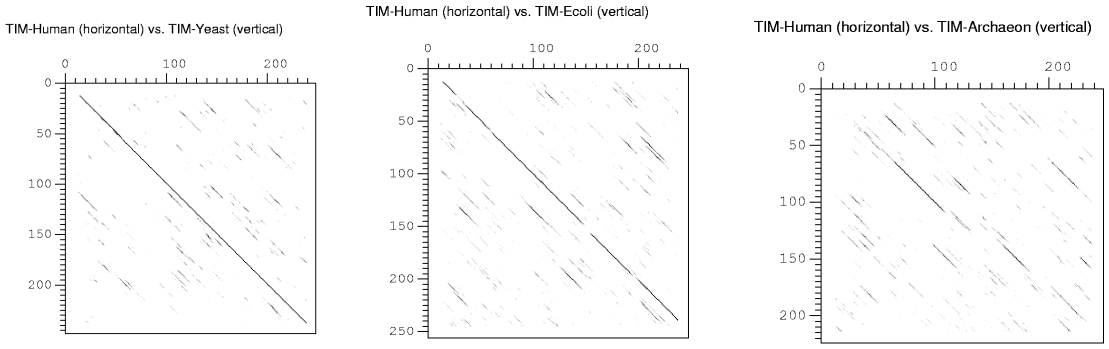
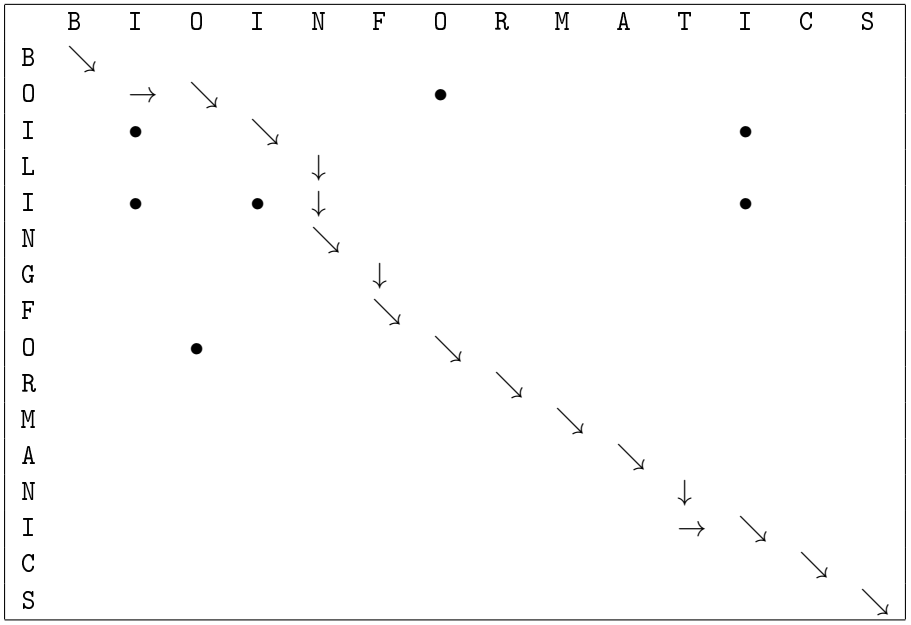


Figure 3.2: Dot plot of the human triosephosphate isomerase with the same protein in yeast, E. coli, and archaeon. Yeast gives the best match as the diagonal is almost complete. E. coli has some breaks in the diagonal. The archaeon shows the weakest similarity but the 3D structure and function is the same in all proteins.



The number of diagonal movements $\searrow$ corresponds to matches and count for the scoring, the "$\rightarrow$" correspond to a "-" in the vertical sequence, the "$\downarrow$" to a "-" in the horizontal sequence and a "$\rightarrow\downarrow$" or a "$\begin{smallmatrix}\downarrow\\\rightarrow\end{smallmatrix}$" combination correspond to a mismatch. Therefore, each way through the matrix corresponds to an alignment and each alignment can be expressed as a way through the matrix.

In above examples one can see that dots on diagonals correspond to matching regions. In Fig. 3.2 we show the dot matrices for comparing the human protein triosephosphate isomerase (TIM) to the same protein in yeast, E. coli (bacteria), and archaeon. For yeast the diagonal is complete and for E. coli small gaps are visible but the archaeon does not show an extended diagonal. Therefore, the human TIM matches best with the yeast TIM, followed by the E. coli TIM and has lower similarity to the archaeon TIM.

Scoring by counting the matches is the simplest way to score but there exist more advanced methods. They address the fact that for some amino acids it is more likely that they mutate into

each other because they share the same chemical properties (other mutations occur but do not survive). These methods also take into account that the occurrence of a deletion /insertion must be higher weighted then its length.

Here we only consider scoring through evaluation of pairs of amino acids (aligned amino acids, one from the first and one from the second sequence). It may be possible to discover other scoring schemes (taking the context into account; aligning pairs to pairs, etc.) but the optimization methods would be complex, as we will see later.

Now we derive methods for evaluating the match of two amino acids, i.e. how much does one match score. The intuitions says that the value should correspond to the probability of the mutation of one amino acid into another one. Here and in the following we focus on amino acid sequences but everything holds analogously for nucleotide sequences.

3.2.2 PAM Matrices

Dayhoff et. al (1978) introduced Percent or Point Accepted Mutation (PAM) matrices. PAM corresponds to a unit of evolution, e.g. 1 PAM = 1 point mutation/100 amino acids and 250 PAM = 250 point mutations/100 amino acids. The unit of evolution is therefore the time that on average n % mutations occur at a certain position and survive. For PAM 250 1/5 of the amino acids remain unchanged (homework: proof with PAM 1). PAM n is obtained from PAM 1 through n-times matrix multiplication. PAM matrices are Markov matrices and have the form

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,20} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,20} \\ \vdots & \vdots & \ddots & \vdots \\ p_{20,1} & p_{20,2} & \cdots & p_{20,20} \end{pmatrix} , \tag{3.1}$$

where $p_{i,j} = p_{j,i}$, $p_{i,j} \geq 0$ and $\sum_j p_{i,j} = 1$.

The original PAM was obtained through the comparison of 71 blocks of subsequences which had >85% mutual identity yielding to 1,572 changes. Phylogenetic trees ($\rightarrow$) were constructed for each of the 71 blocks. The average transition of amino acid i to amino acid j $C_{i,j}$ per tree is counted (see Tab. 3.2) and symmetrized ($C_{i,j} = \frac{1}{2}(C_{i,j} + C_{j,i})$) because the trees are not directed (note, that for two sequences the direction of point mutations is ambiguous).

From the constraint of summing to 1 we obtain

$$\forall_i : p_{i,i} = 1 - \sum_{j \neq i} p_{i,j}. \tag{3.2}$$

f_i is the frequency of the presence of an amino acid in a protein (see Tab. 3.1). Further the assumption of a stationary state was made for the PAM matrix computation

$$f_i \, p_{i,j} = f_j \, p_{j,i} \, , \tag{3.3}$$

i.e. the amino acid distribution remains constant (this assumption is incorrect as found out recently).

Gly	0.089	Val	0.065	Arg	0.041	His	0.034
Ala	0.087	Thr	0.058	Asn	0.040	Cys	0.033
Leu	0.085	Pro	0.051	Phe	0.040	Tyr	0.030
Lys	0.081	Glu	0.050	Gln	0.038	Met	0.015
Ser	0.070	Asp	0.047	Ile	0.037	Trp	0.010

Table 3.1: Amino acid frequencies according to Dayhoff et. al (1978).

Under the assumption that a mutation takes place, the probability that amino acid i mutates into amino acid j is

$$c_{i,j} \;=\; \frac{C_{i,j}}{\sum_{l,l\neq i} C_{i,l}}\,, \tag{3.4}$$

the frequency $C_{i,j}$ of changing i to j divided by the number of changes of amino acid i. Note, that the time scale of one mutation is not taken into account.

The mutation probability $p_{i,j}$ should be proportional to $c_{i,j}$ up to a factor m_i "the relative mutability" of amino acid i. m_i accounts for the fact that different amino acids have different mutation rates. Using above constraints we will determine the value of m_i.

We set

$$p_{i,j} \;=\; m_i\, c_{i,j} \;=\; m_i\, \frac{C_{i,j}}{\sum_{l,l\neq i} C_{i,l}} \tag{3.5}$$

and insert this in the steady state assumption

$$f_i\, p_{i,j} \;=\; f_j\, p_{j,i} \tag{3.6}$$

leading to (note $C_{i,j} \;=\; C_{j,i}$)

$$f_i\, m_i\, \frac{C_{i,j}}{\sum_{l,l\neq i} C_{i,l}} \;=\; f_j\, m_j\, \frac{C_{i,j}}{\sum_{l,l\neq j} C_{j,l}}\,. \tag{3.7}$$

We obtain

$$m_i\, \frac{f_i}{\sum_{l,l\neq i} C_{i,l}} \;=\; m_j\, \frac{f_j}{\sum_{l,l\neq j} C_{j,l}} \;:=\; c\,. \tag{3.8}$$

Using the value c in the right hand side of the last equation and solving for m_i gives

$$m_i \;=\; c\, \frac{\sum_{l,l\neq i} C_{i,l}}{f_i}\,. \tag{3.9}$$

We now insert m_i into the equation for $p_{i,j}$:

$$p_{i,j} \;=\; c\, \frac{\sum_{l,l\neq i} C_{i,l}}{f_i} \frac{C_{i,j}}{\sum_{l,l\neq i} C_{i,l}} \;=\; c\, \frac{C_{i,j}}{f_i}\,. \tag{3.10}$$

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

Table 3.2: Cumulative Data for computing PAM with 1572 changes.

The free parameter c must be chosen to obtain 1 mutation per 100 amino acids, i.e.

$$\sum_i f_i\,(1\,-\,p_{i,i})\;=\;\sum_i\sum_{j\neq i} f_i\,p_{i,j}\;= \tag{3.11}$$
$$c\,\sum_i\sum_{j\neq i} f_i\,\frac{C_{i,j}}{f_i}\;=\;c\,\sum_i\sum_{j\neq i} C_{i,j}\;=\;1/100\,,$$

therefore

$$c\;=\;1/\left(100\,\sum_i\sum_{j\neq i} C_{i,j}\right)\,. \tag{3.12}$$

Finally we obtain an expression for $p_{i,j}$:

$$p_{i,j}\;=\frac{C_{i,j}}{100\;f_i\;\sum_i\sum_{j\neq i} C_{i,j}}\,. \tag{3.13}$$

The result of this computation is presented as the PAM 1 matrix in Tab. 3.3 and Tab. 3.4 shows the according PAM 250 matrix.

Now we want to compute the scoring matrix. Towards this end we want to compare a pairing resulting from mutations occurring in nature with the probability of a random pairing. The probability of a mutation in nature is $f_i\;p_{i,j}$, i.e. the probability that amino acid i is present multiplied

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1
C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Q	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2
G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5
H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33
L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
K	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1	1
M	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0	4
F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2
T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
W	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1	0
Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1
V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901

Table 3.3: 1 PAM evolutionary distance (times 10000).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	13	6	9	9	5	8	9	12	6	8	6	7	7	4	11	11	11	2	4	9
R	3	17	4	3	2	5	3	2	6	3	2	9	4	1	4	4	3	7	2	2
N	4	4	6	7	2	5	6	4	6	3	2	5	3	2	4	5	4	2	3	3
D	5	4	8	11	1	7	10	5	6	3	2	5	3	1	4	5	5	1	2	3
C	2	1	1	1	52	1	1	2	2	2	1	1	1	1	2	3	2	1	4	2
Q	3	5	5	6	1	10	7	3	7	2	3	5	3	1	4	3	3	1	2	3
E	5	4	7	11	1	9	12	5	6	3	2	5	3	1	4	5	5	1	2	3
G	12	5	10	10	4	7	9	27	5	5	4	6	5	3	8	11	9	2	3	7
H	2	5	5	4	2	7	4	2	15	2	2	3	2	2	3	3	2	2	3	2
I	3	2	2	2	2	2	2	2	2	10	6	2	6	5	2	3	4	1	3	9
L	6	4	4	3	2	6	4	3	5	15	34	4	20	13	5	4	6	6	7	13
K	6	18	10	8	2	10	8	5	8	5	4	24	9	2	6	8	8	4	3	5
M	1	1	1	1	0	1	1	1	1	2	3	2	6	2	1	1	1	1	1	2
F	2	1	2	1	1	1	1	1	3	5	6	1	4	32	1	2	2	4	20	3
P	7	5	5	4	3	5	4	5	5	3	3	4	3	2	20	6	5	1	2	4
S	9	6	8	7	7	6	7	9	6	5	4	7	5	3	9	10	9	4	4	6
T	8	5	6	6	4	5	5	6	4	6	4	6	5	3	6	8	11	2	3	6
W	0	2	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	55	1	0
Y	1	1	2	1	3	1	1	1	3	2	2	1	2	15	1	2	2	3	31	2
V	7	4	4	4	4	4	4	4	5	4	15	10	4	10	5	5	5	72	4	17

Table 3.4: 250 PAM evolutionary distance (times 100).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	3				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-2	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

Table 3.5: Log-odds matrix for PAM 250.

with the probability that it is mutated into amino acid j. The probability of randomly selecting a pair (with independent selections) is $f_i\, f_j$. The likelihood ratio is

$$\frac{f_i\, p_{i,j}}{f_i\, f_j} \;=\; \frac{p_{i,j}}{f_j} \;=\; \frac{p_{j,i}}{f_i}\;. \tag{3.14}$$

If each position is independent of the other positions then the likelihood ratio for the whole sequence is the product

$$\prod_k \frac{f_{i_k}\, p_{i_k,j_k}}{f_{i_k}\, f_{j_k}} \;=\; \prod_k \frac{p_{i_k,j_k}}{f_{j_k}}\;. \tag{3.15}$$

To handle this product and avoid numerical problems the logarithm is taken and we get a scoring function

$$\sum_k \log\left(\frac{p_{i_k,j_k}}{f_{j_k}}\right)\;. \tag{3.16}$$

The values $\log\left(\frac{p_{i_k,j_k}}{f_{j_k}}\right)$ are called "log-odds-scores" after they are multiplied by a constant and rounded. The "log-odds-scores" for the PAM 250 are summarized in Tab. 3.5. Positive values of the "log-odds-scores" mean that the corresponding pair of amino acids appears more often in aligned homologous sequences than by chance (vice versa for negative values).

For a detailed example of how to calculate the PAM 1 matrix see Appendix C.

3.2.3 BLOSUM Matrices

The PAM matrices are derived from very similar sequences and generalized to sequences which are less similar to each other by matrix multiplication. However, this generalization is not verified.

Henikoff and Henikoff (1992) derived scoring matrices, called "BLOSUM p" (BLOck SUbstitution Matrix). BLOSUM scoring matrices are directly derived from blocks with specified similarity, i.e. different sequence similarities are not computed based on model assumptions which may be incorrect. The data is based on the Blocks data base (see Chapter 2) where similar subsequences are grouped into blocks. Here p refers to the % identity of the blocks, e.g. BLOSUM 62 is derived from blocks with 62 % identity (ungapped ($\rightarrow$)). **The default and most popular scoring matrix for pairwise alignment is the BLOSUM 62 matrix.**

Calculation the BLOSUM matrices:

1. Sequences with at least $p\%$ identity to each other are clustered. Each cluster generates a frequency sequence (relative amino acid frequencies at every position). The frequency sequence represents all sequences of one cluster and similar sequences are down-weighted. In the following we consider only clusters with one sequence, i.e. there are no frequencies. Frequencies will be treated later.

2. The (frequency) sequences are now compared to one another. Pairs of amino acid i and j are counted by $c_{i,j}$ where amino acids are counted according to their frequency. If in column k there are n_i^k amino acids i and n_j^k amino acids j then the count for column k gives

$$c_{i,j}^k \;=\; \begin{cases} \binom{n_i^k}{2} & \text{for} \quad i = j \\ \\ n_i^k n_j^k & \text{for} \quad i > j \end{cases} . \tag{3.17}$$

   Note, that $\binom{n_i^k}{2} \;=\; \frac{1}{2}\left(n_i^k\, n_i^k \;-\; n_i^k\right)$, where the factor $\frac{1}{2}$ accounts for symmetry and $-n_i^k$ subtracts the counts of mutations of the sequence into itself.

3. Compute $c_{i,j} \;=\; \sum_k c_{i,j}^k$ and $Z \;=\; \sum_{i \geq j} c_{i,j} \;=\; \frac{L\; N\; (N-1)}{2}$, where L is the sequence length (column number) and N the number of sequences. Now the $c_{i,j}$ are normalized to obtain the probability

$$q_{i,j} \;=\; \frac{c_{i,j}}{Z} \, . \tag{3.18}$$

   Finally we set $q_{j,i} \;=\; q_{i,j}$ for $i > j$.

4. The probability of the occurrence of amino acid i is

$$q_i \;=\; q_{i,i} \;+\; \sum_{j \neq i} \frac{q_{i,j}}{2} \, , \tag{3.19}$$

   the probability of i not being mutated plus the sum of the mutation probabilities. Note, that $q_{i,j}$ is divided by 2 because mutations from i to j and j to i are counted in step 2.

5. The likelihood ratios $\frac{q_{i,i}}{q_i^2}$ and $\frac{q_{i,j}/2}{q_i\ q_j}$ as well as the log-odds ratios

$$\text{BLOSUM}_{i,j} \;=\; \begin{cases} 2\log_2 \frac{q_{i,i}}{q_i^2} & \text{for} \quad i = j \\ \\ 2\log_2 \frac{q_{i,j}}{2\ q_i\ q_j} & \text{for} \quad i \neq j \end{cases} \tag{3.20}$$

are computed. Note, that the BLOSUM values are actually rounded to integers.

Here an example for computing the BLOSUM matrix, where the first column gives the sequence number and the second the sequence:

1	`NFHV`
2	`DFNV`
3	`DFKV`
4	`NFHV`
5	`KFHR`

In this example we compute BLOSUM100 and keep even the identical subsequences 1 and 4 (which would form one sequence after clustering). Therefore we do not have clusters and each amino acid obtains a unit weight. For example if we cluster the second and third sequence then the frequency sequence is `[D][F][0.5 N, 0.5 K][V]`. For simplicity we do not cluster in the following.

The values of $c_{i,j}$ are:

	R	N	D	H	K	F	V
R	0	-	-	-	-	-	-
N	0	1	-	-	-	-	-
D	0	4	1	-	-	-	-
H	0	3	0	3	-	-	-
K	0	3	2	3	0	-	-
F	0	0	0	0	0	10	-
V	4	0	0	0	0	0	6

$$Z \;=\; 4 \cdot \frac{5 \cdot 4}{2} \;=\; 40 \;=\; \sum_{i \geq j} c_{i,j} \tag{3.21}$$

The values of $q_{i,j}$ are:

	R	N	D	H	K	F	V
R	0	0	0	0	0	0	0.1
N	0	0.025	0.1	0.075	0.075	0	0
D	0	0.1	0.025	0	0.05	0	0
H	0	0.075	0	0.075	0.075	0	0
K	0	0.075	0.05	0.075	0	0	0
F	0	0	0	0	0	0.25	0
V	0.1	0	0	0	0	0	0.15

To compute the single amino acid probabilities q_i, we compute it for N: $0.025 + \frac{1}{2}\,(0.1 \;+\; 0.075 \;+\; 0.075) \;=$ 0.15. The values of q_i are:

R	0.05
N	0.15
D	0.1
H	0.15
K	0.1
F	0.25
V	0.2

The likelihood ratio values are:

	R	N	D	H	K	F	V
R	-	-	-	-	-	-	5
N	-	1.1	3.3	1.7	2.5	-	-
D	-	3.3	2.5	-	2.5	-	-
H	-	1.7	-	3.3	2.5	-	-
K	-	2.5	2.5	2.5	-	-	-
F	-	-	-	-	-	4	-
V	5	-	-	-	-	-	3.8

The log-odds ratios are:

	R	N	D	H	K	F	V
R	-	-	-	-	-	-	4.6
N	-	0.3	3.5	1.5	2.6	-	-
D	-	3.5	3.4	-	2.6	-	-
H	-	1.5	-	3.4	2.6	-	-
K	-	2.6	2.6	2.6	-	-	-
F	-	-	-	-	-	4	-
V	4.6	-	-	-	-	-	3.8

For the case with frequencies stemming from clustering we define $f^k_{i,l}$ as the frequency of amino acid i in the k-th column for the l-th cluster.

$$
\begin{aligned}
c^k_{i,j} \;&=\; \sum_{l,m:l\neq m} f^k_{i,l}\, f^k_{j,m} \;= \\
&\sum_l f^k_{i,l} \sum_{m:m\neq l} f^k_{j,m} \;= \\
&n^k_i\, n^k_j \;-\; \sum_l f^k_{i,l}\, f^k_{j,l}\,,
\end{aligned}
\tag{3.22}
$$

where

$$
n^k_i \;=\; \sum_l f^k_{i,l} \tag{3.23}
$$

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Table 3.6: BLOSUM62 scoring matrix.

and

$$c_{i,i}^{k} \;=\; \frac{1}{2}\left(\left(n_i^k\right)^2 \;-\; \sum_l \left(f_{i,l}^k\right)^2\right) \tag{3.24}$$

With these new formulas for $c_{i,j}^k$ all other computations remain as mentioned.

For a detailed example of how to calculate the BLOSUM75 matrix see Appendix C.

Tab. 3.6 shows the BLOSUM62 scoring matrix computed as shown above but on the BLOCKs data base with $\geq 62\%$ sequence identity.

If we compare the BLOSUM matrices with the PAM matrices then PAM100 $\approx$ BLOSUM90, PAM120 $\approx$ BLOSUM80, PAM160 $\approx$ BLOSUM60, PAM200 $\approx$ BLOSUM52, and PAM250 $\approx$ BLOSUM45.

PAM assumptions are violated because positions are context dependent, i.e. one substitution makes other substitutions more or less likely (dependency between mutations). Further, mutations with low probability are not as well observed. Only the subsequences which are different in very similar sequences are used to compute mutation probabilities. That may introduce a bias towards mutations, i.e. only mutation-rich regions are used.

BLOSUM is not model based in contrast to PAM as it is empirically computed. For example, it does not take the evolutionary relationships into account.

Literature:

- Altschul, S.F. (1991), Amino acid substitution matrices from an information theoretic perspective, J. Mol. Biol. 219, 555-665 (1991).

- Altschul, S. F., M. S. Boguski, W. Gish and J. C. Wootton (1994), Issues in searching molecular sequence databases, Nature Genetics 6:119-129.
- PAM: Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C. (1978), A model of evolutionary change in proteins, In "Atlas of Protein Sequence and Structure" 5(3) M.O. Dayhoff (ed.), 345 - 352.
- GONNET: Gonnet G.H., Cohen M.A., Benner S.A. (1992), Exhaustive matching of the entire protein sequence database, Science 1992 Jun 5;256(5062):1443-5.
- BLOSUM: Henikoff, S. and Henikoff, J. (1992), Amino acid substitution matrices from protein blocks, Proc. Natl. Acad. Sci. USA. 89(biochemistry): 10915 - 10919. 1992.

These measurements of sequence similarities assume that the measurement is context independent, i.e. point-wise. Therefore, the scoring can be expressed by a 20×20 matrix of pairwise scores. However, more complex scores may be possible. Advantage of the simple scores is that they can be used in algorithms which decompose the alignment in aligned amino acid pairs and, therefore, are efficient.

3.2.4 Gap Penalties

In our example

```
BIOINFORMATICS          BIOI-N-FORMATICS
                  ⟶
BOILING FOR MANICS      B-OILINGFORMANICS
```

we inserted "-" into the strings to account for deletions and insertions. A maximal substring consisting of "-" is called "gap".

Obviously gaps are not desired as many gaps indicate a more remote relationship, i.e. more deletions and insertions. Therefore gaps should contribute negatively to the score. But how should gaps be penalized?

The first approach would be to equally penalize each "-". A gap with length l and gap penalty of d would give a linear score of

$$-\,l\,d\,. \tag{3.25}$$

However, from a biological point of view neighboring insertions and deletions are not statistically independent from each other. Reason is that a single mutation event can delete whole substrings or insert whole substrings. Those events are almost as likely as single insertions/deletions. Another reason is that a sequence with introns and exons is matched against a measured sequence. Here the first sequence may be obtained from genome sequencing and the second sequence may be obtained by measuring proteins by X-ray or NMR. Missing introns should not be penalized by their length.

On the other hand, a linear affine gap penalty function is computationally more efficient (as we will see later the alignment computational cost is the product of the length of the strings for linear affine gap penalties). Therefore the cost for a gap is computed as

$$-\,d\,-\,(l\,-\,1)\,e\,, \tag{3.26}$$

where d is the *gap open penalty* and e is the *gap extension penalty*. The penalty for gaps in the whole alignment is $-\ d\ (\text{numbergaps})\ -\ (\text{number}\text{“}-\text{”}\ -\ \text{numbergaps})\ e$.

The optimal alignment with BLOSUM62 as scoring matrix and with affine gap penalty $d = 20$ and $e = 1$ is the following:

```
RKFFVGGNWKMNGDKKSLNGAKLSADTEVVCGAPSIYLDF
|.|||||:|        ||.|.:..:.|||...|:.|||:
RTFFVGGNFK-------LNTASIPENVEVVICPPATYLDY
```

Here "|" indicates a match, ":" similar amino acids, "." less similar amino acids, and a blank a gap. However the optimal alignment with affine gap penalty $d = 1$ and $e = 1$, i.e. a linear gap penalty, is

```
RKFFVGGNWKMNGDKKSL--NGAKLSADTEVV-CGAPSIYLDF
|.|||||:|:|  ..|:  |   :    ||| | .|:.|||:
RTFFVGGNFKLN--TASIPEN---V----EVVIC-PPATYLDY
```

And the optimal alignment with affine gap penalty $d = 4$ and $e = 4$, i.e. a linear gap penalty with higher penalties, is

```
RKFFVGGNWKMNGDKKSLNGAKLSADTEVVCGAPSIYLDF
|.|||||:|:|  ..|:  .: :.:. |:| .|:.|||:
RTFFVGGNFKLN--TASI--PE-NVEV-VIC-PPATYLDY
```

This shows that the number of gaps is minimized with the affine gap penalty with $e < d$. In the last example the gaps are penalized more, therefore, fewer gaps appear in the optimal alignment. Note that the gap penalty must be related with the BLOSUM62 scoring matrix.

The probability of amino acids aligned to a gap is assumed to be random, i.e. no amino acid has a preference for a gap (is that true?). That means only the gap itself and its length may have a certain probability. Assume that the likelihood of generating a gap is 2^{-d} and the likelihood of extending a gap is 2^{-e} then the $\log_2$-likelihood of a gap is exactly the affine gap penalty. Therefore the gap penalty also fits into the log-odds ratio framework. The expected scoring contribution of gaps in random sequences is an additive constant for a certain sequence length. Therefore it can be neglected in the log-odds ratios.

Until now only the evaluation of alignments, i.e. computing the similarity of sequences was treated. To optimize the alignment is independent of the similarity measure, if it is a point-wise measure. In the following algorithms for finding the optimal alignment are presented.

3.3 Alignment Algorithms

Alignment algorithms optimize the scoring of the sequences by pairing their amino acids and inserting gaps. The sequential order of the original amino acids must be kept. For pairing only shifting the whole sequences and inserting gaps is allowed. The alignment algorithms can be classified as being global or local. Global alignment algorithms optimize the alignment of two

sequences whereas local alignment algorithms search for high scoring local alignments. Local alignments may detect similarities even if the sequences have different lengths because of (alternative) splicing or mutations which glue domains together. Also remote homologous sequences with conservative regions (important for function or structure) may be detected by local alignment.

3.3.1 Global Alignment — Needleman-Wunsch

If you remember our simple game then you know that this was an alignment task. How did you solve it? One possibility would be to try out all possible alignments, compute their scores and choose the best scoring alignment. However that would be too expensive. Assume both sequences have the same length and let us assume that we only match subsequences to one another. $\binom{n}{i}$ subsequences of length i exists for each sequence. The unmatched elements can be computed at once. We have $\binom{n}{i}^2$ sequence pairs which must be evaluated in i steps. The complexity is

$$\sum_i \binom{n}{i}^2 i \geq \sum_i \binom{n}{i}^2 = \binom{2n}{n} \approx \tag{3.27}$$
$$\sqrt{4\,\pi\,n}\,(2n/e)^{2n} / \left(\sqrt{2\pi\,n}\,(n/e)^n\right)^2 = 2^{2n}/\sqrt{\pi\,n}\,.$$

The approximation stems from Stirling's formula.

The naive approach does not work in practice because the number of operations increases exponentially with the sequence length.

1970 Needleman and Wunsch used the idea of dynamic programming (cf. Bellman) to introduce alignment algorithms which are practical.

3.3.1.1 Linear Gap Penalty

The idea is that the alignment of two sequences of length n and m can be reduced to the alignment of two sequences of length $(n-1)$ and $(m-1)$ (match) or of two sequences of length $(n-1)$ and m (gap in the second sequence) or of two sequences of length n and $(m-1)$ (gap in the first sequence).

Consider two sequences ending with `x` and `y`, then the following cases are possible for the optimal alignment:

match	gap	gap
`x`	`?x`	`x-`
`y`	`y-`	`?y`

Either the ends match or the end of one sequence is more to the right than the end of the other sequence. In the latter case either the end of the first sequence is matched with a space or the end of the second sequence.

We obtain immediately a recursion for the optimal score $S(n,m)$ of two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with elements x_i, $1 \leq i \leq n$, and y_j, $1 \leq j \leq m$, respectively. d denotes the gap penalty and s

the scoring function for two amino acids (s maps amino acid pairs to the corresponding entry in the scoring matrix). The recursion is

$$S(i,j) \;=\; \max \begin{cases} S(i-1,j-1) \;+\; s(x_i,y_j) \\ S(i-1,j) \;-\; d \\ S(i,j-1) \;-\; d \end{cases} \tag{3.28}$$

with start

$$S(0,0) \;=\; 0 \;\text{ and }\; S(-1,j) \;=\; S(i,-1) \;=\; -\infty\,. \tag{3.29}$$

This leads to $S(0,j) = -\,j\,d$ and $S(i,0) = -\,i\,d$. The optimal score can be written in an $n \times m$ matrix:

	0	y_1	$\dots$	y_{j-1}		y_j	$\dots$	y_m
0	$S(0,0)$	$S(0,1)$	$\dots$				$\dots$	$S(0,m)$
x_1	$S(1,0)$	$S(1,1)$	$\dots$				$\dots$	$S(1,m)$
x_2			$\dots$				$\dots$	
x_3			$\dots$				$\dots$	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\ddots$	$\vdots$
x_{i-1}				$S(i-1,j-1)$		$S(i-1,j)$		
					$\searrow$	$\downarrow$		
x_i				$S(i,j-1)$	$\rightarrow$	$S(i,j)$		
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\ddots$	$\vdots$
x_n	$S(n,0)$	$S(n,1)$	$\dots$				$\dots$	$S(n,m)$

The table can be filled from the left upper corner to the right lower corner through the recursion.

During filling the matrix also the $S(i-1,j-1), S(i-1,j), S(i,j-1)$ which was chosen by the maximum and from which $S(i,j)$ is computed should be memorized, e.g. in a variable $B(i,j) = (i{-}1,j{-}1)$ or $(i{-}1,j)$ or $(i,j{-}1)$. This variable allows to generate the alignment by determining how $S(n,m)$ is computed. The alignment is generated by backtracking with $B(i,j)$ starting from (n,m). During backtracking the alignment can be generated:

$$\textbf{if } B(i,j) \;=\; \begin{cases} (i-1,j-1) & \textbf{then print } \begin{smallmatrix} x_i \\ y_j \end{smallmatrix} \\ (i-1,j) & \textbf{then print } \begin{smallmatrix} x_i \\ - \end{smallmatrix} \\ (i,j-1) & \textbf{then print } \begin{smallmatrix} - \\ y_j \end{smallmatrix} \end{cases}. \tag{3.30}$$

Alg. 3.1 shows a pseudo code of the Needleman-Wunsch global alignment algorithm with linear gap penalty. It can be seen that the algorithm has time and memory complexity of $O(n\,m)$. An example (from above) for the backtracking algorithm is shown in tables 3.7 and 3.8.

For a detailed example of the Needleman-Wunsch global alignment algorithm with linear gap penalty see Appendix C.

Algorithm 3.1 Needleman-Wunsch with linear gap

Input: two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with length n and m, respectively; scoring matrix s, gap penalty d
Output: optimal global alignment and its score

BEGIN INITIALIZATION
$S(0,0) = 0$, $S(0,j) = -\,j\,d$, $1 \le j \le m$, and $S(i,0) = -\,i\,d$, $1 \le i \le n$
END INITIALIZATION

BEGIN PROCEDURE
for $1 \le i \le n$ **do**
for $1 \le j \le m$ **do**
$a(i-1,j-1) = S(i-1,j-1) + s(x_i, y_j)$, $a(i-1,j) = S(i-1,j) - d$,
$a(i,j-1) = S(i,j-1) - d$
$S(i,j) = \max\{a(i-1,j-1), a(i-1,j), a(i,j-1)\}$
$B(i,j) = \arg\max\{a(i-1,j-1), a(i-1,j), a(i,j-1)\}$
end for
end for
print "Score: " $S(n,m)$

$(i,j) = (n,m)$
while $(i,j) \neq (0,0)$ **do**

$$\textbf{if } B(i,j) = \begin{cases} (i-1,j-1) & \textbf{then print } \begin{smallmatrix} x_i \\ y_j \end{smallmatrix} \\ (i-1,j) & \textbf{then print } \begin{smallmatrix} x_i \\ - \end{smallmatrix} \\ (i,j-1) & \textbf{then print } \begin{smallmatrix} - \\ y_j \end{smallmatrix} \end{cases}$$

$(i,j) = B(i,j)$
end while
END PROCEDURE

	G	A	K	L	S	A	D	T	E	V	V	C	G	A	P	S	I	Y	L	D	F
	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34	-35	-36	-37	-38	-39	-40
R	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34
T	-12	-13	-14	-15	-16	-17	-18	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27	-28
F	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-21	-22	-23	-21
F	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-17
V	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15
G	12	11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
G	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
N	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6
F	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	13	12	11	13
K	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14
L	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19
N	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26
T	46	46	45	44	44	43	42	45	44	43	42	41	40	39	38	37	36	35	34	33	32
A	47	50	49	48	47	48	47	46	45	44	43	42	41	44	43	42	41	40	39	38	37
S	49	49	50	49	52	51	50	49	48	47	46	45	44	43	43	47	46	45	44	43	42
I	52	51	50	52	51	51	50	49	48	51	50	49	48	47	46	46	51	50	49	48	47
P	51	51	50	51	51	50	50	49	48	50	49	48	47	47	54	53	52	51	50	49	48
E	52	51	52	51	51	50	52	51	54	53	52	51	50	49	53	54	53	52	51	52	51
N	←**57**	←**56**	←**55**	54	53	52	51	52	53	51	50	49	51	50	52	54	53	52	51	52	51
V	55	57	56	↖**56**	←**55**	←**54**	←**53**	←**52**	52	57	56	55	54	53	52	53	57	56	55	54	53
E	55	56	58	57	56	55	56	55	↖**57**	56	55	54	53	53	52	52	56	55	54	57	56
V	54	55	57	59	58	57	56	56	56	↖**61**	60	59	58	57	56	55	55	55	56	56	56
V	53	54	56	58	57	58	57	56	55	60	↖**65**	64	63	62	61	60	59	58	57	56	55
I	52	53	55	58	57	57	55	56	55	59	↑**64**	64	63	62	61	60	64	63	62	61	60
C	51	52	54	57	57	57	56	55	54	58	63	↖**73**	←**72**	71	70	69	68	67	66	65	64
P	50	51	53	56	56	56	56	55	54	57	62	72	71	↖**71**	78	77	76	75	74	73	72
P	49	50	52	55	55	55	55	55	54	56	61	71	70	70	↖**78**	77	76	75	74	73	72
A	50	53	52	54	56	59	58	57	56	55	60	70	71	74	77	↖**79**	78	77	76	75	74
T	49	52	52	53	55	58	58	63	62	61	60	69	70	73	76	78	↖**78**	77	76	75	74
Y	48	51	51	52	54	57	57	62	61	61	60	68	69	72	75	77	77	↖**85**	84	83	82
L	47	50	50	55	54	56	56	61	60	62	62	67	68	71	74	76	79	84	↖**89**	88	87
D	47	49	49	54	55	55	62	61	63	62	61	66	67	70	73	75	78	83	88	↖**95**	94
Y	45	48	48	53	54	54	61	60	62	62	61	65	66	69	72	74	77	85	87	94	↖**98**

Table 3.7: Example for backtracking in the Needleman-Wunsch algorithm with linear penalties $d = 1$ and BLOSUM62 (part 1).

		R	K	F	F	V	G	G	N	W	K	M	N	G	D	K	K	S	L	N	G
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20
R	-1	↖**5**	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
T	-2	4	↖**4**	3	2	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-9	-10	-11	-12
F	-3	3	3	↖**10**	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7
F	-4	2	2	9	↖**16**	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	-5	1	1	8	15	↖**20**	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
G	-6	0	0	7	14	19	↖**26**	25	24	23	22	21	20	19	18	17	16	15	14	13	12
G	-7	-1	-1	6	13	18	25	↖**32**	31	30	29	28	27	26	25	24	23	22	21	20	19
N	-8	-2	-1	5	12	17	24	31	↖**38**	37	36	35	34	33	32	31	30	29	28	27	26
F	-9	-3	-2	5	11	16	23	30	37	↖**39**	38	37	36	35	34	33	32	31	30	29	28
K	-10	-4	2	4	10	15	22	29	36	38	↖**44**	43	42	41	40	39	38	37	36	35	34
L	-11	-5	1	3	9	14	21	28	35	37	43	↖**46**	45	44	43	42	41	40	41	40	39
N	-12	-6	0	2	8	13	20	27	34	36	42	45	↖**52**	←**51**	←**50**	49	48	47	46	47	46
T	-13	-7	-1	1	7	12	19	26	33	35	41	44	51	50	50	↖**49**	48	49	48	47	46
A	-14	-8	-2	0	6	11	18	25	32	34	40	43	50	51	50	49	↖**48**	49	48	47	47
S	-15	-9	-3	-1	5	10	17	24	31	33	39	42	49	50	51	50	49	↖**52**	51	50	49
I	-16	-10	-4	-2	4	9	16	23	30	32	38	41	48	49	50	49	48	51	↖**54**	53	52
P	-17	-11	-5	-3	3	8	15	22	29	31	37	40	47	48	49	49	48	50	↑**53**	52	51
E	-18	-12	-6	-4	2	7	14	21	28	30	36	39	46	47	50	50	50	49	↑**52**	53	52
N	-19	-13	-7	-5	1	6	13	20	27	29	35	38	45	46	49	50	50	51	51	↖**58**	←**57**
V	-20	-14	-8	-6	0	5	12	19	26	28	34	37	44	45	48	49	49	50	52	57	55
E	-21	-15	-9	-7	-1	4	11	18	25	27	33	36	43	44	47	49	50	49	51	56	55
V	-22	-16	-10	-8	-2	3	10	17	24	26	32	35	42	43	46	48	49	48	50	55	54
V	-23	-17	-11	-9	-3	2	9	16	23	25	31	34	41	42	45	47	48	47	49	54	53
I	-24	-18	-12	-10	-4	1	8	15	22	24	30	33	40	41	44	46	47	46	49	53	52
C	-25	-19	-13	-11	-5	0	7	14	21	23	29	32	39	40	43	45	46	46	48	52	51
P	-26	-20	-14	-12	-6	-1	6	13	20	22	28	31	38	39	42	44	45	45	47	51	50
P	-27	-21	-15	-13	-7	-2	5	12	19	21	27	30	37	38	41	43	44	44	46	50	49
A	-28	-22	-16	-14	-8	-3	4	11	18	20	26	29	36	37	40	42	43	45	45	49	50
T	-29	-23	-17	-15	-9	-4	3	10	17	19	25	28	35	36	39	41	42	44	44	48	49
Y	-30	-24	-18	-14	-10	-5	2	9	16	19	24	27	34	35	38	40	41	43	43	47	48
L	-31	-25	-19	-15	-11	-6	1	8	15	18	23	26	33	34	37	39	40	42	47	46	47
D	-32	-26	-20	-16	-12	-7	0	7	14	17	22	25	32	33	40	39	39	41	46	48	47
Y	-33	-27	-21	-17	-13	-8	-1	6	13	16	21	24	31	32	39	38	38	40	45	47	45

Table 3.8: Example for backtracking in the Needleman-Wunsch algorithm with linear penalties $d = 1$ and BLOSUM62 (part 2).

3.3.1.2 Affine Gap Penalty

The problem for affine gap penalties is that long term dependencies appear. For linear gap penalties we considered

match	gap	gap
`x`	`?x`	`x-`
`y`	`y-`	`?y`

to derive the recursion. However, for affine gap penalties introducing a gap at the end of a sequence implies a gap opening event earlier in the sequence. Therefore all earlier gap opening events must be considered:

$$S(i,j) \;=\; \max \begin{cases} S(i-1,j-1) \;+\; s(x_i,y_j) \\ S(i-k,j) \;-\; d \;-\; (k-1)\,e\,, \;\; 1 \le k \le i \\ S(i,j-k) \;-\; d \;-\; (k-1)\,e\,, \;\; 1 \le k \le j \end{cases} . \tag{3.31}$$

For two sequences of length n the complexity is $O\left(n^3\right)$, because all $S(i,j)$ must be considered ($O\left(n^2\right)$) and a consideration is of order $O\left(n\right)$ through checking all previous gap openings.

The idea to make the algorithm with affine gap penalties efficient is to propagate 3 matrices:

- $G_d(i,j)$ best score up to position (i,j) and no gap at the end
- $G_y(i,j)$ best score up to position (i,j) with a gap in the sequence $\boldsymbol{y}$ at position j
- $G_x(i,j)$ best score up to position (i,j) with a gap in the sequence $\boldsymbol{x}$ at position i

For the matrices G_x and G_y one has to control whether extending an existing gap or introducing a new gap gives a better score.

The recursion equations are

$$G_y(i,j) \;=\; \max \begin{cases} G_d(i-1,j) \;-\; d \\ G_y(i-1,j) \;-\; e \end{cases} , \tag{3.32}$$

$$G_x(i,j) \;=\; \max \begin{cases} G_d(i,j-1) \;-\; d \\ G_x(i,j-1) \;-\; e \end{cases} \quad \text{and} \tag{3.33}$$

$$G_d(i,j) \;=\; \max\{G_d(i-1,j-1), G_y(i-1,j-1), \tag{3.34}$$

$$G_x(i-1,j-1)\} \;+\; s(x_i,y_j) \tag{3.35}$$

The initialization $G_d(0,0) = 0, G_y(0,0) = -\infty$ and $G_x(0,0) = -\infty$ leads to $G_y(i,0) = -d - (i-1)\,e$, $G_x(i,0) = -\infty$, $G_x(0,j) = -d - (j-1)\,e$, $G_y(0,j) = -\infty$. $G_d(i,0) = G_d(0,j) = -\infty$.

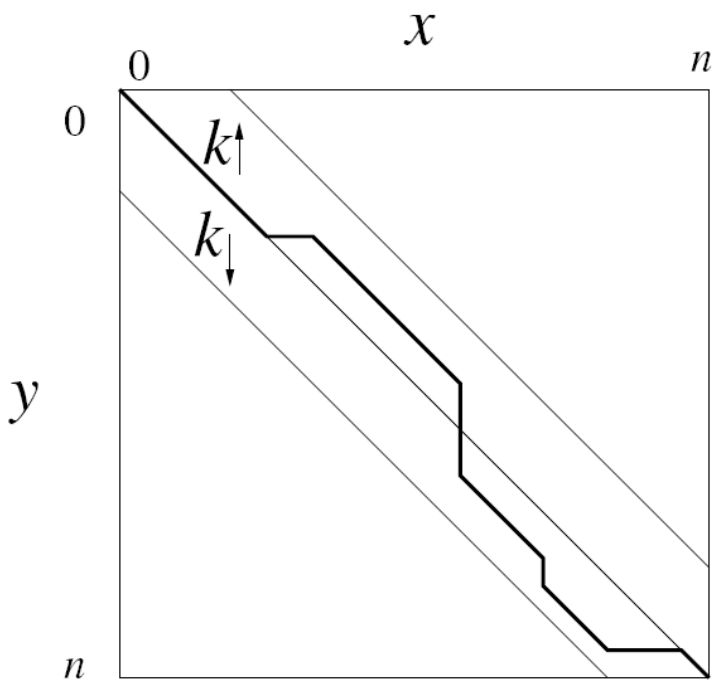


Figure 3.3: The idea of the banded global alignment algorithm is depicted.

Note, that for being mathematically correct, $G_y(i,j)$ has to consider $G_y(i,j-1)$ and $G_x(i,j)$ the previous $G_y(i-1,j)$. However, these two cases mean two consecutive gaps, one in the $\boldsymbol{x}$ and one in $\boldsymbol{y}$, which is biologically not reasonable. It would mean that two indel mutations are more likely than one point mutation. Therefore, for a reasonable setting, two gap opening costs are higher than each mismatch cost. Thus, these cases of two consecutive gaps in different sequences are not possible.

Alg. 3.2 shows a pseudo code of the Needleman-Wunsch algorithm for affine gap penalty. Again the algorithm has time and memory complexity of $O(n\ m)$. An example (from above) for the backtrack algorithms is shown in tables 3.9 and 3.10.

For a detailed example of the Needleman-Wunsch global alignment algorithm with affine gap penalty see Appendix C.

3.3.1.3 KBand Global Alignment

The global alignment algorithms can be made faster if the sequence similarity is high. As seen at the dot matrices for high sequence similarities most backtracking paths are on the main diagonal of the matrix. Therefore, instead of computing the whole matrix only a band around the main diagonal is filled. All solutions are computed within this band. An additional parameter for such an algorithm is k, the width of the band. k constraints the allowed differences of accumulated gaps in the sequences. Fig. 3.3 depicts the idea.

The new parameter k may be set to the estimated gaps in the alignment. If both sequences have the length n, then to leave the band with linear gap penalty costs $-2(k+1)d$, which can be

Algorithm 3.2 Needleman-Wunsch with affine gap

Input: two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with length n and m, respectively; scoring matrix s, gap opening penalty d and gap extend penalty e
Output: optimal global alignment and its score

BEGIN INITIALIZATION
$G_d(0,0) = 0,\ G_x(0,0) = -d-(n+m)\,e,\ G_y(0,0) = -d-(n+m)\,e$
for $1 \le j \le m$
$G_x(0,j) = -d\ -\ (j-1)\,e,\ G_y(0,j) = G_d(0,j) = -d-(n+m)\,e,\ B_x(0,j) =$ "x"
for $1 \le i \le n$
$G_y(i,0) = -d\ -\ (i-1)\,e,\ G_x(i,0) = G_d(i,0) = -d-(m+n)\,e,\ B_y(i,0) =$ "y"
END INITIALIZATION

BEGIN PROCEDURE
for $1 \le i \le n$ **do**
for $1 \le j \le m$ **do**
$G_x(i,j) = \max\{G_d(i,j-1) - d\ ,\ G_x(i,j-1) - e\}$
if $G_x(i,j) = G_d(i,j-1) - d$ **then** $B_x(i,j) =$ "d" **else** $B_x(i,j) =$ "x"
$G_y(i,j) = \max\{G_d(i-1,j) - d\ ,\ G_y(i-1,j) - e\}$
if $G_y(i,j) = G_d(i-1,j) - d$ **then** $B_y(i,j) =$ "d" **else** $B_y(i,j) =$ "y"
$G_d(i,j) = \max\{G_d(i-1,j-1), G_y(i-1,j-1), G_x(i-1,j-1)\} + s(x_i,y_j)$
if $G_d(i,j) = G_d(i-1,j-1) + s(x_i,y_j)$ **then** $B_d(i,j) =$ "d"
if $G_d(i,j) = G_y(i-1,j-1) + s(x_i,y_j)$ **then** $B_d(i,j) =$ "y"
if $G_d(i,j) = G_x(i-1,j-1) + s(x_i,y_j)$ **then** $B_d(i,j) =$ "x"
end for
end for
$score = \max\{G_d(n,m)\ ,\ G_x(n,m)\ ,\ G_y(n,m)\}$
print "Score: " $score$

if $G_d(n,m) = score$ **then** $t =$ "d"
if $G_x(n,m) = score$ **then** $t =$ "x"
if $G_y(n,m) = score$ **then** $t =$ "y"
$(i,j) = (n,m)$
while $(i,j) \neq (0,0)$ **do**

$$\textbf{if } t = \begin{cases} \text{"d"} & \textbf{then print } \begin{smallmatrix} x_i \\ y_j \end{smallmatrix}; \ \ i = i-1,\ j = j-1,\ t = B_d(i,j) \\ \text{"y"} & \textbf{then print } \begin{smallmatrix} x_i \\ - \end{smallmatrix}; \ \ i = i-1,\ t = B_y(i,j) \\ \text{"x"} & \textbf{then print } \begin{smallmatrix} - \\ y_j \end{smallmatrix}; \ \ j = j-1,\ t = B_x(i,j) \end{cases}$$

end while
END PROCEDURE

	G	A	K	L	S	A	D	T	E	V	V	C	G	A	P	S	I	Y	L	D	F
	-39	-40	-41	-42	-43	-44	-45	-46	-47	-48	-49	-50	-51	-52	-53	-54	-55	-56	-57	-58	-59
R	-33	-34	-35	-36	-37	-38	-39	-40	-41	-42	-43	-44	-45	-46	-47	-48	-49	-50	-51	-52	-53
T	-33	-33	-35	-36	-35	-37	-39	-34	-41	-41	-42	-44	-45	-45	-47	-46	-49	-50	-51	-52	-53
F	-26	-27	-28	-29	-30	-31	-32	-33	-34	-35	-36	-37	-38	-39	-40	-41	-42	-43	-44	-45	-46
F	-19	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34	-35	-36	-37	-38	-39
V	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34
G	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27
G	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20
N	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13
F	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-6	-9	-10	-6
K	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5
L	18	17	16	17	14	13	12	11	10	9	8	7	6	5	4	3	2	1	2	-1	-2
N	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
T	↖**26**	25	23	22	23	21	19	24	17	17	16	14	12	13	11	12	9	7	7	6	4
A	27	↖**30**	24	22	23	27	19	19	23	17	17	16	14	16	12	12	11	7	6	5	4
S	25	28	↖**30**	22	26	24	27	20	19	21	15	16	16	15	15	16	10	9	6	6	4
I	26	24	25	↖**32**	20	25	21	26	17	22	24	14	12	15	12	13	20	9	11	3	6
P	23	25	23	22	↖**31**	19	24	20	25	15	20	21	12	11	22	11	10	17	6	10	2
E	31	22	26	20	22	↖**30**	21	23	25	23	13	16	19	11	10	22	8	8	14	8	7
N	27	29	25	24	23	22	↖**31**	21	23	22	20	16	16	17	13	12	19	10	9	15	7
V	44	27	27	26	22	23	19	↖**31**	19	27	26	19	13	16	15	11	15	18	11	6	14
E	25	43	28	24	26	21	25	18	↖**36**	17	25	22	17	12	15	15	9	13	15	13	5
V	27	25	41	29	22	26	18	25	16	↖**40**	21	24	19	17	16	15	18	13	14	12	12
V	22	27	23	42	27	22	23	19	23	20	↖**44**	24	23	22	21	20	19	18	17	16	15
I	21	21	24	25	40	26	19	22	17	26	24	↖**43**	23	22	21	20	24	18	20	16	16
C	20	21	19	23	24	40	23	19	18	18	25	33	↖**40**	23	19	20	19	22	17	17	14
P	20	19	20	20	22	23	39	22	18	17	22	22	31	↖**39**	30	18	17	16	19	16	13
P	19	19	18	19	19	21	22	38	21	17	21	21	20	30	↖**46**	29	25	24	23	22	21
A	20	23	18	18	20	23	19	22	37	21	20	21	21	24	29	↖**47**	28	26	25	24	23
T	17	20	22	17	19	20	22	24	21	37	21	19	19	21	25	30	↖**46**	26	25	24	23
Y	15	15	18	21	15	17	17	20	22	20	36	19	16	17	24	26	29	↖**53**	33	32	31
L	14	14	13	22	19	15	15	16	17	23	21	35	15	15	23	25	28	33	↖**57**	37	36
D	15	13	13	14	22	17	21	14	18	15	20	18	34	14	22	24	24	32	37	↖**63**	43
Y	15	13	11	13	12	20	14	19	13	17	15	18	15	32	21	23	23	31	36	43	↖**66**

Table 3.9: Example for backtracking in the Needleman-Wunsch algorithm with affine penalties $d = 20, e = 1$ and BLOSUM62 (part 1).

		R	K	F	F	V	G	G	N	W	K	M	N	G	D	K	K	S	L	N	G
	0	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33	-34	-35	-36	-37	-38	-39
R	-20	↖ **5**	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27	-28	-29	-30	-31	-32	-33
T	-21	-15	↖ **4**	-16	-17	-17	-19	-20	-20	-22	-23	-24	-24	-26	-27	-28	-29	-28	-31	-31	-33
F	-22	-16	-16	↖ **10**	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26
F	-23	-17	-17	-10	↖ **16**	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19
V	-24	-18	-18	-11	-4	↖ **20**	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
G	-25	-19	-19	-12	-5	0	↖ **26**	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7
G	-26	-20	-20	-13	-6	-1	6	↖ **32**	12	11	10	9	8	7	6	5	4	3	2	1	0
N	-27	-21	-20	-14	-7	-2	5	12	↖ **38**	18	17	16	15	14	13	12	11	10	9	8	7
F	-28	-22	-22	-14	-8	-3	4	11	18	↖ **39**	19	18	17	16	15	14	13	12	11	10	9
K	-29	-23	-17	-16	-9	-4	3	10	17	19	↖ **44**	←**24**	←**23**	←**22**	←**21**	←**20**	←**19**	←**18**	17	16	15
L	-30	-24	-24	-17	-10	-5	2	9	16	18	24	46	26	25	24	23	22	21	↖ **22**	19	18
N	-31	-25	-24	-18	-11	-6	1	8	15	17	23	26	52	32	31	30	29	28	27	↖ **28**	25
T	-32	-26	-26	-19	-12	-7	0	7	14	16	22	25	32	50	31	30	29	30	27	27	↖ **26**
A	-33	-27	-27	-20	-13	-8	-1	6	13	15	21	24	31	32	48	30	29	30	29	25	27
S	-34	-28	-27	-21	-14	-9	-2	5	12	14	20	23	30	31	32	48	30	33	28	30	25
I	-35	-29	-29	-22	-15	-10	-3	4	11	13	19	22	29	28	28	29	45	28	35	25	26
P	-36	-30	-30	-23	-16	-11	-4	3	10	12	18	21	28	27	27	27	28	44	25	33	23
E	-37	-31	-29	-24	-17	-12	-5	2	9	11	17	20	27	26	29	28	28	28	41	25	31
N	-38	-32	-31	-25	-18	-13	-6	1	8	10	16	19	26	27	27	29	28	29	25	47	27
V	-39	-33	-33	-26	-19	-14	-7	0	7	9	15	18	25	24	24	25	27	26	30	27	44
E	-40	-34	-32	-27	-20	-15	-8	-1	6	8	14	17	24	23	26	25	26	27	23	30	25
V	-41	-35	-35	-28	-21	-16	-9	-2	5	7	13	16	23	22	21	24	23	24	28	25	27
V	-42	-36	-36	-29	-22	-17	-10	-3	4	6	12	15	22	21	20	21	22	21	25	25	22
I	-43	-37	-37	-30	-23	-18	-11	-4	3	5	11	14	21	20	19	20	18	20	23	23	21
C	-44	-38	-38	-31	-24	-19	-12	-5	2	4	10	13	20	19	18	19	17	17	19	22	20
P	-45	-39	-39	-32	-25	-20	-13	-6	1	3	9	12	19	18	18	18	18	16	14	21	20
P	-46	-40	-40	-33	-26	-21	-14	-7	0	2	8	11	18	17	17	17	17	17	13	20	19
A	-47	-41	-41	-34	-27	-22	-15	-8	-1	1	7	10	17	18	15	16	16	18	16	19	20
T	-48	-42	-42	-35	-28	-23	-16	-9	-2	0	6	9	16	15	17	15	15	17	17	18	17
Y	-49	-43	-43	-36	-29	-24	-17	-10	-3	0	5	8	15	14	13	15	13	13	16	17	15
L	-50	-44	-44	-37	-30	-25	-18	-11	-4	-2	4	7	14	13	12	13	13	11	17	16	14
D	-51	-45	-45	-38	-31	-26	-19	-12	-5	-3	3	6	13	13	19	12	12	13	8	18	15
Y	-52	-46	-46	-39	-32	-27	-20	-13	-6	-3	2	5	12	11	10	17	10	10	12	14	15

Table 3.10: Example for backtracking in the Needleman-Wunsch algorithm with affine penalties $d = 20, e = 1$ and BLOSUM62 (part 2).

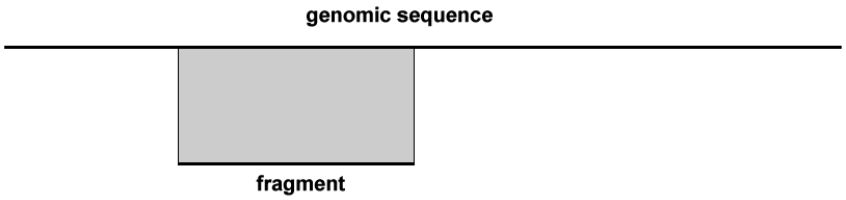


Figure 3.4: A fragment in a DNA sequence.

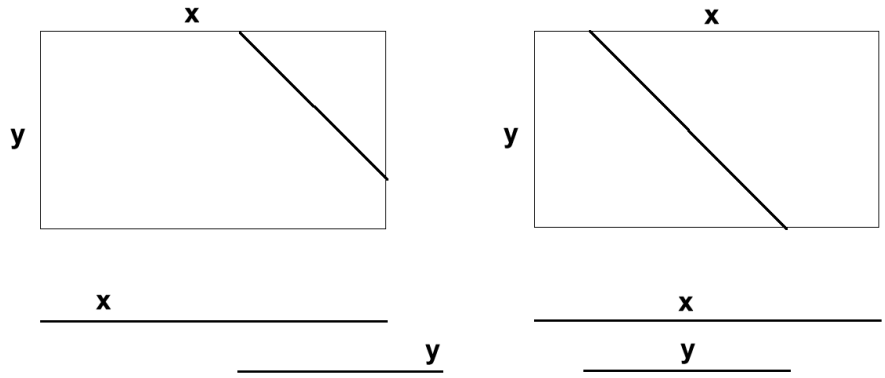


Figure 3.5: Fragments are aligned off the main diagonal.

used to estimate a good k. But also the differences of the sequences' lengths must be taken into account.

The KBand algorithm can be extended to an iterative algorithm with increasing k. However, its running time still depends on the similarity between the sequences.

3.3.2 Local Alignment — Smith-Waterman

The global alignment matches two sequences completely and does not take into account that parts of the sequences match very good. Many proteins which are remotely related (homologous) share subsequences which have much higher similarity than random sequences, even if the global alignment does not result in a high score. Often these similar subsequences are conservative because they are important for the function or for the folding of the protein. Currently most relations between proteins are found by local alignment methods. Also the number of high scoring subsequences is an indicator whether these sequences are homologous or not. For genomic DNA sequences often only fragments are matched (see Fig. 3.4). In these cases also the lengths do not matter and the ends of the fragments need not match, which leads to alignments on diagonals off the main diagonal (see Fig. 3.5).

The main idea of local alignment algorithms is that negative scores are avoided. Negative scores indicate that the subsequence is not homologous. At every position the algorithm can decide whether it will use a prefix match and extend it or to start a new match. The new aspect is to start a new match which can be indicated by deleting the prefix score and setting $S(i, j) = 0$.

Again we start with the linear gap penalty, which gives for the recursion

$$S(i,j) \;=\; \max \begin{cases} 0 \\ S(i-1,j-1) \;+\; s(x_i,y_j) \\ S(i-1,j) \;-\; d \\ S(i,j-1) \;-\; d \end{cases} \tag{3.36}$$

with start

$$S(0,0) \;=\; 0. \tag{3.37}$$

However, this time $S(i,0) = S(0,j) = 0$.

Because we want to find the best matching subsequence, we have to look for the maximal $S(i,j)$ which gives the score. Backtracking is also started at the position with maximal score and may end whenever a 0 is reached.

Alg. 3.3 shows a pseudo code of the Smith-Waterman local alignment algorithm. Again the algorithm has time and memory complexity of $O(n\ m)$.

The Smith-Waterman algorithm for affine gap penalty is analogous to its Needleman-Wunsch counterpart. In contrast to the Needleman-Wunsch algorithm every maximization is kept non-negative by an additional 0 in the set to maximize.

An example (from above) for the local alignment backtracking algorithm is shown in tables 3.11 and 3.12 for affine gap with $d = 20$ and $e = 4$ (e is increased to 4 in order to avoid the global solution). The best local alignment has a score of 52:

```
RKFFVGGNWKMN
|.|||||:|.|
RTFFVGGNFKLN
```

and the second best score of 50 is:

```
GDKKSLNGAKLSADTEVVCGAPSIYLDF
|....||.|.:..:.|||...|:.|||:
GGNFKLNTASIPENVEVVICPPATYLDY
```

For detailed examples of the Smith-Waterman local alignment algorithm with linear or affine gap penalty see Appendix C.

3.3.3 Fast Approximations: FASTA, BLAST and BLAT

The algorithms so far have time complexity of $O(nm)$ (except for KBand algorithms which only work for similar sequences). Because the number of new sequences increases very fast through the various genomes which are sequenced, the $O(nm)$ algorithms are too slow to compare a sequence to all known sequences. For example, the NR data base of non-redundant (non-identic) sequences contains more than 3 mio. sequences. For secondary structure prediction or for protein

Algorithm 3.3 Smith-Waterman with linear gap

Input: two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with length n and m, respectively; scoring matrix s, gap penalty d
Output: optimal local alignment and its score

BEGIN INITIALIZATION
$S(i,0) = S(0,j) = 0$ for $0 \leq j \leq m$ and $0 \leq i \leq n$
END INITIALIZATION

BEGIN PROCEDURE
for $1 \leq i \leq n$ **do**
for $1 \leq j \leq m$ **do**
$a(i-1,j-1) = S(i-1,j-1) + s(x_i, y_j)$, $a(i-1,j) = S(i-1,j) - d$, $a(i,j-1) = S(i,j-1) - d$
$S(i,j) = \max\{0, a(i-1,j-1), a(i-1,j), a(i,j-1)\}$
if $S(i,j) > 0$ **then** $B(i,j) = \arg\max\{0, a(i-1,j-1), a(i-1,j), a(i,j-1)\}$ **else** $B(i,j) = (-1,-1)$
end for
end for
$(i,j) = \arg\max\{S(i,j) \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$
print "Score: " $S(i,j)$

while $S(i,j) \neq 0$ **do**

$$\textbf{if } B(i,j) = \begin{cases} (i-1,j-1) & \textbf{then print } \begin{smallmatrix} x_i \\ y_j \end{smallmatrix} \\ (i-1,j) & \textbf{then print } \begin{smallmatrix} x_i \\ - \end{smallmatrix} \\ (i,j-1) & \textbf{then print } \begin{smallmatrix} - \\ y_j \end{smallmatrix} \end{cases}$$

$(i,j) = B(i,j)$
end while
END PROCEDURE

		R	K	F	F	V	G	G	N	W	K	M	N	G	D	K	K	S	L	N	G
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	↖ **5**	2	0	0	0	0	0	0	0	2	0	0	0	0	2	2	0	0	0	0
T	0	0	↖ **4**	0	0	0	0	0	0	0	0	1	0	0	0	0	1	3	0	0	0
F	0	0	0	↖ **10**	6	0	0	0	0	1	0	0	0	0	0	0	0	0	3	0	0
F	0	0	0	6	↖ **16**	5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
V	0	0	0	0	5	↖ **20**	2	0	0	0	0	1	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	2	↖ **26**	8	2	0	0	0	1	↖ **6**	0	0	0	0	0	1	6
G	0	0	0	0	0	0	8	↖ **32**	12	8	4	0	0	7	↖ **5**	0	0	0	0	0	7
N	0	0	0	0	0	0	2	12	↖ **38**	18	14	10	6	2	8	↖ **5**	0	1	0	6	0
F	0	0	0	6	6	0	0	8	18	↖ **39**	19	15	11	7	3	5	↖ **2**	0	1	0	3
K	0	2	5	0	3	4	0	4	14	19	↖ **44**	24	20	16	12	8	10	↖ **2**	0	1	0
L	0	0	0	5	0	4	0	0	10	15	24	↖ **46**	26	22	18	14	10	8	↖ **6**	0	0
N	0	0	0	0	2	0	4	0	6	11	20	26	↖ **52**	32	28	24	20	16	12	↖ **12**	4
T	0	0	0	0	0	2	0	2	2	7	16	22	32	50	31	27	23	21	15	12	↖ **10**
A	0	0	0	0	0	0	2	0	0	3	12	18	28	32	48	30	26	24	20	13	12
S	0	0	0	0	0	0	0	2	1	0	8	14	24	28	32	48	30	30	22	21	13
I	0	0	0	0	0	3	0	0	0	0	4	10	20	22	25	29	45	28	32	19	17
P	0	0	0	0	0	0	1	0	0	0	0	6	16	18	21	24	28	44	25	30	17
E	0	0	1	0	0	0	0	0	0	0	1	2	12	14	20	22	25	28	41	25	28
N	0	0	0	0	0	0	0	0	6	0	0	0	8	12	15	20	22	26	25	47	27
V	0	0	0	0	0	4	0	0	0	3	0	1	4	6	9	13	18	20	27	27	44
E	0	0	1	0	0	0	2	0	0	0	4	0	1	2	8	10	14	18	17	27	25
V	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	6	8	12	19	19	24
V	0	0	0	0	0	4	1	0	0	0	0	1	2	0	0	0	4	6	13	16	16
I	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	2	8	11	12
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	7	8
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Y	0	0	0	3	3	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	3	4	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0
D	0	0	0	0	0	0	3	0	1	0	0	0	3	0	6	0	0	0	0	5	0
Y	0	0	0	3	3	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	2

Table 3.11: Example for backtracking in the Smith-Waterman algorithm with affine penalties $d = 20, e = 4$ and BLOSUM62 (part 1).

	G	A	K	L	S	A	D	T	E	V	V	C	G	A	P	S	I	Y	L	D	F
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	1	1	0	0	5	0	0	0	0	0	0	0	1	0	0	0	0	0
F	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	3	0	0	6
F	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	4	3	0	6
V	0	0	0	1	0	0	0	0	0	4	5	0	0	0	0	0	3	0	5	0	0
G	6	0	0	0	1	0	0	0	0	0	1	2	6	0	0	0	0	0	0	4	0
G	7	6	0	0	0	1	0	0	0	0	0	0	8	6	0	0	0	0	0	0	1
N	0	5	6	0	1	0	2	0	0	0	0	0	0	6	4	1	0	0	0	1	0
F	3	0	2	6	0	0	0	0	0	0	0	0	0	0	2	2	1	3	0	0	7
K	0	2	5	0	6	0	0	0	1	0	0	0	0	0	0	2	0	0	1	0	0
L	0	0	0	9	0	5	0	0	0	2	1	0	0	0	0	0	4	0	4	0	0
N	4	0	0	0	10	0	6	0	0	0	0	0	0	0	0	1	0	2	0	5	0
T	↖ **10**	4	0	0	1	10	0	11	0	0	0	0	0	0	0	1	0	0	1	0	3
A	12	↖ **14**	3	0	1	5	8	0	10	0	0	0	0	4	0	1	0	0	0	0	0
S	13	13	↖ **14**	1	4	2	5	9	0	8	0	0	0	1	3	4	0	0	0	0	0
I	17	12	10	↖ **16**	0	3	0	4	6	3	11	0	0	0	0	1	8	0	2	0	0
P	17	16	11	7	↖ **15**	0	2	0	3	4	1	8	0	0	7	0	0	5	0	1	0
E	28	16	17	8	7	↖ **14**	2	1	5	1	2	0	6	0	0	7	0	0	2	2	0
N	27	26	19	15	11	7	↖ **15**	2	1	2	0	0	0	4	0	1	4	0	0	3	0
V	44	27	24	20	13	11	4	↖ **15**	0	5	6	0	0	0	2	0	4	3	1	0	2
E	25	43	28	21	20	12	13	3	↖ **20**	0	3	2	0	0	0	2	0	2	0	3	0
V	24	25	41	29	19	20	9	13	1	↖ **24**	4	2	0	0	0	0	5	0	3	0	2
V	16	24	23	42	27	19	17	10	11	5	↖ **28**	8	4	0	0	0	3	4	1	0	0
I	12	15	21	25	40	26	16	16	8	14	8	↖ **27**	7	3	0	0	4	2	6	0	0
C	8	12	13	20	24	40	23	16	12	8	13	17	↖ **24**	7	0	0	0	2	1	3	0
P	5	7	11	14	19	23	39	22	15	11	7	10	15	↖ **23**	14	0	0	0	0	0	0
P	1	4	6	10	13	18	22	38	21	14	10	6	8	14	↖ **30**	13	6	2	0	0	0
A	0	5	3	6	11	17	16	22	37	21	14	10	6	12	13	↖ **31**	12	7	3	0	0
T	0	0	4	2	7	11	16	21	21	37	21	13	9	6	11	14	↖ **30**	10	6	2	0
Y	0	0	0	3	0	5	8	14	19	20	36	19	12	8	4	9	13	↖ **37**	17	13	9
L	0	0	0	4	1	0	3	7	11	20	21	35	15	11	7	3	11	17	↖ **41**	21	17
D	0	0	0	0	4	0	6	2	9	9	17	18	34	14	10	7	2	13	21	↖ **47**	27
Y	2	0	0	0	0	2	0	4	1	8	8	15	15	32	12	8	6	9	17	27	↖ **50**

Table 3.12: Example for backtracking in the Smith-Waterman algorithm with affine penalties $d = 20, e = 4$ and BLOSUM62 (part 2).

classification a new sequence is compared with all sequences in NR. This may take a while with our algorithms.

However, if the perfect alignment need not be ensured then approximations can speed up the alignment. These fast methods are "*seed and extend*" algorithms. They find exact matches of small subsequences of length 2 to 7 with high scores and subsequently extend these matches. Extensions do not require exact matches. Background of these approximations is that in most reasonable alignments there are many 2- to 7-mers (or 2- to 7-grams) of exact matches which highly contribute to the score (cf. main diagonal of the BLOSUM matrices). Another advantage is that the query sequence can be preprocessed for the 2- to 7-mers match search.

3.3.3.1 FASTA

FASTA (fast-aye, Lipman and Pearson, 1985, Pearson and Lipman, 1988) searches for relevant diagonals in the alignment matrix (dot-plot or dynamic programming). See `http://www2.ebi.ac.uk/fasta3/`.

FASTA works as follows (cf. Fig. 3.6):

1. Searching *hot-spots*, i.e. perfect matches of length k, where the default is $k = 2$ for amino acid sequences ($k = 6$ for nucleotides). For scoring the PAM250 is used. The search is sped up by a lookup table of words of length k in the query sequence. The ten best scoring regions are selected based on number of hot-spots and their distance.

2. The ten best scoring regions are re-evaluated by *diagonal runs*. Diagonal u contains all matrix elements (i, j) with $u = i - j$. The diagonal run accumulates the hot spots on the diagonal but also matches shorter that k with a PAM250 scoring matrix and finds maximal scoring subregions. This step is performed also to allow scoring schemes which are different to step 1.

3. The best-scoring sub-alignments are chained to a larger candidate alignment, where gaps (penalty is 20) are allowed. Candidates must be beyond a threshold. In such a way candidates for step 4 are selected.

4. A banded Smith-Waterman algorithm with $k = 23$ generates local alignments around the high-scoring regions. Finally, a full Smith-Waterman alignment can be obtained.

FASTA misses matches when two sequences have similarities but at different positions or if patterns occur repeatedly.

3.3.3.2 BLAST

BLAST (Basic Local Alignment Search Tool, Altschul 1990) and Position-Specific-Iterative-BLAST (PSI-BLAST for data bases) is the most used bioinformatics software these times. For a tutorial see `http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html` and to download it `http://www.ncbi.nlm.nih.gov/BLAST/`. The paper of Altschul 1990 which introduced BLAST is now the most cited paper in Biology.

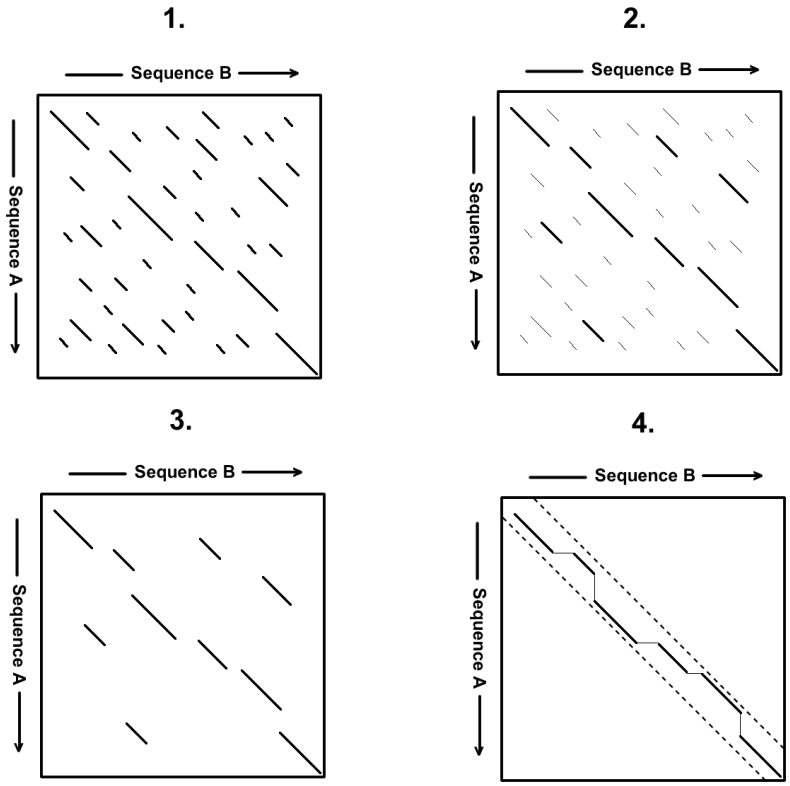


Figure 3.6: The FASTA algorithm. The four steps of the FASTA algorithm.

The idea of BLAST is similar to FASTA. BLAST tries to identify *high-scoring segment pairs* (HSPs). HSPs are local maximal segment pairs (matching subsequences) exceeding a scoring threshold. A segment pair is locally maximal if neither shortening nor extending improves the score.

BLAST works as follows

1. Queue words as k-mers ($k = 3$ for proteins and $k = 11$ for nucleotides) are generated for the query sequence. Queue words score at least T with a non-gapped local alignment with the query sequence (typically 50 words per amino acid). The list can be generated in linear time (length of the list). The parameter T is a trade-off between speed and sensitivity: high T yields shorter lists and is fast but misses weak similarities.

2. The data base (treated as single sequence of length m) is scanned for hits with the queue words. This is a classical problem which can be solved with finite state machines (more precise a Mealy automaton) and a keyword tree (cf. Fig. 3.7) similar to the Aho-Corasick algorithms (see Gusfield, 1999) (cf. Fig. 3.8). The keyword tree can be constructed in about linear time in the list length. The whole search has complexity $O(n + m + u)$, where u are the number of hits.

3. The found hits are gaplessly extended (both directions) to find locally maximal segment pairs (fast without gaps). Non-overlapping hits on the same diagonal with distance below a threshold are joined. Only such extensions are considered which do not drop below a

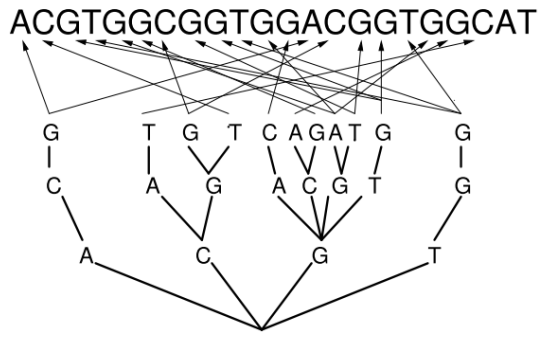


Figure 3.7: The idea of the keyword tree of the BLAST algorithm.

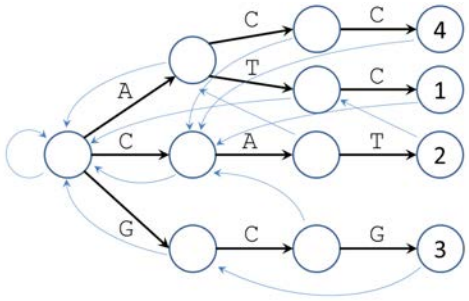


Figure 3.8: Aho Corasick finite state machine for DNA string matching. Blue arrows are failure links, that point to the node where the algorithm jumps to if it hits a mismatch. Using failure links the algorithm does not have to start at the root each time.

certain distance to the best score yet. As in FASTA gapped alignments are constructed with a KBand algorithm.

BLAST can miss HSPs due to the value of k and the thresholds introduced in step 2 and 3.

The BLAST programs are

- BLASTP: compares amino acid sequence query to a protein sequence data base
- BLASTN: compares nucleotide sequence query to a nucleotide sequence data base
- BLASTX: nucleotide sequence query is translated and compared to a protein sequence data base
- TBLASTN: compares amino acid sequence query to a nucleotide sequence data base, where the later is translated
- TBLASTX: compares nucleotide sequence query to a nucleotide sequence data base but the sequences are first translated

It is recommended to use translated amino acid sequences if possible. An e-value of 0.05 is the threshold for significance and indicates interesting results. Repeated segments may confuse BLAST and should be removed.

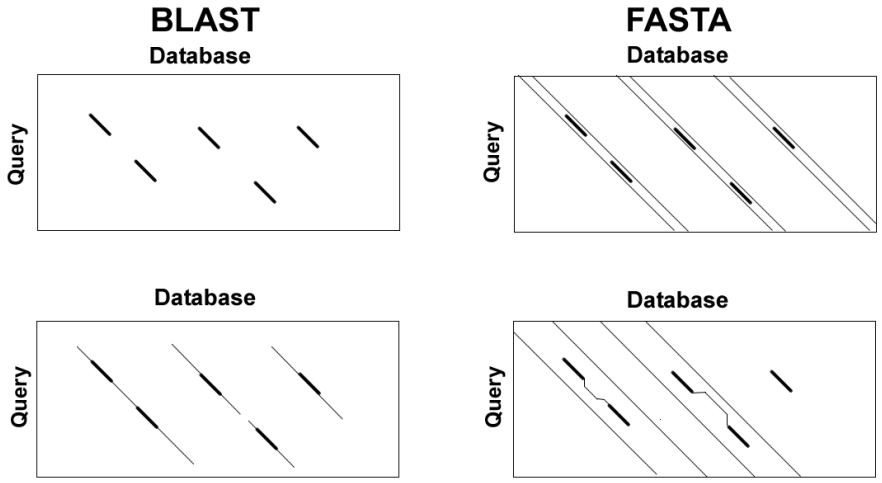


Figure 3.9: Difference between BLAST and FASTA. BLAST extends local regions only on the diagonal whereas FASTA uses banded alignments.

PSI-BLAST is an iterative BLAST search where a profile ($\rightarrow$) or position specific scoring matrix (PSSM) is constructed from a multiple alignment ($\rightarrow$) of the highest scoring hits. The PSSM is a position-specific score for each position in the multiple alignment. Highly conserved positions, i.e. positions with the same amino acid or very similar amino acid in every sequence which was aligned, receive high scores. The scoring profile is subsequently used to perform another BLAST search where in each iteration the profile is refined and sensitivity increased.

The PSI-BLAST profile generation is important for secondary structure prediction and for protein classification. In both cases instead of the original sequence the PSSM is used which generalizes from the individual characteristics of the single sequence. The change from the original sequence to the PSSM has given a performance jump in both disciplines.

PSI-BLAST may generate dirty profiles by including hits which are no longer similar to the original sequence. Especially non-informative patterns should be removed from the query like coiled-coil regions and low complex patterns (e.g. repeating amino acid) to avoid random hits.

Comparison of BLAST and FASTA:

	BLAST	FASTA
output	multiple HSPs	best alignment
better for	proteins	nucleotides
speed	faster	slower
sensitivity	low	higher
homologs	finds low	misses low
false positives	more	less
significance	e-values	estimation

3.3.3.3 BLAT

Faster algorithms for finding homologous sequences have been developed.

BLAT (BLAST-Like Alignment Tool) is about 50 times faster than BLAST at comparable sensitivity (500 times faster for nucleotides). It also searches first for k-mer hits and extends them to HSPs as BLAST does. However the speed up compared to BLAST is due to some differences:

- BLAST generates a list of words from the query sequence and goes linearly through the data base. In contrast to BLAST, BLAT builds an index of the data base and goes linearly through the query
- BLAT uses – in contrast to BLAST – near perfect matches, i.e. perfect matches with one or more amino acids mismatch
- BLAT joins more than two hits by extending the hits
- BLAT chains the high scoring local alignments together

There exist further methods like QUASAR (Q-gram Alignment based Suffix Arrays, Burkhardt et al., Recomb, 1999). Here the data base is an index in a suffix array.

3.4 Alignment Significance

In this section we focus on the result of an alignment. What does a certain score mean? How can it be judged? Is it significant or random? Most alignment tools provide a significance measure. To understand and judge these measures we will derive and discuss them in the following.

3.4.1 Significance of HSPs

BLAST searches for HSPs which will be considered here. What is the distributions of the alignment score of randomly generated sequences? It is not Gaussian. The alignment algorithms produce high scores but avoid low scores which results in a distribution with positively skewed tail. The extreme value distribution (see Fig. 3.10) is an appropriate distribution to model the outcome of optimizing some values:

$$\text{pdf: } p(x) = e^{-x}\, e^{-e^{-x}}, \quad \text{dist.: } P(x) = e^{-e^{-x}}. \tag{3.38}$$

Following assumptions are given:

- the two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ are i.i.d. in their elements according to the letter probabilities p_x and p_y
- $\boldsymbol{x}$ and $\boldsymbol{y}$ are long
- the expected pairwise score for random sequences $\sum_{i,j}\ p_x(i)\ p_y(j) s(i,j)$ is negative
- there exist i, j for which $s(i,j) > 0$ (existence of a positive score)

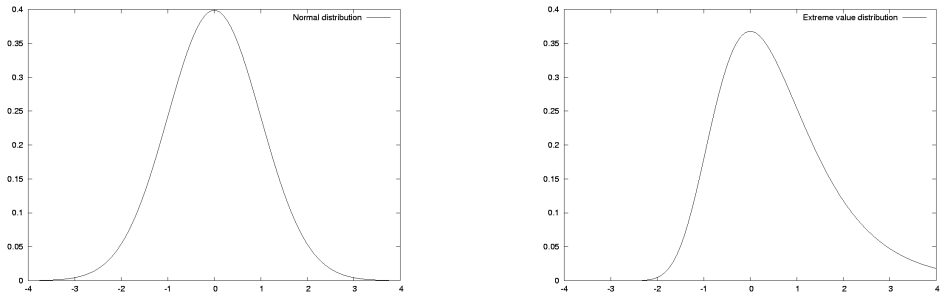


Figure 3.10: The density functions of the normal distribution (left) and the extreme value distribution (right).

Karlin, Dembo, Kawabata, 1990, showed that the maximal segment scores $S_{n,m}$ (n, m are the length' of the sequences) are

$$S_{n,m} \;\propto\; \frac{\ln nm}{\lambda} \;. \tag{3.39}$$

Karlin and Altschul, 1990, and Altschul and Gish, 1996, showed that the centered values $\tilde{S}_{n,m} = S_{n,m} - \frac{\ln nm}{\lambda}$, are distributed according to an extreme value distribution:

$$P\left(\tilde{S}_{n,m} > S\right) \;\approx\; 1 - \exp\left(-K\; m\; n\; e^{-\lambda\, S}\right) \;\approx\; K\; m\; n\; e^{-\lambda\, S} \;. \tag{3.40}$$

The last approximation stems from the fact that $e^h \;=\; 1 + h + O(h^2)$ and for large S, i.e. high scores, $h \;=\; -K\; m\; n\; e^{-\lambda\, S}$ is small to allow for a linear approximation. The scoring matrix can be expressed as log-odds score with $p_{i,j}$ as the target distributions of letter pairs:

$$s(i,j) \;=\; \log\left(\frac{p_{i,j}}{p_x(i)\; p_y(j)}\right) \;/\; \lambda \;. \tag{3.41}$$

The expected contribution of a pair to the score is

$$\sum_{i,j} p_{i,j} s(i,j) \;=\; \sum_{i,j} p_{i,j} \log\left(\frac{p_{i,j}}{p_x(i)\; p_y(j)}\right) / \lambda \;= \tag{3.42}$$

$$\mathrm{KL}\left(p_{i,j} \parallel p_x(i)\; p_y(j)\right) / \lambda \;, \tag{3.43}$$

where "KL" denotes the Kullback-Leibler distance or the relative entropy of $p_{i,j}$ with respect to $p_x(i)\; p_y(j)$. The Kullback-Leibler gives the number of additional bits which are needed to describe a pair (i,j) produced by $p_{i,j}$ if $p_x(i)\; p_y(j)$ is given. Therefore λ can be seen as scaling to bit scores.

To determine the parameter λ we assume two random sequences and delete one pair, then the following should hold

$$E \;=\; P\left(\tilde{S}_{n,m} > S\right) \;\approx \tag{3.44}$$

$$\sum_{i,j} p_x(i)\; p_y(j)\; P\left(\tilde{S}_{n-1,m-1} > S - s(i,j)\right) \;.$$

$P\left(\tilde{S}_{n,m} > S\right)$ is the probability of score larger S. This is split into the probability of having a score without the pair (i, j) larger than $S - s(i, j)$ multiplied by observing pair (i, j) summed over all possible pairs (i, j). Note, that $s(i, j)$ may be deleted from a position outside the HSP and that some amino acids are more probable to be part of an HSP. Both facts are neglected.

If we approximate $\tilde{S}_{n-1,m-1}$ by $\tilde{S}_{n,m}$ then the parameter λ is obtained as positive solution of the equation

$$\sum_{i,j} p_x(i)\, p_y(j)\ \exp(\lambda\, s(i,j))\ =\ 1\,. \tag{3.45}$$

Typical values for the BLOSUM62 matrix are $\lambda = 0.254$ and $K = 0.040$. K can be interpreted as how related amino acids are in the given context (the scoring matrix). λ can be interpreted as the scale of the scoring matrix because a change of the base of the logarithm leads to a re-scaling of the scoring matrix.

The value $P\left(\tilde{S}_{n,m} > S\right)$ is called the e-value and is used as one output of the BLAST algorithm. BLAST uses another output called "bit-score" which is a normalized score independent of λ and K.

We see that the log-probability is

$$\ln(E)\ =\ \ln\left(K\ m\ n\ e^{-\lambda\ S}\right)\ =\ \ln(m\ n)\ +\ \ln(K)\ -\ \lambda\, S\,. \tag{3.46}$$

We set the bit-score $\bar{S}$ to

$$\bar{S}\ :=\ \frac{\lambda S\ -\ \ln(K)}{\ln 2} \tag{3.47}$$

therefore

$$S\ =\ \frac{\bar{S}\ \ln 2\ +\ \ln(K)}{\lambda}. \tag{3.48}$$

This leads to

$$\ln(E)\ =\ \ln(m\ n)\ +\ \ln(K)\ -\ \lambda\ \left(\frac{\bar{S}\ \ln 2\ +\ \ln(K)}{\lambda}\right) \tag{3.49}$$

$$\log_2(E)\ =\ \log_2(m\ n)\ -\ \bar{S} \tag{3.50}$$

$$E\ =\ m\ n\ 2^{-\bar{S}} \tag{3.51}$$

Therefore the e-value p of the bit-score $\bar{S}$ is a better score which is independent of K and λ.

Finally, we compute the probability that l maximal segments scores or HSPs exceed a threshold S. The HSPs follow a Poisson distribution and E is the probability of observing an HSP $> S$ (the average number of events). Therefore we have

$$P(l)\ =\ e^{-E}\ \frac{E^l}{l!}\,. \tag{3.52}$$

3.4.2 Significance of Perfect Matches

In this subsection we focus on the significance of perfect matches or near perfect matches. Therefore methods like BLAT can be evaluated in this framework but also results from BLAST or global alignment if perfect matches are considered.

The expected number of non-overlapping k-mers of the query of length n which match a data base of length m with each of the a letters having the same probability is

$$(n\ -\ k\ +\ 1)\ \frac{m}{k}\ \left(\frac{1}{a}\right)^k\ . \tag{3.53}$$

Here $(n\ -\ k\ +\ 1)$ are the number of k-mers in the query, $\frac{m}{k}$ are the number of non-overlapping k-mers in the data base and $\left(\frac{1}{a}\right)^k$ is the matching probability (context-free).

If we know M the similarity between the query and the data base, then the matching probability is

$$(M)^k\ . \tag{3.54}$$

The probability of at least one matching k-mer in the data base to a certain k-mer in the query is

$$1\ -\ \left(1\ -\ M^k\right)^{m/k}\ . \tag{3.55}$$

This is one minus the probability of that all m/k do not match simultaneously. For small M we can approximate it by

$$1\ -\ \left(1\ -\ M^k\right)^{m/k}\ \approx\ M^k\ . \tag{3.56}$$

Now we consider almost perfect matches. The probability of a match with 1 or less mismatches is simply the sum of probability of a perfect match and the probability of the mismatch:

$$k\ M^{k-1}(1-M)\ +\ M^k\ . \tag{3.57}$$

The first term is the number k of existing $(k\ -\ 1)$-mers multiplied by their probability. Their probability is the probability M^{k-1} of $(k-1)$ matches multiplied by the probability of a mismatch $(1\ -\ M)$. Other probabilities like equal letter probability follow analog, e.g. by replacing M through $\frac{1}{a}$.

For more than one exact match we introduce

$$p_m\ =\ M^k\ . \tag{3.58}$$

For non-overlapping k-mers more than one match can be described by a binomial distribution: match or non-match. The probability to obtain l matches is

$$\binom{m/k}{l}\ p_m^l\ (1\ -\ p_m)^{m/k-l}\ . \tag{3.59}$$

For more than l matches and large m the binomial distribution $\mathcal{B}(m/k, p_m)$ can be either approximated through a Poisson distribution or through a normal distribution depending on $m/k\ M^k$:

$$\mathcal{B}(m/k, p_m) \sim \begin{cases} \mathcal{P}(np) & \text{for} \quad m/k\ M^k \leq 5 \\ \mathcal{N}(np, \sqrt{np_m(1-p_m)}) & \text{for} \quad m/k\ M^k > 5 \end{cases}, \tag{3.60}$$

where $\mathcal{P}$ denotes the Poisson distribution and $\mathcal{N}$ the normal distribution. Note, that approximation of $\mathcal{B}$ through the normal distribution approximation results from the central limit theorem. The approximation through the Poisson distribution is due to its probability-generating function $G(s) = (p\,s + (1-p))^t$

$$(1 + l/t)^t \approx e^l \tag{3.61}$$

$$(p\,s + (1-p))^t = \left(1 + \frac{p\,n}{t}\,(s - 1)\right)^t \approx \exp(n\,p\,(s-1))\,. \tag{3.62}$$

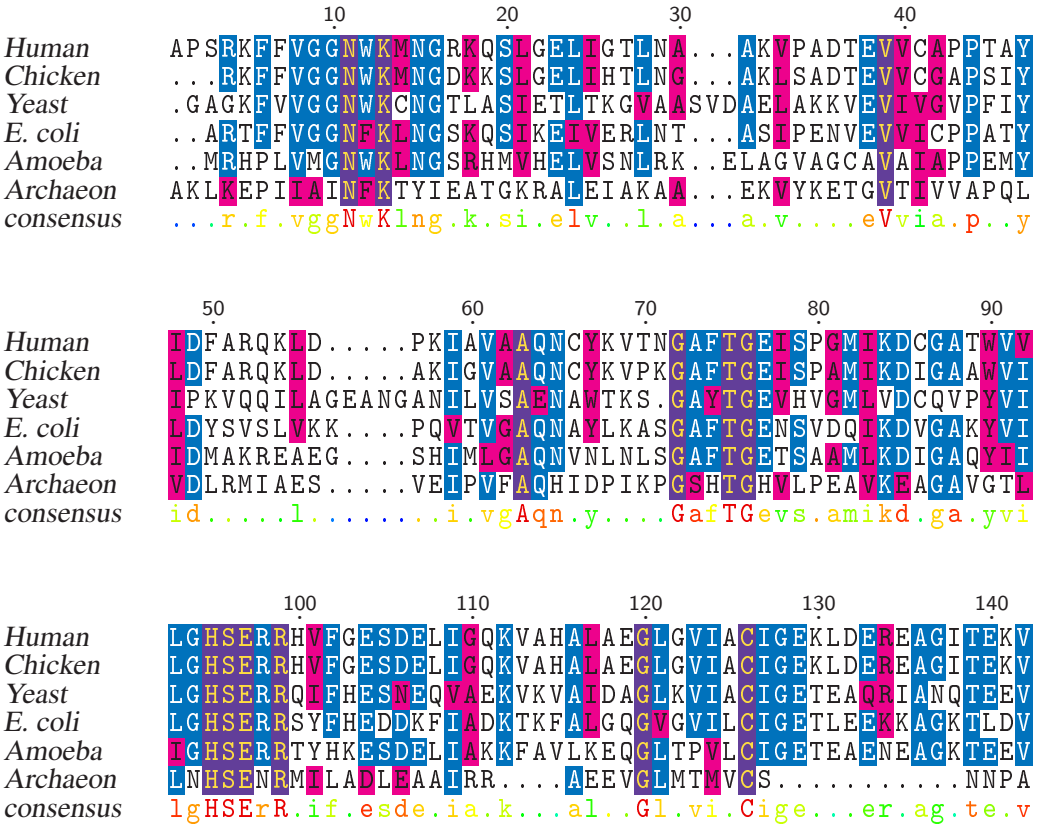
Chapter 4

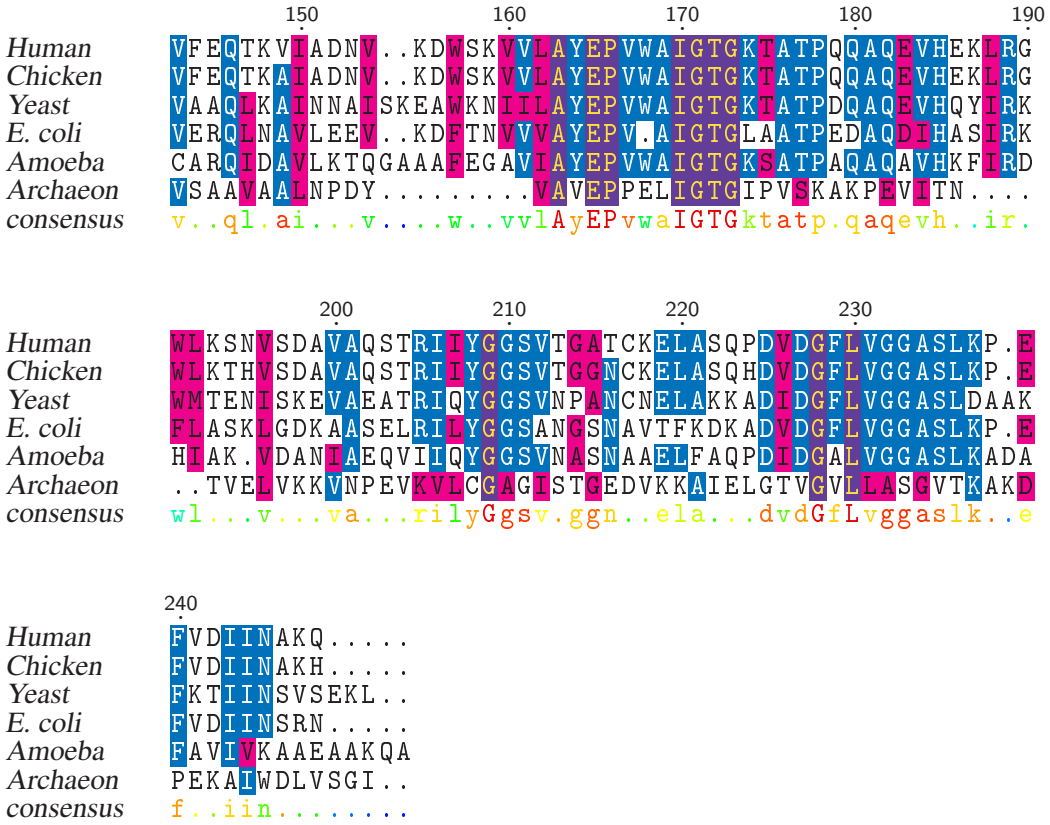
Multiple Alignment

4.1 Motivation

An extension of pairwise sequence alignment is the multiple sequence alignment. Now more than two sequences must be compared to one another by writing each sequence in a separate line. The sequences should be arranged so that in the columns the amino acids match as good as possible. Below is an example of a multiple sequence alignment of triosephosphate isomerase from different species.

triosephosphate isomerase





Multiple sequence alignment is used to

- detect remote homologous regions which are detected through the average of all sequences but cannot be detected through pairwise alignment (A is similar to B and B to C even if B and A are not similar to one another),
- detect motifs (→), i.e. regular patterns, in families of proteins,
- detect conserved regions or positions, e.g. to extract disulfide bonds,
- detect structural blocks like helices or sheets,
- construct phylogenetic trees (→),
- construct a profile for protein families to which a new sequence can be compared,
- construct profiles instead of the sequences to obtain a more reliable input to sequences processing methods,
- sequence genomes by superimposing fragments (nucleotides),
- cluster proteins according to similar regions.

Multiple sequence alignment can be based on the ideas of pairwise sequence alignment but some issues must be reconsidered. As in pairwise alignment we separate scoring schemes and optimization methods.

4.2 Multiple Sequence Similarities and Scoring

For the similarity measure there are different approaches: (1) either measure similarity to a reference sequence, (2) measure similarities between evolutionary adjacent sequences, or (3) measure all pairwise similarities. The next subsections will discuss these different approaches.

4.2.1 Consensus and Entropy Score

The *consensus sequence* is the sequence obtained if for each column in the alignment the most frequent amino acid is chosen (cf. the last line of above multiple alignment example). If the highest frequency does not exceed a threshold (is not significantly higher than other frequencies) then either a wildcard, a gap, or a set of amino acids (set is expressed by one letter) will be inserted. More precisely the consensus sequence contains at a certain column the amino acid or letter representing a group of amino acids which has the highest pairwise score to all other amino acids and gaps in the column.

The *consensus score* is the sum of the pairwise scores between sequences and the consensus sequence. The costs between amino acids were already considered. The cost between amino acids and gaps must be defined.

The consensus score can be generalized by constructing a profile instead of the consensus sequence. A *profile* gives the relative frequency $f_{i,a}$ of each letter a in each column i of the alignment. Later we will see another approach to generate profiles, which was introduced by Karlin and Altschul, 1990.

High entropy of the letter distribution in one column means that all letter are equally probable. Zero entropy means that only one letter occurs in one column. Therefore a good alignment correlates with a low accumulative entropy per position, and the *entropy score* is

$$- \sum_i \sum_a f_{i,a} \log f_{i,a} \; . \tag{4.1}$$

4.2.2 Tree and Star Score

In order to count the mutations (i.e. the similarity), instead of comparing all pairs of letters only those pairs should be compared which are evolutionary adjacent, i.e. are ancestors/successors of each other. For example if only two letters appear at one column but multiple, say `D` and `E`, then it may be only one mutation which changed `D` into `E` or vice versa. All sequences older than the time of the mutation agree to each other as do all sequences after the mutation. Pairwise comparison of letters will overestimate the number of mutations.

To compare evolutionary adjacent sequences we must assume to know a phylogenetic tree ($\rightarrow$) which gives the evolutionary relationship. The tree may be constructed through multiple alignment as preprocessing.

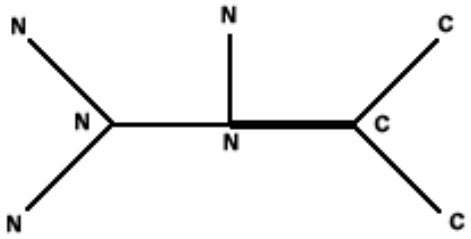


Figure 4.1: Pairwise comparison of letters according to a tree. The edges are the comparisons which are made. Bold edges are mismatches.

If we compare the sequences

```
NNN
NNN
NNN
NNC
NCC
```

then the edges in Fig. 4.1 give the comparisons to make, where bold edges are mismatches.

Instead of a phylogenetic tree also a phylogenetic star may be used where one sequence is considered as ancestor of all others. This scoring scheme is similar to the consensus score as the sequence most similar to the consensus score is more likely the ancestor. Reason for this is that sequences resulting from the ancestor by mutations are independent of each other. Mutations are conservative and, therefore, at every position only few letters differ from the ancestor. Fig. 4.2 give the comparisons to make, where bold edges are mismatches.

4.2.3 Weighted Sum of Pairs Score

The most common score is the weighted sum of pairs, where all pairs of letters per alignment position are mutually compared.

As above also the weighted sum of pairs can be expressed through a graph where each edge means a contribution to the score by comparing the letters at the nodes of the edge. Fig. 4.3 gives all pairwise comparisons to make.

For alignment length L and N sequences, the weighted sum of pairs score is

$$\sum_{i=1}^{L} \sum_{l=1}^{N-1} \sum_{j=l+1}^{N} w_{l,j}\; s\left(x_{i,l}, x_{i,j}\right) \;. \tag{4.2}$$

The weights may be chosen to reduce the influence of closely related sequences to avoid that the final score is governed by close homologous sequences.

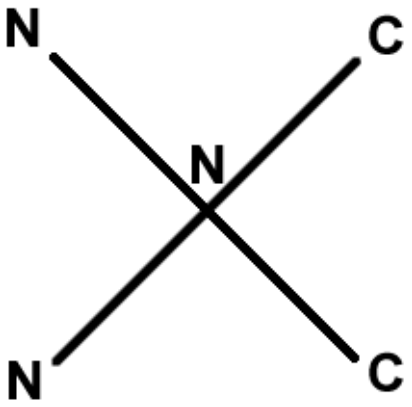


Figure 4.2: Pairwise comparison of letters according to a star. The edges are the comparisons which are made.

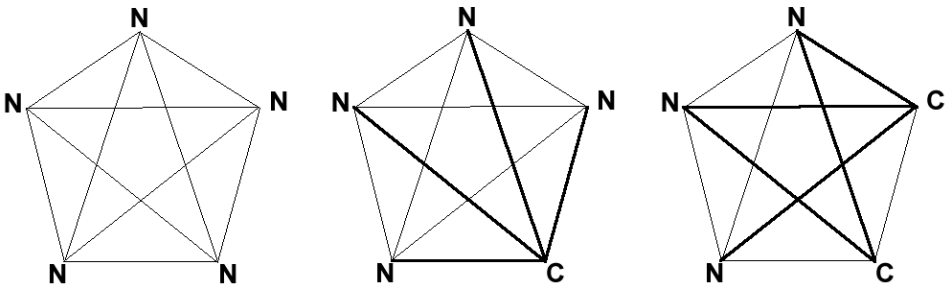


Figure 4.3: Pairwise comparison of letters. The edges are the comparisons which are made. Bold edges are mismatches

The weighted sum of pairs score has disadvantages. Assume that in a column all letters are equal `C` and now we flip one letter to `D`. What is the new relative score compared to the old score? It decreases with respect to the number of sequences N. The old score is

$$S_{\text{old}} \;=\; \frac{N\,(N-1)}{2} s(C,C) \tag{4.3}$$

and the new is

$$S_{\text{new}} \;=\; \frac{N\,(N-1)}{2} s(C,C) \;-\; (N-1)s(C,C) \;+\; (N-1)s(C,D) \tag{4.4}$$

which gives as relative decrease

$$\frac{S_{\text{old}} - S_{\text{new}}}{S_{\text{old}}} \;=\; \frac{2\,(N-1)\,s(C,C) \;-\; 2\,(N-1)\,s(C,D)}{N\,(N-1)\,s(C,C)} \;= \tag{4.5}$$
$$\frac{2}{N}\left(1 \;-\; \frac{s(C,D)}{s(C,C)}\right) .$$

Note, that for all reasonable scoring matrices $s(C,D) < s(C,C)$ from which follows that

$$\left(1 \;-\; \frac{s(C,D)}{s(C,C)}\right) > 0.$$

Therefore, the relative decrease is inverse proportional to the number of sequences. However that is contra-intuitive because a new letter in a column of 100 equal letters is more surprising than a new letter in a column of 3 equal letters. Information theory say the same: a new letter in a column of 100 has higher information gain because of the low entropy whereas the information gain at high entropy is low (the description length of the new letter a is $-\log f_{i,a} \;=\; \log(N)$).

Gaps can be handled as for pairwise algorithms where linear gaps are to prefer as the alignment methods are computationally much more expensive for affine gaps. For the pairwise score the gaps which fall onto gaps are removed.

4.3 Multiple Alignment Algorithms

For many cases the multiple alignment optimization problem is NP-hard. Therefore, the solutions are often approximated if the number of sequences is more than 10 to 15.

The algorithms can be divided into four classes

- exact methods: MSA
- progressive methods: COSA, GSA, clustalW, TCoffee
- iterative and search algorithms: DIALIGN, MultAlin, SAGA, PRRP, Realigner
- local methods (motif/profile): eMotif, Blocks, Dialign, Prosite, HMM, Gibbs sampling
- divide-and-conquer algorithms: DCA, OMA

Tab. 4.1 gives an overview over the multiple alignment programs, their web link, and their associated publication.

Exact alignment methods		
MSA	http://www.ncbi.nlm.nih.gov/CBBresearch/Schaffer/msa.html	Lipman et al. (1989) Gupta et al. (1995)
Progressive alignment methods		
CLUSTALW	ftp://ftp.ebi.ac.uk/pub/software	Thompson et al. (1994/97) Higgins et al. (1996)
PRALINE	http://www.ibi.vu.nl/programs/pralinewww/	Heringa (1999)
Iterative and search algorithms		
DIALIGN segment alignment	http://bibiserv.techfak.uni-bielefeld.de/dialign/	Morgenstern et al. (1996)
MultAlin	http://multalin.toulouse.inra.fr/multalin/	Corpet (1988)
PRRP progressive global alignment	http://www.genome.jp/tools/prrn/	Gotoh (1996)
SAGA genetic algorithm	http://www.tcoffee.org/Projects/saga/	Notredame and Higgins (1996)
Local alignments / motif / profile		
Aligned Segment Statistical Eval. Tool (Asset)	http://iubio.bio.indiana.edu/soft/iubionew/molbio/align/search/Asset/	Neuwald and Green (1994)
BLOCKS	http://blocks.fhcrc.org/blocks/	Henikoff and Henikoff (1991, 1992)
eMOTIF	http://motif.stanford.edu/projects.html	Nevill-Manning et al. (1998)
GIBBS (Gibbs sampler)	http://ccmbweb.ccv.brown.edu/gibbs/gibbs.html	Lawrence et al. (1993), Liu et al. (1995), Neuwald et al. (1995)
HMMER hidden Markov model	http://hmmer.janelia.org/	Eddy (1998)
MACAW	http://iubio.bio.indiana.edu/soft/molbio/ncbi/old/macaw/	Schuler et al. (1991)
MEME (EM method)	http://meme.sdsc.edu/meme/website/	Bailey and Elkan (1995), Grundy et al. (1996, 1997), Bailey and Gribskov (1998)
Profile (UCSD)	http://www.sdsc.edu/projects/profile/	Gribskov and Veretnik (1996)
SAM hidden Markov model	http://compbio.soe.ucsc.edu/sam.html	Krogh et al. (1994), Hughey and Krogh (1996)

Table 4.1: Overview over multiple alignment programs.

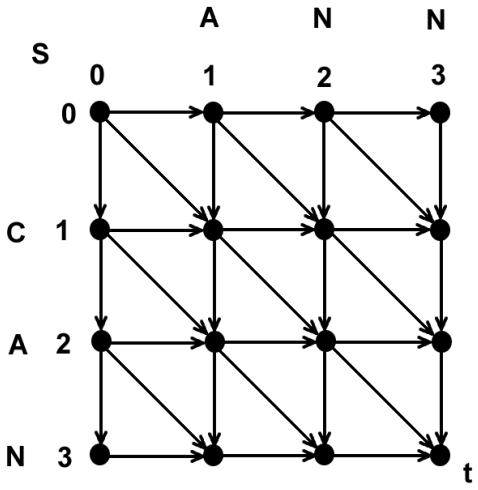


Figure 4.4: Matrix used for pairwise alignment. Each path through the matrix is an alignment of two sequences.

4.3.1 Exact Methods

The MSA algorithm (Lippman et al., 1989, Gupa et al., 1995) generalizes the dynamic programming ideas from pairwise alignment. For three sequences a cube instead of the matrix used for the dynamic programming algorithms for aligning two sequences must be used. At each coordinate direction a sequence is written. Grid points (nodes) contain as entries the best alignment score of the according subsequences. The paths through a matrix (cf. Fig. 4.4) are now paths from the lower, left, front to the upper, right, back through a cube (cf. Fig. 4.5). The alignment of the examples in Fig. 4.5 is:

```
A-BD-E-
ACB--E-
A--DCEE
```

At each node of the cube 7 steps in order to go to other nodes are possible in order to move closer to the upper, right, back node.

```
x x - - x x -
y - y - y - y
z - - z - z z
```

The three dimensions can be generalized to more dimension, i.e. to more sequences. However the memory and computational complexity grow exponentially with the number of sequences. Therefore, these methods are limited to few sequences and to short sequences. Gupa et al., 1995, proposed a more efficient variant of MSA. First pairwise alignments are computed in order to constrain the position of the optimal multiple alignment path. The optimal path can be projected onto 2 dimensional planes on which it represents a pairwise alignment. The constraints reduce the number of entries which must be evaluated on the hypercube.

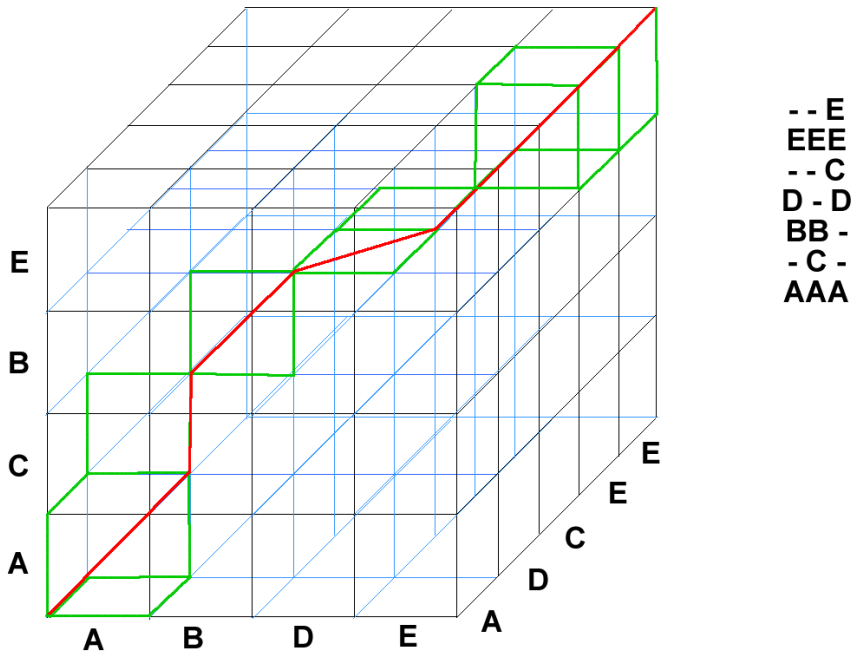


Figure 4.5: Cube for aligning three sequences. Each path from the lower, left, front to the upper, right, back through the cube is an alignment of three sequences.

MSA (Gupa) works as follows:

- compute all pairwise alignment scores $S_{k,l}$
- predict a phylogenetic tree based on the pairwise scores
- compute pairwise weights based on the tree
- construct a temporary multiple alignment based on the phylogenetic tree with score S_t
- compute $B_{k,l}$, a lower bound on $S[k,l]$, the score of the projection of the optimal multiple alignment to the sequences k and l
- constrain the space in the cube where the optimal alignment path goes through $B_{k,l}$; the constraints can be computed similar to the Baum-Welch algorithm used for hidden Markov models
- compute the optimal alignment on the constraint cube; here Dijkstra's shortest path algorithm for DAGs (directed acyclic graphs) with nonnegative edges can be used; using a priority queue avoids to construct the whole graph; non-negativity guarantees monotonically increasing costs and allows to use the priority queues
- compare the weight found in the alignment with the maximal weight

The last step checks whether the actual weight exceeds the maximal weight. If this is the case then a better alignment may be possible. However, larger maximal weight means more computational costs.

The Carillo-Lipman bound is given by

$$B_{k,l} \;=\; S_t \;+\; S_{k,l} \;-\; \sum_{i,j} S_{i,j} \;. \tag{4.6}$$

The inequality

$$S[k,l] \;\geq\; B_{k,l} \tag{4.7}$$

follows from $S[k,l] \;\leq\; S_{k,l}$ (the optimal pairwise alignment scores is better or equal than all other alignments) and $S_t \;\leq\; S$ (S is the optimal multiple alignment score):

$$\begin{aligned}
& S \geq S_t \\
\Leftrightarrow\;& \sum_{i,j} S[i,j] \geq S_t \\
\Rightarrow\;& \sum_{(i,j)\neq(k,l)} S_{i,j} \;+\; S[k,l] \geq S_t \\
\Leftrightarrow\;& S[k,l] \geq\; S_t \;-\; \sum_{(i,j)\neq(k,l)} S_{i,j} \\
\Leftrightarrow\;& S[k,l] \;\geq\; S_t \;+\; S_{k,l} \;-\; \sum_{i,j} S_{i,j} \\
\Leftrightarrow\;& S[k,l] \;\geq\; B_{k,l} \;.
\end{aligned} \tag{4.8}$$

The MSA can be improved according to Lermen and Reinert, 1997, through the $\mathcal{A}^*$ algorithm (see Alg. B.1). The $\mathcal{A}^*$ algorithm uses the bounds derived above as approximative distance to the goal.

The MSA algorithm in its original form uses the weighted sum of pairs as score and a linear gap penalty. The weighting is computed as the difference of the pairwise alignment and the score of the projected multiple alignment. Therefore the weight measures the difference between the multiple alignment reduced to the special pair and the pairwise alignment. Larger difference means higher weight. As a consequence, similar sequences pull the multiple alignment towards them and, therefore, are down-weighted. In conclusion, the effect of similar sequences heavily influencing the alignment is reduced. However the weights through the phylogenetic tree are vice versa where weights between distant sequences are removed, i.e. set to zero.

Summing up all the weights results in another value which measures the overall divergence of the sequences.

4.3.2 Progressive Algorithms

Progressive methods are the most popular methods for multiple alignment. Especially ClustalW (Thomson,Higgins,Gibson, 1994) and TCoffee (Notredame, Higgins, Heringa, 2000) are most prominent.

ClustalW and TCoffee work as follows:

- perform pairwise alignment for each pair
- a weight matrix of pairwise weights is computed; the entry in this weight matrix is one minus the ratio of perfect matches in the pairwise alignment.
- construct a phylogenetic tree from the alignments with evolutionary distances (Neighbor-Joining method ($\rightarrow$))
- start with the closest distance and compute alignments between pairs of sequences, pairs of sequence and alignment, and pairs of alignments. In this way alignments are propagated through the tree.

Initial alignments for the phylogenetic tree may be found through local alignment methods. The phylogenetic tree supplies also the weighting factors as in MSA.

The progressive methods are prone to getting stuck in local minima as many optimization methods do. For closely related sequences other scoring matrices should be used than for remotely related sequences, however the same scoring matrix is used for all sequences. For remotely related sequences the gap penalty parameters are an important hyper-parameter which may be sensitive.

4.3.2.1 ClustalW

Above mentioned problems of progressive methods are partly addressed by clustalW:

- **gap penalties are adjusted context dependent**: for protein sequences gaps in hydrophobic regions (corresponding to the interior of the protein) are more penalized than gaps in hydrophilic regions (loops or random coil); gaps which are close to other gaps (within eight amino acids) but do not match them are more penalized in order to construct gaps at the same position; gaps in regions where earlier gaps are found obtain a lower gap opening penalty; gap penalties are amino acid dependent (e.g. the smallest amino acid `G` indicates loops where gaps are)
- **scoring matrices are adapted**: a special scoring matrix from the PAM or the BLOSUM families can be used according to the expected evolutionary distance
- **sequences are weighted through a phylogenetic tree**: sequences which are similar to one another obtain lower weights; this accounts for unbalanced data sets (e.g. very similar sets of sequences); the initial phylogenetic tree supplies the weights according to the edge length/weightening; the score for a column is $\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w_i \; w_j \; s(i,j)$, where w_i denotes the weight of sequence i;
- **adaptive phylogenetic tree**: for insufficient scores the tree can be changed

The initial gap penalty parameters are chosen according to the scoring matrix, the similarity of the sequences (% identity), length of the sequences (logarithm of the length of the shorter sequences is added to the base gap opening penalty), difference of the length of the sequences (gap extension penalty is multiplied by $(1 + |\log(n/m)|)$ to avoid gaps in the shorter sequence).

ClustalW comes with an option where the alignment can be computed fast but only approximative.

4.3.2.2 TCoffee

TCoffee (Tree based Consistency Objective Function For alignmEnt Evaluation) leads on average to better alignments than clustalW.

TCoffee work as follows:

- compute libraries of pairwise alignments based on both global (clustalW) and local (FASTA) alignments; the combination of local and global alignments gives more reliable relationships between sequences
- library weights are computed according to % identity
- libraries are combined and extended: if a pair of sequences is duplicated between two libraries, it is merged and gets a weight equal to the sum of the two weights; the library is extended by aligning two sequences through a third sequence where matching letters (both are aligned to the same letter of the third sequence) of the original pair are up-weighted
- perform progressive alignment with a distance matrix based on the extended library

4.3.3 Other Multiple Alignment Algorithms

4.3.3.1 Center Star Alignment

The *Center Star Alignment* selects first the center sequence $\bar{i}$ which is the sequence with minimal pairwise costs if all pairwise alignment costs $C(i,j)$ to other sequences are summed up:

$$\bar{i} \;=\; \arg\min_i \sum_j C(i,j)\;. \tag{4.9}$$

In the following the center sequence is called sequence 1.

A new sequence is added to the set of already aligned sequences by a pairwise alignment to the first multiple aligned sequence (center sequence). The addition may cause new gaps in the multiple alignment.

Gusfield, 1993, showed that this procedure produces an alignment with a cost which is less then twice the optimal cost, if the symmetric alignment cost fulfills

$$C(i,i) = 0 \quad \text{and} \quad C(i,j) \;\leq\; C(i,k) \;+\; C(k,j)\;. \tag{4.10}$$

Problem is to ensure the assumptions on the cost.

Assume we have a scoring matrix s with

$$s(-,-) \;=\; 0 \tag{4.11}$$
$$s(-,i) \;<\; 0 \tag{4.12}$$
$$s(k,k) \;\geq\; s(i,k) \;+\; s(k,j) \;-\; s(i,j)\;. \tag{4.13}$$

Note that the last inequality holds for gaps instead of i, j, and k. The last inequality is:

```
AB   AB
|| > ||
AC   CA
```

With these assumptions

$$C(i,j) \;=\; S_{i,i} \;-\; 2\,S_{i,j} \;+\; S_{j,j}, \tag{4.14}$$

where $S_{i,j}$ is the alignment score between sequences i and j, is a distance, which fulfills above conditions. The first condition is trivial to show. The second condition equivalent to

$$\begin{aligned} & S_{i,i} \;-\; 2\,S_{i,j} \;+\; S_{j,j} \;\leq\; S_{i,i} \;-\; 2\,S_{i,k} \;+\; S_{k,k} \;+\; S_{k,k} \;-\; 2\,S_{k,j} \;+\; S_{j,j} \\ & \Leftrightarrow \;\; S_{i,j} \;\geq\; S_{i,k} \;+\; S_{k,j} \;-\; S_{k,k}\,. \end{aligned} \tag{4.15}$$

To show this inequality we construct an alignment of i and j as follows. First align i to k and then j to k. Now align i,j, and k based on the pairwise alignments, where the alignment obtains a gap if the gap was present in the first or second alignment. If S is the score of the multiple alignment, then per construction $S[i,k] = S_{i,k}$ because the projection of the multiple alignment onto (i,k), $S[i,k]$, leads to pairs observed already in the pairwise alignment or to gap-pairs which contribute with zero to the score. Analogously $S[k,j] = S_{k,j}$ and $S[k,k] = S_{k,k}$ can be derived.

Because in the multiple alignment componentwise

$$s(i,j) \;\geq\; s(i,k) \;+\; s(k,j) \;-\; s(k,k) \tag{4.16}$$

holds, we obtain

$$\begin{aligned} & S[i,j] \;\geq\; S[i,k] \;+\; S[k,j] \;-\; S[k,k] \;\text{ and} \\ & S[i,j] \;\geq\; S_{i,k} \;+\; S_{k,j} \;-\; S_{k,k}\,. \end{aligned} \tag{4.17}$$

The inequality to show follows from

$$S_{i,j} \;\geq\; S[i,j]\,. \tag{4.18}$$

Now we want to show the idea of the proof of Gusfield that the center star alignment has only twice of the cost of the optimal alignment cost.

The center sequence alignment has cost C with

$$\begin{aligned} & C \;=\; \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} C(i,j) \;\leq \\ & \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} C(i,1) \;+\; C(1,j) \;= \\ & 2\,(N-1) \sum_{i=2}^{N} C(i,1)\,. \end{aligned} \tag{4.19}$$

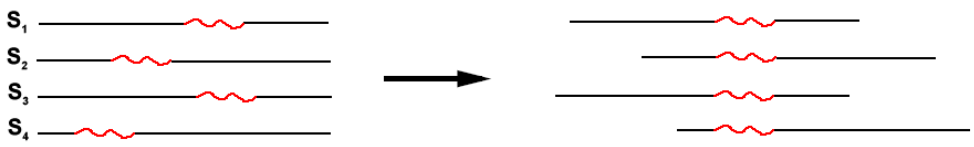


Figure 4.6: The figure depicts how motifs can be aligned in order to obtain landmarks for multiple alignments.

For the optimal cost C^* we have

$$C^* = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} C(i,j) \geq \tag{4.20}$$
$$\sum_{i=1}^{N} \sum_{j=2}^{N} C(i,1) = N \sum_{i=2}^{N} C(i,1) \, .$$

Therefore

$$\frac{C}{C^*} \leq \frac{2(N-1)}{N} \leq 2 \, . \tag{4.21}$$

4.3.3.2 Motif- and Profile-based Methods

If motifs, i.e. patterns, are found then they can be superimposed onto each other and landmarks for the alignment are generated. Fig. 4.6 shows how motifs/patterns can be aligned in order to get landmarks for multiple alignments.

Profiles and blocks can be derived from multiple alignments of protein families. These can be used to find patterns in new sequences.

4.3.3.3 Probabilistic and Model-based Methods

SAGA (Sequence Alignment by Genetic Algorithm) uses a genetic algorithm and MSASA (Multiple Sequence Alignment by Simulated Annealing) simulated annealing for finding good alignments. Gibbs sampling is another method to find solutions of discrete problems (it is very similar to genetic algorithms).

HMMs (hidden Markov models) can be used to find motifs but they almost always get stuck in local minima without initializing them with conservative regions.

4.3.3.4 Divide-and-conquer Algorithms

All N sequences are broken up into two subsequences and the two groups of subsequences are multiple aligned (cf. Fig. 4.7). These idea suffers from finding the optimal cut positions. If an alignment is divided then the global alignment must go through the cut (cf. Fig. 4.8). Stoye introduces additional cost matrices for each sequence pair which evaluate each cut position by assuming that the alignment must go through this position.

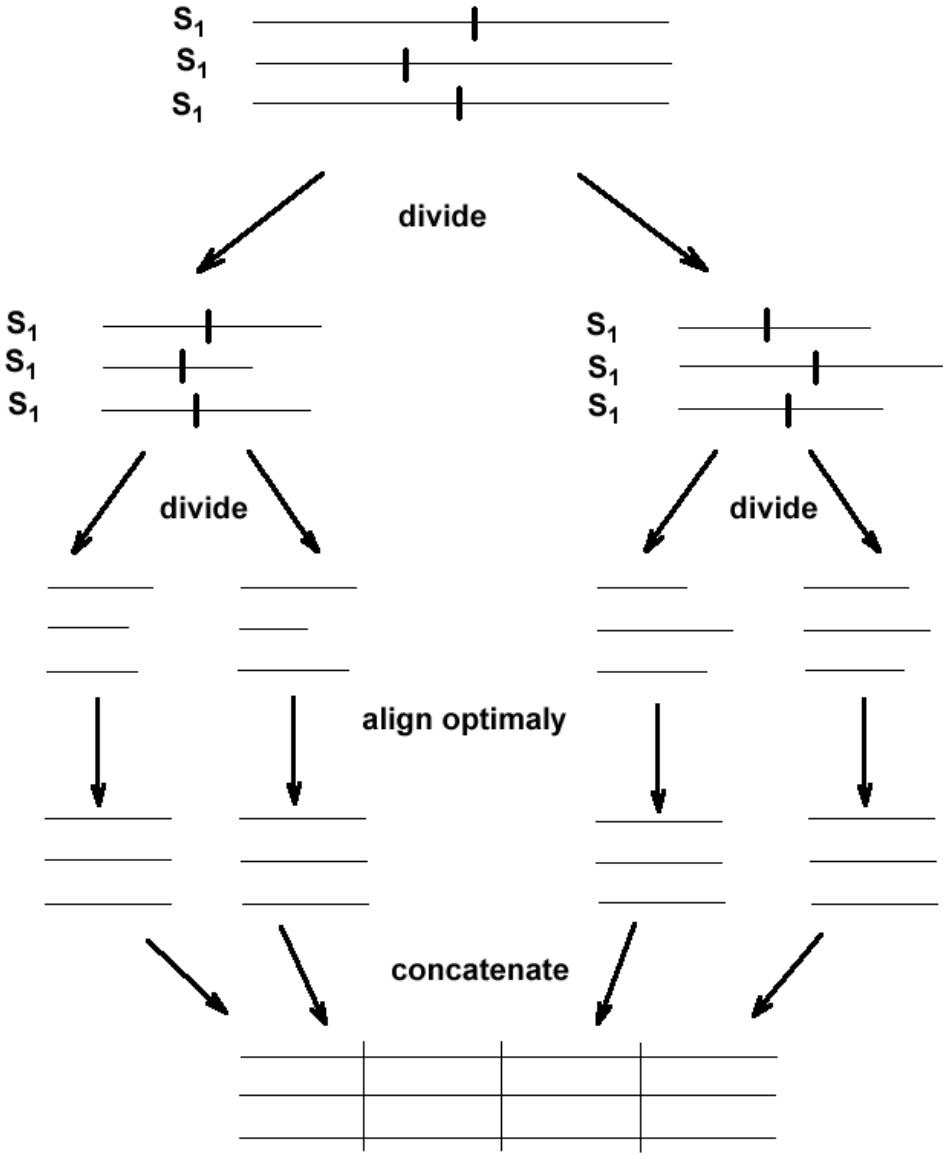


Figure 4.7: The idea of the divide and conquer algorithm is depicted. The sequences are cut and the cuts separately aligned whereafter the alignments are concatenated.

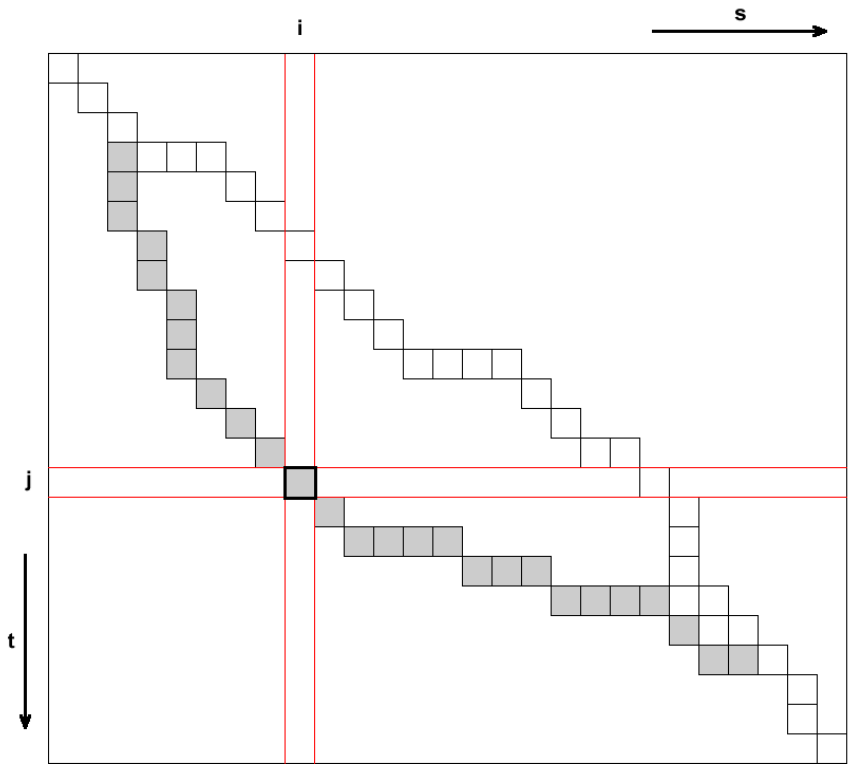


Figure 4.8: A cut position i, j is depicted. The global alignment must go through the cut position.

4.4 Profiles and Position Specific Scoring Matrices

As in section 3.4.1 for pairwise alignments Karlin and Altschul, 1990, showed similar results for single sequences. For gapped alignment see Altschul et al., 1997, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

The following assumptions are given:

- the sequence $\boldsymbol{x}$ is i.i.d. in its elements according to the letter probabilities p_x
- n the length of $\boldsymbol{x}$ is large
- the expected letter score for random sequences $\sum_i \; p_x(i) \; s(i)$ is negative
- there exist i for which $s(i) \; > \; 0$ (existence of a positive score)

The following is analog to section 3.4.1. The score for a sequence of length n is

$$S_n \; = \; \sum_{i=1}^{n} \; s(i) \; . \tag{4.22}$$

the centered value $\tilde{S}_n \; = \; S_n \; - \; \frac{\ln n}{\lambda}$, is distributed according to an extreme value distribution:

$$P\left(\tilde{S}_n > S\right) \; \approx \; 1 - \exp\left(-K \; e^{-\lambda \; S}\right) \; \approx \; K \; e^{-\lambda \; S} \; , \tag{4.23}$$

where λ is the solution of

$$\sum_i p_x(i) \; \exp(\lambda \; s(i)) \; = \; 1 \; . \tag{4.24}$$

Let q_i be the frequency of a letter a_i in a column of a multiple alignment. Karlin and Altschul, 1990, showed that for sufficiently high scoring segments

$$\lim_{n \to \infty} \; q_i \; = \; p_x(i) \; \exp(\lambda \; s(i)) \; . \tag{4.25}$$

Therefore

$$s(i) \; = \; \ln\left(\frac{q_i}{p_x(i)}\right) / \lambda \tag{4.26}$$

is the score for the letter i at this position. Typical values for λ and K are $\lambda \; = \; 0.254$ and $K = 0.040$.

These scores per position are called “Position Specific Scoring Matrices” (PSSMs) or profiles. They serve to evaluate a new sequence where high scores mean that the sequence is similar to the sequences from which the multiple alignment was constructed.

To estimate the probability $p_e(i \mid c)$ of amino acid i to be in column c PSI-BLAST uses pseudo-counts g_i:

$$g_i = \sum_j \frac{q_j}{p_x(i)} q_{ij} , \tag{4.27}$$

where q_{ij} are the target frequencies given by the entries of the substitution matrix.

The pseudo-count results from summing out the hidden variables:

$$q_{ij} = p(i,j) \tag{4.28}$$
$$p_x(i) = p(i) \tag{4.29}$$
$$\frac{p(i,j)}{p(i)} = p(i \mid j) = p(i \mid j,c) \tag{4.30}$$
$$q_j = p(j \mid c) \tag{4.31}$$
$$p_p(i \mid c) = \sum_{j=1}^{20} p(i \mid j,c)\, p(j \mid c) , \tag{4.32}$$

where in the last equation the hidden variable j is integrated (summed) out.

Finally the pseudo-counts are averaged with the observed frequencies:

$$p_e(i \mid c) = \frac{\alpha p(i \mid c) + \beta g_i}{\alpha + \beta} \tag{4.33}$$

Using N_c, the number of different residues in column c, the default values are

$$\alpha = N_c - 1 \tag{4.34}$$
$$\beta = 10 . \tag{4.35}$$

N_c is a measurement of independence of the sequences. For example if we have 10 identical sequences then we would overestimate the according amino acids.

The independence of sequences can be counted in different ways. Sunyaev et al. (1999) count in the PSIC-approach the number of identical positions to estimate the independence. The PSIC-method was extended in Mittelman el al. (2003) by grouping together the sequences with the same amino acid in a certain column and then computing the effective frequencies by the PSIC-method.

Chapter 5

Phylogenetics

5.1 Motivation

5.1.1 Tree of Life

One central field in biology is to infer the relation between species. Do they possess a common ancestor? When did they separate from each other? The relation is presented in form of a tree with a unique root. Leaves of the tree are currently observable species and are called taxa. The branches represent the relationship ("is ancestor of") between nodes. The root is the start of life on earth. From the first living organism all other life forms are derived. An edge in the tree of life means that a species is the ancestor of another one. A node means a split of a species into two.

The construction of the tree of life is called phylogeny (phylo = tribe and genesis). Classical biological phylogeny is divided into the cladistic and the phenetic approach. Cladistic trees are based on conserved characters and phenetic trees on the measure of distance between the leaves of the tree (the phenetic approach considers the distance as a whole and not based on single features). Problems of the phenetic approach are simultaneous development of features and different evolution rates. There may be convergent evolution e.g. finding the best form in water.

For example the phylogenetic relation between some well known animals is depicted in Fig. 5.1. The phylogenetic relation between humans and apes is shown in Fig. 5.2. Interesting questions appear at the root of the tree of life. How did life start? Fig. 5.3 shows the root of the tree of life as it divides into the three kingdoms Bacteria, Archaea, and Eukarya.

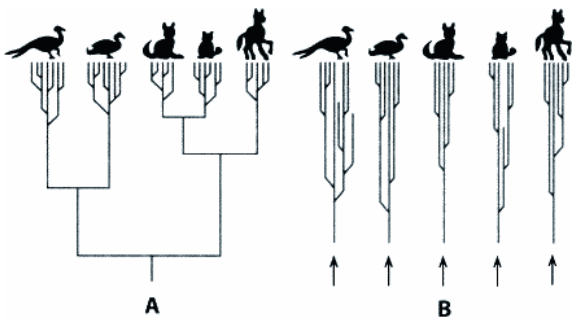


Figure 5.1: Tree of life for some animals. Birds are separated.

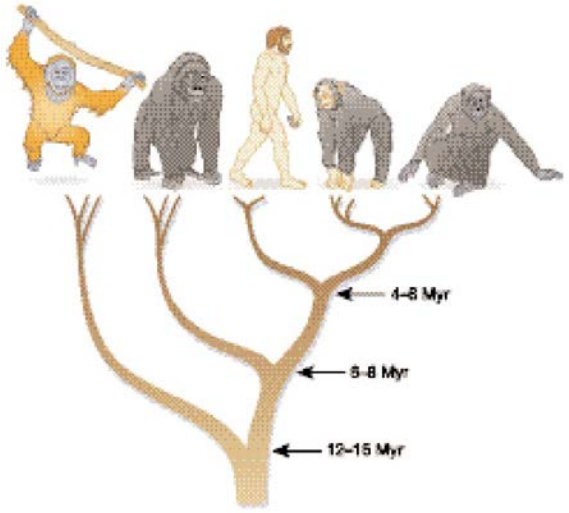


Figure 5.2: Tree of life focused on the relation between human and apes.

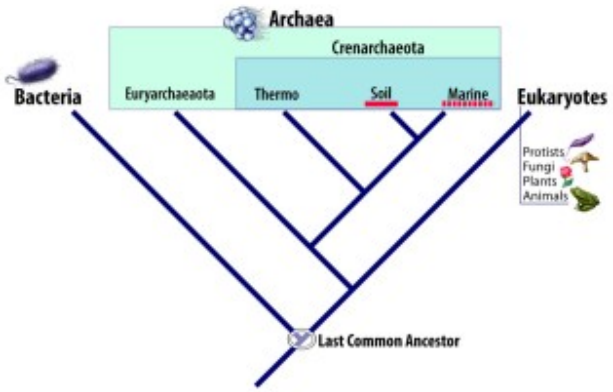


Figure 5.3: The root of the tree of life.

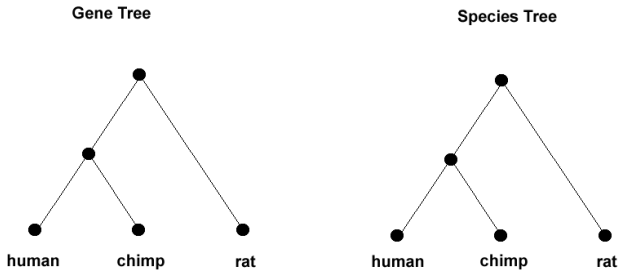


Figure 5.4: The gene tree for the gene α-hemoglobin compared to the species tree. Both match because the gene evolved from common ancestors.

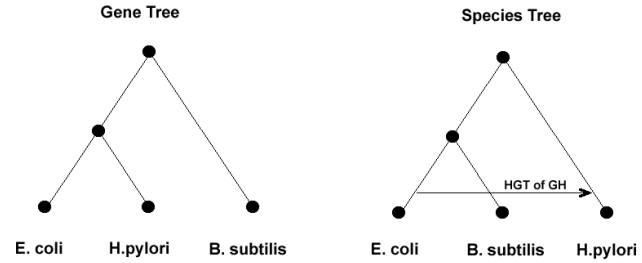


Figure 5.5: The gene tree for the gene Glycosyl Hydrolase compared to the species tree. The trees do not match because of the horizontal gene transfer (HGT).

5.1.2 Molecular Phylogenies

We are focusing on molecular phylogenies in contrast to phylogenies based on characteristics like wings, feathers, etc, i.e. morphological characters. With molecular phylogenetics, the differences between organisms are measured on the proteins and RNA coded in the DNA, i.e. on amino acid and nucleotide sequences. In Fig. 5.4 the species tree and the gene tree for α-hemoglobin are depicted – they match. Molecular phylogenetics is more precise than its counterpart based on external features and behavior and can also distinguish small organism like bacteria or even viruses. Further advantages of molecular phylogenetics are that the DNA must be inherited and connects all species, the molecular phylogenetics can be based on mathematical and statistical methods and is even model-based as mutations can be modeled, remote homologies can be detected, the distance is not only based on one feature, but on many genes.

There are difficulties in constructing a phylogenetic tree. First, different regions in the DNA mutate at different rates. Which means that the distances may look differently. Secondly, horizontal transfer of genetic material (Horizontal Gene Transfer, HGT) between species is possible, e.g. through a virus, DNA transformation, symbiosis, or other mechanism. For example Glycosyl Hydrolase was transferred from E.coli to B.subtilis (see Fig. 5.5).

The branches of the tree represent time measured in number of mutations. In the molecular clock hypothesis it is assumed that different branches have the same evolution / mutation rate. The number of substitutions is assumed to follow a Poisson distribution. For sequences the mutation rate is assumed to be equally distributed over the sequence.

However, phylogenetics can be used to infer gene functions, find regions with high or low mutation rate and, therefore, conservative regions,

5.1.3 Methods

The first step in doing phylogenetics is to choose the sequences from which the tree should be constructed. Very popular sequences to construct phylogenetic trees are the sequences of rRNA (the RNA the ribosome is build of) and mitochondrial genes. These genetic material is present in almost all organisms and they have enough mutations to reliably construct a tree.

The second step is to construct pairwise and multiple sequence alignments from these sequences.

The third step is to choose a method for constructing a phylogenetic tree. There exist 3 categories: distance-based, maximum parsimony, and maximum likelihood.

Maximum parsimony should be chosen for strong sequence similarities because too much variation results in many possible trees. For the same reason only few sequences (less than 15) should be used. Distance based methods (e.g. clustalW) require less similarity among the sequences than maximum parsimony methods but sequence similarities should be present. Some sequences should be similar to one another and others are less similar. Distance based methods can be applied to a set of many sequences. Maximum likelihood methods may be used for very variable sequences but the computational costs increase with the number of sequences as every possible tree must be considered.

Software for phylogenetic analysis:

Name	Author	URL
PHYLIP	Felsenstein 1989, 1996	`http://evolution.genetics.washington.edu/phylip.html`
PAUP	Sinauer Associates	`http://paup.csit.fsu.edu/`

5.2 Maximum Parsimony Methods

The number of mutations should be minimized. Mutations are represented as branches in the tree which explains the evolution of the sequences. If surviving mutations are rare, i.e. occur with small probability, then the tree with minimal mutations is the most likely explanation of the evolution of the observed sequences.

PHYLIP programs DNAPARS, DNAPENNY, DNACOMP, DNAMOVE, and PROTPARS are based on maximum parsimony methods.

5.2.1 Tree Length

The *maximum parsimony tree* is the tree with smallest tree length. *Tree length* is the number of substitutions represented by the tree, where sequence symbols are assigned to the internal nodes.

For example an aligned subsequence of the protein triosephosphate isomerase for the taxa “Human”, “Pig”, “Rye”, “Rice”, and “Chicken” is:

taxa	position					
	1	2	3	4	5	6
Human	I	S	P	G	M	I
Pig	I	G	P	G	M	I
Rye	I	S	A	E	Q	L
Rice	V	S	A	E	M	L
Chicken	I	S	P	A	M	I

If we focus on column 4:

taxa	amino acid
Human	G
Pig	G
Rye	E
Rice	E
Chicken	A

Fitch, 1971, proposed an algorithm for computing the tree length for a tree topology with taxa assigned to the leaves. Note, that here the amino acids at a certain position of certain protein sequences represent the taxa. See Fig. 5.6 for an example of a tree topology.

1. Root node is added to an arbitrary branch and the taxa are replaced by the according symbol. See the tree resulting from adding a root to the example in Fig. 5.7.

2. A bottom-up pass generates sets of symbols (amino acids) which are possible at a hypothetical sequence at this node. The hypothetical sequences are chosen to minimize the number of mutations by finding the maximal agreement of the subtrees, i.e. by choosing the maximal set which is subset of both subtree roots. This set allows to avoid a mutation at the actual node. The bottom up pass starts at the leaves.

$$m_{12} = \begin{cases} \{\text{“leaf symbol”}\} & \text{if} \quad m_1 = m_2 = \emptyset \\ m_1 \cup m_2 & \text{if} \quad m_1 \cap m_2 = \emptyset \\ m_1 \cap m_2 & \text{if} \quad m_1 \cap m_2 \neq \emptyset \end{cases} \tag{5.1}$$

   In the first case m_{12} is leaf, the second case enforces a mutation, and the third case avoids a mutation. Fig. 5.8 depicts the relation between the sets m_{12}, m_1, and m_2. Fig. 5.9 shows the result after the bottom-up pass for the example.

3. A top down pass generates special hypothetical sequences at the interior nodes of the tree and counts the number of mutations. The top down pass starts at the root.

$$m_{1/2} = \begin{cases} x \in m_{1/2} \cap m_{12} & \text{if} \quad m_{1/2} \cap m_{12} \neq \emptyset \\ x \in m_{1/2} & \text{if} \quad m_{1/2} \cap m_{12} = \emptyset \end{cases} \tag{5.2}$$

   $m_{1/2}$ means that the formula hold both for m_1 and for m_2. Any x can be chosen which fulfills the conditions. Fig. 5.10 shows one result after the top-down pass for the example.

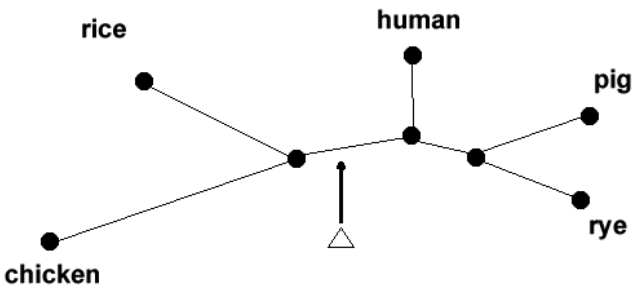


Figure 5.6: A tree topology to which a root node is added.

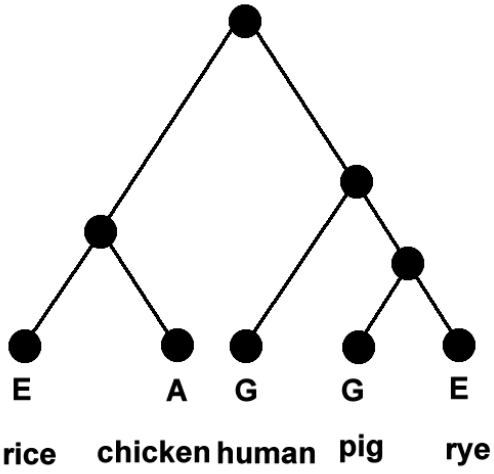


Figure 5.7: The tree after the root node is added.

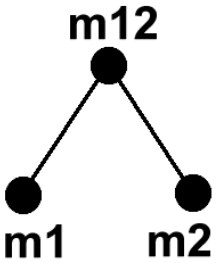


Figure 5.8: Root set m_{12} is constructed from left set m_1 and right set m_2.

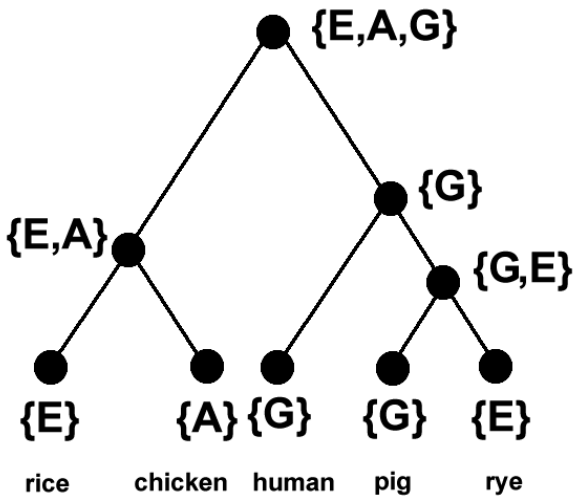


Figure 5.9: The tree after the bottom up pass. Each node has its set of symbols constructed from the subtrees of the node.

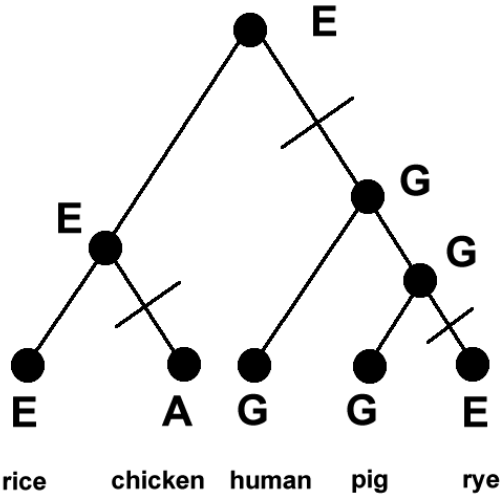


Figure 5.10: The tree after the top down pass. Mutations are now marked by a crossed branch. This tree contains three mutations at the column under consideration. Branches which are not marked have mutations at other columns.

The number of mutation can be either counted during the bottom-up or top-down pass.

Note that columns with maximal or minimal entropy, i.e. where all symbols occur only once in the column or with only one symbol in the column, are not used. Only one symbol in the column means no mutation and symbols which occur only once lead to the same number of mutation in all topologies. Also if one symbol occurs multiple times and others only once then the number of mutation is the same for each reasonable tree topology because each single symbol is obtained by one mutation from another symbol.

Next we give some evaluation criteria for the different trees. The *consistency index* c_i is the minimal number of substitutions m_i for a column divided by the number of substitutions s_i for the topology at hand:

$$c_i = \frac{m_i}{s_i} . \tag{5.3}$$

High values of c_i support the according tree as being plausible. Using g_i, the number of maximal substitutions at a column for a star tree topology with the most frequent symbol in the center, the *retention index* r_i (a measurement how informative the column is) and the *re-scaled consistency index* rc_i are

$$r_i = \frac{g_i - s_i}{g_i - m_i} \text{ and} \tag{5.4}$$

$$rc_i = r_i \, c_i . \tag{5.5}$$

5.2.2 Tree Search

For a small number of sequences all trees can be constructed and their tree length computed, i.e. the trees are found by exhaustive search. However for a larger number of sequences not all trees can be considered. Therefore heuristics will find the most plausible trees.

5.2.2.1 Branch and Bound

The "branch and bound" algorithm has been introduced by Hendy and Penny, 1982. 20 and more taxa can be processed.

The algorithm works as follows:

1. This step determines the addition order for the taxa used in later steps. The algorithm starts with a core tree of three taxa which has the maximal length of all three taxa trees. Next taxa are added to each of the three branches. The taxa is chosen next and inserted into the queue which leads to a tree with maximal tree length. For the tree with four branches we determine the next taxa which can be added and leads to the maximal tree length. At the end we have a sorted taxa list.

2. This step determines an upper bound for the tree length. The upper bound can be computed either through distance based methods like neighbor joining (see next section) or a heuristic search like the stepwise addition algorithm (see below).

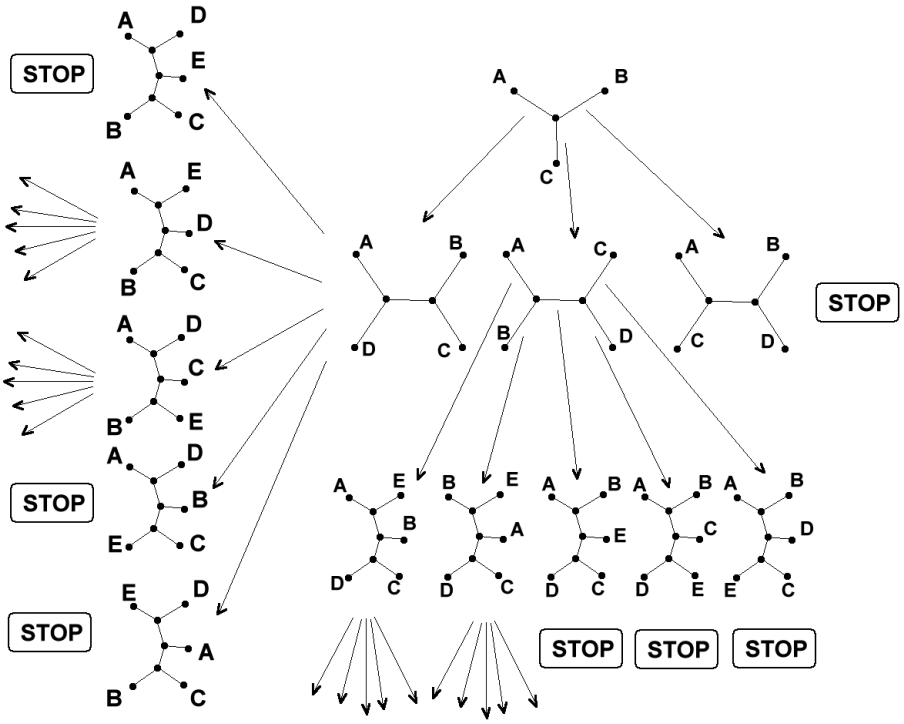


Figure 5.11: Example for constructing trees with the brunch-and-bound method. Trees with STOP mark do not possess successors because their tree length exceeds the upper bound.

3. The algorithm starts with the core tree of three taxa.

4. Construct new tree topologies. The topologies are constructed by stepwise adding new taxa to the trees which do not possess a STOP mark. The next taxa is chosen according to the list in step 1. The next taxa is added to each tree without a STOP mark at all of its branches. All tree lengths are computed.

5. Assign STOP marks. If a tree length reaches or exceeds the upper bound then this tree will not produce successors and will receive a STOP mark. If all trees possess a STOP mark then terminate else go to step 4.

See Fig. 5.11 for an example for the Branch and Bound algorithm.

5.2.2.2 Heuristics for Tree Search

5.2.2.2.1 Stepwise Addition Algorithm At each step only the tree with shortest length does not obtain a STOP mark and is extended. If all taxa are inserted then the tree is optimized by branch swapping (see below). Instead of adding only one taxa also small sets of taxa e.g. all triplets can be considered.

5.2.2.2.2 Branch Swapping Either nearest (1) neighbor interchange, (2) subtree pruning and re-grafting, or (3) bisection-reconnection can be applied. The first algorithm exchanges in each optimization step two taxa connected to the same interior branch. The second algorithm cuts the tree into two parts: the pruned subtree and the residual tree (the part taken away from the original tree giving the pruned subtree). The residual tree is now attached with its root to each branch of the pruned tree. The third algorithm extends the second algorithms. Here the original tree is cut into two subtrees by removing a branch. Then two branches of the subtrees (one of each) are joined by inserting a new branch to a new topology. This extends the second algorithm because not only the root is attached but all branches.

5.2.2.2.3 Branch and Bound Like Instead of selecting the maximal tree length in step 1. of the branch-and-bound algorithm the minimal tree is selected. The upper bound is updated during constructing new trees. Towards this end upper bounds U_n are constructed for n taxa added. In this way the branch-and-bound algorithm is modified to local bounds.

5.2.3 Weighted Parsimony and Bootstrapping

The type of substitution is weighted according to a substitution matrix. We already encountered PAM and BLOSUM matrices. This gives a more precise value of tree length because the survival of substitutions is taken into account, e.g. the chemical similarity of amino acids.

Bootstrapping is used to access the variability of the tree with respect to the data ("variance" in statistics) and to identify structures which do not vary with the data. Thus a value for uncertainty is obtained. The data for parsimony trees are the columns of the alignment. The temporal order of these columns does not matter. Therefore columns can be chosen randomly to generate a new data set from which another tree can be constructed. Bootstrap cannot access the quality of a method ("bias" in statistics) but only the robustness with respect to the data.

5.2.4 Inconsistency of Maximum Parsimony

In Fig. 5.12 an example is shown, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are similar to each other. Sequences $\boldsymbol{c}$ and $\boldsymbol{d}$ are not similar to any other sequence. Assume $\boldsymbol{a}$ and $\boldsymbol{b}$ match to 99%. $\boldsymbol{c}$ or $\boldsymbol{d}$ match by chance to other sequences in 5% (1 out of 20) of the cases. Informative columns are those which contain only two symbols and each symbol appears twice. We obtain for the probabilities of informative columns and their rate (number of informative columns of this kind divided through all informative columns):

$$a_i = b_i,\ c_i = d_i: \quad \text{prob: } 0.0495(0.99 \cdot 0.05) \quad \text{rate: } 0.908 \tag{5.6}$$

$$a_i = c_i,\ b_i = d_i: \quad \text{prob: } 0.0025(0.05 \cdot 0.05) \quad \text{rate: } 0.046 \tag{5.7}$$

$$a_i = d_i,\ b_i = c_i: \quad \text{prob: } 0.0025(0.05 \cdot 0.05) \quad \text{rate: } 0.046 \tag{5.8}$$

In more than 90% of the cases of informative columns we observe $c_i = d_i$. Maximum parsimony will judge $\boldsymbol{c}$ and $\boldsymbol{d}$ as being similar to one another as $\boldsymbol{a}$ and $\boldsymbol{b}$ and will construct a wrong tree.

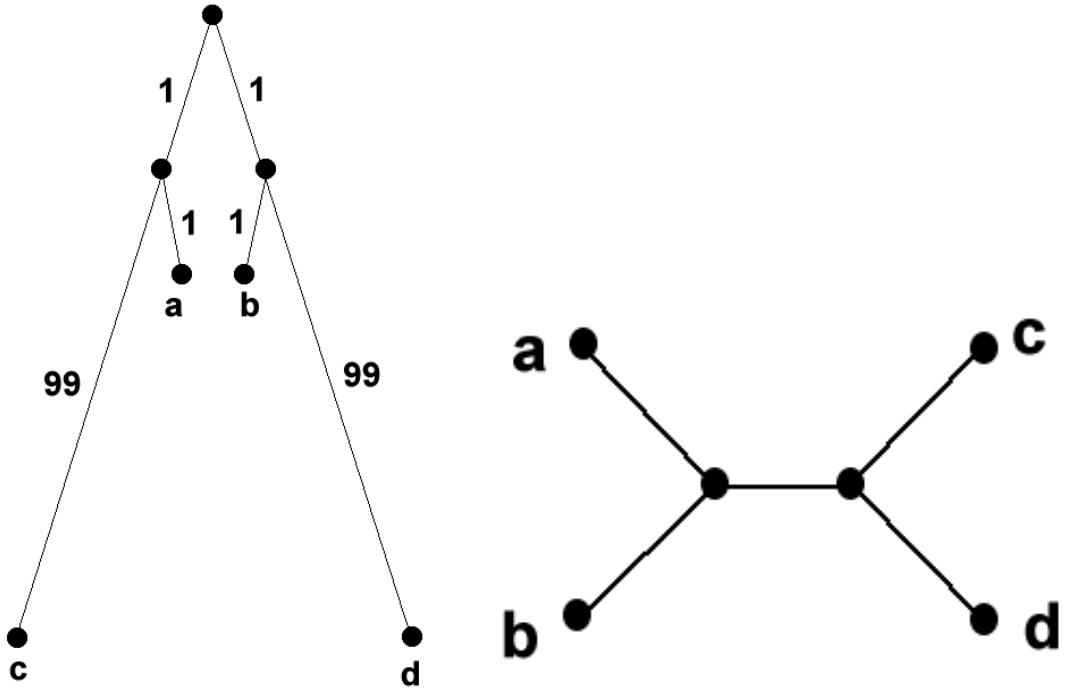


Figure 5.12: An example where maximum parsimony is inconsistent. Left: the true tree. Right: tree from maximum parsimony. Sequence a and b are similar to each other. Sequences c and d are not similar to any other sequence. For long sequences maximum parsimony may be inconsistent.

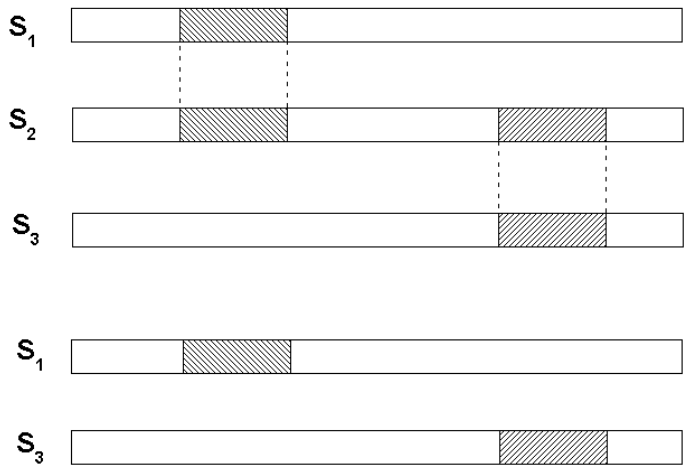


Figure 5.13: Three sequences where the triangle inequality does not hold for the e-value ($d(1,3) \leq d(1,2) + d(2,3)$). Sequence 1 and 2 as well as sequence 2 and 3 have matching regions but sequence 1 and 3 do not have matching regions.

5.3 Distance-based Methods

We assume that we are given the matrix of pairwise distances between the sequences. A distance D is produced by a metric d on objects x indexed by i, j, k. The distance D_{ij} between object x_i and object x_j is

$$D_{ij} \;=\; d(x_i, x_j)\;, \tag{5.9}$$

where the metric d must fulfill

$$d(x_i, x_j) \;\geq\; 0\;, \tag{5.10}$$
$$d(x_i, x_j) \;=\; 0 \text{ for } i = j\;, \tag{5.11}$$
$$d(x_i, x_j) \;=\; d(x_j, x_i)\;, \tag{5.12}$$
$$d(x_i, x_j) \;\leq\; d(x_i, x_k) \;+\; d(x_k, x_j)\;. \tag{5.13}$$

The last condition is the triangle inequality.

How to compute distances is considered later. Note, that not all scoring schemes are a metric. For example the e-value from the BLAST search. Small e-value means related sequences, i.e. small distance. But as depicted in Fig. 5.13 the triangle inequality may be violated. The e-value between sequence 1 and 3 is larger than the sum of the e-values between sequences 1 and 2 and sequences 2 and 3.

5.3.1 UPGMA

One of the classical tree construction algorithms is the Unweighted Pair Group Method using arithmetic Averages (UPGMA). UPGMA is actually a constructive clustering method based on joining pairs of clusters.

It works as follows:

1. Initially, each sequence i is a cluster c_i with one element $n_i = 1$. The height l_i of each cluster is zero. Put all i into a list.
2. Select cluster pair (i, j) from the list with minimal distance D_{ij} and create a new cluster c_k by joining c_i and c_j. Assign the height $l_k = D_{ij}/2$ and the number of elements $n_k = n_i + n_j$.
3. Compute the distances for the new cluster c_k to other clusters c_m:

$$D_{km} = \frac{n_i\, D_{mi} + n_j\, D_{mj}}{n_i + n_j} \,. \tag{5.14}$$

   The formula ensures that D_{km} is the average distance of all elements in c_k and c_m.
4. Remove i and j from the list and add k to the list. If the list contains only one element then terminate else go to step 2.

For a detailed example of the UPGMA algorithm see Appendix C.

The assumption of constant rate of evolution in the different lineages must hold to ensure that the distance measure really measures evolutionary distances.

Again bootstrapping can evaluate the reliability of the result to data variation (robustness). Whether all interior branches are positive can be tested to evaluate the quality of the tree.

5.3.2 Least Squares

The idea of the least squares method is to minimize the differences between the observed distance D_{ij} and the distances E_{ij} in the tree. E_{ij} is the sum of distances (assigned to branches) in the tree on the path from taxa i to taxa j (the path metric). Therefore, E_{ij} is an estimate how far taxa i evolved apart from the common ancestor of taxa i and j plus how far taxa j evolved apart from the common ancestor.

The objective is

$$\sum_{i<j} (D_{ij} - E_{ij})^2 \,. \tag{5.15}$$

Fitch and Margoliash, 1967, extended this approach to the weighted least squares method by using as objective

$$\sum_{i<j} (D_{ij} - E_{ij})^2 / D_{ij}^2 \,. \tag{5.16}$$

The objective is optimized under the constraint of nonnegative branch length.

If matrix $\boldsymbol{A}$ is the binary topology matrix with $N(N-1)/2$ rows – one for each D_{ij} – and v columns for the v branches of the topology. In each row (i,j) all branches contained in the path from i to j are marked by 1 and all other branches are 0.

$\boldsymbol{l}$ is the v-dimensional vector of branch weights. We obtain

$$\boldsymbol{E} \;=\; \boldsymbol{A}\,\boldsymbol{l} \tag{5.17}$$

The least squares assumption is that D_{ij} deviates from E_{ij} according to a Gaussian distribution ϵ_{ij} with mean 0 and variance D_{ij}^2:

$$\boldsymbol{D} \;=\; \boldsymbol{E} \;+\; \boldsymbol{\epsilon} \;=\; \boldsymbol{A}\,\boldsymbol{l} \;+\; \boldsymbol{\epsilon}\,. \tag{5.18}$$

The maximum likelihood estimator (least squares) is

$$\hat{\boldsymbol{l}} \;=\; \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{D}\,. \tag{5.19}$$

The Gaussianity assumption is justified by sufficiently large sequences because in such cases the l_i are Gaussian and, therefore, also D_{ij}.

5.3.3 Minimum Evolution

The objective is the sum of branch lengths l

$$L \;=\; \sum_{ij} \hat{l}_{ij}\,. \tag{5.20}$$

Given an unbiased branch length estimator $\hat{l}$, the expected value of L is smallest for the true topology independent of the number of sequences (Rzhetsky and Nei, 1993).

Minimum evolution is computationally expensive.

5.3.4 Neighbor Joining

The neighbor joining method was introduced by Saitou and Nei, 1987, and and simplifies the minimum evolution method (for fewer than six taxa both methods give the same result).

The taxa are grouped together like in the UPGMA method.

Neighbors are taxa that are connected by a single node.

For an *additive metric* d any four elements can be indexed by i, j, k and m in order to fulfill

$$d(i,j) \;+\; d(k,m) \;\leq\; d(i,k) \;+\; d(j,m) \;=\; d(i,m) \;+\; d(j,k)\,. \tag{5.21}$$

The path metric (counting the branch/edge weights of the path) is an additive metric. An additive metric can be represented by a unique additive tree.

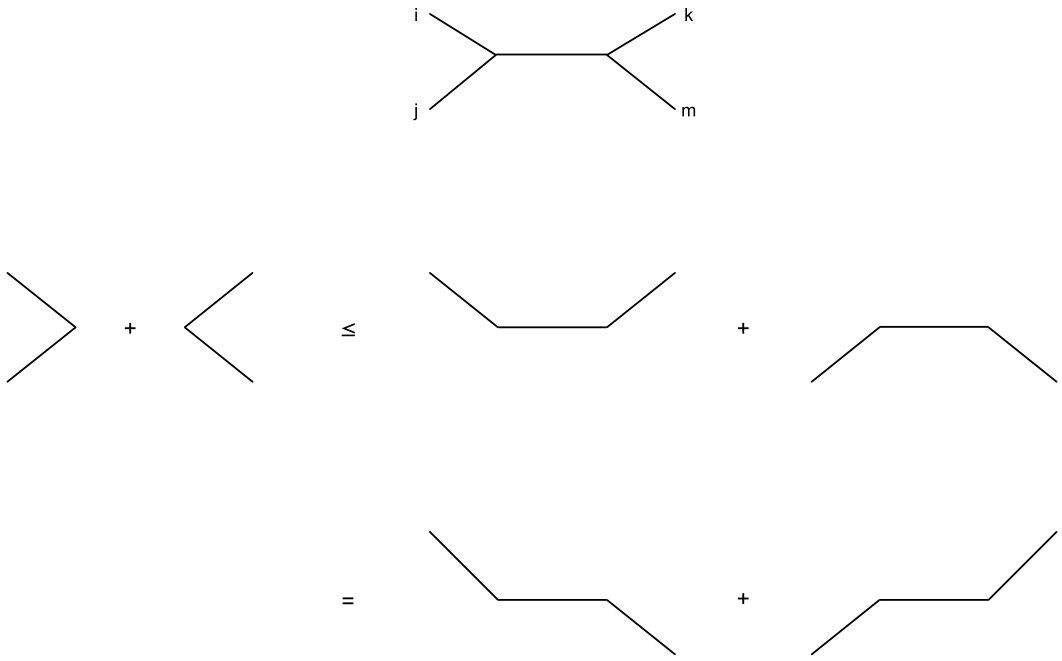


Figure 5.14: Four point condition of an additive metric.

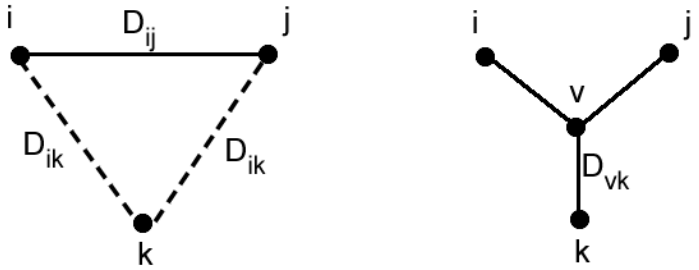


Figure 5.15: Construction of an additive tree from an additive metric. Node v is inserted.

In Fig. 5.15 the construction of an additive tree is depicted. A node v is inserted with distances:

$$D_{vk} \;=\; \frac{1}{2}\left(D_{ik} \;+\; D_{jk} \;-\; D_{ij}\right) \tag{5.22}$$

$$D_{iv} \;=\; \frac{1}{2}\left(D_{ij} \;+\; D_{ik} \;-\; D_{jk}\right) \tag{5.23}$$

$$D_{jv} \;=\; \frac{1}{2}\left(D_{ij} \;+\; D_{jk} \;-\; D_{ik}\right)\;. \tag{5.24}$$

The additive tree conditions fulfill above constraints, and vice versa.

$$D_{ij} \;=\; D_{iv} \;+\; D_{vj} \tag{5.25}$$

$$D_{ik} \;=\; D_{iv} \;+\; D_{vk} \tag{5.26}$$

$$D_{jk} \;=\; D_{jv} \;+\; D_{vk}\;. \tag{5.27}$$

The objective of the neighbor joining algorithm is S the sum of all branch lengths l_{ij} (the branch between taxa or node i and taxa or node j). The algorithm starts with a star tree.

In subfigure a) of Fig. 5.16 the initial star tree is depicted with internal node X.

We assume N taxa. The initial (star tree) objective S_0 is

$$S_0 \;=\; \sum_{i=1}^{N} l_{iX} \;=\; \frac{1}{N-1}\sum_{i,j;i<j} D_{ij}\;, \tag{5.28}$$

where the $\frac{1}{N-1}$ comes from the fact that $D_{ij} = l_{iX} + l_{Xj}$, therefore l_{iX} is part of $N-1$ distances D_{ij}.

In the next step taxa 1 and 2 are joined and a new internal node Y is introduced as depicted in subfigure b) of Fig. 5.16.

Because we assume an additive tree the branch length l_{XY} can be computed:

$$l_{XY} \;=\; \frac{1}{2(N-2)} \tag{5.29}$$
$$\left(\sum_{i=3}^{N}\left(D_{1i}+D_{2i}\right) \;-\; (N-2)\;\left(l_{1Y} \;+\; l_{2Y}\right) \;-\; 2\sum_{i=3}^{N} l_{Xi}\right)\;.$$

This equation is obtained from setting all paths from i to j containing l_{XY} equal to D_ij and solving for l_{XY}. These are all paths from node 1 to nodes $i \geq 3$ and all paths from node 2 to nodes $i \geq 3$. Therefore $(N-2)$ paths start from node 1 and $(N-2)$ paths start from node 2 giving $2(N-2)$ paths. l_{1Y} is contained in all paths starting from node 1 (similar for l_{2Y}). The tail l_{iX} is contained in one path starting from 1 and one path starting from 2. We have $2(N-2)$ solutions for l_{XY}. A least square fit just averages over these solutions and we obtain above equation.

Similar as for the initial star tree we obtain

$$\sum_{i=3}^{N} l_{Xi} \;=\; \frac{1}{N-3}\sum_{i,j;3\leq i<j} D_{ij}\;, \tag{5.30}$$

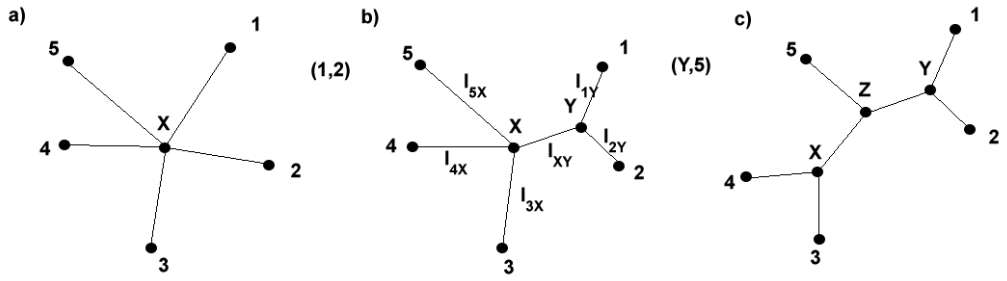


Figure 5.16: a) An initial star tree; b) the tree resulting from the star tree if neighbors A and B are joined; c) the tree resulting from the tree in b) by joining AB and D.

which can be inserted into the last equation.

For the new objective S_{12} after joining taxa 1 and 2 we obtain

$$S_{12} \;=\; l_{1Y} \;+\; l_{2Y} \;+\; l_{XY} + \sum_{i=3}^{N} l_{Xi} = \tag{5.31}$$

$$\frac{1}{2(N-2)} \sum_{i=3}^{N} (D_{1i} + D_{2i}) \;+\; \frac{1}{2} D_{12} \;+\; \frac{1}{N-2} \sum_{i,j;3 \leq i < j} D_{ij} \; .$$

The last equation can be generalized from joining 1 and 2 to joining k and l. We introduce variables r_k called *net divergences*, which give the accumulated distances of k to all other taxa:

$$r_k \;=\; \sum_{i=1}^{N} D_{ki} \; . \tag{5.32}$$

Now the objective of joining k and l can be expressed as

$$S_{kl} \;=\; \frac{2 \sum_{i,j;i<j} D_{ij} \;-\; r_k \;-\; r_l}{2(N-2)} \;+\; \frac{D_{kl}}{2} \; . \tag{5.33}$$

Because

$$\frac{2 \sum_{i,j;i<j} D_{ij}}{2(N-2)} \tag{5.34}$$

is constant for all objectives S_{kl} an equivalent objective after a linear transformation (adding a constant and re-scaling) is

$$Q_{kl} \;=\; (N-2)\, D_{kl} \;-\; r_k \;-\; r_l \; . \tag{5.35}$$

If k and l are evolutionary neighbors but D_{kl} is large due to fast evolution of k and/or l, then this will result in large r_k and/or r_l. Therefore, Q_{kl} will be small.

The algorithm works as follows:

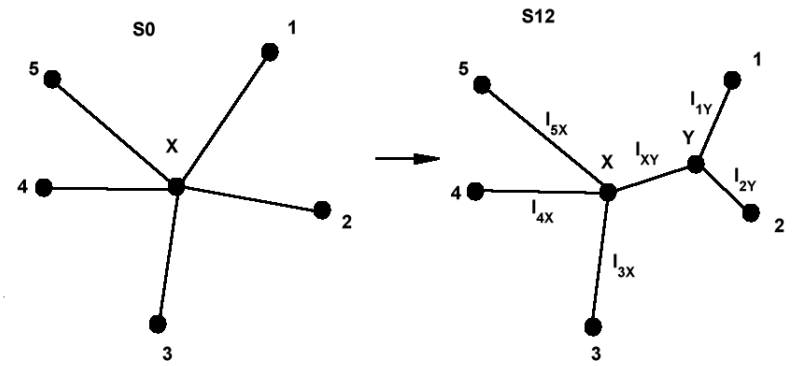


Figure 5.17: An initial star tree with center X and the tree resulting from the star tree if neighbors 1 and 2 are joined and hidden node Y is introduced.

1. Given D_{ij} start with a star tree (the taxa are the leaves). Put all taxa in a set of objects.

2. For each leaf i compute

$$r_i \;=\; \sum_{k=1}^{N} D_{ik} \;. \tag{5.36}$$

3. For each pairs of leaves (i, j) compute

$$Q_{ij} \;=\; (N-2)\, D_{ij} \;-\; r_i \;-\; r_j \;. \tag{5.37}$$

4. Determine the minimal Q_{ij}. Join the according leaves i and j to a new leaf u. Compute branch lengths

$$\begin{aligned} l_{iu} \;&=\; \frac{D_{ij}}{2} \;+\; \frac{r_i \;-\; r_j}{2(N-2)} \\ l_{ju} \;&=\; D_{ij} \;-\; l_{iu} \;. \end{aligned} \tag{5.38}$$

   Compute new distances of u:

$$D_{ku} \;=\; \frac{D_{ik} \;+\; D_{jk} \;-\; D_{ij}}{2} \;. \tag{5.39}$$

   Delete i and j from the set of objects. Add u to the set of objects. Stop if the set of objects contains only one object otherwise go to Step 1.

For a detailed example of the neighbor joining algorithm see Appendix C.

Neighbor joining is an $O(N^3)$ algorithm and is useful for larger data sets. The formula for Q_{ij} accounts for differences in evolution rates. The objective S (sum of all branch lengths) is not minimized directly.

ClustalW uses neighbor-joining as guide for multiple alignments.

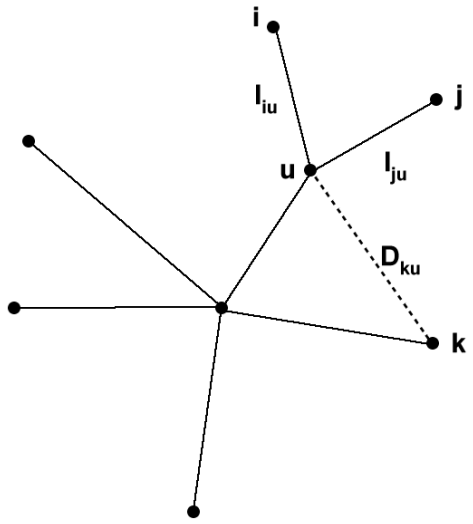


Figure 5.18: Leaves i and j are joined to new leaf u.

5.3.5 Distance Measures

Here we focus on distances between nucleotides.

The basis for deriving distances is an assumption on the substitution rates between nucleotides.

5.3.5.1 Jukes Cantor

The probability of a mutation is

$$r \;=\; 3\alpha\,. \tag{5.40}$$

Given two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$, the probability that an identical position remains identical in the next step is the product that the position does not mutate in both sequences:

$$(1 \;-\; r)^2 \;\approx\; 1 \;-\; 2\,r\,. \tag{5.41}$$

The probability that a position with different nucleotides will have the same in the next step is

$$\frac{2\,r}{3}\,. \tag{5.42}$$

This value is obtained because either the nucleotide in $\boldsymbol{x}$ or the nucleotide in $\boldsymbol{y}$ may change and the other remains constant: $\alpha\,(1 \;-\; r) \;=\; \frac{r}{3}\,(1 \;-\; r)$. Two of these events exist, therefore we obtain: $\frac{2\,r}{3}\,(1 \;-\; r) \;\approx\; \frac{2\,r}{3}$. We now can formulate a difference equation:

$$q_{t+t} \;=\; (1 \;-\; 2\,r)\;q_t \;+\; \frac{2\,r}{3}\,(1 \;-\; q_t) \tag{5.43}$$

	A	T	C	G
		Jukes Cantor		
A		α	α	α
T	α		α	α
C	α	α		α
G	α	α	α	
		Kimura		
A		β	β	α
T	β		α	β
C	β	α		β
G	α	β	β	
		Felsenstein / Tajima-Nei		
A		$\alpha\, g_T$	$\alpha\, g_C$	$\alpha\, g_G$
T	$\alpha\, g_A$		$\alpha\, g_C$	$\alpha\, g_G$
C	$\alpha\, g_A$	$\alpha\, g_T$		$\alpha\, g_G$
G	$\alpha\, g_A$	$\alpha\, g_T$	$\alpha\, g_C$	
		Tamura		
A		$\beta\ (g_A + g_T)$	$\beta\ (g_G + g_C)$	$\alpha\ (g_G + g_C)$
T	$\beta\ (g_A + g_T)$		$\alpha\ (g_G + g_C)$	$\beta\ (g_G + g_C)$
C	$\beta\ (g_A + g_T)$	$\alpha\ (g_A + g_T)$		$\beta\ (g_G + g_C)$
G	$\alpha\ (g_A + g_T)$	$\beta\ (g_A + g_T)$	$\beta\ (g_G + g_C)$	
		Hasegawa		
A		$\beta\, g_T$	$\beta\, g_C$	$\alpha\, g_G$
T	$\beta\, g_A$		$\alpha\, g_C$	$\beta\, g_G$
C	$\beta\, g_A$	$\alpha\, g_T$		$\beta\, g_G$
G	$\alpha\, g_A$	$\beta\, g_T$	$\beta\, g_C$	
		Tamura-Nei		
A		$\beta\, g_T$	$\beta\, g_C$	$\alpha_{AG}\, g_G$
T	$\beta\, g_A$		$\alpha_{TC}\, g_C$	$\beta\, g_G$
C	$\beta\, g_A$	$\alpha_{TC}\, g_T$		$\beta\, g_G$
G	$\alpha_{AG}\, g_A$	$\beta\, g_T$	$\beta\, g_C$	
		Reversible		
A		$\alpha_{AT}\, g_T$	$\alpha_{AC}\, g_C$	$\alpha_{AG}\, g_G$
T	$\alpha_{AT}\, g_A$		$\alpha_{TC}\, g_C$	$\alpha_{TG}\, g_G$
C	$\alpha_{AC}\, g_A$	$\alpha_{TC}\, g_T$		$\alpha_{CG}\, g_G$
G	$\alpha_{AG}\, g_A$	$\alpha_{TG}\, g_T$	$\alpha_{CG}\, g_C$	
		General		
A		a_{12}	a_{13}	a_{14}
T	a_{21}		a_{23}	a_{24}
C	a_{31}	a_{32}		a_{34}
G	a_{41}	a_{42}	a_{43}	

Table 5.1: Different models of nucleotide substitution. A matrix entry a_{ij} is the substitution rate from the nucleotide in the i-th row to the nucleotide in the j-th column. g_A, g_T, g_C, and g_G are the nucleotide frequencies.

for the proportion of identical nucleotides q_t at time step t. This is

$$q_{t+t} \; - \; q_t \; = \; \frac{2\,r}{3} \; - \; \frac{8\,r}{3}\,q_t \tag{5.44}$$

and leads in a continuous model to

$$\dot{q} \; = \; \frac{2\,r}{3} \; - \; \frac{8\,r}{3}\,q \tag{5.45}$$

with the solution ($q(0) = 1$)

$$q(t) \; = \; 1 \; - \; \frac{3}{4}\left(1 \; - \; \exp\left(-\frac{8\,r\,t}{3}\right)\right). \tag{5.46}$$

The substitution per position, denoted by d, for two sequences is $2rt$ ($2r$ is the approximated probability of a change in one time step) leading to

$$d \; = \; -\frac{3}{4}\ln\left(1 \; - \; \frac{3}{4}p\right) \; , \; \; p \; = \; 1 \; - \; q\,. \tag{5.47}$$

Estimating q by $\hat{q}$ (the observed identical nucleotides) and inserting the estimated value in above equation leads to the estimate $\hat{d}$ of d.

The variance of $\hat{d}$ is

$$\mathrm{Var}(\hat{d}) \; = \; \frac{9p\,(1 \; - \; p)}{(3 \; - \; 4p)^2\,n}\,. \tag{5.48}$$

5.3.5.2 Kimura

In contrast to previous model, r must be changed to

$$r \; = \; \alpha \; + \; 2\beta\,. \tag{5.49}$$

Let us group nucleotide pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ as follows:

$$\begin{aligned} P \; &= \; \{\mathrm{AG}, \mathrm{GA}, \mathrm{TC}, \mathrm{CT}\} \\ Q \; &= \; \{\mathrm{AT}, \mathrm{TA}, \mathrm{AC}, \mathrm{CATG}, \mathrm{GT}, \mathrm{CG}, \mathrm{GC}\} \end{aligned} \tag{5.50}$$

We obtain

$$\begin{aligned} P \; &= \; \frac{1}{4}\left(1 \; - \; 2\,\exp\left(-4\,(\alpha \; + \; \beta)\,t\right) \; + \; \exp\left(-8\,\beta\,t\right)\right) \\ Q \; &= \; \frac{1}{2}\left(1 \; - \; \exp\left(-8\,\beta\,t\right)\right) \end{aligned} \tag{5.51}$$

leading to

$$d = 2\,r\,t = 2\,\alpha\,t + 4\,\beta\,t = \tag{5.52}$$
$$-\frac{1}{2}\,\ln(1 - 2\,P - Q) - \frac{1}{2}\,\ln(1 - 2\,Q)\ .$$

The variance is given by

$$\mathrm{Var}(\hat{d}) = \frac{1}{n}\left(c_1^2\,P + c_2^2\,Q - (c_1\,P + c_2\,Q)^2\right) \tag{5.53}$$
$$c_1 = (1 - 2\,P - Q)^{-1}$$
$$c_2 = \frac{1}{2}\left((1 - 2\,P - Q)^{-1} + (1 - 2\,Q)^{-1}\right)\ .$$

Advantage of this model is that transitional ($2\ \alpha\ t$) and transversional ($4\ \beta\ t$) nucleotide substitutions can be estimated.

The equilibrium frequency of each nucleotide is 0.25, which is not in agreement with the observations. For example the occurrence of GC (θ) differs from 0.5 as in *Drosophila* mitochondrial DNA with 0.1.

5.3.5.3 Felsenstein / Tajima-Nei

Using x_{ij} for the relative frequency of nucleotide pair (i, j) we define

$$b = \frac{1}{2}\left(1 - \sum_{i=1}^{4} g_i^2 + \frac{p^2}{c}\right) \tag{5.54}$$
$$c = \sum_{i=1}^{3}\sum_{j=i+1}^{4} \frac{x_{ij}^2}{2\,g_i\,g_j}\ .$$

The number of nucleotide substitutions and its variance is given by

$$d = -b\,\ln\left(1 - \frac{p}{b}\right) \tag{5.55}$$
$$\mathrm{Var}(\hat{d}) = \frac{b^2\,p\,(1 - p)}{(b - b)^2 n}\ . \tag{5.56}$$

5.3.5.4 Tamura

This model extends Kimura's model for GC content θ different from 0.5.

The number of nucleotide substitutions is

$$d = -h\,\ln\left(1 - \frac{P}{h} - Q\right) - \frac{1}{2}\,(1 - h)\,\ln(1 - 2\,Q) \tag{5.57}$$
$$h = 2\,\theta\,(1 - \theta)\ .$$

5.3.5.5 Hasegawa (HKY)

Hasegawa's model (HKY) is a hybrid of Kimuras's model and the model of Felsenstein / Tajima-Nei. It accounts for GC content and for transition / transversion.

5.3.5.6 Tamura-Nei

This model includes Hasegawa's model.

The expression for d is more complicated than in previous models.

First we define

$$g_R \;=\; g_A \;+\; g_G \tag{5.58}$$

$$g_Y \;=\; g_T \;+\; g_C \tag{5.59}$$

$$c_1 \;=\; \frac{2\, g_A\, g_G}{g_R} \tag{5.60}$$

$$c_2 \;=\; \frac{2\, g_T\, g_C}{g_Y} \tag{5.61}$$

and obtain

$$\begin{aligned} d \;=\; & -\, c_1 \ln\Big(1 \;-\; c_1^{-1}\, P_1 \;-\; (2\, g_R)^{-1}\; Q\Big) \;- \\ & c_2 \ln\Big(1 \;-\; c_2^{-1}\, P_2 \;-\; (2\, g_Y)^{-1}\; Q\Big) \;- \\ & (2\, g_R\, g_Y \;-\; c_1\, g_Y \;-\; c_2\, g_R)\, \ln\Big(1 \;-\; (2\, g_R\, g_Y)^{-1}\, Q\Big)\;. \end{aligned} \tag{5.62}$$

P_1 is the proportion of transitional differences between A and G. P_2 is the proportion of transitional differences between T and C. Q is the proportion of transversional differences.

5.4 Maximum Likelihood Methods

The probability of the tree is the product of the mutation rates in each branch. The mutation rate is the product between the substitution rate and the branch length. Models for the substitution rate are given in Tab. 5.1.

The data $\boldsymbol{D}$ is the multiple alignment of the sequences from the N taxa. $\boldsymbol{D}_k$ is the N-dimensional vector at position k of the multiple alignment. We are given a tree with topology $\boldsymbol{A}$ and branch length $\boldsymbol{l}$. Let H be the number of hidden nodes of a given topology $\boldsymbol{A}$. Hidden nodes are indexed from 1 to H and taxa are indexed from $H+1$ to $H+N$. Without loss of generality, the root node is indexed by 1. Further we assume a model $\mathcal{M}$ for the nucleotide substitution. Let $\mathcal{A}$ be the set of letters. The likelihood of the tree at the k-th position is

$$\begin{aligned} & L(\boldsymbol{D}_k \mid \boldsymbol{l}, \boldsymbol{A}, \mathcal{M}) \;= \\ & \sum_{h=1}^{H} \sum_{a_h \in \mathcal{A}} P_r\,(a_1) \prod_{i,j;\; 1 \le i \le H,\; i < j \le N+H,\; A_{ij}=1} P_{a_i a_j}\,(l_{ij})\;, \end{aligned} \tag{5.63}$$

where $P_r\,(a_1)$ is the prior probability of the root node assigned with letter $a_1\ in\mathcal{A}$. The expression $A_{ij}\ =\ 1$ indicates an existing branch $i\ \rightarrow\ j$ in the topology $\boldsymbol{A}$. $P_{a_i a_j}\,(l_{ij})$ is the probability of branch length l_{ij} between nucleotides a_i and a_j. The hidden states are summed out in above likelihood expression. The prior is obtained from the nucleotide frequencies or is estimated.

If $\mathcal{M}$ is the Felsenstein / Tajima-Nei equal-input model, the branch length probabilities are

$$P_{a_i a_i}\,(l_{ii})\ =\ g_{a_i}\ +\ (1\ -\ g_{a_i})\ e^{-l_{ii}} \tag{5.64}$$

$$P_{a_i a_j}\,(l_{ij})\ =\ g_{a_j}\left(1\ -\ e^{-l_{ij}}\right)\ . \tag{5.65}$$

For $g_{a_i}\ =\ \frac{1}{4}$ and $l_{ij}\ =\ 4rt$ we obtain the Jukes-Cantor model from above, where $P_{a_i a_i}$ was denoted by q.

For reversible models

$$g_{a_i}\ P_{a_i a_j}\,(l)\ =\ g_{a_j}\ P_{a_j a_i}\,(l) \tag{5.66}$$

the choice of the root does not matter because branch lengths count independent of their substitution direction.

Under the assumption that all positions are independent from each other the likelihood is given by

$$L(\boldsymbol{D}\mid \boldsymbol{l}, \boldsymbol{A}, \mathcal{M})\ =\ \prod_k L(\boldsymbol{D}_k\mid \boldsymbol{l}, \boldsymbol{A}, \mathcal{M})\ . \tag{5.67}$$

The likelihood $L(\boldsymbol{D}_k\ \mid\ \boldsymbol{l}, \boldsymbol{A}, \mathcal{M})$ for a position k can be computed via Felsenstein's (1981) pruning algorithm.

$P_i(a)\ =\ P_i\,(a\mid \boldsymbol{D}_k, \boldsymbol{l}, \boldsymbol{A}, \mathcal{M})$ denotes the probability of a letter a at node i given data $\boldsymbol{D}_k$, branch length $\boldsymbol{l}$, topology $\boldsymbol{A}$ and model $\mathcal{M}$. We obtain the recursive formula

$$P_i(a_i)\ =\ \delta_{a_i\ D_{k(i-H)}}\ \text{ for }\ i > H(i\ \text{ taxa}) \tag{5.68}$$

$$P_i(a_i)\ =\ \prod_{j;\ A_{ij}=1}\left(\sum_{a_j\in\mathcal{A}} P_{a_i a_j}\,(l_{ij})\ \ P_j\,(a_j)\right)\ \text{ for }\ i \leq H(i\ \text{hidden})\ , \tag{5.69}$$

where $\delta_{a\ b}$ is 1 for $a\ =\ b$ and 0 otherwise. Using the idea of dynamic programming the values $P_i(a)$ can be computed from the taxa (leaves) to the root, where zero values are not propagated further.

The Likelihood can be computed as

$$L(\boldsymbol{D}_k\mid \boldsymbol{l}, \boldsymbol{A}, \mathcal{M})\ =\ \sum_{a_1\in\mathcal{A}} P_r\,(a_1)\,P_1\,(a_1)\ . \tag{5.70}$$

To find the best tree both the branch length and the topology must be optimized.

To optimize the branch length the likelihood can be optimized directly via gradient based approaches like the Newton method. Alternatively an EM (expectation-maximization, Dempster et al., 1977) algorithm can be used, which maximizes a lower bound on the likelihood.

For the optimization of the tree topology Felsenstein (1981) applied a growing (constructive) algorithm (cf. neural networks: cascade correlation). Start with 3 taxa and for the k-th taxa test all $(2k-5)$ branches as insertion point. The tree is then further optimized by local changing the topology.

For small N all topologies can be tested. Also local changes similar to the parsimony tree optimization can be applied.

The ML estimator is computationally expensive. A fast heuristic was introduced by Strimmer and v. Haeseler (1996) which optimizes all topologies of 4 taxa and then constructs the final tree (software: `http://www.tree-puzzle.de/`).

The ML estimation is unbiased, i.e. for the sequence length going to infinity (asymptotically) the ML tree is the true tree. The ML method is asymptotically efficient, i.e. the estimator with minimal variance (variance of the solutions if the same number of examples are randomly drawn).

5.5 Examples

In this experiment we compare triosephosphate isomerase of different species in order to construct a phylogenetic tree. We used the PHYLIP (Phylogeny Inference Package) Version 3.5c for constructing the trees.

In the first experiment we construct a phylogenetic tree for:

EColi	Escherichia coli	Bacterium
VibMar	Vibrio marinus	Bacterium
Chicken	Gallus gallus	Animal
Human	Homo sapiens	Animal
Nematode	Caenorhabditis elegans	Worm
Yeast	Saccharomyces cerevisiae	Yeast
Pfalcip	Plasmodium falciparum	single cell
Amoeba	Entamoeba histolytica	single cell
TBrucei	Trypanosoma brucei	single cell
TCruzi	Trypanosoma cruzi	single cell
LeiMex	Leishmania mexicana	single cell
Bacillus	Bacillus stearothermophilus	Bacterium
ThMar	Thermotoga maritima	Bacterium
Archaeon	Pyrococcus woesei	Archaeon

Figures 5.19, 5.20, 5.21, and 5.22 show the results.

In the second experiment we construct a phylogenetic tree again based on triosephosphat isomerase for: Human, Monkey, Mouse, Rat, Cow, Pig, Goose, Chicken, Zebrafish, Fruit FLY, Rye, Rice, Corn, Soybean, Bacterium. Figures 5.23, 5.24, 5.25, and 5.26 show the results.

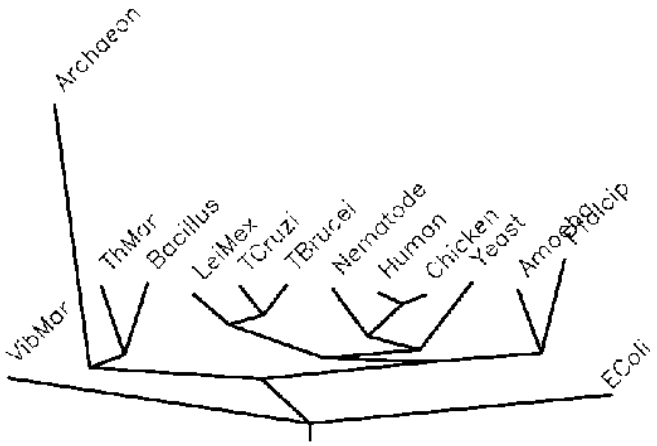


Figure 5.19: The Fitch-Margoliash method for constructing a phylogenetic tree for the taxa of experiment 1.

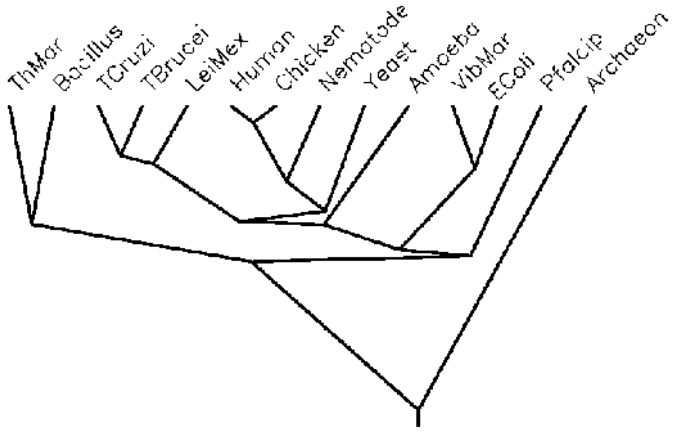


Figure 5.20: The Fitch-Margoliash method under the assumption of molecular clock (“kitsch”) for constructing a phylogenetic tree for the taxa of experiment 1.

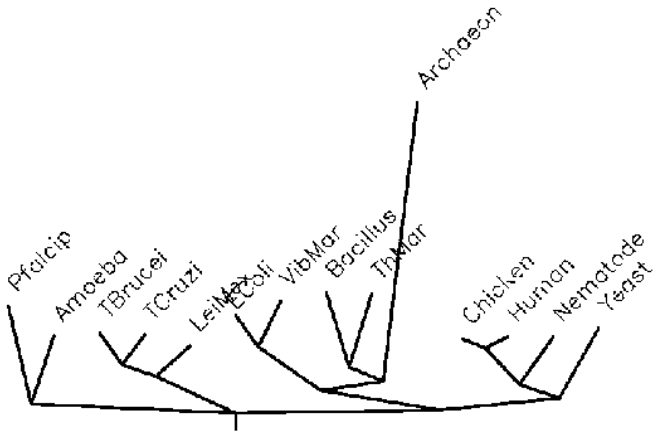


Figure 5.21: The neighbor joining method for constructing a phylogenetic tree for the taxa of experiment 1.

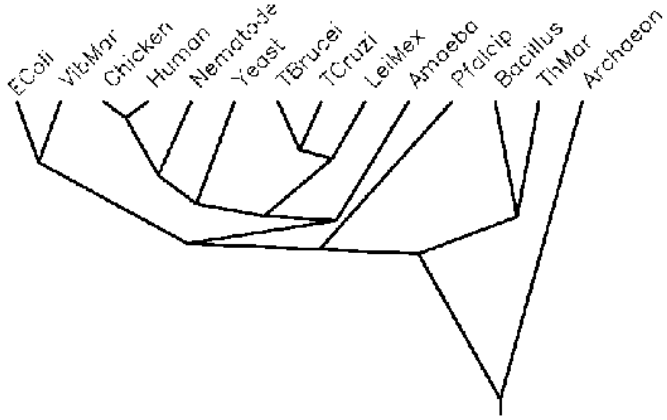


Figure 5.22: The UPGMA method for constructing a phylogenetic tree for the taxa of experiment 1.

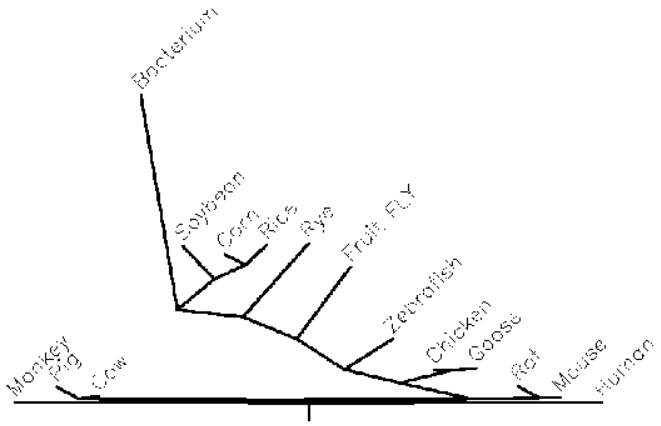


Figure 5.23: The Fitch-Margoliash method for constructing a phylogenetic tree for the taxa of experiment 2.

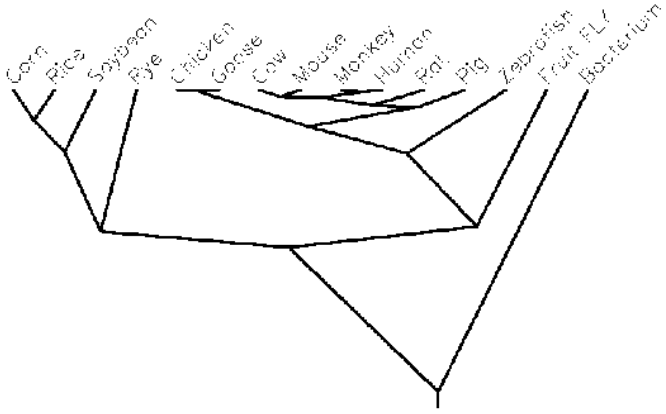


Figure 5.24: The Fitch-Margoliash method under the assumption of molecular clock (“kitsch”) for constructing a phylogenetic tree for the taxa of experiment 2.

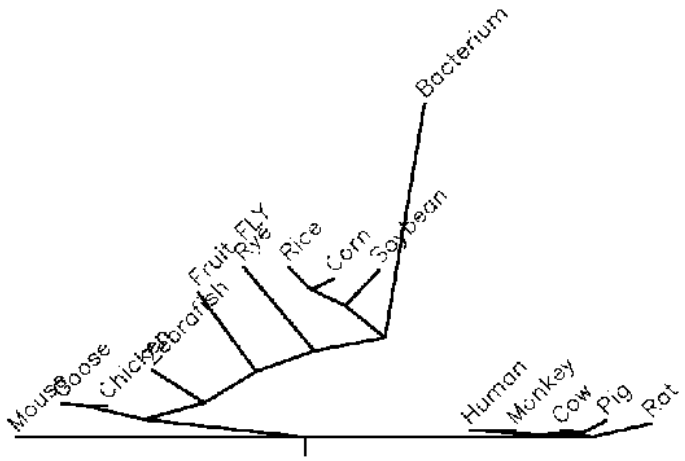


Figure 5.25: The neighbor joining method for constructing a phylogenetic tree for the taxa of experiment 2.

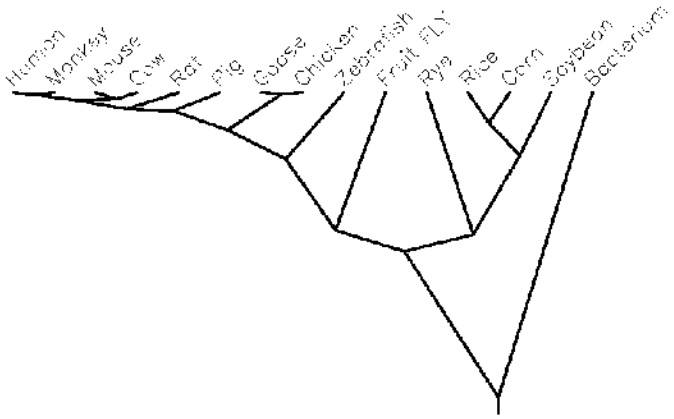


Figure 5.26: The UPGMA method for constructing a phylogenetic tree for the taxa of experiment 2.

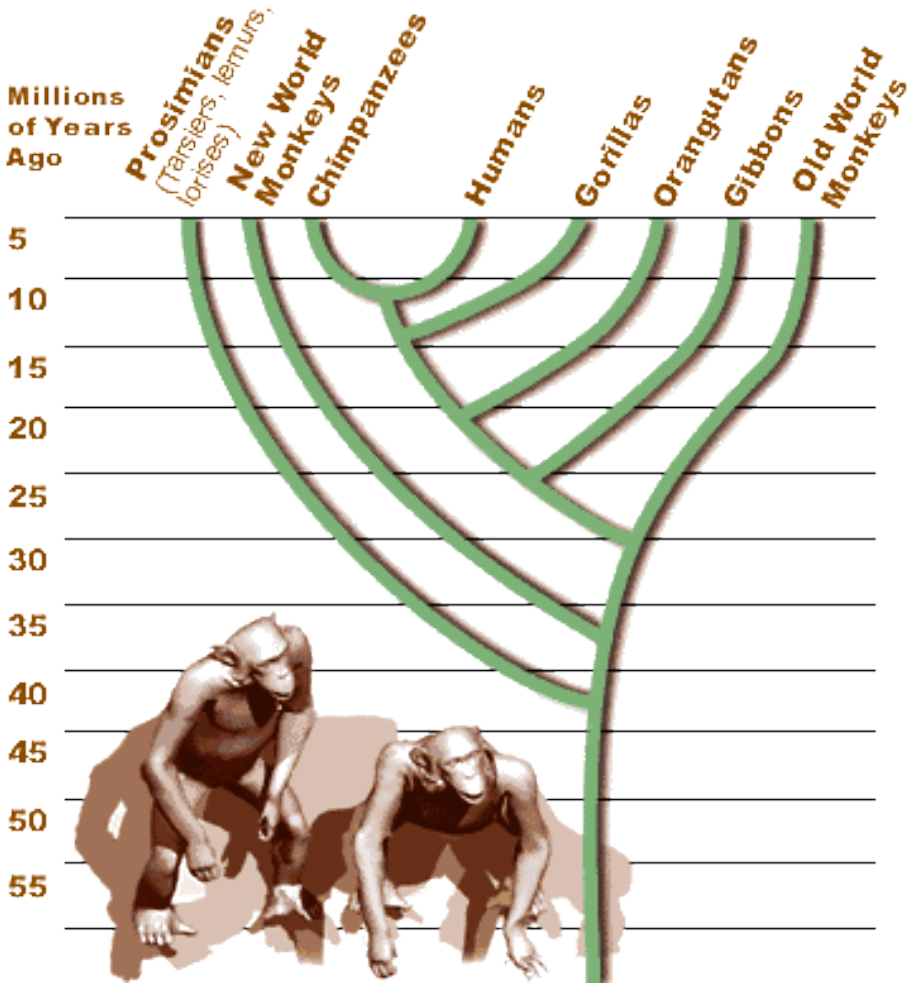


Figure 5.27: Relation humans, chimpanzees, gorillas, oran utans and gibbons (part 1).

An interesting scientific research focus on the relationship between humans and apes. Figures 5.27 and 5.28 show the latest results on this topic.

Finally a phylogenetic tree from a very special point of view is given in Fig. 5.29.

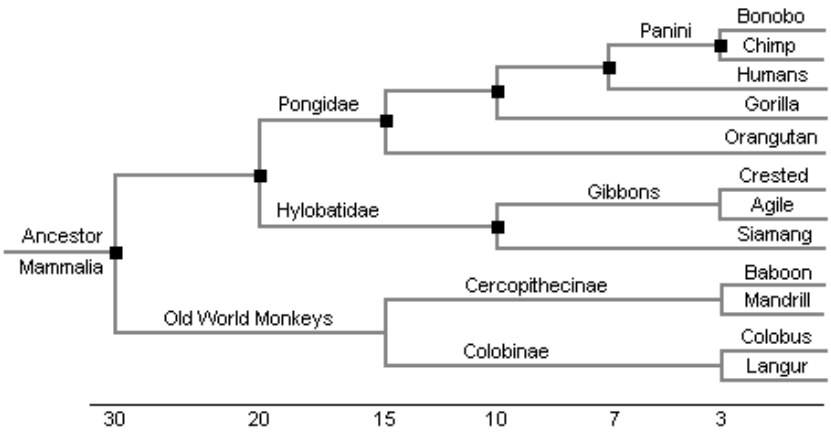


Figure 5.28: Relation humans, chimpanzees, gorillas and oran utans (part 2).

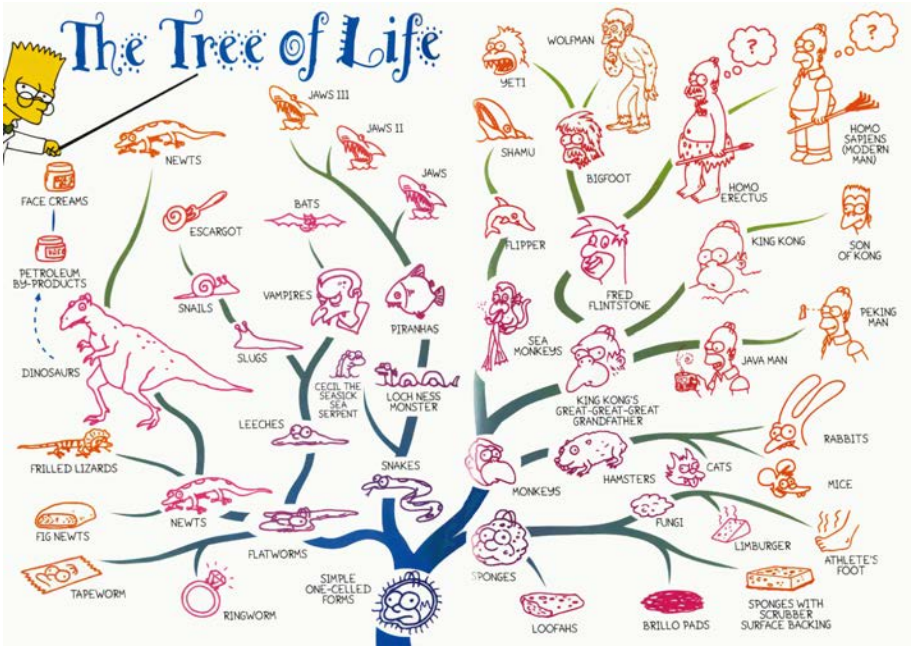


Figure 5.29: Tree of life from a special perspective.

Appendix A

Amino Acid Characteristics

Amino acid features are listed. The mass, surface area, volume, pKa (acid-ionization constant, indicates the extent of dissociation of hydrogen ions from an acid), pI (isoelectric point, pH at which the molecule carries no electrical charge), solubility, density (see Tab. A.2) and solvent accessible area (see Tab. A.1) is given.

Amino Acid	SEA >30 Å^2	SEA <10 Å^2	30 Å^2 $>$ SEA >10 Å^2
S	0.70	0.20	0.10
T	0.71	0.16	0.13
A	0.48	0.35	0.17
G	0.51	0.36	0.13
P	0.78	0.13	0.09
C	0.32	0.54	0.14
D	0.81	0.09	0.10
E	0.93	0.04	0.03
Q	0.81	0.10	0.09
N	0.82	0.10	0.08
L	0.41	0.49	0.10
I	0.39	0.47	0.14
V	0.40	0.50	0.10
M	0.44	0.20	0.36
F	0.42	0.42	0.16
Y	0.67	0.20	0.13
W	0.49	0.44	0.07
K	0.93	0.02	0.05
R	0.84	0.05	0.11
H	0.66	0.19	0.15

Table A.1: Solvent accessibility of amino acids in known protein structures (D. Bordo and P. Argos, J. Mol. Biol. 217(1991)721-729). “SEA” means solvent exposed area.

	mass [dalton]	surface [Å2]	volume [Å3]	pKa [side ch.]	pI [25°C]	solubility [g/100g]	density [g/ml]
A	71.09	115	88.6	-	6.107	16.65	1.401
R	156.19	225	173.4	12	10.76	15	1.1
D	114.11	150	111.1	4.5	2.98	0.778	1.66
N	115.09	160	114.1	-	-	3.53	1.54
C	103.15	135	108.5	9.1-9.5	5.02	very high	-
E	129.12	190	138.4	4.6	3.08	0.864	1.460
Q	128.14	180	143.8	-	-	2.5	-
G	57.05	75	60.1	-	6.064	24.99	1.607
H	137.14	195	153.2	6.2	7.64	4.19	-
I	113.16	175	166.7	-	6.038	4.117	-
L	113.16	170	166.7	-	6.036	2.426	1.191
K	128.17	200	168.6	10.4	9.47	very high	-
M	131.19	185	162.9	-	5.74	3.381	1.340
F	147.18	210	189.9	-	5.91	2.965	-
P	97.12	145	112.7	-	6.3	162.3	-
S	87.08	115	89.0	-	5.68	5.023	1.537
T	101.11	140	116.1	-	-	very high	-
W	186.12	255	227.8	-	5.88	1.136	-
Y	163.18	230	193.6	9.7	5.63	0.0453	1.456
V	99.14	155	140.0	-	6.002	8.85	1.230

Table A.2: Chemical properties of amino acids. Given are surface (C.Chothia, J. Mol. Biol., 105(1975)1-14), volume (A.A. Zamyatin, Prog. Biophys. Mol. Biol., 24(1972)107-123), pKa (C. Tanford, Adv. Prot. Chem., 17(1962)69-165), and pI/solubility/density (The Merck Index, Merck & Co. Inc., Nahway, N.J., 11(1989); CRC Handbook of Chem.& Phys., Cleveland, Ohio, 58(1977)).

Appendix B

A^*-Algorithm

The A^*-algorithm can be used to improve the MSA. Bounds on the score of the projection of the optimal multiple alignment are used as approximative distance to the goal.

Algorithm B.1 A^*-algorithm.

Input: `graph` (the graph), `start` (start node), `goal` (goal node), `h(s)` approximation of the distance of node `s` to the goal, `S` (priority queue), `N` (list of visited nodes)
Output: list `P` of the shortest path

```
BEGIN FUNCTION
  insert (start,S)
  while not isEmpty(S) do
    current_node = pop(S)
    if current_node in N then {no path from start to goal}
      return "no path"
    end if
    insert (current_node, N)
    if current_node = goal then
      reconstruct_shortest_path(start,goal, graph)
    else {find all nodes accessible from current node}
      successors = expand(current_node, graph)
      save_predecessor_in_graph(current_node, graph)
      for all s in successors do {save nodes which lead to s}
        predecessor(s) = current_node {compute and store costs}
        cost(s) = cost(current_node) + edge(graph,current_node,s)
        all_cost(s) = cost(s) + h(s)
        insert(s,S) {according to all_cost(s)}
      end for
    end if
  end while
  return "no path found"
END FUNCTION

BEGIN SUBFUNCTION
 reconstruct_shortest_path (start, node, graph)  {shortest path P as list}
  if node not= start then
    push(node, P) {get predecessor}
    predecessor = getPredecessor(node, graph)
    reconstruct_shortest_path (start, predecessor, graph)
  else
    return P
  end if
END SUBFUNCTION
```

Appendix C

Examples

This chapter gives examples for some of the algorithms discussed in the lecture.

C.1 Pairwise Alignment

C.1.1 PAM Matrices

The PAM1 matrix is calculated as follows:

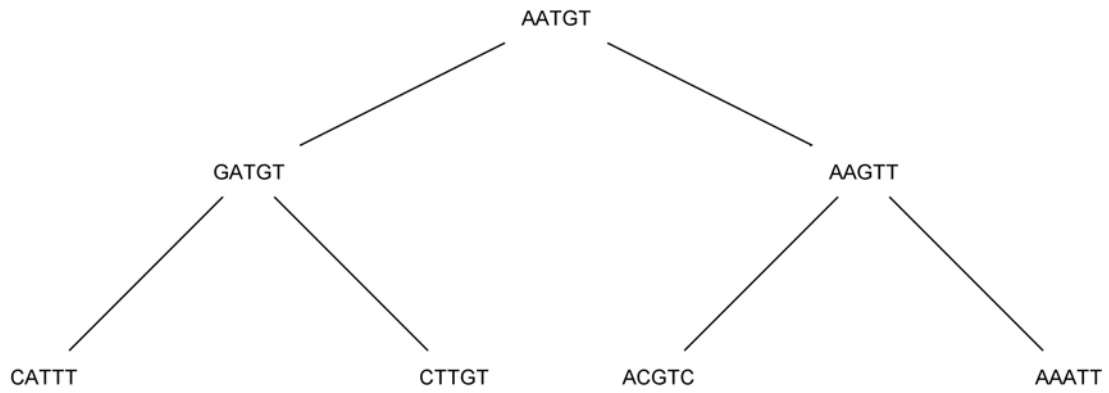


Figure C.1: Phylogenetic tree for the calculation of the PAM1 matrix.

1. Start at the top of the tree and count all transitions of amino acid i to j to get $C_{i,j}^{unsym}$:

$C_{i,j}^{unsym}$:

A -> C:	I	C -> A:	0
A -> G:	I	G -> A:	I
A -> T:	I	T -> A:	0
C -> G:	0	G -> C:	II
C -> T:	0	T -> C:	I
G -> T:	II	T -> G:	I

$C^{unsym} =$

	A	C	G	T
A	-	1	1	1
C	0	-	0	0
G	1	2	-	2
T	0	1	1	-

2. Symmetrize $C_{i,j}^{unsym}$ using the formula $C_{i,j}^{sym} = \frac{1}{2}\left(C_{i,j}^{unsym} + C_{j,i}^{unsym}\right)$:

$C^{sym} =$

	A	C	G	T
A	-	0.5	1	0.5
C	0.5	-	1	0.5
G	1	1	-	1.5
T	0.5	0.5	1.5	-

3. Determine the relative frequencies of the individual amino acids by dividing their number of occurrence in the sequences of the tree by the total number of amino acids in the tree:
   $f_A = \frac{\text{no. of As in all sequences}}{\text{length of sequences * no. of sequences}} = \frac{10}{35}$
   $f_C = \frac{4}{35}$
   $f_G = \frac{6}{35}$
   $f_T = \frac{15}{35}$

4. The PAM1 is obtained by calculating the values for $p_{i,j}$ using the formulas:
   $p_{i,j} = \frac{C_{i,j}}{100\ f_i\ \sum_k \sum_{l \neq k} C_{k,l}}$ and
   $p_{i,i} = 1 - \sum_{j \neq i} p_{i,j}$
   e.g.: $p_{AC} = \frac{C_{AC}}{100\ f_A 10}$
   $p_{AA} = 1 - (p_{AC} + p_{AG} + p_{AT})$

$PAM1 = p =$

	A	C	G	T
A	0.99300	0.00175	0.00350	0.00175
C	0.00438	0.98250	0.00875	0.00438
G	0.00583	0.00583	0.97958	0.00875
T	0.00117	0.00117	0.00350	0.99417

5. The odds scores are calculated by comparing the probability that amino acid i is present and mutates into amino acid j with the probability of randomly selecting amino acids i and j, which results in the following formula:

$$\frac{f_i\, p_{i,j}}{f_i\, f_j} = \frac{p_{i,j}}{f_j} = \frac{p_{j,i}}{f_i}$$

$$odds\ scores = \begin{array}{c|cccc} & \text{A} & \text{C} & \text{G} & \text{T} \\ \text{A} & 3.47550 & 0.01531 & 0.02047 & 0.00408 \\ \text{C} & 0.01531 & 8.59688 & 0.05104 & 0.01020 \\ \text{G} & 0.02047 & 0.05104 & 5.71424 & 0.02042 \\ \text{T} & 0.00408 & 0.01020 & 0.02042 & 2.31972 \end{array}$$

6. Finally we apply the base 2 logarithm to get the log-odds scores and round the resulting values to integers to get the PAM1 scoring matrix:

$$PAM1\ scoring\ matrix = \begin{array}{c|cccc} & \text{A} & \text{C} & \text{G} & \text{T} \\ \text{A} & 2 & -6 & -6 & -8 \\ \text{C} & -6 & 3 & -4 & -7 \\ \text{G} & -6 & -4 & 3 & -6 \\ \text{T} & -8 & -7 & -6 & 1 \end{array}$$

C.1.2 BLOSUM Matrices

We use the following sequences to calculate BLOSUM75:

V	I	I	L
V	I	I	I
L	I	V	V
L	L	V	I
V	L	L	I
V	L	L	L

1. For BLOSUM75 we need to cluster sequences with at least 75% identity which corresponds to 3 identical amino acids for sequences of length 4.

c_1	=	V	I	I	[0.5 L 0.5 I]
c_2	=	L	I	V	V
c_3	=	L	L	V	I
c_4	=	V	L	L	[0.5 I 0.5 L]

2. For each column k of the sequences we calculate the occuring pairs of amino acids i and j using

$$c_{i,j}^k = \begin{cases} \frac{1}{2}\left(\left(n_i^k\right)^2 - \sum_l \left(f_{i,l}^k\right)^2\right) & \text{for} \quad i = j \\ n_i^k \, n_j^k - \sum_l f_{i,l}^k \, f_{j,l}^k & \text{for} \quad i \neq j \end{cases}$$

where $f_{i,l}^k$ is the frequency of amino acid i in the k-th column for the l-th cluster and $n_i^k = \sum_l f_{i,l}^k$

For columns without frequencies we get:

$$c_{i,j}^k = \begin{cases} \binom{n_i^k}{2} & \text{for} \quad i = j \\ n_i^k n_j^k & \text{for} \quad i \neq j \end{cases}$$

For our sequences we get the following table:

k	II	IL	IV	LL	LV	VV
1	0	0	0	1	4	1
2	1	4	0	1	0	0
3	0	1	2	0	2	1
4	1.25	1.5	2	0.25	1	0

3. Next we compute $c_{i,j} = \sum_k c_{i,j}^k$ and $Z = \sum_{i,j \leq i} c_{i,j} = \frac{L \, N \, (N-1)}{2}$, where L is the sequence length (number of columns) and N the number of clusters.

$$c = \begin{array}{c|ccc} & \text{I} & \text{L} & \text{V} \\ \text{I} & 2.25 & - & - \\ \text{L} & 6.5 & 2.25 & - \\ \text{V} & 4 & 7 & 2 \end{array}$$

$Z = \frac{4 \cdot 4 \cdot 3}{2} = 24$

4. Now the $c_{i,j}$ are normalized to obtain the probability $q_{i,j} = \frac{c_{i,j}}{Z}$ and we set $q_{j,i} = q_{i,j}$ for $i > j$.

$$q = \begin{array}{c|ccc} & \text{I} & \text{L} & \text{V} \\ \text{I} & 0.094 & 0.271 & 0.167 \\ \text{L} & 0.271 & 0.094 & 0.292 \\ \text{V} & 0.167 & 0.292 & 0.083 \end{array}$$

The probability of the occurrence of amino acid i is

$q_i = q_{i,i} + \sum_{j \neq i} \frac{q_{i,j}}{2}$,

the probability of i not being mutated plus the sum of the mutation probabilities.

$q_I = 0.3125$
$q_L = 0.375$
$q_V = 0.3125$

5. The log-odds ratios are calculated using

$$log - odds\ ratios = \begin{cases} 2 \log_2 \frac{q_{i,i}}{q_i^2} & \text{for} \quad i = j \\ 2 \log_2 \frac{q_{i,j}}{2\ q_i\ q_j} & \text{for} \quad i \neq j \end{cases}$$

$$log - odds\ ratios: \begin{array}{c|ccc} & \text{I} & \text{L} & \text{V} \\ \text{I} & -0.12 & 0.42 & -0.46 \\ \text{L} & 0.42 & -1.17 & 0.63 \\ \text{V} & -0.46 & 0.63 & -0.46 \end{array}$$

6. To get the BLOSUM scoring matrix, these values need to be rounded to integers.

$$BLOSUM75 = \begin{array}{c|ccc} & \text{I} & \text{L} & \text{V} \\ \text{I} & 0 & 0 & 0 \\ \text{L} & 0 & -1 & 1 \\ \text{V} & 0 & 1 & 0 \end{array}$$

C.1.3 Global Alignment — Needleman-Wunsch

C.1.3.1 Linear Gap Penalty

Given the two sequences BAC and BABABC we want to compute all optimal global alignments using the ***Needleman-Wunsch*** algorithm with ***linear*** gap penalty.
We use a score of 2 for a match and -1 for a mismatch. The gap penalty $d = 2$.

We align the first sequence (sequence x) vertically and the second sequence (sequence y) on top. The initial score in the top left corner is set to 0. For initializing the rest of the first row and first column, we have to substract the gap penalty from the score for each step to the right or down. Arrows point at the element that was used for the computation.

S_{ij}	★		B		A		B		A		B		C
★	0	←	-2	←	-4	←	-6	←	-8	←	-10	←	-12
	↑												
B	-2												
	↑												
A	-4												
	↑												
C	-6												

Next we start filling the matrix row by row. For each element there are three possibilities to obtain a score:

1. a gap in sequence x: this value is calculated by substracting the gap penalty from the score of the element to the left, therefore it is marked by an arrow pointing to the left.
2. a gap in sequence y: this value is calculated by substracting the gap penalty from the score of the element above, therefore it is marked by an upward arrow.
3. no gap: this value is calculated by adding the respective match or mismatch score to the score of the element on the upper left diagonal and is therefore marked by a diagonal arrow.

We take the maximum of these three scores and use the respective arrow to mark how this value was obtained. All arrows leading to the maximal score are stored in order to get all optimal global alignments.

S_{ij}	★		B		A		B		A		B		C
★	0	←	-2	←	-4	←	-6	←	-8	←	-10	←	-12
	↑	↖				↖				↖			
B	-2		2	←	0	←	-2	←	-4	←	-6	←	-8
	↑												
A	-4												
	↑												
C	-6												

The calculations are continued row by row and lead to the following matrix:

S_{ij}	★		B		A		B		A		B		C
★	0	←	-2	←	-4	←	-6	←	-8	←	-10	←	-12
	↑	↖				↖				↖			
B	-2		2	←	0	←	-2	←	-4	←	-6	←	-8
	↑		↑	↖				↖					
A	-4		0		4	←	2	←	0	←	-2	←	-4
	↑		↑		↑	↖		↖		↖		↖	
C	-6		-2		2		3	←	1	←	-1		0

The optimal score of the global alignment can be found in the bottom right corner. Using the arrows we can backtrack the according alignments starting at that position.

S_{ij}	★		B		A		B		A		B		C
★	0	←	-2	←	-4	←	-6	←	-8	←	-10	←	-12
	↑	**↖**				↖				↖			
B	-2		2	←	0	←	-2	←	-4	←	-6	←	-8
	↑		↑	**↖**				↖					
A	-4		0		4	**←**	2	**←**	0	**←**	-2	←	-4
	↑		↑		↑	↖		↖		↖		**↖**	
C	-6		-2		2		3	←	1	←	-1		0

S_{ij}	★		B		A		B		A		B		C
★	0	←	-2	←	-4	←	-6	←	-8	←	-10	←	-12
	↑	**↖**				↖				↖			
B	-2		2	**←**	0	**←**	-2	←	-4	←	-6	←	-8
	↑		↑	↖				**↖**					
A	-4		0		4	←	2	←	0	**←**	-2	←	-4
	↑		↑		↑	↖		↖		↖		**↖**	
C	-6		-2		2		3	←	1	←	-1		0

S_{ij}	★		B		A		B		A		B		C
★	0	**←**	-2	**←**	-4	←	-6	←	-8	←	-10	←	-12
	↑	↖				**↖**				↖			
B	-2		2	←	0	←	-2	←	-4	←	-6	←	-8
	↑		↑	↖				**↖**					
A	-4		0		4	←	2	←	0	**←**	-2	←	-4
	↑		↑		↑	↖		↖		↖		**↖**	
C	-6		-2		2		3	←	1	←	-1		0

We start by aligning the last character of the alignment and go backwards. At a diagonal arrow we write the character of the current position of both sequences, an arrow pointing to the left creates a gap in sequence x, and the upward arrow creates a gap in sequence y. Each fork creates a new alignment, where we first copy the alignment we got so far. This leads to the following three optimal alignments with a score of 0:

Alignment 1:

↖	↖	←	←	←	↖
B	A	-	-	-	C
B	A	B	A	B	C

Alignment 2:

↖	←	←	↖	←	↖
B	-	-	A	-	C
B	A	B	A	B	C

Alignment 3:

←	←	↖	↖	←	↖
-	-	B	A	-	C
B	A	B	A	B	C

C.1.3.2 Affine Gap Penalty

Given the two sequences BAC and BABABC we want to compute all optimal global alignments using the ***Needleman-Wunsch*** algorithm with ***affine*** gap penalty.
We use the identity matrix as scoring matrix i.e. a scoring of 1 for matches and a scoring of 0 for mismatches. The gap open penalty $d = 2$, whereas the gap extension penalty $e = 1$.

Again, we align the first sequence (sequence x) vertically and the second sequence (sequence y) on top. This time we need three matrices:

- $G_d(i, j)$: best score up to position (i, j) and no gap at the end
- $G_x(i, j)$: best score up to position (i, j) with a gap in sequence $\boldsymbol{x}$ at position i
- $G_y(i, j)$: best score up to position (i, j) with a gap in sequence $\boldsymbol{y}$ at position j

In each of the matrices only one kind of step is possible, therefore we only have to look, whether staying in the same matrix gives a higher score than coming from a different matrix. This means that for the G_d matrix we have to look for the highest value in all three matrices whereas for the other matrices we have to consider, whether we want to use the G_d score while introducing a new gap (gap open penalty), or stay in the same matrix and extend the gap (gap extension penalty). We have to remember that we do not want to have a gap in sequence x followed by a gap in sequence y, or the other way around. Therefore we only need to consider matrices G_d and G_x for calculating G_x and analogously G_d and G_y for calculating G_y.

The initial score in the top left corner of the G_d matrix is again set to 0. The remaining first row and first column, as well as the first column of the G_x matrix and the first row of the G_y matrix are initialized with $-\infty$. The rest of the initialization is done using the 0 in the G_d and a gap open penalty in the first step followed by gap extension penalties.

G_d	★	B	A	B	A	B	C
★	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
B	$-\infty$						
A	$-\infty$						
C	$-\infty$						

G_x	★		B		A		B		A		B		C
★	$-\infty$	$\leftarrow^d$	-2	$\leftarrow^x$	-3	$\leftarrow^x$	-4	$\leftarrow^x$	-5	$\leftarrow^x$	-6	$\leftarrow^x$	-7
B	$-\infty$												
A	$-\infty$												
C	$-\infty$												

G_y	★	B	A	B	A	B	C
★	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^{d}$						
B	-2						
	$\uparrow^{y}$						
A	-3						
	$\uparrow^{y}$						
C	-4						

Next we start filling the second row of all three matrices. This time we need to store which matrix was used to calculate the best score. Again, it is possible to store more than one matrix symbol if needed.

G_d	★	B	A	B	A	B	C
★	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\nwarrow^{d}$	$\nwarrow^{x}$	$\nwarrow^{x}$	$\nwarrow^{x}$	$\nwarrow^{x}$	$\nwarrow^{x}$	
B	$-\infty$	1	-2	-2	-4	-4	-6
A	$-\infty$						
C	$-\infty$						

G_x	★	B	A	B	A	B	C
★	$-\infty$	$\leftarrow^{d}$ -2	$\leftarrow^{x}$ -3	$\leftarrow^{x}$ -4	$\leftarrow^{x}$ -5	$\leftarrow^{x}$ -6	$\leftarrow^{x}$ -7
B	$-\infty$	$-\infty$	$\leftarrow^{d}$ -1	$\leftarrow^{x}$ -2	$\leftarrow^{x}$ -3	$\leftarrow^{x}$ -4	$\leftarrow^{x}$ -5
A	$-\infty$						
C	$-\infty$						

G_y	★	B	A	B	A	B	C
★	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^{d}$						
B	-2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^{y}$						
A	-3						
	$\uparrow^{y}$						
C	-4						

The calculations are continued row by row and lead to the following matrices:

G_d	★		B		A		B		A		B		C
★	0		$-\infty$		$-\infty$		$-\infty$		$-\infty$		$-\infty$		$-\infty$
		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$	
B	$-\infty$		1		-2		-2		-4		-4		-6
		$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$	
A	$-\infty$		-2		2		-1		-1		-3		-4
		$\nwarrow^{y}$		$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$		$\nwarrow^{x}$	
C	$-\infty$		-3		-1		2		0		-1		-1

G_x	★		B		A		B		A		B		C
★	$-\infty$	$\leftarrow^{d}$	-2	$\leftarrow^{x}$	-3	$\leftarrow^{x}$	-4	$\leftarrow^{x}$	-5	$\leftarrow^{x}$	-6	$\leftarrow^{x}$	-7
B	$-\infty$		$-\infty$	$\leftarrow^{d}$	-1	$\leftarrow^{x}$	-2	$\leftarrow^{x}$	-3	$\leftarrow^{x}$	-4	$\leftarrow^{x}$	-5
A	$-\infty$		$-\infty$	$\leftarrow^{d}$	-4	$\leftarrow^{d}$	0	$\leftarrow^{x}$	-1	$\leftarrow^{x}$	-2	$\leftarrow^{x}$	-3
C	$-\infty$		$-\infty$	$\leftarrow^{d}$	-5	$\leftarrow^{d}$	-3	$\leftarrow^{d}$	0	$\leftarrow^{x}$	-1	$\leftarrow^{x}$	-2

G_y	★	B	A	B	A	B	C
★	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^{d}$						
B	-2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^{y}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$
A	-3	-1	-4	-4	-6	-6	-8
	$\uparrow^{y}$	$\uparrow^{y}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$	$\uparrow^{d}$
C	-4	-2	0	-3	-3	-5	-6

The optimal score of the global alignment is the maximum of the scores in the bottom right corner of all three matrices. This is where the backtracking starts. Again, we use the arrows to backtrack the according alignments and change matrices according to the stored symbols.

G_d	★		B		A		B		A		B		C
★	0		$-\infty$		$-\infty$		$-\infty$		$-\infty$		$-\infty$		$-\infty$
		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$		$\nwarrow^{x}$	
B	$-\infty$		1		-2		-2		-4		-4		-6
		$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$	
A	$-\infty$		-2		2		-1		-1		-3		-4
		$\nwarrow^{y}$		$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{d,x}$		$\nwarrow^{x}$	
C	$-\infty$		-3		-1		2		0		-1		-1

G_x	★		B		A		B		A		B		C
★	$-\infty$	$\leftarrow^d$	-2	$\leftarrow^x$	-3	$\leftarrow^x$	-4	$\leftarrow^x$	-5	$\leftarrow^x$	-6	$\leftarrow^x$	-7
B	$-\infty$		$-\infty$	$\leftarrow^d$	-1	$\leftarrow^x$	-2	$\leftarrow^x$	-3	$\leftarrow^x$	-4	$\leftarrow^x$	-5
A	$-\infty$		$-\infty$	$\leftarrow^d$	-4	$\leftarrow^d$	0	$\leftarrow^x$	-1	$\leftarrow^x$	-2	$\leftarrow^x$	-3
C	$-\infty$		$-\infty$	$\leftarrow^d$	-5	$\leftarrow^d$	-3	$\leftarrow^d$	0	$\leftarrow^x$	-1	$\leftarrow^x$	-2

G_y	★	B	A	B	A	B	C
★	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^d$						
B	-2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$\uparrow^y$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$
A	-3	-1	-4	-4	-6	-6	-8
	$\uparrow^y$	$\uparrow^y$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$	$\uparrow^d$
C	-4	-2	0	-3	-3	-5	-6

In our example the optimal score is -1 and the alignment ends without a gap because the maximum is found in the G_d matrix. We again start with the last characters of the alignment and go back. As above, we write the character of the current position of both sequences at a diagonal arrow (a step in the G_d matrix), an arrow pointing to the left (a step in the G_x matrix) creates a gap in sequence x, and the upward arrow (a step in the G_y matrix) creates a gap in sequence y. After each step we look whether we need to change the matrix according to the symbols above the arrows. If there is more than one symbol, we start a new alignment, where we first copy the alignment we got so far.

This gives one optimal alignment for our example:

$\nwarrow^d$	$\nwarrow^d$	$\leftarrow^d$	$\leftarrow^x$	$\leftarrow^x$	$\nwarrow^x$
B	A	-	-	-	C
B	A	B	A	B	C

C.1.4 Local Alignment — Smith-Waterman

C.1.4.1 Linear Gap Penalty

Given the two sequences CGAC and CGTTACT we want to compute all optimal local alignments using the ***Smith-Waterman*** algorithm with ***linear*** gap penalty. We use a score of 2 for a match and -1 for a mismatch. The gap penalty $d = 2$.

We align the first sequence (sequence x) vertically and the second sequence (sequence y) on top. In contrast to the Needleman-Wunsch algorithm, negative entries in the matrix are set to 0, where we mark the 0 with a star to distinguish it from a "normal" 0.

S_{ij}	★	C	G	T	T	A	C	T
★	0	0^*	0^*	0^*	0^*	0^*	0^*	0^*
C	0^*							
G	0^*							
A	0^*							
C	0^*							

Next we start filling the matrix row by row. For each element there are three possibilities to obtain a score:

1. a gap in sequence x: this value is calculated by substracting the gap penalty from the score of the element to the left, therefore it is marked by an arrow pointing to the left.
2. a gap in sequence y: this value is calculated by substracting the gap penalty from the score of the element above, therefore it is marked by an upward arrow.
3. no gap: this value is calculated by adding the respective match or mismatch score to the score of the element on the upper left diagonal and is therefore marked by a diagonal arrow.

We take the maximum of these three scores and use the respective arrow to mark how this value was obtained. All arrows leading to the maximal score are stored in order to get all optimal local alignments. If the maximal score is negative, it is set to 0^* and no arrow is needed.

S_{ij}	★		C		G	T	T	A		C		T
★	0		0^*		0^*	0^*	0^*	0^*		0^*		0^*
		↖							↖			
C	0^*		2	←	0	0^*	0^*	0^*		2	←	0
G	0^*											
A	0^*											
C	0^*											

The calculations are continued row by row and lead to the following matrix:

S_{ij}	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		↖										↖			
C	0*		2	←	0		0*		0*		0*		2	←	0
			↑	↖									↑	↖	
G	0*		0		4	←	2	←	0		0*		0		1
					↑	↖		↖		↖					
A	0*		0*		2		3	←	1		2	←	0		0*
		↖			↑	↖	↑	↖		↖	↑	↖			
C	0*		2	←	0		1		2	←	0		4	←	2

The optimal score of the local alignment is the maximal score found in the matrix. Using the arrows we can backtrack the according alignments starting at this position. If multiple entries have the same maximal score, each of them constitutes a new starting point.

Similar to the global alignment, we start by aligning the last character of the alignment and go backwards. At a diagonal arrow we write the character of the current position of both sequences, an arrow pointing to the left creates a gap in sequence x, and the upward arrow creates a gap in sequence y. Again, each fork creates a new alignment, where we first copy the alignment we got so far. The 0^* serves as a stop sign and therefore terminates the alignment. The "normal" 0 leads to two different alignments: one uses the 0 as a stop sign, the other continues the alignment.

The following three optimal alignments achieve a score of 4:

S_{ij}	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		↖										↖			
C	0*		2	←	0		0*		0*		0*		2	←	0
			↑	↖									↑	↖	
G	0*		0		4	←	2	←	0		0*		0		1
					↑	↖		↖		↖					
A	0*		0*		2		3	←	1		2	←	0		0*
		↖			↑	↖	↑	↖		↖	↑	↖			
C	0*		2	←	0		1		2	←	0		4	←	2

S_{ij}	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		↖										↖			
C	0*		2	←	0		0*		0*		0*		2	←	0
			↑	↖									↑	↖	
G	0*		0		4	←	2	←	0		0*		0		1
					↑	↖		↖		↖					
A	0*		0*		2		3	←	1		2	←	0		0*
		↖			↑	↖	↑	↖		↖	↑	↖			
C	0*		2	←	0		1		2	←	0		4	←	2

S_{ij}	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		↖										↖			
C	0*		2	←	0		0*		0*		0*		2	←	0
			↑	↖									↑	↖	
G	0*		0		4	←	2	←	0		0*		0		1
					↑	↖		↖		↖					
A	0*		0*		2		3	←	1		2	←	0		0*
		↖			↑	↖	↑	↖		↖	↑	↖			
C	0*		2	←	0		1		2	←	0		4	←	2

Alignment 1:

↖	↖
A	C
A	C

Alignment 2:

↖	↖	←	←	↖	↖
C	G	-	-	A	C
C	G	T	T	A	C

Alignment 3:

↖	↖
C	G
C	G

C.1.4.2 Affine Gap Penalty

Given the two sequences CGAC and CGTTACT we want to compute all optimal local alignments using the ***Smith-Waterman*** algorithm with ***affine*** gap penalty.
We use a score of 2 for a match and -1 for a mismatch. The gap open penalty $d = 2$, whereas the gap extension penalty $e = 1$.

Again, we align the first sequence (sequence x) vertically and the second sequence (sequence y) on top. This time we need three matrices:

- $G_d(i, j)$: best score up to position (i, j) and no gap at the end
- $G_x(i, j)$: best score up to position (i, j) with a gap in sequence $\boldsymbol{x}$ at position i
- $G_y(i, j)$: best score up to position (i, j) with a gap in sequence $\boldsymbol{y}$ at position j

In each of the matrices only one kind of step is possible, therefore we only have to look, whether staying in the same matrix gives a higher score than coming from a different matrix. This means that for the G_d matrix we have to look for the highest value in all three matrices whereas for the other matrices we have to consider, whether we want to use the G_d score while introducing a new gap (gap open penalty), or stay in the same matrix and extend the gap (gap extension penalty). We have to remember that we do not want to have a gap in sequence x followed by a gap in sequence y, or the other way around. Therefore we only need to consider matrices G_d and G_x for calculating G_x and analogously G_d and G_y for calculating G_y.

Again, all negative values are replaced by 0^*. Therefore, all three matrices are initialized with 0^*.

G_d	★	C	G	T	T	A	C	T
★	0	0^*	0^*	0^*	0^*	0^*	0^*	0^*
C	0^*							
G	0^*							
A	0^*							
C	0^*							

G_x	★	C	G	T	T	A	C	T
★	0^*	0^*	0^*	0^*	0^*	0^*	0^*	0^*
C	0^*							
G	0^*							
A	0^*							
C	0^*							

G_y	★	C	G	T	T	A	C	T
★	0*	0*	0*	0*	0*	0*	0*	0*
C	0*							
G	0*							
A	0*							
C	0*							

Next we start filling the second row of all three matrices. Again, we need to store which matrices were used to calculate the best score.

G_d	★		C	G	T	T	A		C	T
★	0		0*	0*	0*	0*	0*		0*	0*
		$\nwarrow^{d,x,y}$						$\nwarrow^{d,x,y}$		
C	0*		2	0*	0*	0*	0*		2	0*
G	0*									
A	0*									
C	0*									

G_x	★	C		G	T	T	A	C		T
★	0*	0*		0*	0*	0*	0*	0*		0*
C	0*	0*	$\leftarrow^{d}$	0	0*	0*	0*	0*	$\leftarrow^{d}$	0
G	0*									
A	0*									
C	0*									

G_y	★	C	G	T	T	A	C	T
★	0*	0*	0*	0*	0*	0*	0*	0*
C	0*	0*	0*	0*	0*	0*	0*	0*
G	0*							
A	0*							
C	0*							

The calculations are continued row by row and lead to the following matrices:

G_d	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		$\nwarrow^{d,x,y}$										$\nwarrow^{d,x,y}$			
C	0*		2		0*		0*		0*		0*		2		0*
				$\nwarrow^{d}$										$\nwarrow^{d}$	
G	0*		0*		4		0*		0*		0*		0*		1
						$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{x}$					
A	0*		0*		0*		3		1		3		0*		0*
		$\nwarrow^{d,x,y}$				$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{d,x}$		$\nwarrow^{d}$		$\nwarrow^{x}$	
C	0*		2		0*		1		2		0		5		0

G_x	★	C		G		T		T		A		C		T
★	0*	0*		0*		0*		0*		0*		0*		0*
C	0*	0*	$\leftarrow^{d}$	0		0*		0*		0*		0*	$\leftarrow^{d}$	0
G	0*	0*		0*	$\leftarrow^{d}$	2	$\leftarrow^{x}$	1	$\leftarrow^{x}$	0		0*		0*
A	0*	0*		0*		0*	$\leftarrow^{d}$	1	$\leftarrow^{x}$	0	$\leftarrow^{d}$	1	$\leftarrow^{x}$	0
C	0*	0*	$\leftarrow^{d}$	0		0*		0*	$\leftarrow^{d}$	0		0*	$\leftarrow^{d}$	3

G_y	★	C	G	T	T	A	C	T
★	0*	0*	0*	0*	0*	0*	0*	0*
C	0*	0*	0*	0*	0*	0*	0*	0*
		$\uparrow^{d}$					$\uparrow^{d}$	
G	0*	0	0*	0*	0*	0*	0	0*
			$\uparrow^{d}$					
A	0*	0*	2	0*	0*	0*	0*	0*
			$\uparrow^{y}$	$\uparrow^{d}$		$\uparrow^{d}$		
C	0*	0*	1	1	0*	1	0*	0*

The optimal score of the local alignment is the maximal entry of all three matrices. This is where the backtracking starts. Again, we use the arrows to backtrack the according alignments and change matrices according to the stored symbols. If multiple entries have the same maximal score, each of them constitutes a new starting point.

In our example the optimal score is 5 and the alignment ends without a gap since a gap at the end would decrease the score.

G_d	★		C		G		T		T		A		C		T
★	0		0*		0*		0*		0*		0*		0*		0*
		$\nwarrow^{d,x,y}$										$\nwarrow^{d,x,y}$			
C	0*		2		0*		0*		0*		0*		2		0*
				$\nwarrow^{d}$										$\nwarrow^{d}$	
G	0*		0*		4		0*		0*		0*		0*		1
						$\nwarrow^{d}$		$\nwarrow^{x}$		$\nwarrow^{x}$					
A	0*		0*		0*		3		1		3		0*		0*
		$\nwarrow^{d,x,y}$				$\nwarrow^{y}$		$\nwarrow^{d}$		$\nwarrow^{d,x}$		$\nwarrow^{d}$		$\nwarrow^{x}$	
C	0*		2		0*		1		2		0		5		0

G_x	★	C		G		T		T		A		C		T
★	0*	0*		0*		0*		0*		0*		0*		0*
C	0*	0*	$\leftarrow^{d}$	0		0*		0*		0*		0*	$\leftarrow^{d}$	0
G	0*	0*		0*	$\leftarrow^{d}$	2	$\leftarrow^{x}$	1	$\leftarrow^{x}$	0		0*		0*
A	0*	0*		0*		0*	$\leftarrow^{d}$	1	$\leftarrow^{x}$	0	$\leftarrow^{d}$	1	$\leftarrow^{x}$	0
C	0*	0*	$\leftarrow^{d}$	0		0*		0*	$\leftarrow^{d}$	0		0*	$\leftarrow^{d}$	3

G_y	★	C	G	T	T	A	C	T
★	0*	0*	0*	0*	0*	0*	0*	0*
C	0*	0*	0*	0*	0*	0*	0*	0*
		$\uparrow^{d}$					$\uparrow^{d}$	
G	0*	0	0*	0*	0*	0*	0	0*
			$\uparrow^{d}$					
A	0*	0*	2	0*	0*	0*	0*	0*
			$\uparrow^{y}$	$\uparrow^{d}$		$\uparrow^{d}$		
C	0*	0*	1	1	0*	1	0*	0*

We start by aligning the last character of the alignment and work backwards. Again, we write the character of the current position of both sequences at a diagonal arrow (a step in the G_d matrix), an arrow pointing to the left (a step in the G_x matrix) creates a gap in sequence x, and the upward arrow (a step in the G_y matrix) creates a gap in sequence y. After each step we look whether we need to change the matrix according to the symbols above the arrows. If there is more than one symbol, we start a new alignment, where we first copy the alignment we got so far. The 0^* serves as a stop sign and therefore terminates the alignment. The "normal" 0 leads to two different alignments: one uses the 0 as a stop sign, the other continues the alignment.

In our example we have one optimal alignment:

$\nwarrow$	$\nwarrow^{d}$	$\leftarrow^{d}$	$\leftarrow^{x}$	$\nwarrow^{x}$	$\nwarrow^{d}$
C	G	-	-	A	C
C	G	T	T	A	C

C.2 Phylogenetics

C.2.1 UPGMA

Initially, each sequence i is a cluster c_i with one element ($n_i = 1$). The height l_i of each cluster is zero.
For a given set of sequences from A to E we get the following list of clusters with the respective sizes N:

$Clusters^{(0)} = (\{A\},\{B\},\{C\},\{D\},\{E\})$
$N^{(0)} = (1, 1, 1, 1, 1)$

We start with the following pairwise distances obtained by pairwise alignment:

$Distances^{(0)}$:

	A	B	C	D	E
A	0	-	-	-	-
B	0.012	0	-	-	-
C	0.043	0.042	0	-	-
D	0.038	0.033	0.037	0	-
E	0.095	0.092	0.097	0.093	0

Now we repeat the following steps until the list contains only one element:

1. Select cluster pair (i, j) from the list with minimal distance D_{ij} and create a new cluster c_k by joining c_i and c_j. Assign the height $l_k = D_{ij}/2$ and the number of elements $n_k = n_i + n_j$.
2. Compute the distances for the new cluster c_k to all other clusters c_m:
   $D_{km} = \frac{n_i\, D_{mi} + n_j\, D_{mj}}{n_i + n_j}$.
   The formula ensures that D_{km} is the average distance between all elements in c_k and c_m.
3. Remove i and j from the list and add k to the list.

In our example the smallest distance is between clusters {A} and {B}. We join these clusters to a new cluster {A, B} with $n_{AB} = 1 + 1 = 2$ and $l_{AB} = 0.012/2 = 0.006$.
We get:

$Clusters^{(1)} = (\{A, B\},\{C\},\{D\},\{E\})$
$N^{(1)} = (2, 1, 1, 1)$

The new distances are:

$Distances^{(1)}$:

	AB	C	D	E
AB	0	-	-	-
C	0.0425	0	-	-
D	0.0355	0.037	0	-
E	0.0935	0.097	0.093	0

E.g.: The distance between clusters {A ,B} and {C} is calculated using $D_{AB,C} = \frac{1{\cdot}0.043 + 1{\cdot}0.042}{1 + 1}$.

Now we look again for the smallest distance and therefore, join clusters {A, B} and {D} to cluster {A, B, D} with height $l_{ABD} = 0.0355/2 = 0.01775$.

We get:

$Clusters^{(2)} = (\{A, B, D\},\{C\},\{E\})$
$N^{(2)} = (3, 1, 1)$

The new distances are:

$Distances^{(2)}$:

	ABD	C	E
ABD	0	-	-
C	0.0406	0	-
E	0.093	0.097	0

E.g.: The distance between clusters {A, B, D} and {C} is calculated using $D_{ABD,C} = \frac{2 \cdot 0.0425 + 1 \cdot 0.037}{2 + 1}$.

Now the smallest distance leads to joining of clusters {A, B, D} and {C} to cluster {A, B, C, D} with height $l_{ABCD} = 0.0406/2 = 0.0203$.

We get:

$Clusters^{(3)} = (\{A, B, C, D\},\{E\})$
$N^{(3)} = (4, 1)$

The new distances are:

$Distances^{(3)}$:

	ABCD	E
ABCD	0	-
E	0.094	0

E.g.: The distance between clusters {A, B, C, D} and {E} is calculated using $D_{ABCD,E} = \frac{3 \cdot 0.093 + 1 \cdot 0.097}{3 + 1}$.

The next step is trivial. The remaining clusters are joined to one large cluster and the algorithm terminates:

$Clusters^{(4)} = (\{A, B, C, D, E\})$
$N^{(4)} = (5)$

$l_{ABCDE} = 0.094/2 = 0.047$.

The phylogenetic tree is constructed using the joining order and the calculated height values.

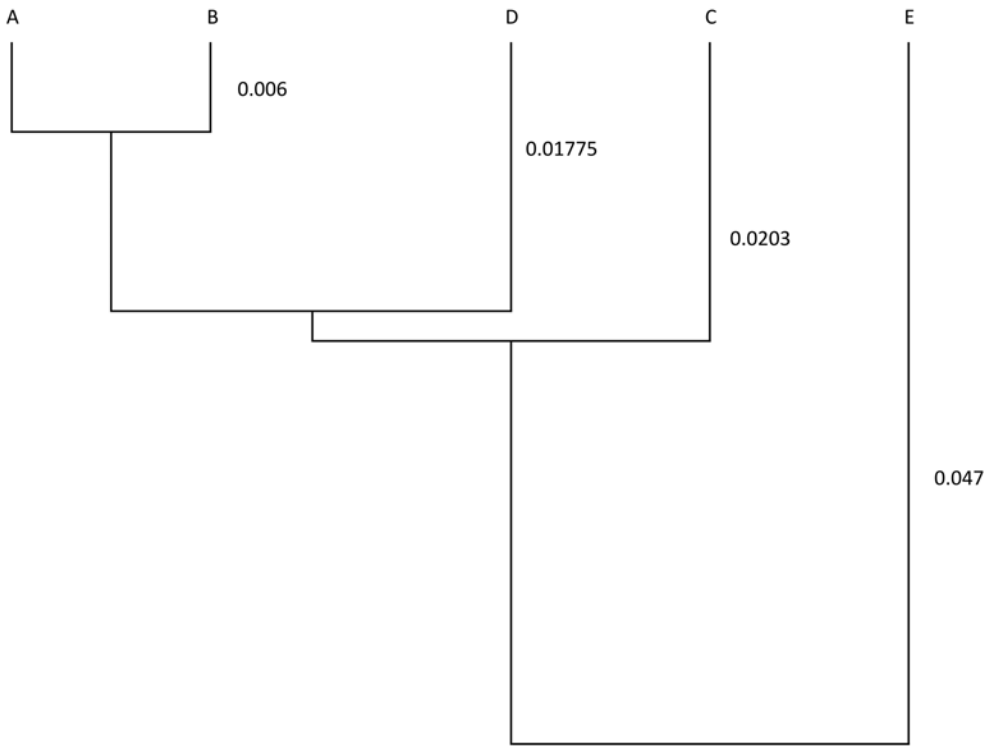


Figure C.2: Phylogenetic tree constructed by UPGMA.

C.2.2 Neighbor Joining

The neighbor joining algorithm works as follows:

Given the pairwise distances D_{ij}, start with a star tree (the taxa are the leaves) and put all taxa in a set of objects.

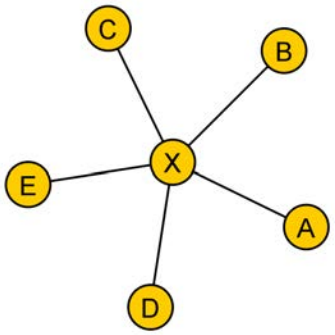


Figure C.3: Initial star tree for neighbor joining.

1. For each leaf i compute
   $r_i = \sum_{k=1}^{N} D_{ik}$,
   where N is the number of objects in the set.
2. For each pair of leaves (i, j) compute
   $Q_{ij} = (N-2)\, D_{ij} - r_i - r_j$.
3. Determine the minimal Q_{ij}. Join the according leaves i and j to a new leaf u.
   Compute branch lengths
   $l_{iu} = \frac{D_{ij}}{2} + \frac{r_i - r_j}{2(N-2)}$ and
   $l_{ju} = D_{ij} - l_{iu}$.
   Compute new distances of u:
   $D_{ku} = \frac{D_{ik} + D_{jk} - D_{ij}}{2}$.
   Delete i and j from the set of objects. Add u to the set of objects. Stop if the set of objects contains only two objects, otherwise go to Step 1.
4. Connect the last two objects by using $l_{ij} = D_{ij}$.

Again, we start with the following pairwise distances obtained by pairwise alignment:

$D^{(0)}$:

	A	B	C	D	E
A	0	-	-	-	-
B	0.012	0	-	-	-
C	0.043	0.042	0	-	-
D	0.038	0.033	0.037	0	-
E	0.095	0.092	0.097	0.093	0

1.1: At the beginning $N = 5$. Therefore, we have to compute the following five r_i values:

$$r_A = 0.012 + 0.043 + 0.038 + 0.095 = 0.188$$
$$r_B = 0.012 + 0.042 + 0.033 + 0.092 = 0.179$$
$$r_C = 0.043 + 0.042 + 0.037 + 0.097 = 0.219$$
$$r_D = 0.038 + 0.033 + 0.037 + 0.093 = 0.201$$
$$r_E = 0.095 + 0.092 + 0.097 + 0.093 = 0.377$$

1.2: $Q^{(0)}$:

	A	B	C	D	E
A	-	-	-	-	-
B	-0.331	-	-	-	-
C	-0.278	-0.272	-	-	-
D	-0.275	-0.281	-0.309	-	-
E	-0.280	-0.280	-0.305	-0.299	-

E.g.: $Q_{AB} = (5-2) \cdot 0.012 - 0.188 - 0.179 = -0.331$

1.3: The smallest Q_{ij} is -0.331 therefore leaves A and B are joined to leaf U_1.
The branch lengths are: $l_{AU_1} = \frac{0.012}{2} + \frac{0.188 - 0.179}{2(5-2)} = 0.0075$ and $l_{BU_1} = 0.012 - 0.0075 = 0.0045$.

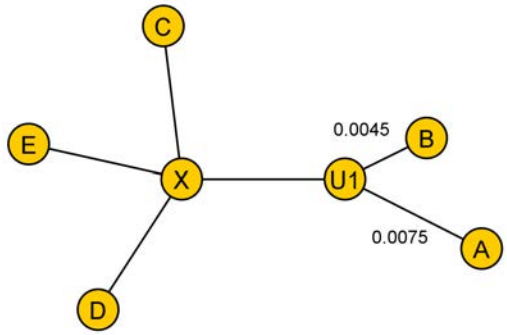


Figure C.4: First join of neighbor joining. Leaves A and B are joined to leaf U_1.

The new distances are:

$D^{(1)}$:

	U_1	C	D	E
U_1	0	-	-	-
C	0.0365	0	-	-
D	0.0295	0.037	0	-
E	0.0875	0.097	0.093	0

E.g.: $D_{CU_1} = \frac{0.043 + 0.042 - 0.012}{2} = 0.0365$

2.1: Since we removed leaves A and B and only added one leaf, U_1, we have now $N = 4$.

$$r_{U_1} = 0.0365 + 0.0295 + 0.0875 = 0.1535$$
$$r_C = 0.0365 + 0.037 + 0.097 = 0.1705$$
$$r_D = 0.0295 + 0.037 + 0.093 = 0.1595$$

$r_E = 0.0875 + 0.097 + 0.093 = 0.2775$

2.2: $Q^{(1)}$:

	U_1	C	D	E
U_1	-	-	-	-
C	-0.251	-	-	-
D	-0.254	-0.256	-	-
E	-0.256	-0.254	-0.251	-

E.g.: $Q_{U_1C} = (4-2) \cdot 0.0365 - 0.1535 - 0.1705 = -0.251$

2.3: The minimal Q_{ij} is -0.256 for pairs U_1E and CD. We have to choose one pair and therefore join leaves C and D to leaf U_2.

The branch lengths are: $l_{CU_2} = \frac{0.037}{2} + \frac{0.1705 - 0.1595}{2(4-2)} = 0.02125$ and $l_{DU_2} = 0.037 - 0.02125 = 0.01575$.

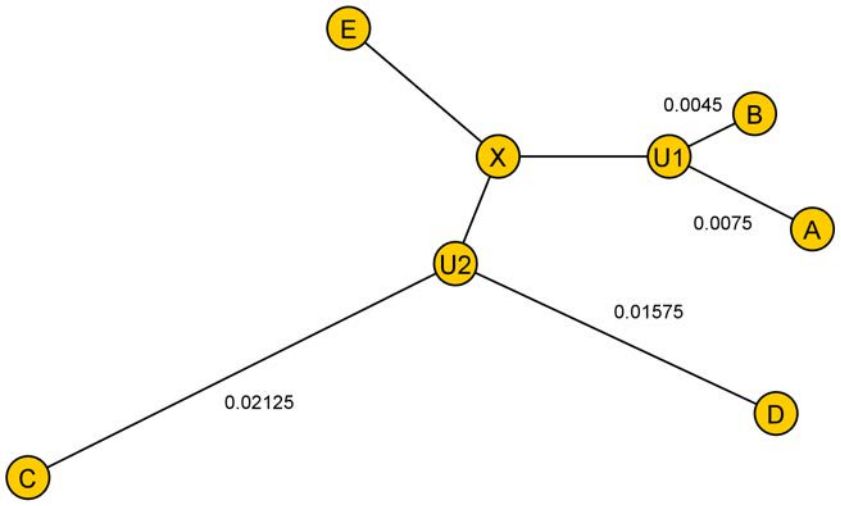


Figure C.5: Second join of neighbor joining. Leaves C and D are joined to leaf U_2.

The new distances are:

$D^{(2)}$:

	U_1	U_2	E
U_1	0	-	-
U_2	0.0145	0	-
E	0.0875	0.0765	0

3.1: $N = 3$, which means we have to compute three r_i values:

$r_{U_1} = 0.0145 + 0.0875 = 0.102$
$r_{U_2} = 0.0145 + 0.0765 = 0.091$
$r_E = 0.0875 + 0.0765 = 0.164$

3.2: $Q^{(2)}$:

	U_1	U_2	E
U_1	-	-	-
U_2	-0.1785	-	-
E	-0.1785	-0.1785	-

E.g.: $Q_{U_1U_2} = (3-2) \cdot 0.0145 - 0.102 - 0.091 = -0.1785$

3.3: Since all Q_{ij} are the same we can choose one pair and join leaves U_1 and E to leaf U_3. The branch lengths are: $l_{U_1U_3} = \frac{0.0875}{2} + \frac{0.102 - 0.164}{2(3-2)} = 0.01275$ and $l_{EU_3} = 0.0875 - 0.01275 = 0.07475$.

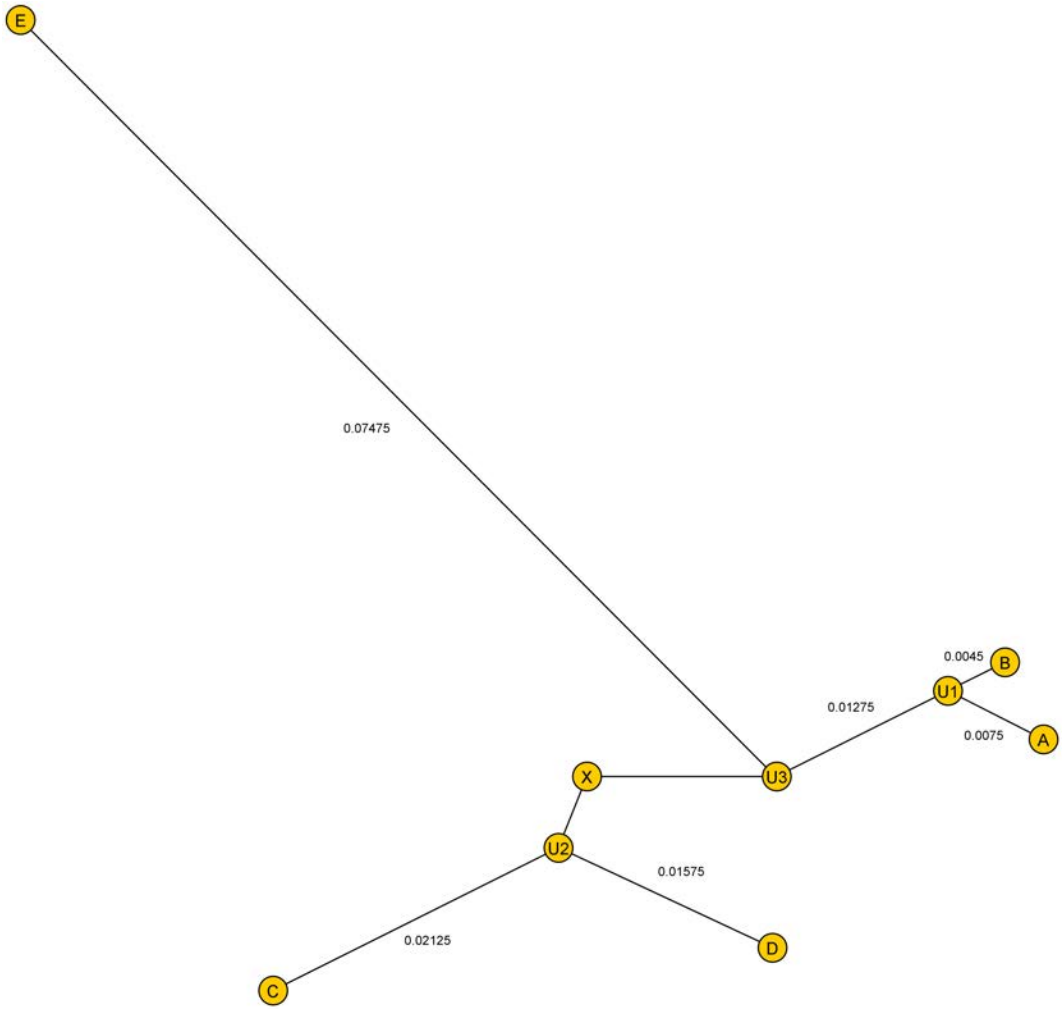


Figure C.6: Third join of neighbor joining. Leaves U_1 and E are joined to leaf U_3.

The new distances are:

$D^{(3)}$:

	U_2	U_3
U_2	0	-
U_3	0.00175	0

4. The last branch length is equal to the distance between U_2 and U_3 and therefore 0.00175.

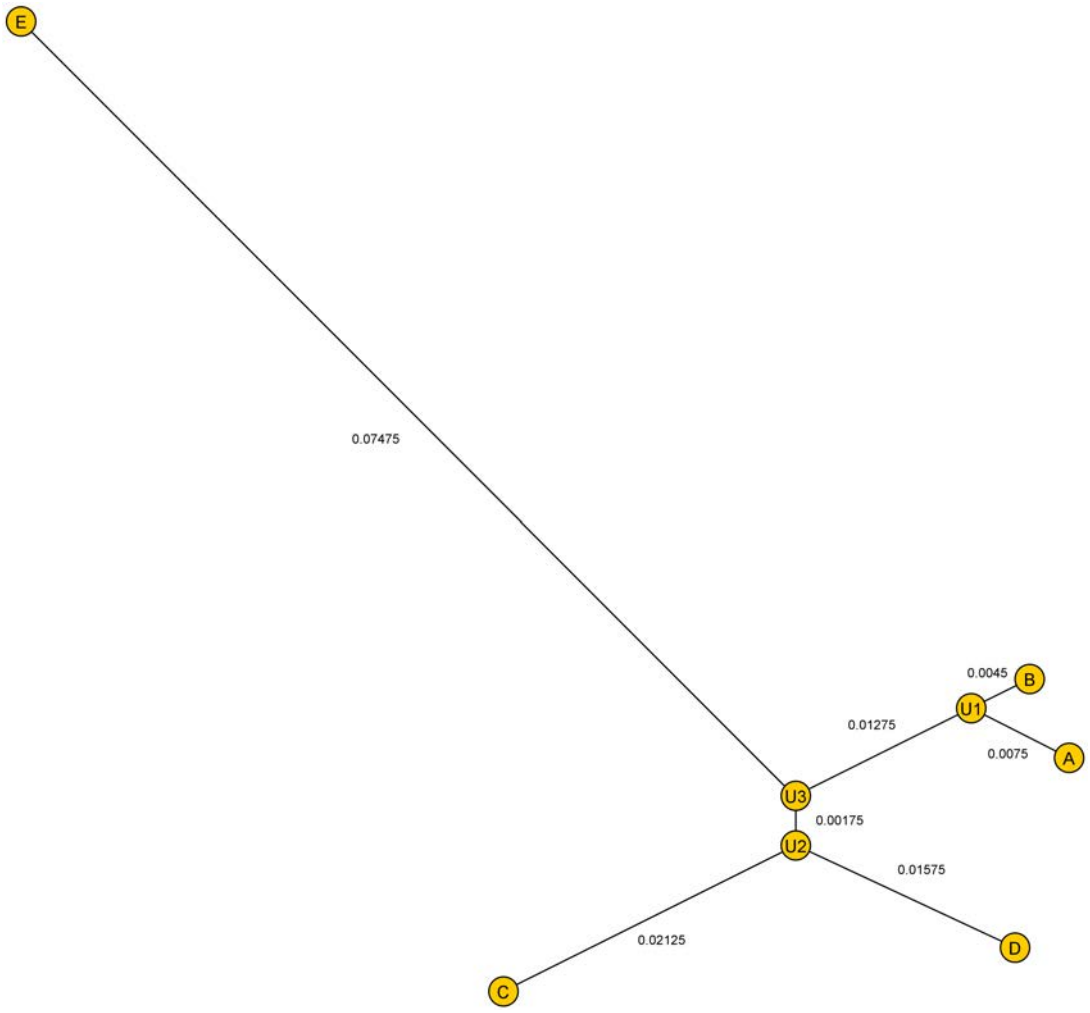


Figure C.7: Phylogenetic tree built by neighbor joining.