Algorithms in Bioinformatics I

Wintersemester 2002/3,
Zentrum für Bioinformatik Tübingen,
WSI-Informatik, Universität Tübingen

Prof. Dr. Daniel Huson

huson@informatik.uni-tuebingen.de

0.1 Recommended reading

- R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological Sequence Analysis, Cambridge, 1998
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, CSHL 2001.
- Dan Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge 1997
- T.K. Attwood and D.J. Parry-Smith. Introduction to bioinformatics, Prentice Hall 1998.
- J. Setubal and J. Meidanis. Introduction to Computational Molecular Biology. PWS, 1997.

Weitere Literaturangaben befinden sich in dem Skript.

0.2 What is bioinformatics?

Is bioinformatics a field of science in which biology, computer science, and information technology merge into a one discipline?

Technological advances, such as high-throughput sequencers, DNA-arrays, protein mass-spectrometry, faster processors and larger computer memories, have helped transform molecular biology into a high-throughtput science, in which huge amounts of data are being generated ever faster.

There are three important sub-disciplines in bioinformatics:

- the development of algorithms and statistics with which to assess relationships among members of large data sets,
- the analysis and interpretation of various types of data including nucleotide and amino acid sequences, protein domains, and protein structures, and
- the development and implementation of tools that enable efficient access and management of different types of information.

1 Probabilities and probabilistic models

The following is based on: R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological Sequence Analysis, Cambridge, 1998

1.1 Probabilistic models

A *model* means a system that simulates an object under consideration.

A *probabilistic model* is a model that produces different outcomes with different probabilities. Hence, a probabilistic model can simulate a *whole class* of objects, assigning each an associated probability.

In bioinformatics, the objects usually are DNA or protein sequences and a model might describe a family of related sequences.

1.2 Examples

1. The roll of a six-sided die. This model has six parameters $p_1, p_2, \ldots, p_6$, where p_i is the probability of rolling the number i. For probabilities, $p_i > 0$ and $\sum_i p_i$.
2. Three rolls of a die: the model might be that the rolls are independent, so that the probability of a sequence such as $[2, 4, 6]$ would be $p_2 p_4 p_6$.
3. An extremely simple model of any DNA or protein sequence is a string over a 4 (nucleotide) or 20 (amino acid) letter alphabet. Let q_a denote the probability, that residue a occurs at a given position, at random, independent of all other residues in the sequence.

   Then, for a given length n, the probability of the sequence $x_1, x_2, \ldots, x_n$ is

   $$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} q_{x_i}.$$

1.3 Maximum Likelihood estimation

Probabilistic models have parameters, which are usually *estimated* from large sets of trusted examples, called a *training set.*

For example, the probability q_a for seeing amino acid a in a protein sequence can be estimated as the observed frequency f_a of a in a database of known protein sequences, such as SWISS-PROT.

This way of estimating models is called *Maximum likelihood estimation*, because it can be shown that using the observed frequencies maximizes the total probability of the training set, given the model.

In general, given a model with parameters θ and a set of data D, the *maximum likelihood estimate (MLE)* for θ is the value which maximizes $P(D \mid \theta)$.

1.4 Overfitting and pseudo-counts

When estimating parameters from a small training set, there is a danger of *overfitting*: the model fits the training set very well, but does not *generalize* to new data.

For example, consider a coin-toss model. If our training set consists of three flips $[tail, tail, tail]$, then MLE gives us
$p_{tail} := f_{tail} = 1$ and $p_{head} := f_{head} = 0$.

Similarly, if we have a small training set of protein sequences in which a certain amino acid a is not present, then MLE leads to $f_a = 0$.

The problem of overfitting can be addressed using *pseudocounts*: Add a small number to each count of occurrences of an object in the training set:

For example, in the coin-toss model above, set

$$f_{tail} = \frac{4}{5} = \frac{3+1}{3+2} \text{ and } f_{head} = \frac{1}{5} = \frac{0+1}{3+2}.$$

1.5 Conditional, joint, and marginal probabilities

Suppose we have two dice D_1 and D_2. The probability of rolling an i with die D_1 is called the *conditional probability* $P(i \mid D_1)$.

If we pick a die at random, with probability $P(D_j)$ $(j = 1, 2)$, then the probability for picking die j and rolling an i is the product $P(i, D_j) = P(D_j)P(i \mid D_j)$, called the *joint probability.*

In general, $P(X, Y) = P(X \mid Y)P(Y)$.

Given conditional or joint probabilities, a *marginal* probability is calculated by removing one of the variables, which is accomplished by summing over all possible events:

$$P(X) = \sum_Y P(X, Y) = \sum_Y P(X \mid Y)P(Y).$$

1.6 Prior- and posterior probability

An occasionally dishonest casino uses two kinds of dice, of which 99% *are fair, but* 1% *are loaded, so that a* 6 *appears* 50% *of the time.*

We pick up a dice and roll $[6, 6, 6]$. This looks like a loaded die, is it? This is an example of a *model comparison* problem.

I.e., our hypothesis D_{loaded} is that the die is loaded. The other alternative is D_{fair}. Which model fits the observed data better? We want to calculate:

$$P(D_{loaded} \mid [6, 6, 6]),$$

the *posterior probability* that the dice is loaded, given the observed data.

Note that the *prior probability* of this hypothesis is $\frac{1}{100}$. It is called *prior* because it is our best guess about the dice *before* having seen any information about the it.

1.7 Comparing models using Bayes' theorem

We can easily compute the *likelihood* of the hypothesis D_{loaded}:

$$P([6,6,6] \mid D_{loaded}) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}.$$

We can calculate posterior probabilities using Bayes' theorem:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)}.$$

We set $X = D_{loaded}$ and $Y = [6,6,6]$, thus obtaining

$$P(D_{loaded} \mid [6,6,6]) = \frac{P([6,6,6] \mid D_{loaded})P(D_{loaded})}{P([6,6,6])}.$$

The probability $P(D_{loaded})$ of picking a loaded die is 0.01.

The probability $P([6,6,6] \mid D_{loaded})$ of rolling three sixes using a loaded die is $0.5^3 = 0.125$.

The total probability $P([6,6,6])$ of three sixes is

$$P([6,6,6] \mid D_{loaded})P(D_{loaded}) + P([6,6,6] \mid D_{fair})P(D_{fair}).$$

Now,

$$\begin{aligned} P(D_{loaded} \mid [6,6,6]) &= \frac{(0.5^3)(0.01)}{(0.5^3)(0.01)+(\frac{1}{6}^3)(0.99)} \\ &= 0.21. \end{aligned}$$

Thus, the die is probably $\boxed{\text{loaded}}$.

1.8 Biological example

Lets assume that extracellular (*ext*) proteins have a slightly different composition than intercellular (*int*) ones. We want to use this to judge whether a new protein sequence $x_1, \ldots, x_n$ is *ext* or *int*.

To obtain training data, classify all proteins in SWISS-PROT into *ext*, *int* and unclassifiable ones.

Determine the frequencies f_a^{ext} and f_a^{int} of each amino acid a in *ext* and *int* proteins, respectively.

To be able to apply Bayes' theorem, we need to determine the priors p^{int} and p^{ext}, i.e. the probability that a new (unexamined) sequence is extracellular or intercellular, respectively.

We have:

$$P(x \mid ext) = \prod_{i=1}^{n} q_{x_i}^{ext} \text{ and } P(x \mid int) = \prod_{i=1}^{n} q_{x_i}^{int}.$$

If we assume that any sequence is either extracellular or intercellular, then we have

$$P(x) = p^{ext}P(x \mid ext) + p^{int}P(x \mid int).$$

By Bayes' theorem, we obtain

$$P(ext \mid x) = \frac{P(ext)P(x \mid ext)}{P(x)} = \frac{p^{ext}\prod_i q_{x_i}^{ext}}{p^{ext}\prod_i q_{x_i}^{ext} + p^{int}\prod_i q_{x_i}^{int}},$$

the posterior probability that a sequence is extracellular.

(In reality, many transmembrane proteins have both intra- and extracellular components and more complex models such as HMMs are appropriate.)

1.9 Bayes' theorem and parameter estimation

Given training data D and a probabilistic model with parameters θ. Recall that the maximum likelihood estimation chooses parameter values for θ that

$$\text{maximize } P(D \mid \theta).$$

As discussed earlier, this is effected by overfitting, if the given training set D is too small.

However, if we have some prior knowledge $P(\theta)$ about θ, then Bayes' theorem allows us to use it:

$$P(\theta \mid D) = \frac{P(\theta)P(D \mid \theta)}{\int_{\theta'} P(\theta')P(D \mid \theta')}.$$

We can then choose the parameter values for θ that

$$\text{maximize } P(\theta \mid D).$$

This is called maximum *a posteriori* or *MAP* estimation.

1.10 Probability vs likelihood

If we consider $P(X \mid Y)$ as a function of X, then this is called a *probability.*

If we consider $P(X \mid Y)$ as a function of Y, then this is called a *likelihood.*

2 Pairwise alignment

We will discuss:

1. Dot matrix method for comparing sequences
2. The probabilistic model of pairwise alignment
3. Global-, local-, repeat- and overlap alignment of two sequences using dynamic programming

Sources for this lecture:

- R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological Sequence Analysis, Cambridge, 1998
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, CSHL 2001.
- R. Giegerich. Sequence similarity and dynamic programming, 2002.
  `http://www.zbit.uni-tuebingen.de/biss2002/handouts/pdf/giegerich_part1.pdf`
- J. Setubal & J. Meidanis, Introduction to Computational Molecular Biology, 1997.

2.1 Importance of sequence alignment

The first fact of biological sequence analysis: In biomolecular sequences (DNA, RNA, or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity.

"Duplication with modification": The vast majority of extant proteins are the result of a continuous series of genetic duplications and subsequent modifications. As a result, redundancy is a built-in characteristic of protein sequences, and we should not be surprised that so many new sequences resemble already known sequences. ...all of biology is based on enormous redundancy...

We didn't know it at the time, but we found everything in life is so similar, that the same genes work in flies are the ones that work in humans. (Eric Wieschaus, cowinner of the 1995 Nobel prize in medicine for work on the genetics of *Drosophia* development.)

Dan Gusfield, 1997, 212 ff

2.2 Sequence alignment

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple-alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in both sequences.

Two sequences are aligned by writing them across a page in two rows. Identical or similar characters are placed in the same column, whereas non-identical characters are either placed in the same column as a mismatch or are opposite a gap in the other sequence.

```
Two strings:          →   Alignment:
I MISS MISSISSIPPI        I-MISSMISSISIPPI-
                          |||||||||||||||||
MY MISS IS A HIPPIE       MYMISS-ISAH-IPPIE
```

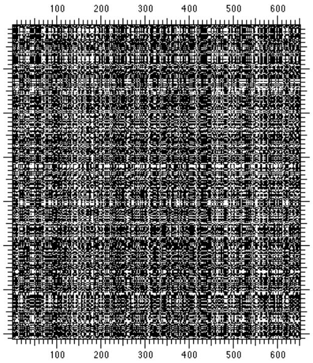
2.3 Dot matrix sequence comparison

A dot matrix analysis is primarily a method for comparing two sequences to look for a possible alignment of characters between the sequences.

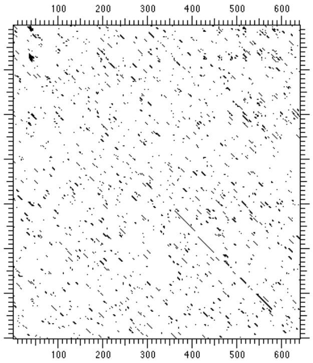
A matrix relating two sequences is produced. A dot is placed at each cell for which the corresponding symbols match:

```
  IMISSMISSISSIPPI
M  .   .
Y
M  .   .
I . .   .  .  .  .
S    ..  .. ..
S    ..  .. ..
I . .   .  .  .  .
S    ..  .. ..
A
H
I . .   .  .  .  .
P              ..
P              ..
I . .   .  .  .  .
E
```

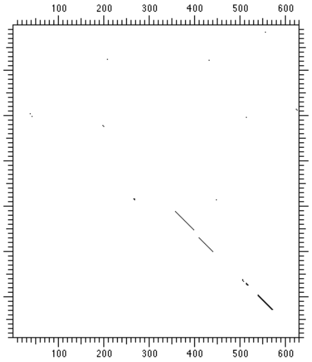
Example: DNA sequences which encode the phage lambda and P22 repressor sequences:



$w = 1, s = 1$



$w = 11,\ s = 7$

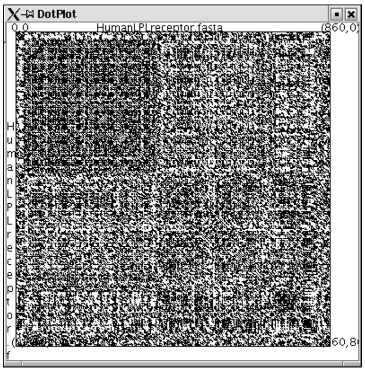


$w = 23,\ s = 15$

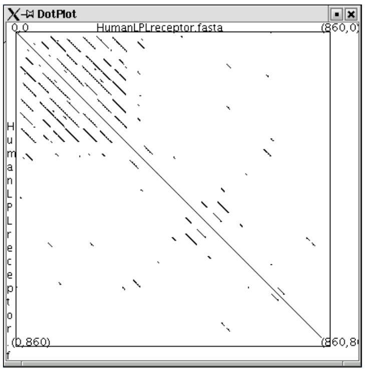
To obtain cleaner pictures, a *window size* w and a *stringency* s are used and a dot is only drawn at point (x, y) if in the next w positions at least s characters are equal.

2.4 Dot matrix repeat detection

Dot matrix analysis of human LDL receptor against itself (protein sequence):



$w = 1,\ s = 1$



$w = 23,\ s = 7$

This reveals many repeats in the first 300 positions.

2.5 Pairwise alignment

1. Alignment between very similar human alpha- and beta globins:

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
            G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN   GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL
```

2. Plausible alignment to leghaemoglobin from yellow lupin:

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
            ++ ++++H+ KV   + +A  ++          +L+ L+++H+ K
LGB2_LUPLU NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```

3. A spurious high-scoring alignment of human alpha globin to a nematode glutathione S-transferase homologue:

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD----LHAHKL
            GS+ + G +   +D L  ++ H+ D+  A +AL D    ++AH+
F11G11.2    GSGYLVGDSLTFVDLLVAQHTADLL--AANAALLDEFPQFKAHQE
```

In (1), there are many positions at which the two corresponding residues are identical. Many others are functionally conserved, e.g. the D-E pairs, both negatively charged amino acids.

In (2), we also see a biologically meaningful alignment, as it is known that the two proteins are evolutionarily related, have the same 3D structure and both have the same function. However, there are many fewer identities and gaps have been introduced in the sequences.

In (3), we see an alignment with a similar number of identities or conservative changes. However, this is a spurious alignment between two proteins that have completely different structure and function.

The goal is to use similarity to uncover *homology*, while avoiding *homoplasy*.

2.6 The scoring model

When comparing two biological sequences, we want to determine whether and how they diverged from a common ancestor by a process of mutation and selection.

The basic mutational processes are *substitutions*, *insertions* and *deletions*. The latter two give rise to *gaps*.

The total score assigned to an alignment is the sum of terms for each aligned pair of residues, plus terms for each gap.

In a probabilistic interpretation, this will correspond to the the logarithm of the relative likelihood that the sequences are related, compared to being unrelated.

Using such an additive scoring scheme is based on the assumption that mutations at different sites occur independently of each other. This is often reasonable for DNA and proteins, but not for structural RNA, where base pairing introduces very important long-range dependences.

2.7 Substitution matrices

To be able to score an alignment, we need to determine score terms for each aligned residue pair.

Given two sequences

$$x = (x_1, x_2, \ldots, x_n) \text{ and } y = (y_1, y_2, \ldots, y_m).$$

The symbols come from some alphabet $\mathcal{A}$, e.g. the four bases $\{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ for DNA or, in the case of amino acids, the 20 symbols $\{\mathtt{A}, \mathtt{R}, \mathtt{N}, \mathtt{D}, \mathtt{C}, \mathtt{Q}, \mathtt{E}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{L}, \mathtt{K}, \mathtt{M}, \mathtt{F}, \mathtt{P}, \mathtt{S}, \mathtt{T}, \mathtt{W}, \mathtt{Y}, \mathtt{V}\}$.

For now we will only consider non-gapped alignments such as:

```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
           G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN  GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL
```

Given a pair of aligned sequences (without gaps), we want to assign a score to the alignment that gives a measure of the relative likelihood that the sequences are related (model M) as opposed to being unrelated (model R), as

$$\frac{P(x, y \mid M)}{P(x, y \mid R)}.$$

The unrelated or *random* model R assumes that the letter a occurs independently with some frequency q_a, and hence the probability of the two sequences is the product:

$$P(x, y \mid R) = \prod_i q_{x_i} \prod_j q_{y_j}.$$

In the *match* model M, aligned pairs of residues occur with a joint probability p_{ab}, which is the probability that a and b have each evolved from some unknown original residue c as their common ancestor.

Thus, the probability for the whole alignment is:

$$P(x, y \mid M) = \prod_i p_{x_i y_i}.$$

The ratio of theses two likelihoods is known as the *odds ratio*:

$$\frac{P(x, y \mid M)}{P(x, y \mid R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}.$$

To obtain an additive scoring scheme, we take the logarithm to get the *log-odds ratio:*

$$S = \sum_i s(x_i, y_i),$$

with

$$s(a, b) := \log\left(\frac{p_{ab}}{q_a q_b}\right).$$

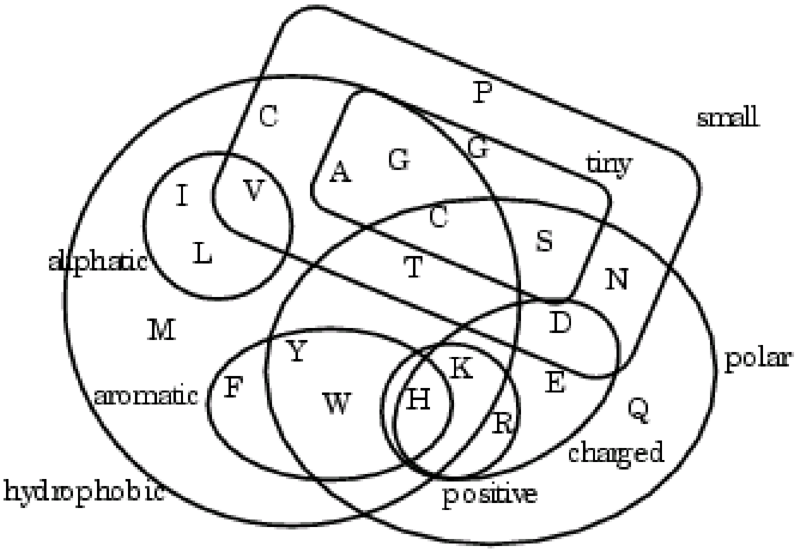
We thus obtain a matrix $s(a, b)$ that determines a score for each aligned residue pair, known as a *score* or *substitution* matrix.

For amino-acid alignments, commonly used matrices are the PAM and BLOSUM matrices, for example BLOSUM50 (to be discussed in detail later):

2.8 BLOSUM50

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

2.9 Classification of amino acids



2.10 Gap penalties

Gaps are undesirable and thus penalized. The standard cost associated with a gap of length g is given either by a linear score

$$\gamma(g) = -gd$$

or an affine score

$$\gamma(g) = -d - (g-1)e,$$

where d is the *gap open* penalty and e is the *gap extension* penalty.

Usually, $e < d$, with the result that less isolated gaps are produced, as shown in the following comparison:

Linear gap penalty:

```
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
GSAQVKGHGKK--------VA--D----A-SALSDLHAHKL
```

Affine gap penalty:

```
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
GSAQVKGHGKKVADA---------------SALSDLHAHKL
```

2.11 Probabilistic model of gap penalties

Assume that the probability of a gap occurring at a particular site in a given sequence is

$$p(gap) = f(g) \prod_{x_i \text{ in gap}} q_{x_i},$$

where $f(g)$ is a function of the length g of the gap, and $\prod_{i \text{ in gap}} q_{x_i}$ is the combined probability of the inserted residues.

We can assume that the q_{x_i} are the probabilities given by the random model, because they correspond to unmatched residues. Hence, in the computation of the log-odds ratio for the gapped region, the q_{x_i} cancel and the remaining term is

$$\gamma(g) = \log\left(\frac{f(g)\prod_{x_i \text{ in gap}} q_{x_i}}{\prod q_{x_i}}\right) = \log(f(g)).$$

This shows that gap penalties correspond to the log probability of a gap of the given length.

2.12 The number of possible alignments

How many different gapped alignments are possible between two sequences $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$?

A sequences of pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$ is called a *subsequence of indices* of x and y, if $1 \leq i_1 \leq i_2 \leq \ldots \leq i_r \leq n$ and if $1 \leq j_1 \leq j_2 \leq \ldots \leq j_r \leq m$.

We use such a subsequence to specify the set of all positions that are paired by a given alignment of x and y.

```
          Alignment                 Subsequence
          a - c g t g t a - c       a c g t g t a c
Example:  ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕   ⇔   |  \  |  / /  |
          a g c - t - g a c c       a g c t g a c c
```

Here the subsequence is:

$$(1,1),(2,3),(4,4),(6,5),(7,6),(8,8).$$

Lemma The number of possible alignments between x and y equals the number of possible subsequences of indices,

$$N(n,m)=\sum_{r=0}^{\min(n,m)}\binom{n}{r}\binom{m}{r}.$$

Proof: For each $r\in\{0,1,\dots,\min(n,m)\}$: the number of ordered selections of $i_1,i_2,\dots,i_r$ in $1,2,\dots,n$ is $\binom{n}{r}$ and the number of ordered selections of $j_1,j_2,\dots,j_r$ in $1,2,\dots,m$ is $\binom{m}{r}$. All these possibilities can be combined. □

The number $N(n,n)=\sum_{r=0}^{n}\binom{n}{r}\binom{n}{r}=\binom{2n}{n}$ can be approximated by $\frac{2^{2^n}}{\sqrt{\pi n}}$, using Stirling's formula.

Hence, $N(1000,1000)\approx 10^{600}$.

2.13 Alignment algorithms

Given a scoring scheme, we need to have an algorithm that computes the highest-scoring alignment of two sequences.

We will discuss alignment algorithms based on *dynamic programming*. Dynamic programming algorithms play a central role in computational sequence analysis. They are guaranteed to find the optimal scoring alignment.

However, for large sequences they can be too slow and heuristics (such as BLAST, FASTA, MUMMER etc) are then used that usually perform very well, but will miss the best alignment for some sequence pairs.

Depending on the input data, there are a number of different variants of alignment that are considered, among them *global alignment*, *local alignment* and *overlap alignment*.

We will use two short amino acid sequences for illustration:

`HEAGAWGHEE` and `PAWHEAE`.

To score the alignment we will use the BLOSUM50 matrix and a gap cost of $d=-8$. (Later, we will also use affine gap costs.)

Here they are arranged to show a matrix of corresponding BLOSUM50 values:

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	**5**	0	**5**	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	**15**	-3	-3	-3	-3
H	**10**	0	-2	-2	-2	-3	-3	**10**	0	0
E	0	**6**	-1	-3	-1	-3	-3	0	**6**	**6**
A	-2	-1	**5**	0	**5**	-3	0	-2	-1	-1
E	0	**6**	-1	-3	-1	-3	-3	0	**6**	**6**

2.14 Global alignment: Needleman-Wunsch algorithm

Saul Needleman and Christian Wunsch (1970), improved by Peter Sellers (1974).

Consider the problem of obtaining the best *global* alignment of two sequences. The Needleman-Wunsch algorithm is a dynamic program that solves this problem.

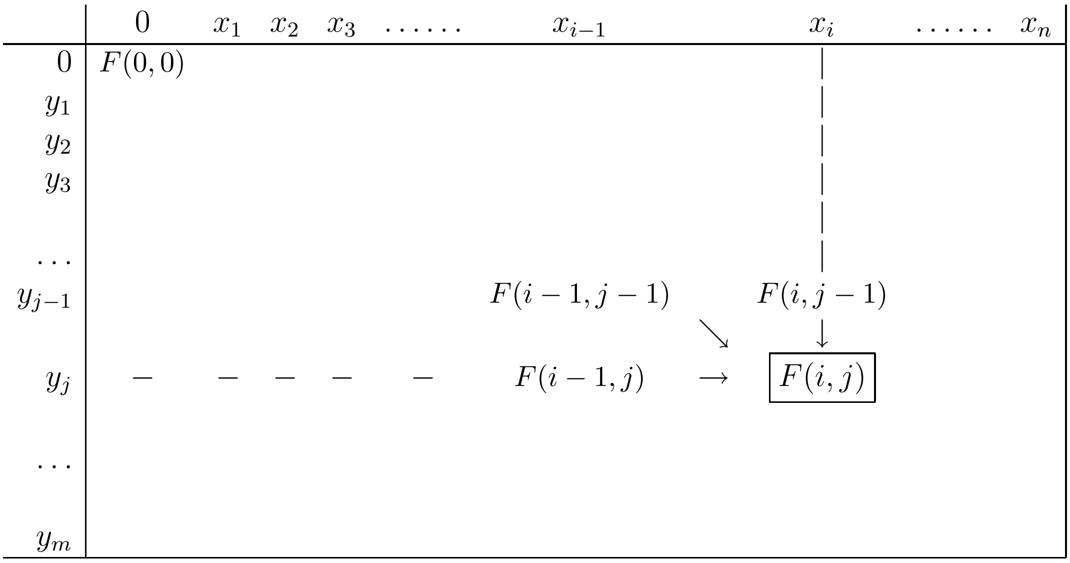
Idea: Build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences.

Given two sequences $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$. We will compute a matrix

$$F : \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} \to \mathbb{R}$$

in which $F(i, j)$ equals the best score of the alignment of the two prefixes $(x_1, x_2, \ldots, x_i)$ and $(y_1, y_2, \ldots, y_j)$.

This will be done recursively by setting $F(0, 0) = 0$ and then computing $F(i, j)$ from $F(i-1, j-1)$, $F(i-1, j)$ and $F(i, j-1)$:



2.15 The recursion

There are three ways in which an alignment can be extended up to (i, j):

x_i aligns to y_i:	x_i aligns to a gap:	y_i aligns to a gap:
`I G A` x_i `L G V` y_j	`A I G A` x_i `G V` y_j `- -`	`G A` x_i `- -` `S L G V` y_j

We obtain $F(i, j)$ as the largest score arising from these three options:

$$F(i, j) := \max \begin{cases} F(i-1, j-1) + s(x_i, y_i) \\ F(i-1, j) - d \\ F(i, j-1) - d. \end{cases}$$

This is applied repeatedly until the whole matrix $F(i, j)$ is filled with values.

To complete the description of the recursion, we need to set the values of $F(i, 0)$ and $F(0, j)$ for $i \neq 0$ and $j \neq 0$:

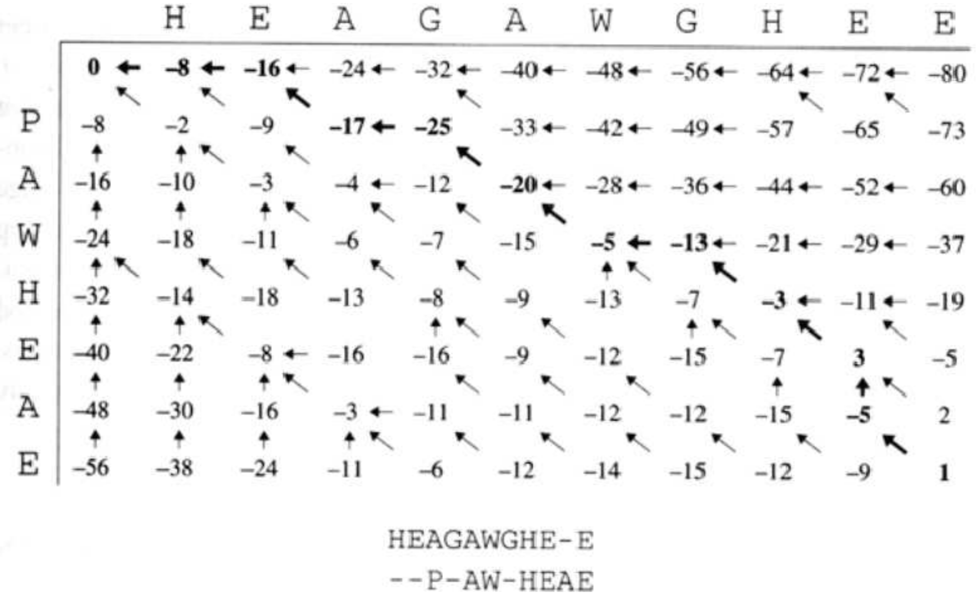
We set $F(i, 0) =$ ________ for $i = 0, 1, \ldots, n$ and
we set $F(0, j) =$ ________ for $j = 0, 1, \ldots, m$.

The final value $F(n, m)$ contains the score of the best global alignment between x and y.

To obtain an alignment corresponding to this score, we must find the path of choices that the recursion made to obtain the score. This is called a *traceback*.

2.16 Example of global alignment matrix

Alignment matrix made using $d = -8$ and BLOSUM50 scores:



		H	E	A	G	A	W	G	H	E	E
	0	**-8**	**-16**	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	**-17**	**-25**	-33	-42	-49	-57	-65	-73
A	-16	-10	-3	-4	-12	**-20**	-28	-36	-44	-52	-60
W	-24	-18	-11	-6	-7	-15	**-5**	**-13**	-21	-29	-37
H	-32	-14	-18	-13	-8	-9	-13	-7	**-3**	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	**3**	-5
A	-48	-30	-16	-3	-11	-11	-12	-12	-15	**-5**	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	**1**

```
HEAGAWGHE-E
--P-AW-HEAE
```

Durbin *et al.* (1998)

2.17 Needleman-Wunsch algorithm

```
Input: two sequences $x$ and $y$
Output: optimal alignment and score $\alpha$
Initialization: Set $F(i,0) := -id$ for all $i = 0,1,2,\dots,n$
Set $F(0,j) := -jd$ for all $j = 0,1,2,\dots,m$
For $i = 1,2,\dots,n$ do:
    For $j = 1,2,\dots,m$ do:
        Set $F(i,j) := \max \begin{cases} F(i-1,j-1) + s(x_i, y_i) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$
        Set backtrace $T(i,j)$ to the maximizing pair $(i',j')$
The best score is $\alpha := F(n,m)$
Set $(i,j) := (n,m)$

repeat
    if $T(i,j) = (i-1,j-1)$ print $\binom{x_{i-1}}{y_{j-1}}$
    else if $T(i,j) = (i-1,j)$ print $\binom{x_{i-1}}{-}$ else print $\binom{-}{y_{j-1}}$
    Set $(i,j) := T(i,j)$
until $(i,j) = (0,0)$.
```

2.18 Complexity

Complexity of the Needleman-Wunsch algorithm:

We need to store $(n+1)\times(m+1)$ numbers. Each number takes a constant number of calculations to compute: three sums and a max.

Hence, the algorithm requires $O(nm)$ time and memory.

For biological sequence analysis, we prefer algorithms that have time and space requirements that are linear in the length of the sequences. Quadratic time algorithms are a little slow, but feasible. $O(n^3)$ algorithms are only feasible for very short sequences.

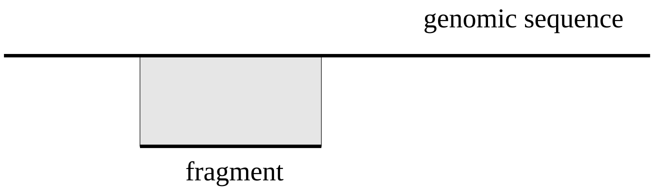
Something to think about: if we are only interested in the best score, but not the actual alignment, then it is easy to reduce the space requirement to linear.

2.19 Local alignment: Smith-Waterman algorithm

Temple Smith and Mike Waterman (1981).

Global alignment is applicable when we have two similar sequences that we want to align from end-to-end, e.g. two homologous genes from related species.

Often, however, we have two sequences x and y and we would like to find the best match between *subsequences* of both. For example, we may want to find the position of a fragment of DNA in a genomic sequence:



The best scoring alignment of two subsequences of x and y is called the best *local alignment.*

The Smith-Waterman local alignment algorithm is obtained by making two simple modifications to the global alignment algorithm.

(1) In the main recursion, we set the value of $F(i,j)$ to zero, if all attainable values at position (i,j) are negative:

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

The value $F(i,j) = 0$ indicates that we should start a new alignment at (i,j). This is because, if the best alignment up to (i,j) has a negative score, then it is better to start a new one, rather than to extend the old one.

Note that, in particular, we have $F(i,0) = 0$ and $F(0,j) = 0$ for all $i = 0,1,2,\ldots,n$ and $j = 0,1,2,\ldots,m$.

(2) Instead of starting the traceback at (n,m), we start it at the cell with the highest score, $\arg\max F(i,j)$. The traceback ends upon arrival at a cell with score 0, with corresponds to the start of the alignment.

For this algorithm to work, we require that the expected score for a random match is negative, i.e. that

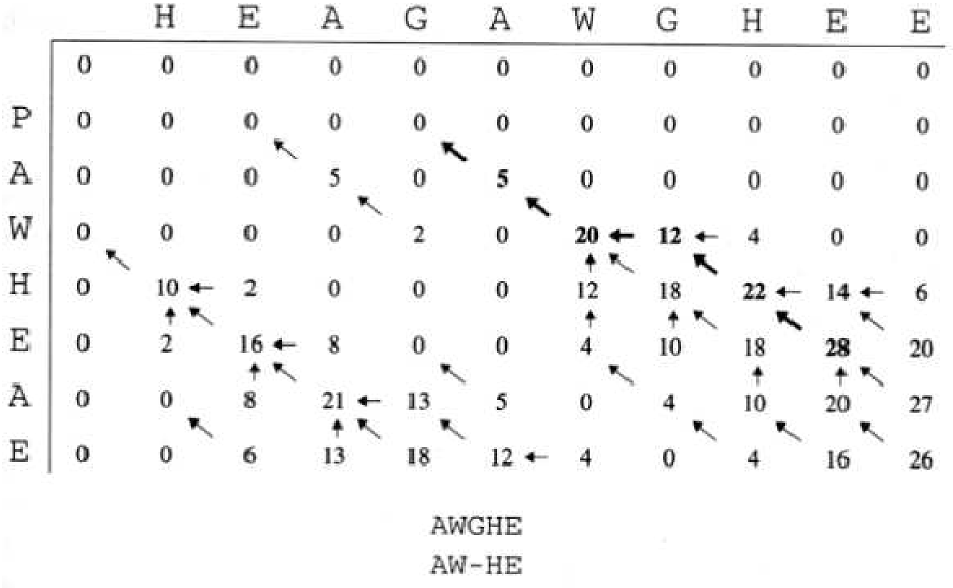
$$\sum_{a,b} q_a q_b s(a,b) < 0,$$

where q_a and q_b are the probabilities for the seeing the symbol a or b at any given position, respectively.

Otherwise, matrix entries will tend to be positive, producing long matches between random sequences.

2.20 Example of a local alignment matrix

Alignment matrix made using $d = -8$ and BLOSUM50 scores:



		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	5	0	0	0	0	0
W	0	0	0	0	2	0	20	12	4	0	0
H	0	10	2	0	0	0	12	18	22	14	6
E	0	2	16	8	0	0	4	10	18	28	20
A	0	0	8	21	13	5	0	4	10	20	27
E	0	0	6	13	18	12	4	0	4	16	26

```
AWGHE
AW-HE
```

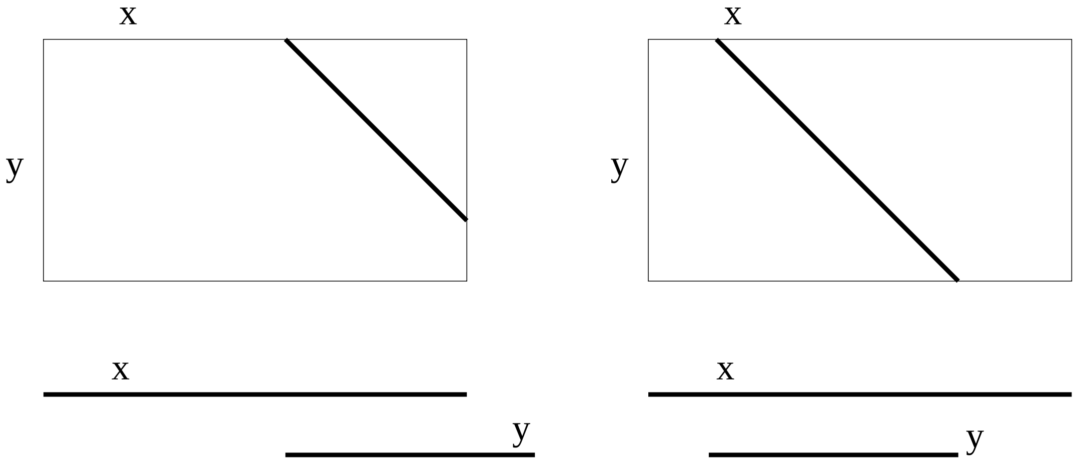
Durbin *et al.* (1998)

2.21 Smith-Waterman algorithm

```
**Input:** two sequences $x$ and $y$
**Output:** optimal local alignment and score $\alpha$
**Initialization:** Set $F(i,0) := 0$ for all $i = 0, 1, 2, \ldots, n$
Set $F(0,j) := 0$ for all $j = 0, 1, 2, \ldots, m$
**For** $i = 1, 2, \ldots, n$ **do**:
    **For** $j = 1, 2, \ldots, m$ **do**:
        Set $F(i,j) := \max \begin{cases} 0 \\ F(i-1,j-1) + s(x_i, y_i) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$
        Set backtrace $T(i,j)$ to the maximizing pair $(i', j')$
Set $(i,j) := \arg\max\{F(i,j) \mid i = 1, 2, \ldots, n, j = 1, 2, \ldots, m\}$
The best score is $\alpha := F(i,j)$
**repeat**
    **if** $T(i,j) = (i-1, j-1)$ **print** $\binom{x_{i-1}}{y_{j-1}}$
    **else if** $T(i,j) = (i-1, j)$ **print** $\binom{x_{i-1}}{-}$ **else print** $\binom{-}{y_{j-1}}$
    Set $(i,j) := T(i,j)$
**until** $F(i,j) = 0$.
```

2.22 Algorithm for finding overlap alignments

If we are given different fragments of genomic DNA that we would like to piece together, then we need an alignment method that does not penalize overhanging ends:



For an overlap alignment, matches should be allowed to start anywhere on the top or left boundary of the matrix, and should be allowed to end anywhere on the bottom or right boundary.

To allow the former, simply set $F(i, 0) = 0$ and $F(0, j) = 0$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$. The recurrence relations are those used for global alignment.

To allow the latter, start the traceback at the best scoring cell contained in the bottom row or rightmost column, i.e. start at

$$\arg\max\{F(i, j) \mid i = n \text{ or } j = m\}.$$

2.23 Example of an overlap alignment

Given two sequences $x =$ `acatatt` and $y =$ `ttttac`. We use 1, −1 and 2 for the match, mismatch and gap scores, respectively:

	0	a	c	a	t	a	t	t
0								
t								
t								
t								
t								
a								
c								

2.24 Dynamic programming with more complex models

So far, we have considered the case of a linear gap score. This type of scoring scheme is not ideal for biological sequences, as it penalizes additional gap steps as much as the initial one. However, gaps are often longer than one residue and one would like a scheme that makes it expensive to open a gap, but once open, makes it less expensive to extend a gap.

The simplest solution is to let the gap score be a function γ of the length of the gap, like this:

$$F(i,j) := \max \begin{cases} F(i-1,j-1) + s(x_i, y_j), & \\ F(k,j) + \gamma(i-k), & k = 0,\dots,i-1, \\ F(i,k) + \gamma(j-k), & k = 0,\dots,j-1. \end{cases}$$

Problem: requires $O(n^3)$ time, because for each cell we need to inspect $i+j+1$ predecessors.

2.25 Affine gap scores

The standard alternative to using the above recursion is to use an *affine gap score*

$$\gamma(g) = -d - (g-1)e,$$

with d the *gap-open score* and e the *gap-extension* score.

We will discuss how to modify the Needleman-Wunsch algorithm for global alignment so as to incorporate affine gap costs. Approach is due to Osamu Gotoh (1982).

Instead of using one matrix $F(i,j)$ to represent the best score attainable up to x_i and y_j, we will now use three matrices M, I_x and I_y:

Case 1	Case 2	Case 3
x_i aligns to y_j:	x_i aligns to a gap:	y_j aligns to a gap:
I G A x_i	A I G A x_i	G A x_i - -
L G V y_j	G V y_j - -	S L G V y_j

1. $M(i,j)$ is the best score up to (i,j), given that x_i is aligned to y_j,
2. $I_x(i,j)$ is the best score up to (i,j), given that x_i is aligned to a gap, and
3. $I_y(i,j)$ is the best score up to (i,j), given that y_j is aligned to a gap.

2.26 Recursion for global alignment with affine gap costs

Initialization:

$$\begin{aligned} &M(0,0) = 0,\ I_x(0,0) = I_y(0,0) = -\infty \\ &M(i,0) = I_x(i,0) = -d - (i-1)e,\ I_y(i,0) = -\infty,\ \text{for } i = 1,\dots,n, \text{ and} \\ &M(0,j) = I_y(0,j) = -d - (j-1)e,\ I_x(0,j) = -\infty,\ \text{for } j = 1,\dots,m. \end{aligned}$$

Recursion:

$$
\begin{aligned}
M(i,j) &= \max\begin{cases} M(i-1,j-1)+s(x_i,y_j), \\ I_x(i-1,j-1)+s(x_i,y_j), \\ I_y(i-1,j-1)+s(x_i,y_j); \end{cases} \\
I_x(i,j) &= \max\begin{cases} M(i-1,j)-d, \\ I_x(i-1,j)-e; \end{cases} \\
I_y(i,j) &= \max\begin{cases} M(i,j-1)-d, \\ I_y(i,j-1)-e. \end{cases}
\end{aligned}
$$

2.27 Example of a global alignment with affine gap costs

Given two sequences x = `ttagat` and y = `ttgt`. We use 1, -1, 2 and 1 for the match-, mismatch-, gap-open and gap-extension scores, respectively:

		0	t	t	a	g	t
0	M I_x I_y						
t	M I_x I_y						
t	M I_x I_y						
g	M I_x I_y						
t	M I_x I_y						

2.28 Alignment in linear space

Can we compute a best alignment between two sequences
$x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, m)$ using only linear space?

The best score of an alignment is easy to compute in linear space, as $F(i,j)$ is computed locally from the values in the previous and current column only.

However, to obtain an actual alignment in linear space, we need to replace the traceback matrix.

We will discuss this the case of global alignments.

Idea: Divide-and-conquer! Consider the middle column $u = \lfloor \frac{n}{2} \rfloor$ of the F matrix. If we knew at which cell (u,v) the best scoring alignment passes through the u^{th} column, then we could split the dynamic-programming problem into two parts:

- align from $(0,0)$ to (u,v), and then
- align from (u,v) to (n,m).

$y \setminus x$	$0\ \ 1\ \ \ldots$	u	$\ldots\ \ n$
0			
1			
$\ldots$			
v		(u, v)	
$\ldots$			
m			

Concatenate the two solutions to obtain the final result.

How to determine v, the row in which a best path crosses the u^{th} column?

For $i > u$, define $c(i, j)$ such that $(u, c(i, j))$ is on an optimal path from $(1, 1)$ to (i, j).

As we compute $F(i, j)$ column for column, we update $c(i, j)$:

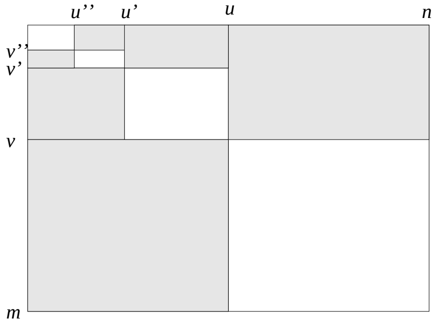
Let (i', j') be the cell from which $F(i, j)$ is obtained. Set

$$c(i, j) := \begin{cases} j', & \text{if } i' = u, \\ c(i', j'), & \text{else} \end{cases} \qquad (\text{ for } i > u).$$

.

Note that $c(i, j)$ is computed locally from the values in the previous and current column only. The final value is $v = c(n, m)$.

Once we have determined (u, v), we recurse, as indicated here:



If we implement $c(i, j)$ and $F(i, j)$ using only two vectors of length m for each, then the algorithm only requires linear space. We obtain the actual alignment as a sequence of pairs $(1, v_1), (2, v_2), \ldots (n, v_n)$.

What is the time complexity? We first look at $1 \times nm$ cells, then at $\frac{1}{2}nm$ cells, then at $\frac{1}{4}nm$ cells etc. As $\sum_{i=0}^{n} \frac{1}{2^i} < 2$, this algorithm is only twice as slow as the quadratic-space one!

2.29 Simplifying the affine-gap algorithm

In practice, the affine-gap formulation of the dynamic programs for sequence alignment usually use only two matrices, M and I, corresponding to an alignment of two symbols and an insertion in one of the two sequences, respectively. For global alignment, the recursions are:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ I(i-1,j-1) + s(x_i, y_j); \end{cases}$$

$$I(i,j) = \max \begin{cases} M(i-1,j) - d, \\ I(i-1,j) - e, \\ M(i,j-1) - d, \\ I(i,j-1) - e. \end{cases}$$

This produces the same result as the original algorithm, if the lowest mismatch score is $> -2e$. But, even if this does not hold, then the difference in score and alignment will be insignificant (in a poorly matched gapped region).

Assume that the original algorithm (using M, I_x and I_y) produces the following optimal alignment:

```
x x x x - - - a x x x x x x x x
y y y y y y y b - - - - y y y y
```

If $s(a,b) < -2e$, then the modified algorithm (using M and I) will produce the following higher scoring alignment, adding a gap before `a` and one after `b`:

```
x x x x - - - - a x x x x x x x x
y y y y y y y b - - - - - y y y y
```

However, situations in which $\binom{-}{b}$ is directly followed by $\binom{a}{-}$ (or vice-versa) are uninteresting and so the original algorithm rules them out.

2.30 Where do we stand?

So far, we have discussed:

- the dot matrix for visual comparision of two sequences,
- global alignments and the Needleman-Wunsch algorithm,
- local alignments and the Smith-Waterman algorithm, and
- overlap alignments and a corresponding dynamic program.

A naive implementation of any of the dynamic programming algorithms uses $O(nm)$ time and $O(nm)$ space. We saw that:

- the space complexity can be reduced to $O(n)$.

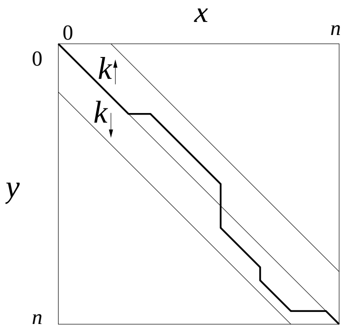
Now we ask:

- can we reduce the time complexity, too?

2.31 Banded global alignment

For simplicity, we consider DNA sequences, assume $n = m$ and use a linear gap score d.

Idea: Instead of computing the whole matrix F, use only a *band* of cells along the main diagonal:



Let $2k$ denote the height of the band. Obviously, the time complexity of the banded algorithm will be $O(kn)$.

Questions: Will this algorithm produce an optimal global alignment? What should k be set to?

2.32 The KBand algorithm

Input: two sequences x and y of equal length n, integer k
Output: best score α of global alignment at most k diagonals away from main diagonal
Initialization: Set $F(i,0) := -id$ for all $i = 0,1,2,\ldots,k$.
Set $F(0,j) := -jd$ for all $j = 0,1,2,\ldots,k$.

```
for $i = 1$ to $n$ do
    for $h = -k$ to $k$ do
        $j := i + h$
        if $1 \leq j \leq n$ then
            $F(i,j) := F(i-1,j-1) + s(x_i, y_j)$
            if insideBand$(i-1,j,k)$ then
                $F(i,j) := \max\{F(i,j), F(i-1,j) - d\}$
            if insideBand$(i,j-1,k)$ then
                $F(i,j) := \max\{F(i,j), F(i,j-1) - d\}$
return $F(n,n)$
```

To test whether (i,j) is inside the band, we use:

$$\text{insideBand}(i,j,k) := (-k \leq i - j \leq k).$$

2.33 Searching for high-identity alignments

We can use the KBand algorithm as a fast method for finding high-identity alignments:

If we know that the two input sequences are highly similar and we have a bound b on the number of gaps that will occur in the best alignment, then the KBand algorithm with $k = b$ will compute an optimal alignment.

For example, in forensics, one must sometimes determine whether a sample of human mtDNA obtained from a victim matches a sample obtained from a relative (or from a hair brush etc). If two such sequences differ by more than a couple of base-pairs or gaps, then they are not considered a match.

2.34 Optimal alignments using KBand

Given two sequences x and y of the same length n. Let M be the match score and d the gap penalty.

Question: Let α_k be the best score obtained using the KBand algorithm for a given k. When is α_k equal to the optimal global alignment score α?

Lemma If $\alpha_k \geq M(n-k-1) - 2(k+1)d$, then $\alpha_k = \alpha$.

Proof If there exists an optimal alignment with score α that does not leave the band, then clearly $\alpha_k = \alpha$. Else, all optimal alignments leave the band somewhere. This requires insertion of at least $k+1$ gaps in each sequence, and allows only at most $n-k-1$ matches, giving the desired bound. □

2.35 Optimal alignment using repeated KBand

The following algorithm computes an optimal alignment by repeated application of the KBand algorithm, with larger and larger k:

Input: two sequences x and y of the same length
Output: an optimal global alignment of x and y

```
Initialize $k := 1$
repeat
        compute $\alpha_k$ using KBand
        if $\alpha_k \geq M(n-k-1) - 2(k+1)d$ then
          return $\alpha_k$
        $k := 2k$
end
```

As usual, we omit details of the traceback.

2.36 Analysis of time complexity

The algorithm terminates when:

$$\alpha_k \geq M(n-k-1) - 2(k+1)d \qquad \Leftrightarrow$$

$$\alpha_k - Mn + M + 2d \geq -(M+2d)k \qquad \Leftrightarrow$$

$$-\alpha_k + Mn - (M+2d) \leq (M+2d)k \qquad \Leftrightarrow$$

$$\frac{Mn - \alpha_k}{M+2d} - 1 \leq k$$

At this point, the total complexity is:

$$n + 2n + 4n + \ldots + kn \leq 2kn.$$

So far, this doesn't look better than nn. To bound the total complexity, we need a bound on k.

When the algorithm stops for k, we must have:

$$\frac{k}{2} < \frac{Mn - \alpha_{\frac{k}{2}}}{M + 2d} - 1.$$

There are two cases: If $\alpha_{\frac{k}{2}} = \alpha_k = \alpha$, then

$$k < 2(\frac{Mn - \alpha}{M + 2d} - 1).$$

Otherwise, $\alpha_{\frac{k}{2}} < \alpha_k = \alpha$. Then any optimal alignment must have more than $\frac{k}{2}$ spaces, and thus

$$\alpha \leq M(n - \frac{k}{2} - 1) + 2(\frac{k}{2} + 1)d \quad \Rightarrow \quad k \leq 2(\frac{Mn - \alpha}{M + 2d} - 1).$$

As $M + 2d$ is a constant, it follows that k is bounded by $O(\Delta)$, with $\Delta = Mn - \alpha$, and thus the total bound is $O(\Delta n)$. □

In consequence, the more similar the sequences, the faster the KBand algorithm will run!

We can easily generalize to $n \neq m$. Space saving, how much do we need?

3 Multiple alignment

Sources for this lecture:

- P. Clote and R. Backofen. Computational molecular biology, 2000.
- R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological sequence analysis, Cambridge, 1998
- D. Gusfield, Algorithms on string, trees and sequences, 1997.
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, 2001.
- J. Setubal & J. Meidanis, Introduction to computational molecular biology, 1997.
- M. Waterman. Introduction to computational biology, 1995.

3.1 What is a multiple sequence alignment (msa)?

A multiple sequence alignment is simply an alignment of more than two sequences, like this:

```
SRC_RSVP    -FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERG
YES_AVISY   -FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELVTKGRVPYPGMVNR-EVLEQVERG
ABL_MLVAB   -FPIKWTAPESLAY---NKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKD
FES_FSVGA   QVPVKWTAPEALNY---GRYSSESDVWSFGILLWETFSLGASPYPNLSNQ-QTREFVEKG
FPS_FUJSV   QIPVKWTAPEALNY---GWYSSESDVWSFGILLWEAFSLGAVPYANLSNQ-QTREAIEQG
KRAF_MSV36  TGSVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRG
```

(A small section of six tyrosine kinase protein sequences.)

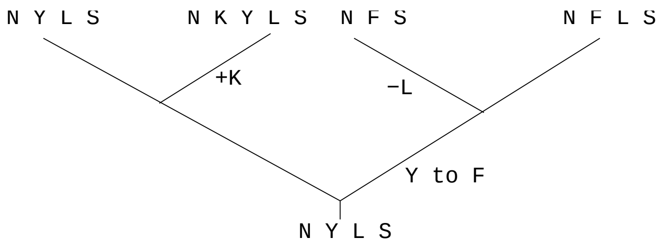
Multiple sequence alignment is applied to a set of sequences that are assumed to be homologous (have a common ancestor sequence) and the goal is to detect homologous residues and place them in the same column of the multiple alignment.

3.2 msa and evolutionary trees

One main application of msa's is in phylogenetic analysis. Given an msa:

$$\begin{array}{rccccc} a_1^* = & \mathtt{N} & \mathtt{-} & \mathtt{F} & \mathtt{L} & \mathtt{S} \\ a_2^* = & \mathtt{N} & \mathtt{-} & \mathtt{F} & \mathtt{-} & \mathtt{S} \\ a_3^* = & \mathtt{N} & \mathtt{K} & \mathtt{Y} & \mathtt{L} & \mathtt{S} \\ a_4^* = & \mathtt{N} & \mathtt{-} & \mathtt{Y} & \mathtt{L} & \mathtt{S} \end{array}$$

We would like to reconstruct the evolutionary tree that gave rise to these sequences, e.g.:



The problem of computing phylogenetic trees will be discussed later.

3.3 Protein families

Assume we have established a family $a_1, a_2, \ldots, a_r$ of homologous protein sequences. Does a new sequence a_0 belong to the family?

One method of answering this question would be to align a_0 to each of $a_1, \ldots, a_r$ in turn. If one of these alignments produces a high score, then we may decide that a_0 belongs to the family.

However, perhaps a_0 does not align particularly well to any one specific family member, but does well in a multiple alignment, due to common motifs etc.

We will return to this type of problem later.

3.4 Definition of an msa

Suppose we are given r sequences

$$A := \begin{cases} a_1 = (a_{11}, a_{12}, \ldots, a_{1n_1}) \\ a_2 = (a_{21}, a_{22}, \ldots, a_{2n_2}) \\ \qquad \ldots \\ a_r = (a_{r1}, a_{r2}, \ldots, a_{rn_r}) \end{cases}$$

A *multiple sequence alignment (msa)* is obtained by inserting gaps ('-') into the original sequences so as to produce a configuration of the form:

$$A^* := \begin{cases} a_1^* = (a_{11}^*, a_{12}^*, \ldots, a_{1L}^*) \\ a_2^* = (a_{21}^*, a_{22}^*, \ldots, a_{2L}^*) \\ \qquad \ldots \\ a_r^* = (a_{r1}^*, a_{r2}^*, \ldots, a_{rL}^*), \end{cases}$$

such that no column consists of -'s only.

Note that the aligned sequences a_i^* all have the same length

$$L \geq \max\{n_i \mid i = 1, \ldots, r\}.$$

3.5 Scoring an msa

In the case of a linear gap penalty, if we assume independence of the different columns of an msa, then the score $\alpha(A^*)$ of an msa A^* can be defined as a sum of column scores:

$$\alpha(A^*) := \sum_{i=1}^{L} s(a_{1i}, a_{2i}, \ldots, a_{ri}).$$

Here we assume that $s(a_{1i}, a_{2i}, \ldots, a_{ri})$ is a function that returns a score for every combination of r symbols (including the gap symbol).

How to define s? For two protein sequences, s is usually given by a BLOSUM or PAM matrix. For more than two sequences, providing such a matrix is not practical, as the number of possible combinations is too large.

3.6 The sum-of-pairs (SP) score

Given an msa A^*. Consider two sequences a_p^* and a_q^* in the alignment. For two aligned symbols u and v we define:

$$s(u,v) := \begin{cases} \text{match score for } u \text{ and } v, & \text{if } u \text{ and } v \text{ are residues,} \\ -d & \text{if either } u \text{ or } v \text{ is a gap, or} \\ 0 & \text{if both } u \text{ and } v \text{ are gaps.} \end{cases}$$

(Note that $u = -$ and $v = -$ can occur simultaneously.)

We define the *sum-of-pairs (SP) score* as:

$$\alpha_{SP}(A^*) := \sum_{1 \le p < q \le r} s(a_p^*, a_q^*) = \sum_{i=1}^{L} s_{SP}(a_{1i}^*, a_{2i}^*, \dots, a_{ri}^*),$$

with

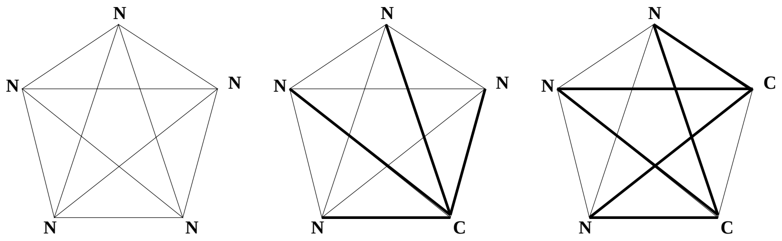
$$s(a_p^*, a_q^*) := \sum_{i=1}^{L} s(a_{pi}^*, a_{qi}^*) \text{ and } s_{SP}(a_{1i}^*, \dots, a_{ri}^*) := \sum_{1 \le p < q \le r} s(a_{pi}^*, a_{qi}^*).$$

Thus, we obtain a score for a multiple alignment based on a pairwise-scoring matrix.

Multiple alignment:

```
                (1)       (2)       (3)
Seq. 1   ...    N    ...  N    ...  N    ...
Seq. 2   ...    N    ...  N    ...  N    ...
Seq. 3   ...    N    ...  N    ...  N    ...
Seq. 4   ...    N    ...  N    ...  C    ...
Seq. 5   ...    N    ...  C    ...  C    ...
```

Comparisons: (1) (2) (3)

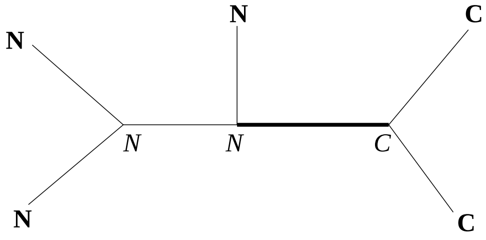


N-N pairs:	10	6	3
N-C pairs:	0	4	6
BLOSUM50:	70	34	22

(BLOSUM50 scores: N-N: 7, N-C: -2, C-C: 13)

3.7 Scoring along a tree

Assume we have a *phylogenetic tree* T for the sequences that we want to align, i.e. a tree whose leaves are labeled by the sequences. Instead of comparing *all pairs* of residues in a column of a msa, one may instead determine an optimal labeling of the internal nodes of the tree by symbols in a given column (in this case (3)) and then sum over all edges in the tree:

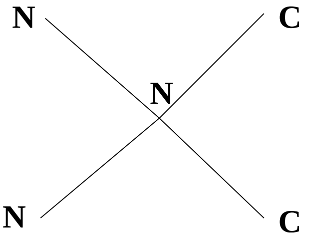


Such an optimal *most parsimonious labeling* of internal nodes can be computed in polynomial time using the *Fitch* algorithm, which we will discuss in detail later.

Based on this tree, the scores for columns (1), (2) and (3) are: $7 \times 7 = 49$, $6 \times 7 - 2 = 40$ and $4 \times 7 - 2 + 2 \times 13 = 52$.

3.8 Scoring along a star

In a third alternative, one sequence is treated as the ancestor of all others others in a so-called *star phylogeny*:



Based on this star phylogeny, assuming that sequence 1 is at the center of the star, the scores for columns (1), (2) and (3) are: $4 \times 7 = 28$, $3 \times 7 - 2 = 19$ and $2 \times 7 - 2 \times 2 = 10$.

At present, there is no conclusive argument that gives any one scoring scheme more justification than the others. The sum-of-pairs score is widely used, but is problematic.

3.9 An undesirable property of the SP score

Consider a position i in an SP-optimal multi-alignment A^* that is *constant*, i.e., has the same residue in all sequences.

What happens when we add a new sequence? If the number of aligned sequences is small, then we would not be too surprised, if the new sequence shows a different residue at the previously constant position i.

However, if the number of sequences is large, then we would expect the constant position i to remain constant, if possible.

Unfortunately, the SP score favors the opposite behavior, when scoring using a BLOSUM matrix: the more sequences there are in an msa, the easier it is, relatively speaking, for a differing residue to be placed in an otherwise constant column.

Example:

$$\begin{array}{rl} a_1^* = & \ldots \quad \texttt{E} \quad \ldots \\ a_2^* = & \ldots \quad \texttt{E} \quad \ldots \\ \ldots & \\ a_{r-1}^* = & \ldots \quad \texttt{E} \quad \ldots \\ a_r^* = & \ldots \quad \texttt{E} \quad \ldots \end{array} \quad \leftrightarrow \quad \begin{array}{rl} a_1^* = & \ldots \quad \texttt{E} \quad \ldots \\ a_2^* = & \ldots \quad \texttt{E} \quad \ldots \\ \ldots & \\ a_{r-1}^* = & \ldots \quad \texttt{E} \quad \ldots \\ a_r^* = & \ldots \quad \texttt{C} \quad \ldots \end{array}$$

Using BLOSUM50, we obtain:

$$s_{SP}(\mathtt{E}^r) = \frac{r(r-1)}{2} s(\mathtt{E},\mathtt{E}) = 6\frac{r(r-1)}{2}.$$

The value for $s_{SP}(\mathtt{E}^{r-1}\mathtt{C})$ is obtained from this by subtracting $(r-1)s(\mathtt{E},\mathtt{E})$ and then adding $(r-1)s(\mathtt{E},\mathtt{C})$. So, the difference between $s_{SP}(\mathtt{E}^r)$ and $s_{SP}(\mathtt{E}^{r-1},\mathtt{C})$ is:

$$(r-1)s(\mathtt{E},\mathtt{E}) - (r-1)s(\mathtt{E},\mathtt{C}) = (r-1)6 - (r-1)(-3) = 9(r-1).$$

Therefore, the relative difference is

$$\frac{s_{SP}(\mathtt{E}^r) - s_{SP}(\mathtt{E}^{r-1}\mathtt{C})}{s_{SP}(\mathtt{E}^r)} = \frac{9(r-1)}{6r(r-1)/2} = \frac{3}{r},$$

which *decreases* as the number of sequences r increases!

3.10 The dynamic program for an msa

Although local alignments are biologically more relevant, it is easier to discuss global msa. Dynamic programs developed for pairwise alignment can be modified to msa. We discuss how to compute a global msa for three sequences, in the case of a linear gap penalty. Given:

$$A = \begin{cases} a_1 = & (a_{11}, a_{12}, \ldots, a_{1n_1}) \\ a_2 = & (a_{21}, a_{22}, \ldots, a_{2n_2}) \\ a_3 = & (a_{31}, a_{32}, \ldots, a_{3n_3}). \end{cases}$$

We proceed by computing the entries of a $(n1+1) \times (n2+1) \times (n3+1)$-matrix $F(i,j,k)$.

After the computation, $F(n_1, n_2, n_3)$ will contain the best score α for a global alignment A^*. As in the pairwise case, we can use traceback to recover an actual alignment.

The main recursion is:

$$F(i,j,k) = \max \begin{cases} F(i-1,j-1,k-1) + s(a_{1i}, a_{2j}, a_{3k}), \\ \\ F(i-1,j-1,k) + s(a_{1i}, a_{2j}, -), \\ F(i-1,j,k-1) + s(a_{1i}, -, a_{3k}), \\ F(i,j-1,k-1) + s(-, a_{2j}, a_{3k}), \\ \\ F(i-1,j,k) + s(a_{1i}, -, -), \\ F(i,j-1,k) + s(-, a_{2j}, -), \\ F(i,j,k-1) + s(-, -, a_{3k}), \end{cases}$$

$$\text{for } 1 \leq i \leq n_1, 1 \leq j \leq n_2, 1 \leq k \leq n_3,$$

where $s(a,b,c)$ returns a score for a given column of symbols a,b,c; for example, $s = s_{SP}$, the sum-of-pairs score.

Clearly, this algorithm generalizes to r sequences.

It has time complexity $O(n^r)$, where n is the sequence length. Hence, it is only practical for small numbers of short sequences.

Computing an msa with optimal SP-score has been proved to be NP-complete (so, most likely, no polynomial time algorithm exists that solves this problem).

Question: how to initialize the boundary cells?

3.11 Minimizing the distance between sequences

So far, we have studied alignment in the light of *maximizing a similarity score.* This approach was motivated by a probabilistic analysis that gave rise to log-likelihood rations as a measure of similarity for biological sequences.

Alternatively, one can study alignment with the goal of *minimizing a distance score.* For example, a spell-checking program may compare an unknown word found in a document with a list of known ones stored in a database and return those words as possible correct spellings, whose distance to the misspelled word is small.

In this type of formulation, we are given two sequences x and y and a set of edit operations and the goal is to transform x into y using a minimal number of operations.

3.12 Edit distance

Given two sequences x and y. The *edit distance* between the two is obtained using three different operations, namely

- *insertion (I)* of a character into the first string x,
- *deletion (D)* of a character from the first string x, pr
- *replacement (R)* of a character in the first string x by a character in the second string.

The first two types of operations are collectively known as *indels.* We use M to denote the match of a character in x to one in y.

Example:

```
transcript: R I M D M D M M I
         x: v   i n t n e r
         y: w r i   t   e r s
```

Definition A string over the alphabet I, D, R and M that describes a transformation of one string x into another y is called an *(edit) transcript* of the two strings.

Definition The *edit distance* (or *Levenshtein distance*) between two strings x and y is the minimum number $D(x, y)$ of indels and replacements needed to transform x into y.

Definition The *edit distance problem* is to compute the edit distance between two given strings, along with an optimal transcript.

Definition A scoring scheme satisfies the *triangle inequality*, if $s(x, z) \leq s(x, y) + s(y, z)$ for any three characters x, y, z.

The edit distance problem can be solved by dynamic programming. *The details are an exercise.*

3.13 String alignment

An edit transcript is one way to represent a transformation of a string x into a string y. Alternatively, such a transformation can be displayed as an explicit global alignment:

```
R I M D M D M M I
v   i n t n e r
w r i   t   e r s
```

$$\Leftrightarrow$$

```
v - i n t n e r -
w r i - t - e r s
```

Note that both descriptions are mathematically equivalent. However, an edit transcript determines how sequence x transforms to sequence y, whereas an alignment simply displays the relationship between the two sequences. The distinction is between *process* and *product*.

3.14 Generalizations of edit distance

In the *operation-weighted* edit distance, each type of operation I, D, R and M is given a specific weight and the distance is calculated using the sum of weights of the operations in a transcript.

In the *character-weighted* edit distance, additionally, the weight of a replacement R or match M depends on the two characters involved.

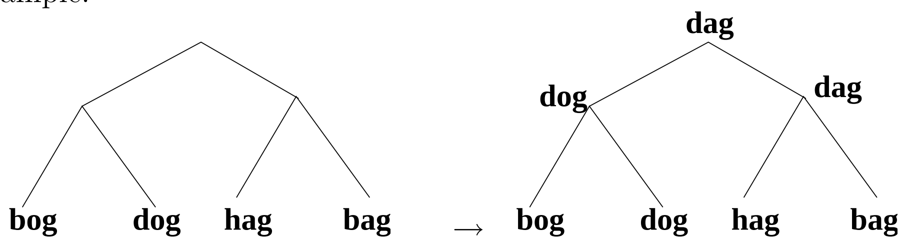
When comparing DNA sequences, the operation-weighted edit distance is usually used. For example, the default values for BLAST are $+5$ for M and -4 for R.

When comparing protein sequences, the alphabet-weight edit distance is used, using distance scores based on the BLOSUM or PAM matrices.

3.15 Multiple alignment to a tree

Given a set of distinct sequences $A = \{a_1, a_2, \dots, a_r\}$ and a tree T, whose leaves are labeled by the sequences. A *phylogenetic alignment* for T, is an assignment of a string to each internal node. In the following we will use T to refer to a phylogenetic alignment of T and we will not always distinguish between the label of a node and the node itself.

Example:



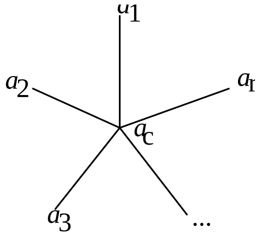
Tree T with sequences at leaves. A phylogenetic alignment.

If $e = (x, y)$ is an edge that joins sequences x and y, then we define the *edge distance* of e as $D(x, y)$. The *distance $D(T)$ of a phylogenetic alignment* T is the sum of it's edge distances.

Definition A phylogenetic alignment T is called *consistent*, if for each pair of sequences x and y that label adjacent nodes, the induced alignment has score $D(x, y)$.

3.16 Star approx. of SP alignment

Given a set of sequences $A = \{a_1, \dots, a_r\}$. Choose a *center* string $a_c \in A$ for which $\sum_{p \neq c} D(a_c, a_p)$ is minimal and place it at the center of a star tree T_c, labeling all leaves with the remaining sequences:



Let A_c^* denote the multiple alignment consistent with the center star tree T_c.

Theorem If the pairwise distances satisfy the triangle inequality, then $D(A_c^*) < 2D_{sp}(A^*)$. That is, the center star alignment approximates the best SP score within a factor of 2.
(Proof: see Gusfield, pg. 350)

3.17 The phylogenetic alignment problem

Phylogenetic alignment problem Given a set of distinct sequences A that label the leaves of a tree T, find an assignment of strings to internal nodes of T that minimizes the distance of the alignment.

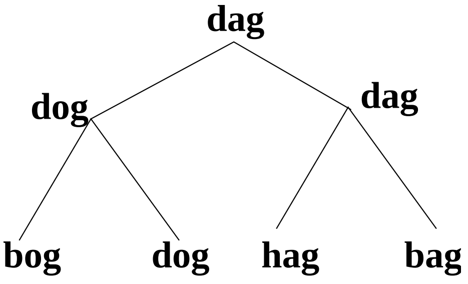
The general phylogenetic alignment problem is NP-complete. We will look at a polynomial method that obtains a result that has distance at most twice the optimal distance.

(This is under the reasonable assumption that the pairwise distance function D satisfies the triangle inequality.)

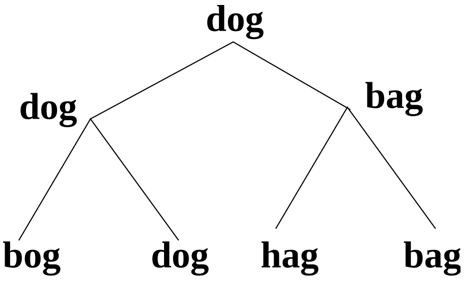
3.18 Lifted phylogenetic alignment

A tree alignment is called a *lifted alignment*, if for every internal node v, the string assigned to v is also assigned to a child of v.

Example:



Not lifted.



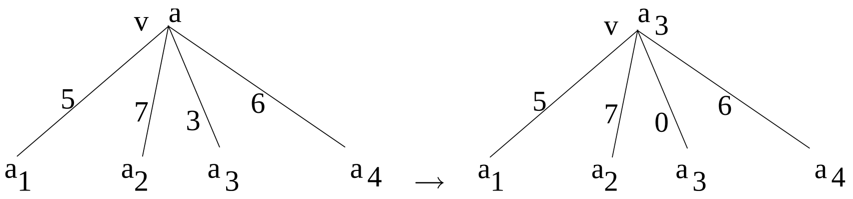
Lifted.

3.19 Obtaining a lifted alignment from an optimal one

Given sequences $A = \{a_1, \ldots, a_r\}$ and leaf-labeled tree T. Assume that we have an optimal phylogenetic alignment T^*.

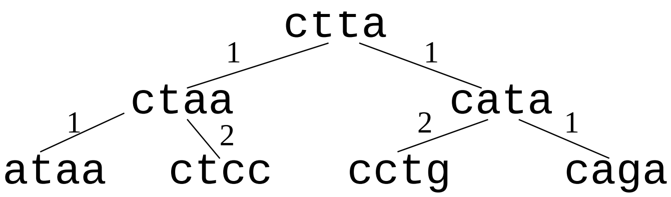
Assume that, initially, all nodes are labeled by T^*. In the following construction, we say a node is *lifted*, if it is labeled by a string in the leaf set A. By definition, all leaves are lifted.

Bottom-up, we lift each node v whose children are all lifted. To do so, we replace the label of v by the label of a child that has shortest distance to v, e.g:

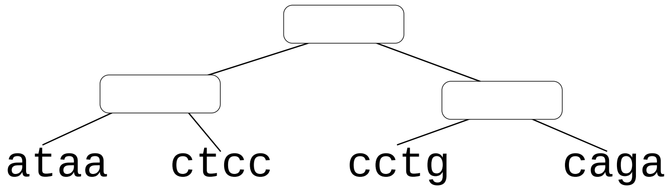


This transforms T^* into a lifted alignment T^L.

Example (no indels), optimal phylogenetic alignment T^*:



Obtained lifted alignment T^L:



This construction is not performed in practice, because if an optimal alignment is known, why bother to compute a suboptimal one from it? But it is conceptually useful, because it gives rise to an error bound for an optimal lifted alignment.

3.20 Error analysis

Theorem Given the assumptions above. The total distance of T^L is at most twice the distance of an optimal alignment T^*.

Proof Consider an edge $e = (v, w)$ in T and let a_j be the label of v, and a_i the label of w, in the lifted alignment T^L.

If $a_i = a_j$, then the distance of e in T^L is 0.

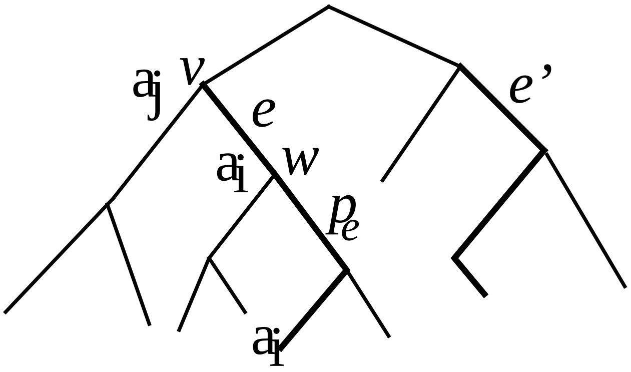
Now, assume that $a_i \neq a_j$. Then we have:

$$D(a_j, a_i) \leq D(a_j, a_v^*) + D(a_v^*, a_i),$$

where a_v^* is the label of v in T^*, by the triangle inequality. This implies

$$D(a_j, a_i) \leq 2 \times D(a_v^*, a_i),$$

because at the point in the lifting process where v was labeled, a_i was a candidate and so we must have $D(a_v^*, a_j) \leq D(a_v^*, a_i)$.



Now consider the path in T^* from v to leaf a_i and call it p_e. By the triangle inequality, $D(a_v^*, a_i)$ is at most the sum of edge distances along p_e in T^*.

Hence, using the inequality established above, the distance $D(a_j, a_i)$ of the edge e in T^L is at most twice the total distance of edges of p_e in T^*.

By construction, the following statements hold: The node v is the only node in p_e that is not labeled by a_i in T^L. Also, no node outside of p_e is labeled by a_i in T^L. If $e' = (v', w')$ is another edge with non-zero distance in T^L, then p_e and $p_{e'}$ are disjoint.

Hence, there exists a mapping between edges e with non-zero distance $D(e)$ in T^L and paths p_e in T^* such that $D(e) \leq 2 \times$ total distance of path p_e. Finally, no edge in T^* is mapped-to more than once.

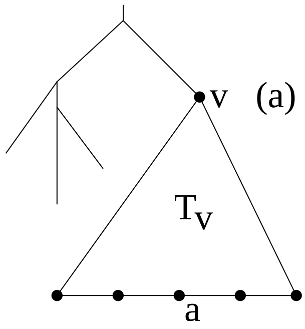
In summary, the sum of all edges distances in T^L is bounded by 2× the sum of all total disjoint-path distances in T^*. □

3.21 Computing a best lifted alignment

Corollary The best lifted alignment has distance $\leq 2D(T^*)$.

It can be computed using dynamic programming, as we will now see.

Definition Let T_v denote the subtree rooted at v. Let $d(v, a)$ denote the distance of the best lifted alignment of the tree T_v, given that the node v is labeled with sequence $a \in A$, assuming that the leaf labeled a is contained in T_v.



Bottom-up, we will compute $d(v, a)$ for all subtrees T_v and all contained leaves a. Once this has been done, then the optimal lifted alignment score can be found at the root node r:

$$\alpha_L(T) = \min_{a \in A}\{d(r, a)\}.$$

How do we compute $d(v, a)$?

- For all leaves v, set $d(v, a) := 0$, where a is the label of v.
- If v is a node whose children are all leaves, then set:

$$d(v, a) := \sum_{a' \text{ leaf under } v} D(a, a').$$

- For a general internal node, the d.p. recurrence is:

$$d(v, a) = \sum_{v' \text{ child of } v} \min_{a' \text{ leaf under } v'} \{D(a, a') + d(v', a')\}.$$

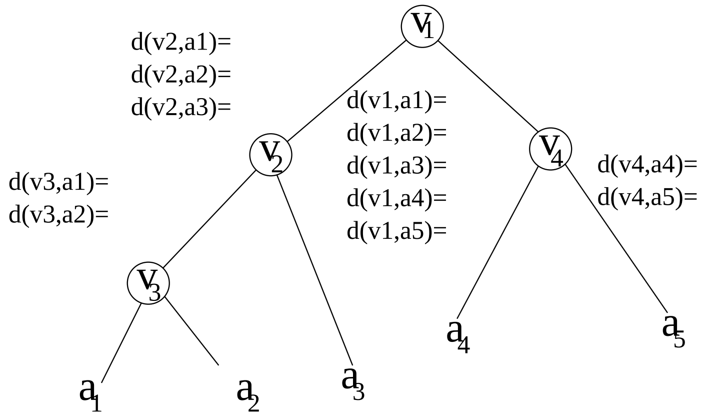
The actual lifted alignment can be found by traceback.

3.22 Example

Given $A = \{a_1, a_2, \ldots, a_5\}$ with pairwise alignment distances:

	a_1	a_2	a_3	a_4	a_5
a_1	0	2	2	3	4
a_2	2	0	1	3	5
a_3	2	1	0	2	3
a_4	3	3	2	0	2
a_5	4	5	3	2	0

Bottom-up, compute the values for $d(v, a)$:



3.23 Time analysis

Let k be the number of sequences (i.e., leaves). In a preprocessing step, we compute all $\binom{k}{2}$ pairwise distances $D(a, a')$. This takes $O(N^2)$ time, where N is the total length of all sequences.

At each internal node, we perform $O(k^2)$ operations.

As there are $O(k)$ internal nodes, we obtain a time complexity of $O(N^2 + k^3)$, which can be reduced to $O(N^2 + k^2)$ by a slight modification of the main step.

In summary, although the optimal phylogenetic alignment problem is NP-complete, an approximation of within a factor of 2 can be computed in polynomial time.

3.24 Profile alignment

Given two msa (called *profiles* in this context) $A_1 = (a_1, \ldots, a_r)$ and $A_2 = (a_{r+1}, \ldots, a_n)$. Here, we discuss the alignment of profiles in the case of the SP-score and linear gap scores. In this case, we can set $s(-, a) = s(a, -) = g$ and $s(-, -) = 0$ for all $a \in A_1$ or A_2.

A *profile alignment* of A_1 and A_2 is an msa

$$A^* = \left\{ \begin{array}{lcllll} a_1^* & = & a_{11}^*, & a_{12}^*, & \ldots, & a_{1L}^* \\ & \ldots & & & & \\ a_r^* & = & a_{r1}^*, & a_{r2}^*, & \ldots, & a_{rL}^* \\ & & & & & \\ a_{r+1}^* & = & a_{r+1,1}^*, & a_{r+1,2}^*, & \ldots, & a_{r+1,L}^* \\ & \ldots & & & & \\ a_n^* & = & a_{n1}^*, & a_{n2}^*, & \ldots, & a_{nL}^*, \end{array} \right.$$

obtained by inserting gaps in *whole columns* of A_1 or A_2, without changing the alignment of either of the two profiles.

The SP-score of the global alignment A^* is:

$$\sum_{1\leq p<q\leq n} \sum_{i=1}^{L} s(a^*_{pi}, a^*_{qi}) = \sum_{i=1}^{L} \sum_{1\leq p<q\leq n} s(a^*_{pi}, a^*_{qi}) =$$

$$\sum_{i=1}^{L} \sum_{1\leq p<q\leq r} s(a^*_{pi}, a^*_{qj}) + \sum_{i=1}^{L} \sum_{r<p<q\leq n} s(a^*_{pi}, a^*_{qj}) + \sum_{i=1}^{L} \sum_{1\leq p\leq r<q\leq n} s(a^*_{pi}, a^*_{qj}).$$

The first two sums of the latter are the alignment scores of A_1 and A_2. The third sum consists of all cross terms and can be optimized using standard pairwise alignment, with the modification that columns are scored against columns by adding their pair scores.

Clearly, either or both profiles may consist of a single sequence. In the former case, we are aligning a single sequence to a profile and in the latter case, we are simply aligning two sequences.

3.25 Progressive alignment

Probably, the most commonly used approach to msa is *progressive alignment.* In general, this works by constructing a series of pairwise alignments, first starting with pairs of sequences and then later also aligning sequences to existing alignments (profiles) and profiles to profiles.

Progressive alignment is a heuristic and does not directly optimize any known global scoring function of alignment correctness. However, it is fast and efficient, and often provides reasonable results.

3.26 CLUSTALW

CLUSTALW is one of the most popular programs for computing an msa. The weighted version, CLUSTAL**W** is the successor of CLUSTAL, which was introduced about ten years ago. It is similar to the older *Feng-Doolittle* method. Another program, PILEUP (GCG package), works similar to CLUSTALW, but lacks some of its more recent features.

Algorithm CLUSTALW progressive alignment

1. Construct a distance matrix of all $\frac{r(r-1)}{2}$ pairs by pairwise dynamic programming alignment followed by approximate conversion of similar scores to evolutionary distances.
2. Construct a *guide* tree using the *Neighbor Joining* tree-building method.
3. Progressively align sequences at nodes of tree in order of decreasing similarity, using sequence-sequence, sequence-profile and profile-profile alignment.

There are no provable performance guarantees associated with the program. However, it works well in practice and the following features contribute to its accuracy:

- Sequences are weighted to compensate for the defects of the SP score.

- The substitution matrix used is chosen based on the similarity expected of the alignment, e.g. BLOSUM80 for closely related sequences and BLOSUM50 for less related ones.
- Position-specific gap-open profile penalties are multiplied by a modifier that is a function of the residues observed at the position (hydrophobic residues give higher gap penalties than hydrophilic or flexible ones.)
- Gap-open penalties are also decreased if the position is spanned by a consecutive stretch of five or more hydrophilic residues.
- Gap-open and gap-extension penalties increase, if there are no gaps in the column, but gaps near by.
- In the progressive alignment stage, if the score of an alignment is low, then the low scoring alignment may be deferred until later.

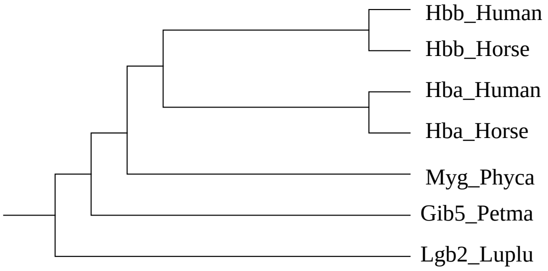
T-Coffee works similar to CLUSTALW, but retains and uses the initial pairwise alignments to produce a better alignment.

3.27 Example

For a set of sequences A, we compute all pairwise distances:

Hbb_Human	1	0					
Hbb_Horse	2	0.17	0				
Hba_Human	3	0.59	0.60	0			
Hba_Horse	4	0.59	0.59	0.13	0		
Myg_Phyca	5	0.77	0.77	0.75	0.75	0	
Gib5_Petma	6	0.81	0.82	0.73	0.74	0.80	0
Lgb2_Luplu	7	0.87	0.86	0.86	0.88	0.93	0.90
		1	2	3	4	5	6

This gives rise to the following tree:

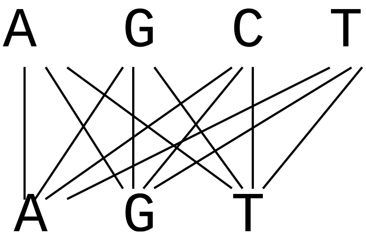


First we align Hbb_Human with Hbb_Horse and Hba_Human with Hba_Horse. Then we align the two resulting profiles etc.

3.28 The alignment graph

Given two sequences $a_1 =$ `A G C T` and $a_2 =$ `A G T` .

The *complete alignment graph* is the following bipartite graph $G = (V, E)$, with node set V and edge set E:



Each edge $e = (u, v)$ has a weight $\omega(e) = s(u, v)$, the score for placing v under u.

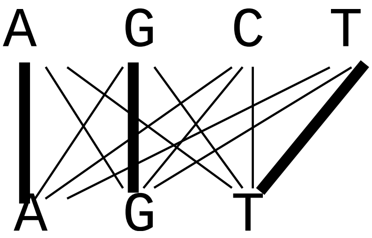
An *alignment graph* is any subgraph of the complete alignment graph.

3.29 The trace of an alignment

Given an alignment such as:

```
A G C T
A G - T
```

, we say that an edge in the alignment graph is *realized*, if the corresponding positions are aligned:



The set of realized edges is called the *trace* of the alignment. An arbitrary subset $T \subseteq E$ of edges is called a *trace*, if there exists some alignment that it realizes.

Similarly, we define the (complete) alignment graph and trace for multiple alignments. For r sequences, the resulting graph will be r-partite.

3.30 Maximum-weight trace problem

Problem Given sequences A and a corresponding alignment graph $G = (V, E)$ with edge weights ω. The maximum-weight trace problem is to find a trace $T \subseteq E$ of maximum weight.

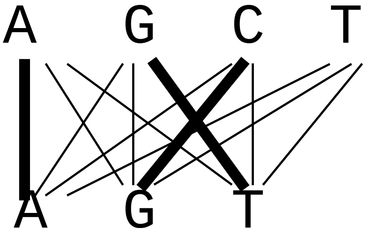
Note for two sequences, this is the maximum-weight bipartite matching problem, which can be solved in polynomial time.

3.31 Characterization of traces

We have seen that an alignment can be described by a trace in the complete alignment graph $G = (V, E)$.

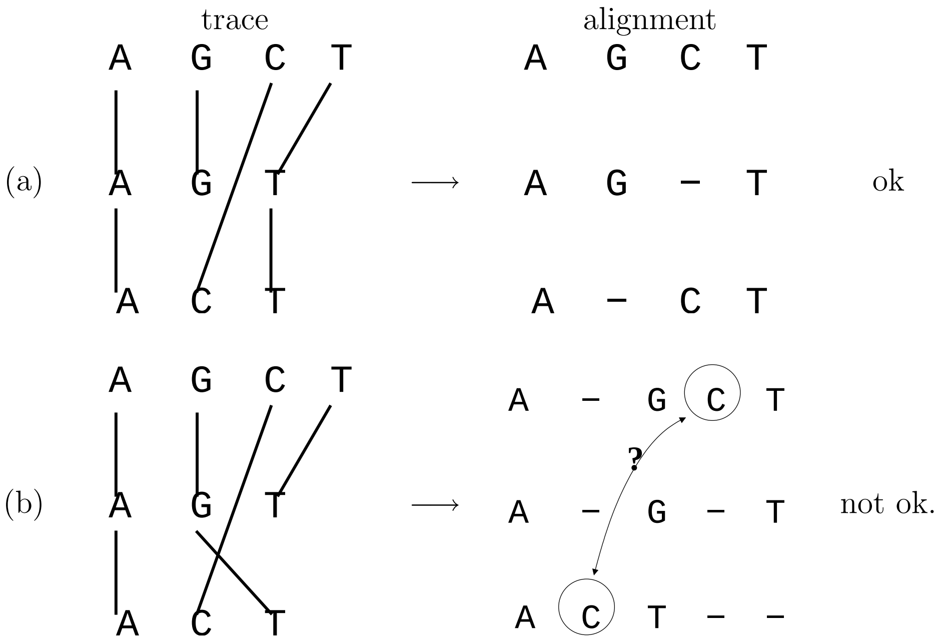
Question: Is *every* subset $T \subseteq E$ the trace of some alignment?

Clearly, the answer is *no*:



Goal: Characterize all legal traces.

Here are two examples:



3.32 Partial orders

A binary relation $\leq$ is a *partial order*, if it is

1. *reflexive*, i.e., $a \leq a$,
2. *antisymmetric*, i.e., $a \leq b$ and $b \leq a$ implies $a = b$, and
3. *transitive*, i.e., $a \leq b$ and $b \leq c$ implies $a \leq c$.

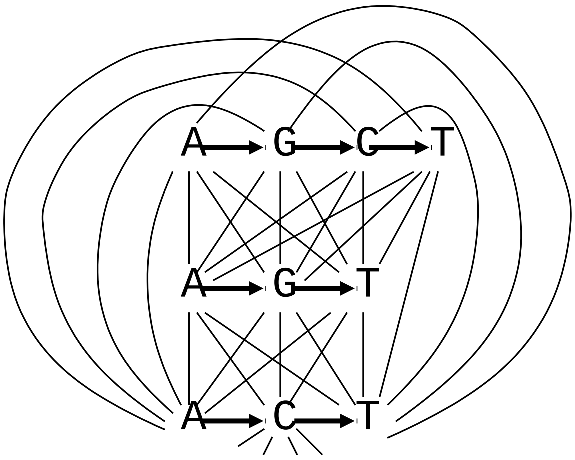
A binary relation $<$ is a *strict partial order*, if it is

1. *irreflexive*, i.e., $a \not< a$, and
2. *transitive*, i.e., $a < b$ and $b < c$ implies $a < c$.

Given a binary relation $\prec$, the *transitive closure* of $\prec$ is a binary relation $\prec^*$ such that $x \prec^* x'$ if there exists a sequence of elements $x = x_1, x_2, \ldots, x_k = x'$ with $x_1 \prec x_2 \prec \ldots x_k$.

3.33 The extended alignment graph

We define a partial order $\prec$ on the cells of the matrix $A = \{a_{ij}\}$ by writing $a_{ij} \prec a_{ij'}$, if $j' = j + 1$, and indicate the pairs $(a_{ij}, a_{i,j+1})$ by a set H of directed edges in the alignment graph:



Let $\prec^*$ denote the *transitive closure* of $\prec$, i.e. we write $a_{ij} \prec^* a_{ij'}$, if $j' > j$.

Consider two sets of nodes $X \subseteq V$ and $Y \subseteq V$. We define

$$X \lhd Y,$$

if and only if

$$\exists\, x \in X\ \exists\, y \in Y : x \prec y.$$

We define $\lhd^*$ to be the transitive closure of $\lhd$, that is, we write $X \lhd^* Y$, if for one of the sequences $a_p \in A$ we have that X contains a node representing a position a_{pj} in a_p and Y contains a node representing another position a_{pk} in a_p, with $j < k$.

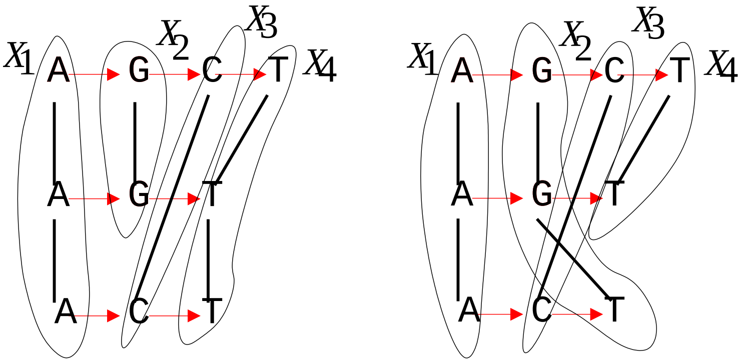
In other words, we write

$$X \lhd^* Y,$$

if and only if

$$\exists\, x \in X\ \exists\, y \in Y : x \prec^* y.$$

Consider the two examples again and define sets $X_1, X_2, \ldots$ via the two given traces:



(a) (b)

In (a), $X_1 \lhd^* X_2 \lhd^* X_3 \lhd^* X_4$ and we have a strict partial order.

In (b), we have $X_1 \lhd^* X_2, X_3, X_4$; $X_2 \lhd^* X_3, X_4$; and $X_3 \lhd^* X_2, X_4$. In this case, the condition for a partial or for a strict partial order is not fulfilled.

3.34 Characterization of traces

Theorem (John Kececioglu) Given a set of sequences A. Let $G = (V, E, H)$ be an extended alignment graph for A. A subset $T \subseteq E$ of edges is a trace, if and only if $\lhd^*$ is a strict partial order on the connected components of $G' = (V, T)$.

(Recall that a *connected component* of a graph is a maximal set of nodes $U \subseteq V$ such that any two nodes $v, u \in U$ are connected by a path of edges in the graph.)

3.35 Proof

We first prove "$\Rightarrow$": Assume that T is the trace of an alignment A^* of A, with L columns. We know that $(a_{pi}, a_{qj}) \in T$ only if the i^{th} position of sequence a_p and the j^{th} position of sequence a_q are aligned in the same column. So, all nodes contained in the same connected component of $G' = (V, T)$ correspond to symbols aligned in the same column and thus, we have the following partition of the nodes:

$$V = X_1 \cup X_2 \cup \ldots \cup X_L,$$

where X_i contains all nodes labeled a^*_{pi} for $p = 1, 2, \ldots, r$.

So clearly, $\lhd^*$ is a strict partial order on $X_1, \ldots, X_L$.

We now prove "$\Leftarrow$": Assume $T \subseteq E$ such that $\lhd^*$ is a strict partial order on the connected components of $G' = (V, T)$. Extend $\lhd^*$ to a total order $\lhd^*_{tot}$, always possible.

Let $X_1 \cup X_2 \cup \ldots \cup X_m = V$ be the connected components of $G = (E, T)$, ordered such that $i < j$ implies $X_i \lhd^*_{tot} X_j$.

For each sequence a_p, we define

$$a^*_{pj} = \begin{cases} a_{pi}, & \text{if } a_{pi} \in X_j, \\ -, & \text{else,} \end{cases}$$

obtaining $a^*_p = a^*_{p1}, \ldots a^*_{pm}$.

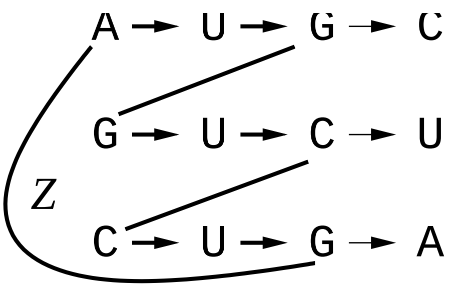
We have to show that $A^* = (a^*_1, \ldots, a^*_r)$ is an alignment.

(1) Note that a^*_p contains all positions of a_p, because each a_{pi} is contained in some component X_j and no such component can contain two different positions from the same sequence, due to the strictness of $\lhd^*$.

(2) Note that the order of the symbols in a_p is preserved in a^*_p: Assume a^*_{pi} and a^*_{pj} are two symbols whose order have been reversed in a^*_p, with $i < j$. By the strictness of $\lhd^*$, there exist two distinct components X_s and X_t, with $X_s \lhd^* X_t$, such that $a_{pi} \in X_s$ and $a_{pj} \in X_t$. On the other hand, the reversal of the order of a^*_{pi} and a^*_{pj} in a^*_p implies $X_t \lhd^* X_s$, a contradiction to the assumed strictness. □

3.36 Mixed cycles

A *mixed cycle* Z is a cycle in the extended alignment graph $G = (V, E, H)$ that contains both undirected and directed edges, from E and H, respectively, the latter all in the same direction:



A mixed cycle Z is called *critical*, if all nodes in $Z \cap a_p$ occur consecutively in Z, for all sequences $a_p \in A$. That is, the cycle enters and leaves each sequence at most once.

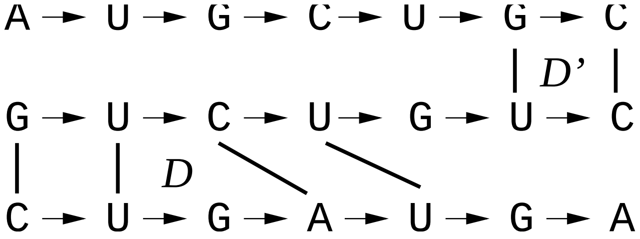
We have the following result:

Lemma A subset $T \subseteq E$ is a trace, if and only if $G' = (V, H \cup T)$ does not contain a critical mixed cycle.

3.37 Block partition

Given a set of sequences $A = \{a_1, a_2, \ldots, a_r\}$. The complete alignment graph is usually too big to be useful.

Often, we are give a set of *block matches* between pairs of the sequences a_p and a_q, where a match relates a substring of a_p and a substring of a_q via a run of *non-crossing* edges (called a *block*), as shown here for two blocks D and D':



In the following, we will assume that the edges of the alignment graph $G = (V, E)$ were obtained from a set of matches and we are given a partition of E into blocks.

3.38 Generalized maximum trace problem

Given a partition $\mathcal{D}$ of the edges of $G = (V, E)$ obtained from a set of matches. Each block D is assigned a positive weight $\omega(D)$.

Problem Given an extended alignment graph $G = (V, E, H)$ and a partition $\mathcal{D}$ of E into blocks with weights $\omega(D)$. The *generalized maximum trace problem (GMT)* is to determine a set $M \subseteq \mathcal{D}$ of maximum total weight such that the edges in $\bigcup_{D \in M} D$ do not induce a mixed cycle on G.

3.39 Simplification

Although blocks play an important role in practice, to simplify the following discussion, we will not use them explicity. However, everything that follows is easily adjusted to the case that a set of blocks is given.

3.40 Linear programming

A *linear program (LP)* consists of a set of linear inequalities,

$$\begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n & \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n & \leq b_2 \\ \ldots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots a_{mn}x_n & \leq b_m, \end{array}$$

together with an *objective function*

$$c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

to be optimized, i.e. minimized or maximized.

Linear programs can be efficiently solved using the *simplex method*, developed by George Dantzig in 1947.

There exist powerful computer programs for solving LPs, even when huge numbers of variables and inequalities are involved.

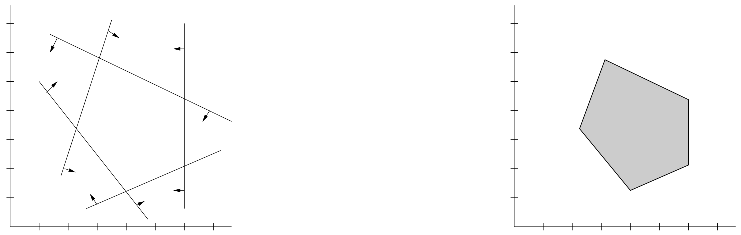
CPLEX is a very powerful commercial LP solver. Moderate size problems can be solved using `lp_solve`, which is free for academic purposes.

The inequalities describe a convex *polyhedron*, which is called a *polytope*, if it is bounded.

For example, the inequalities

$$\begin{array}{rcr} -1x_1 - 1x_2 & \leq & 5 \\ -2x_1 + 1x_2 & \leq & -1 \\ 1x_1 + 3x_2 & \leq & 18 \end{array} \qquad \begin{array}{rcr} 1x_1 + 0x_2 & \leq & 6 \\ 1x_1 - 2x_2 & \leq & 2 \end{array}$$

describe the following hyperplanes and polytope:



For example, the objective function $2x_1 - 3x_2$ takes on a maximum of 6, for $x_1 = 6$ and $x_2 = 2$, and a minimum of -9, for $x_1 = 3$ and $x_2 = 5$.

3.41 Integer linear program

An *integer linear program (ILP)* is a linear program with the additional constraint that the variables x_i are only allowed to take on integer values.

Solving ILPs has been shown to be NP-hard. (See the book by Garey and Johnson 1979, for this and many other NP-completeness results.)

There exist a number of different strategies for approximating or solving such an ILP. These strategies usually first attempt to solve *relaxations* of the original problem, which are obtained by dropping some of the inequalities. They usually also rely on the *LP-relaxation* of the ILP, which is the LP obtained by dropping the integer condition.

3.42 Integer LP for the GMT problem

How to encode the GMT problem as an integer LP?

Assume we are given an extended alignment graph $G = (V, E, H)$, with $E = \{e_1, e_2, \ldots, e_n\}$.

Each edge $e_i \in E$ is represented by a variable x_i, that will take on value 1, if e_i belongs to the best scoring trace, and 0, if not.

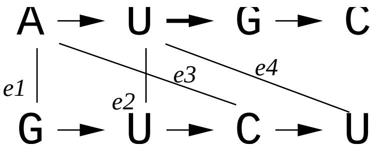
Hence, our variables are $x_1, x_2, \ldots, x_n$.

To ensure that the variables are *binary*, we add constraints $x_i \leq 1$ and $-x_i \leq 0$.

Additional inequalities must be added to prevent mixed cycles.

(This and the following is from: Knut Reinert, A Polyhedral Approach to Sequence Alignment Problems, Dissertation, Saarbrücken 1999.)

For example, consider:



There are three possible critical mixed cycles in the graph, one using e_1 and e_3, one using e_2 and e_3, and one using e_2 and e_4. We add the constraints

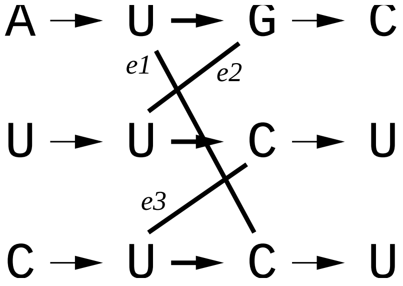
$$x_1 + x_3 \leq 1,$$

$$x_2 + x_3 \leq 1,$$

$$x_2 + x_4 \leq 1.$$

to ensure that none of the critical mixed cycles is realized.

For example, consider:



with three edges e_1, e_2 and e_3 that all participate in a critical mixed cycle. The constraint

$$x_i + x_j + x_k \leq 2$$

prevents them from being realized simutaneously.

In summary, given an extended alignment graph $G = (V, E, H)$ with $E = \{e_1, e_2, \ldots, e_n\}$, and a score ω_i defined for every edge edge $e_i \in E$.

We can obtain a solution to the GMT problem by solving the following ILP:

Maximize

$$\sum_{e_i \in E} \omega_i x_i,$$

subject to

$$\begin{array}{ll} \sum_{e_i \in C \cap E} x_i \leq |C \cap E| - 1, & \text{for all critical} \\ & \text{mixed cycles } C, \text{ and} \\ x_i \in \{0, 1\} & \text{for all variables} \\ & i = 1, \ldots, n. \end{array}$$

3.43 Solving the ILP using branch-and-cut

Solving IPs and ILPs is a main topic in combinatorial optimization. We will take a brief look at the *branch-and-cut* approach.

Branch-and-cut makes use of two techniques:

- Cutting: to solve an ILP, one considers the LP-relaxation of the problem and repeatedly cuts away parts of the polytope (by adding new constraints) in the hope of obtaining an integer solution.
- Branch-and-bound: an enumeration tree of all possible choices of parameters is partially traversed, computing local upper- and global lower-bounds, which are used to avoid parts of the tree that cannot produce the optimal value.

First note that the number of mixed cycles grows exponentially with the size of the graph. So, initially, we select a polynomial number of constraints. That is, we consider a relaxation of the original problem.

We further relax the problem by solving the LP-relaxation.

If the solution $\hat{x}$ is not an integer, or is not feasible, then we add an unused constraint to the LP to **cut** away a part of the polyhedron that contains $\hat{x}$. (This is a non-trivial operation that we won't discuss).

This is repeated until a integer solution is found that fulfills all constraints, or until we get stuck.

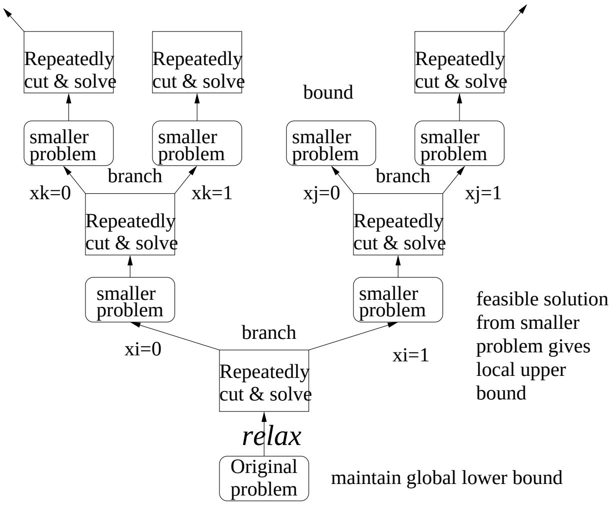
If no appropriate *cut plane* can be found, then we **branch**. That is, we choose a variable x_i and solve two sub-cases, namely the case $x_i = 0$ and the case $x_i = 1$. Repeated application produces a enumeration tree of possible cases.

We call an upper bound for the original ILP *local*, if it is obtained from considering such a subproblem in the enumeration tree.

If the solution found for a subproblem is feasible for the original problem and has a higher score that any solution found so for, then it is recorded and its value becomes the new *global lower bound* for the original objective function.

Subsequently, we only pursue subproblems whose local upper bound is greater or equal to the global lower bound.

General strategy:



As the details are quite involved, we will skip them.
(See Knut Reinert's thesis for a very clear exposition)

3.44 An ILP for pairwise alignment

We discuss how to formulate the ILP for the problem of aligning two sequences (see the Exercises).

Given two sequences $a = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_m)$. Let $s(f, g)$ denote the score for aligning symbols f and g.

The objective function that we would like to maximize is:

$$\sum_{\substack{1 \le i \le n \\ 1 \le j \le m}} s(a_i, b_j) x_{ij},$$

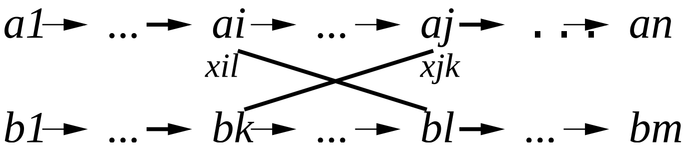
where x_{ij} is a variable that will indicate whether the edge from node a_i to node b_j belongs to the trace, or not.

To ensure that every variable x_{ij} is binary, we use the inequalities

$$x_{ij} \le 1 \text{ and } -x_{ij} \le 0,$$

for all i, j with $1 \le i \le n$ and $1 \le j \le m$.

In the case of two sequences, every critical mixed cycle is given by an ordered pair of positions (i, j) in sequence a and an ordered pair of positions (k, l) in sequence b:



This gives rise to the following set of inequalities:

$$x_{il} + x_{jk} \le 1,$$

for all i, j, k, l with $1 \le i \le j \le n$, $1 \le k \le l \le m$ and, additionally, $i \ne j$ or $k \ne l$.

For example, given sequences $a =$ `GCT` and $b =$ `GT`, and assume the match- and mismatch scores are 1 and -1, respectively.

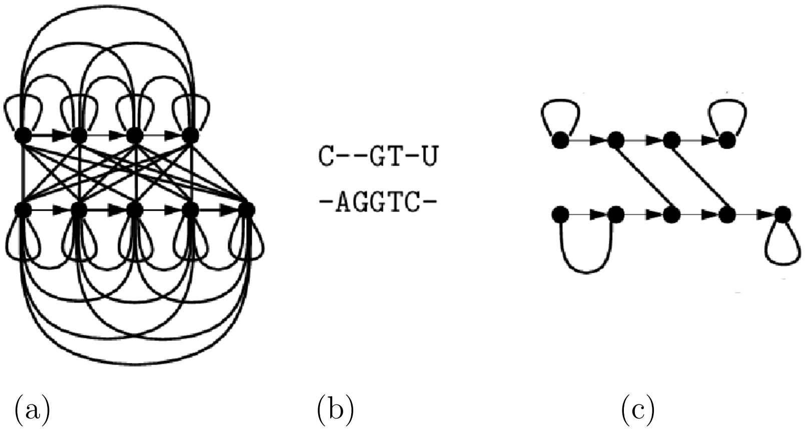
In the format used by the program `lp_solve`, the ILP has the following formulation:

```
max: +1*x1001-1*x1002-1*x2001-1*x2002-1*x3001+1*x3002;
 x1001<1;
 x1002<1;
 x2001<1;                       x1001+x3001<1;
 x2002<1;                       x1002+x3001<1;
 x3001<1;                       x1002+x3002<1;
 x3002<1;                       x2002+x2001<1;
x1002+x1001<1;                  x2001+x3001<1;
x1001+x2001<1;                  x2002+x3001<1;
x1002+x2001<1;                  x2002+x3002<1;
x1002+x2002<1;                  x3002+x3001<1;

int x1001, x1002, x2001, x2002, x3001, x3002;
```

Here, we use the variable $x_{1000i+j}$ to represent the edge from a_i to b_j for all i, j. The program `lp_solve` interprets "<" as "$\leq$" and assigns only non-negative values to variables.

3.45 The gapped extended alignment graph

The extended alignment graph $G = (V, E, H)$ does not explicitly model gaps. To allow the scoring of gaps, we add a new set B of edges to the graph, joining any two consecutive nodes in a sequence, as indicated here:



For two sequences $a =$ `CGTU` and $b =$ `AGGTC` we see (a) the gapped extended alignment graph, (b) an alignment and (c) the *gapped trace* that realizes the gapped alignment.

Given a gapped extended alignment graph $G = (V, E, H, B)$. When modeling gaps in a trace, we require that, for any *pair* of sequences, a node must be either

- incident to an alignment edge between the two sequences, or
- it must be enclosed by exactly one gap edge.

Additionally, we require that a consecutive run of gap characters is regarded as one gap. Without going into details, this gives rise to the definition of a *gapped trace* (T, C), with $T \subseteq E$ and $C \subseteq B$.

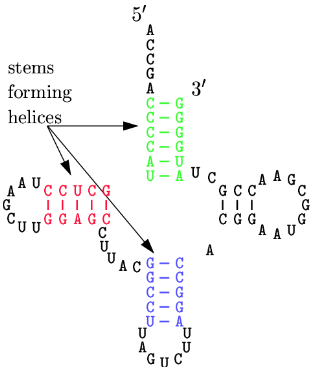
Given weights ω for all edges in E and B, a gapped trace can be scored as follows:

$$\alpha((T, C)) = \sum_{e \in T} \omega(e) - \sum_{g \in C} \omega(g).$$

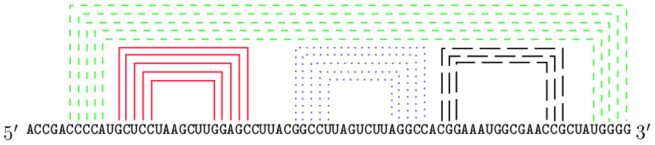
In the gapped trace formulation, it is trivial to encode, linear, affine, or any other reasonable gap cost function.

3.46 Secondary structure

RNA is a single stranded molecule that folds intra-molecularly to form hydrogen-bonded base pairs, mostly `C-G` and `A-U`. This configuration is called the *secondary structure* of RNA.



This can also be depicted as:


5′ ACCGACCCCAUGCUCCUAAGCUUGGAGCCUUACGGCCUUAGUCUUAGGCCACGGAAAUGGCGAACCGCUAUGGGG 3′

(We will discuss RNA secondary structure in detail in a later Chapter.)

3.47 Structured sequences

Given a set of sequences $A = \{a_1, \ldots, a_r\}$. How can we use information about their secondary structures to obtain biologically better alignments?

Let a_t be a sequence of length n. An *interaction* is a pair (i, j) of positions in a_t, with $a_{ti} \neq$ '-', $a_{tj} \neq$ '-', and $1 \leq i < j \leq n$. Two interactions (i, j) and (k, l) are in *conflict*, if $\{i, j\} \cap \{k, l\} \neq \emptyset$.

A set of non-conflicting interactions for a_t is called a *(secondary) structure* for a_t, and we call (a_t, P) a *structured sequence.*

The sequence and interactions depicted in the previous example form a structured sequence.

3.48 Structural alignment

Given a set of structured sequences

$$A = \{(a_1, P_1), (a_2, P_2), \ldots, (a_r, P_r)\}.$$

A *structural multiple sequence alignment* of A is a set

$$\hat{A} = \{(\hat{a}_1, \hat{P}_1), \ldots, (\hat{a}_r, \hat{P}_r)\}$$

of structured sequences such that:

1. $\{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_r\}$ is a msa for $\{a_1, a_2, \ldots, a_r\}$, and
2. for all $t \in \{1, 2, \ldots, r\}$ and all $(i, j) \in \hat{P}_t$, we have

$$(i - \#\text{gaps}(\hat{a}_{t1}, \ldots, \hat{a}_{ti}), j - \#\text{gaps}(\hat{a}_{t1}, \ldots, \hat{a}_{tj})) \in P_t.$$

The second condition ensures that the structural msa only contains interactions that are present in the input.

3.49 Scoring a structural alignment

Goal: Define a scoring scheme that reflects the biological relatedness of the sequences under consideration, taking their secondary structure into account.

In general terms, a scoring function simply assigns a real number to every possible structural alignment.

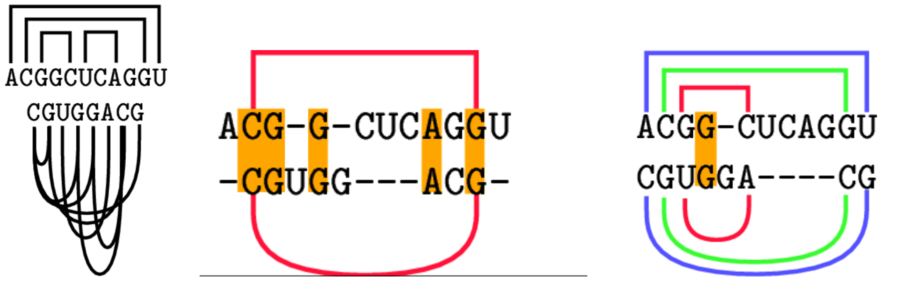
In practice, scoring functions usually consist of a weighted sum of two parts, one that evaluates the alignment of the characters in the sequences and one that evaluates the alignment of the interactions in the secondary structure.

In the pairwise case, for example, one could use a pairwise sequence alignment score *sim* to score the sequence alignment and then define an *interaction score isim* : $\Sigma^4 \to \mathbb{R}$ that assigns a score to two pairs of aligned characters that are defined by two matching interactions (where Σ denotes the alphabet of characters, excluding '-').

The following structural alignment score function $rna(\hat{A})$ can be used to identify pairwise structural alignments of RNA sequences that have both high sequence similarity and high structure conservation:

$$rna(\hat{A}) = \sum_{i=1}^{L} sim(\hat{a}_{1i}, \hat{a}_{2i}) + \sum_{\substack{(j,l) \in \hat{P}_1, (q,r) \in \hat{P}_2 \\ (q,r) = (j,l)}} isim(\hat{a}_{1j}, \hat{a}_{1l}, \hat{a}_{2q}, \hat{a}_{2r}).$$

Example:



interactions

alignment with high sequence score

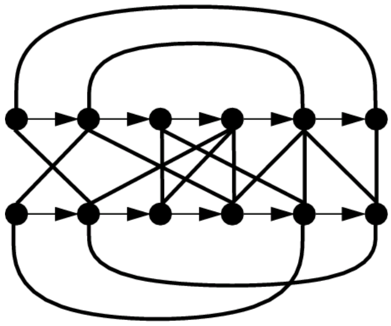
alignment with high interaction score

3.50 Structural alignment graph

Given a set of structured sequences $A = \{(a_1, P_1), \ldots, (a_r, P_r)\}$. Let $G = (V, G, H)$ denote a extended alignment graph associated with $\{a_1, \ldots, a_r\}$.

A *structural (extended) alignment graph* $G' = (V, E, H, I)$ is obtained by introducing a set of new edges I that represent all given interactions.

Example:

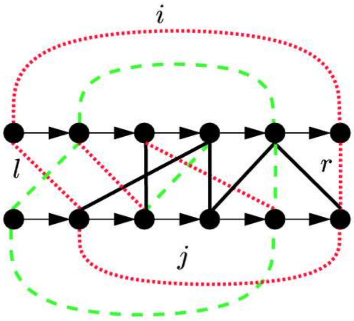


3.51 Structural traces

Let $G = (V, E, H, I)$ be a structural alignment graph for a set of structured sequences $A = \{(a_1, P_1), \dots, (a_r, P_r)\}$. A *structural trace* of G is a pair (T, B) with $T \subseteq E$ and $B \subseteq I$ such that

- the induced subgraph $(V, T \cup H)$ does not contain a critical mixed cycle, and
- there are no two conflicting edges in B.

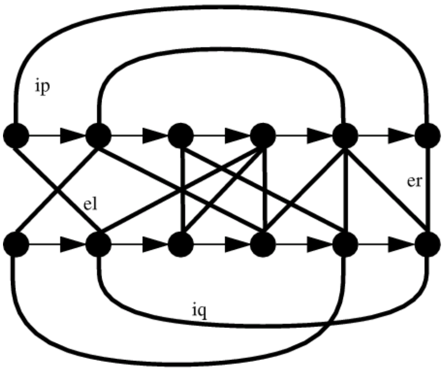
Here are two possible structural traces:



Lemma Structural traces correspond to struct. alignments.

3.52 Scoring a structural trace

Let i_p and i_q denote two interaction edges in a structural trace that are contained in the secondary structures of two difference sequences. We say that i_p *matches* i_q, if the two alignment edges e_l and e_r joining the two left respectively two right nodes of i_p and i_q are realized, i.e. if $e_l, e_r \in T$.



We call any such set $\{i_p, i_q, e_l, e_r\}$ an *interaction match.*

Let (T, B) be a structural trace. As for conventional traces, each edge $e \in T$ is given a weight $\omega(e)$ which is simply the score for aligning the two corresponding symbols.

Any interaction match $\{i_p, i_q, e_l, e_r\}$ is completely specified by the two edges e_l and e_r and so we can denote it by m_{lr} and assign a weight $\omega(l, r)$ to it. Let M denote the set of all interaction matches:

$$M = \{m_{lr} \mid m_{lr} = \{i_p, i_q, e_l, e_r\} \text{ is an interaction match in } G\}.$$

We define the *score of a structural trace* as:

$$S((T, B)) = \sum_{e \in T} \omega(e) + \sum_{\substack{i_p, i_q \in B,\, e_l, e_r \in T \\ \{i_p, i_q, e_l, e_r\} \in M}} \omega(l, r).$$

We can find the maximal scoring trace by solving an ILP.

3.53 Maximal scoring structural trace

Given a structural alignment graph $G = (V, E, H, I)$, the score ω_i for realizing an edge $e_i \in E$ and the score ω_{lr} for realizing an interaction match m_{lr}.

The problem of maximizing the score of a structural trace can be formulated as the following ILP:

$$\begin{array}{rll} \text{Maximize} & \sum_{e_i \in E} \omega_i x_i + \sum_{m_{ij} \in M} \omega_{ij} x_{ij}, & \\ \text{subject to} & \sum_{e_i \in C \cap E} x_i \leq |C \cap E| - 1, & \text{for all critical mixed cycles } C, \\ & x_{ij} \leq x_i \text{ and } x_{ij} \leq x_j, & \text{for all } i, j, \text{ and} \\ & x_i, x_{ij} \in \{0, 1\} & \text{for all variables.} \end{array}$$

Lemma Any solution to this ILP corresponds to a structural trace.

Proof The set of all variables x_i with $x_i = 1$ define a set of edges $T \subseteq E$ and the critical mixed cycle inequalities ensure that T is a trace.

The set of all variables x_{ij} with $x_{ij} = 1$ define a set of matched interactions $B \subseteq I$. To establish that B does indeed give rise to a structural trace, note that the second set of constraints in the ILP ensure that

- an interaction match m_{lr} can only be realized if both e_l and e_r are realized, and
- only one interaction match can "use" any specific alignment edge e_i as its left- or right-connecting edge, and so no conflicts can arise. □

4 RNA Secondary Structure

Sources for this lecture:

- R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological sequence analysis, Cambridge, 1998
- J. Setubal & J. Meidanis, Introduction to computational molecular biology, 1997.
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, 2001.
- M. Zuker, Algorithms in Computational Molecular Biology. Lectures, 2002, `http://www.rpi.edu/~zukerm/MATH-4961/PostScript/rnafold.ps`.

4.1 RNA

RNA, *DNA* and *proteins* are the basic molecules of life on Earth. Recall that:

- DNA is used to store and replicate genetic information,
- proteins are the basic building blocks and active players in the cell, and
- RNA plays a number of different important roles in the production of proteins from instructions encoded in the DNA.

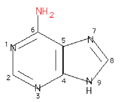
In eukaryotes, DNA is transcribed into pre-mRNA, from which introns are spliced to produce mature mRNA, which is then translated by ribosomes to produce proteins with the help of tRNAs. A substantial amount of a ribosome consists of RNA.

The *RNA-world* hypothesis suggests that originally, life was based on RNA and over time RNA delegated the data storage problem to DNA and the problem of providing structure and catalytic functionality to proteins.

4.2 RNA secondary structure

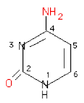
An RNA molecule is a polymer composed of four types of (ribo)nucleotides, each specified by one of four bases:

A



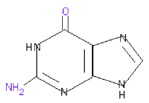
adenine

C



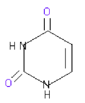
cytosine

G



guanine

U



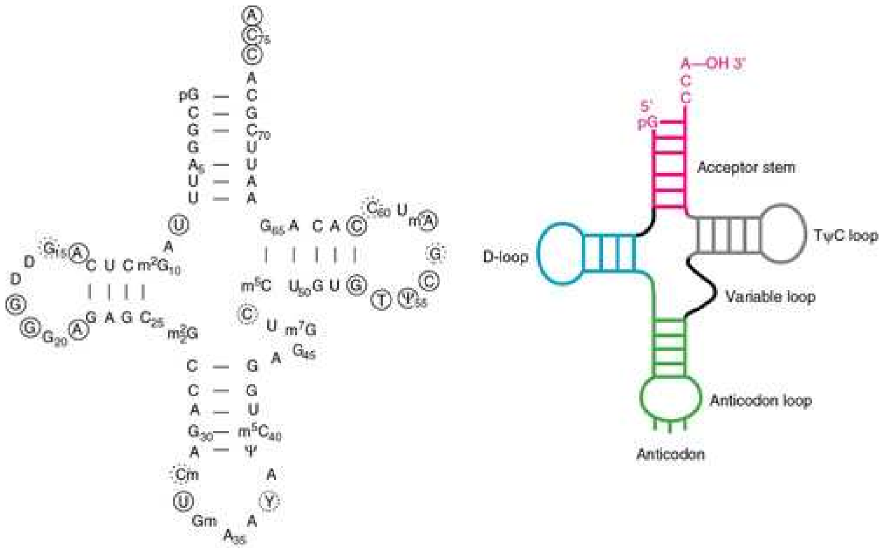
uracil.

`(source: Zuker)`

Unlike DNA, RNA is single stranded. However, complementary bases `C` − `G` and `A` − `U` form stable *base pairs* with each other using hydrogen bonds. These are called *Watson-Crick* pairs. Additionally, one sometimes considers the weaker `U` − `G` *wobble pairs*. These are all called *canonical base pairs*.

When base pairs are formed between different parts of a RNA molecule, then these pairs are said to define the *secondary structure* of the RNA molecule.

The secondary structure of a tRNA:



This particular tRNA is from yeast and is for the amino acid phenylalanine. (source: http://www.blc.arizona.edu/marty/411/Modules/ribtRNA.html)

4.3 Definition of RNA secondary structure

The *true secondary structure* of a real RNA molecule is the set of base pairs that occur in its three-dimensional structure.

Definition For our purposes, a *RNA molecule* is simply a string

$$x = (x_1, x_2, \ldots, x_L),$$

with $x_i \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ for all i.

Definition A *secondary structure* for x is a set P of ordered *base pairs*, written (i, j), with $1 \leq i < j \leq L$, satisfying:

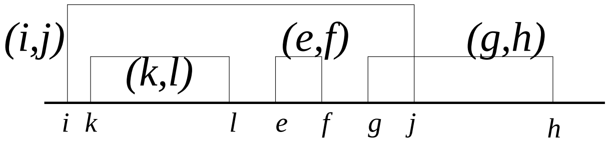
1. $j - i > 3$, i.e. the bases are not too close to each other, (although we will ignore this condition below), and
2. $\{i, j\} \cap \{i', j'\} = \emptyset$, i.e. the base pairs don't conflict.

Definition A secondary structure is called *nested*, if for any two base pairs (i, j) and (i', j'), w.l.o.g. $i < i'$, we have either

1. $i < j < i' < j'$, i.e. (i, j) precedes (i', j'), or
2. $i < i' < j' < j$, i.e. (i, j) includes (i', j').

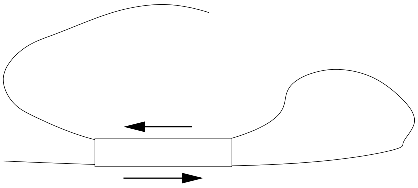
4.4 Nested structures

In the following, we only will consider *nested* secondary structures, as the more complicated non-nested structures are not tractable with the methods we will discuss.

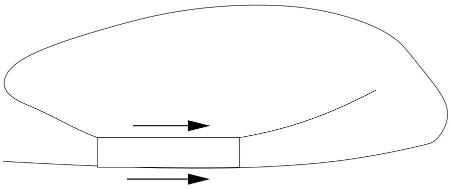


Here, the interactions (i, j) and (g, h) are not nested.

Interactions that are not nested give rise to a *pseudo knot* configuration in which segments of sequence are bonded in the "same direction":

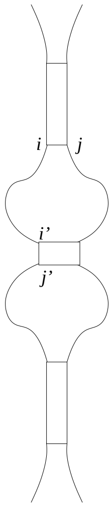


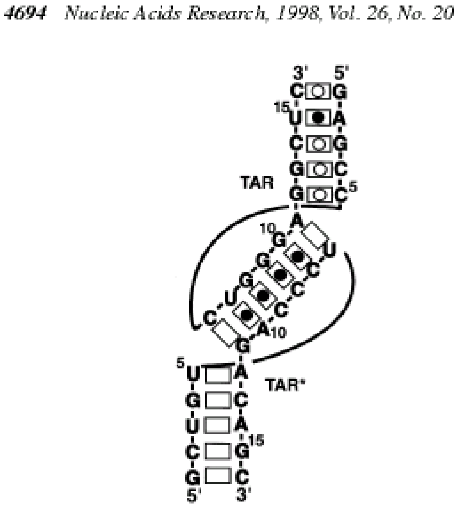
unknoted



pseudo knot

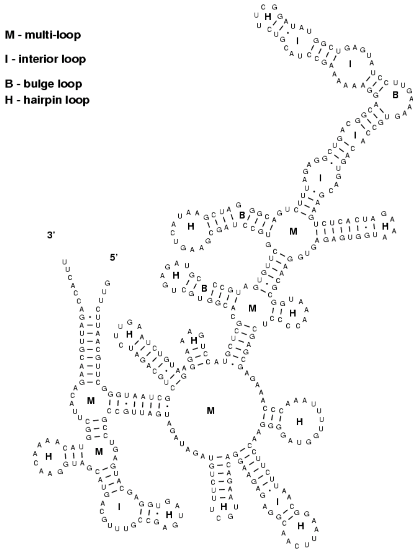
The nested requirement excludes other types of configurations, as well, such as *kissing hairpins*, for example:

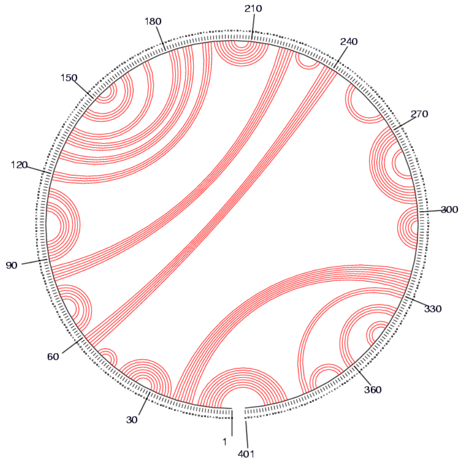




4.5 Example of secondary structure

Predicted structure for Bacillus Subtilis RNAase P RNA:





(source: Zuker)

This example shows the different types of single- and double-stranded regions in RNA secondary structures:

- single-stranded RNA,

- double-stranded RNA helix of stacked base pairs,
- stem and loop or hairpin loop,
- bulge loop,
- interior loop, and
- junction or multi-loop.

4.6 Prediction of RNA secondary structure

The problem of predicting the secondary structure of RNA has some similarities to DNA alignment, except that the sequence folds back on itself and aligns complementary bases rather than similar ones.

The goal of aligning two or more biological sequences is to determine whether they are homologous or just similar. In contrast, a secondary structure for an RNA is a simplification of the complex three-dimensional folding of the RNA molecule.

Problem Determine the true secondary structure of an RNA.

Variants: find a secondary structure that:

1. maximizes the number of base pairs,
2. minimizes the "free energy", or
3. is optimal, given a family of related sequences.

4.7 The Nussinov folding algorithm

The simplest approach to predicting the secondary structure of RNA molecules is to find the configuration with the greatest number of paired bases. The number of possible configurations to be inspected grows exponentially with the length of the sequence.

Fortunately, we can employ dynamic programming to obtain an efficient solution. In 1978 Ruth Nussinov et al. published a method to do just that.

The algorithm is recursive. It calculates the best structure for small subsequences, and works its way outward to larger and larger subsequences. The key idea of the recursive calculation is that there are only four possible ways of the getting the best structure for i, j from the best structures of the smaller subsequences.

Idea: There are four ways to obtain an optimal structure for a sequence i, j from smaller substructures:

```
     o                 o                 o              o          o
   o   o             o   o             o   o          o   o      o   o
    o-o               o-o               o-o            o-o        o-o
    o-o               o-o               o-o            o-o        o-o
i+1 o-o j           i o-o j-1       i+1 o-o j-1      i o-o--o--o--o-o j
 i o                     o j          i o-o j               k k+1

(1) i unpaired   (2) j unpaired     (3) i,j pair       (4) bifurcation
```

1. Add an unpaired base i to the best structure for the subsequence $i+1, j$,
2. add an unpaired base j to the best structure for the subsequence $i, j-1$,
3. add paired bases $i-j$ to the best structure for the subsequence $i+1, j-1$, or
4. combine two optimal substructures i, k and $k+1, j$.

Given a sequence $x = (x_1, \dots, x_L)$ of length L. We set $\delta(i,j) = 1$, if $x_i - x_j$ is a canonical base pair and 0, else.

The dynamic programing algorithm has two stages:

In the *fill stage*, we will recursively calculate scores $\gamma(i,j)$ which are the maximal number of base pairs that can be formed for subsequences $(x_i, \dots, x_j)$.

In the *traceback* stage, we traceback through the calculated matrix to obtain one of the maximally base paired structures.

4.8 The fill stage

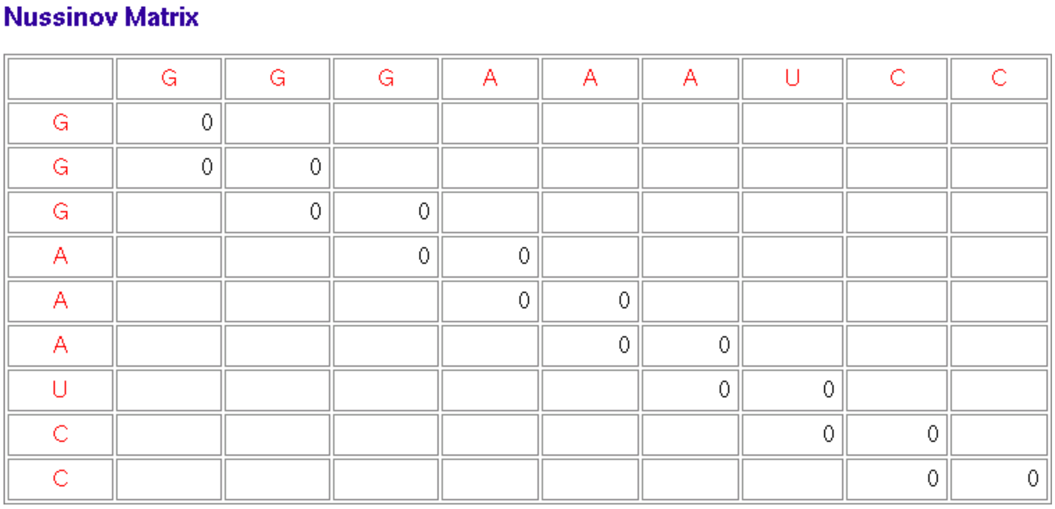
Algorithm (Nussinov RNA folding, fill stage)
Input: Sequence $x = (x_1, x_2, \dots, x_L)$
Output: Maximal number $\gamma(i,j)$ of base pairs for $(x_i, \dots, x_j)$.

Initialization:

$$\begin{aligned} \gamma(i, i-1) &= 0 \qquad \text{for } i = 2 \text{ to } L, \\ \gamma(i,i) &= 0 \qquad \text{for } i = 1 \text{ to } L; \end{aligned}$$

```
for  $n = 2$ to $L$ do        // longer and longer subsequences
    for $j = n$ to $L$ do
        Set $i = j - n + 1$
        Set $\gamma(i,j) = \max \begin{cases} \gamma(i+1, j), \\ \gamma(i, j-1), \\ \gamma(i+1, j-1) + \delta(i,j), \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)]. \end{cases}$
```

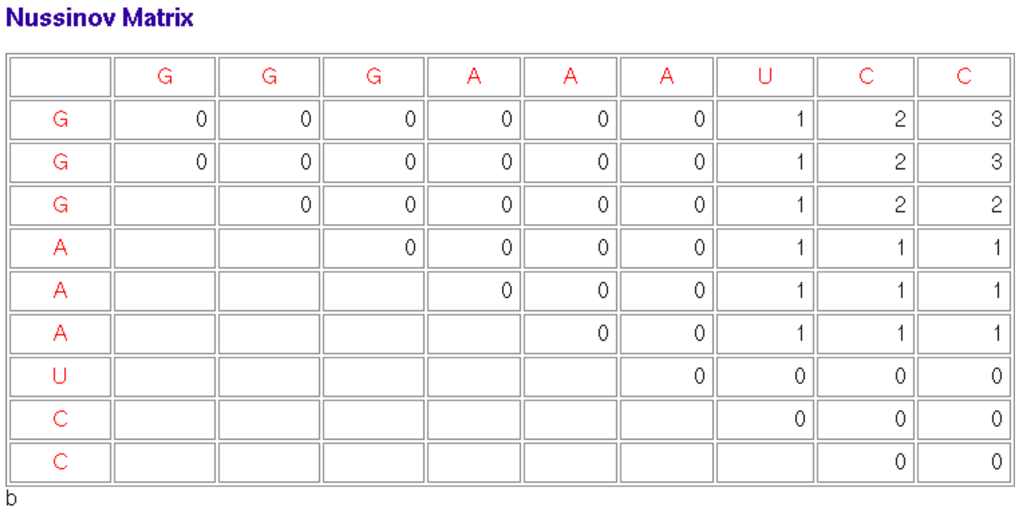
Consider the sequence $x =$ `GGGAAAUCC`. Here is the matrix γ after initialization ($i :\downarrow$, $j :\rightarrow$):



Nussinov Matrix

	G	G	G	A	A	A	U	C	C
G	0								
G	0	0							
G		0	0						
A			0	0					
A				0	0				
A					0	0			
U						0	0		
C							0	0	
C								0	0

Here is the matrix γ after executing the recursion ($i :\downarrow$, $j :\rightarrow$):



Nussinov Matrix

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	3
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

b

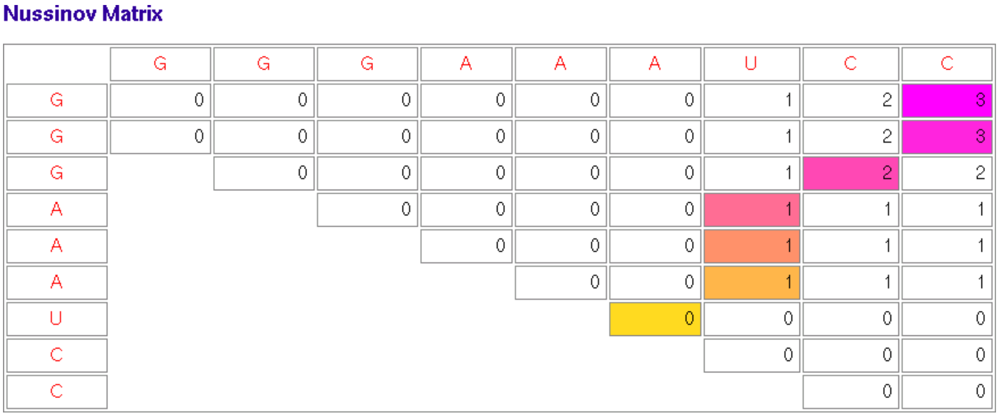
Values obtained using $\delta(a,b) = \begin{cases} 1 & \text{if } \{a,b\} = \{\mathtt{A},\mathtt{U}\} \text{ or } \{\mathtt{C},\mathtt{G}\}, \\ 0 & \text{else.} \end{cases}$

4.9 The traceback stage

Algorithm traceback(i,j)
Input: Matrix γ and positions i,j.
Output: Secondary structure maximizing the number of base pairs.
Initial call: traceback$(1,L)$.

```
if $i < j$ then
    if $\gamma(i,j) = \gamma(i+1,j)$ then                              // case (1)
        traceback$(i+1,j)$
    else if $\gamma(i,j) = \gamma(i,j-1)$ then                         // case (2)
        traceback$(i,j-1)$
    else if $\gamma(i,j) = \gamma(i+1,j-1) + \delta(i,j)$ then         // case (3)
        print base pair $(i,j)$
        traceback$(i+1,j-1)$
    else for $k = i+1$ to $j-1$ do                                     // case (4)
        if $\gamma(i,j) = \gamma(i,k) + \gamma(k+1,j)$ then
            traceback$(i,k)$
            traceback$(k+1,j)$
            break
end
```

Here is the traceback through γ ($i :\downarrow$, $j :\rightarrow$):

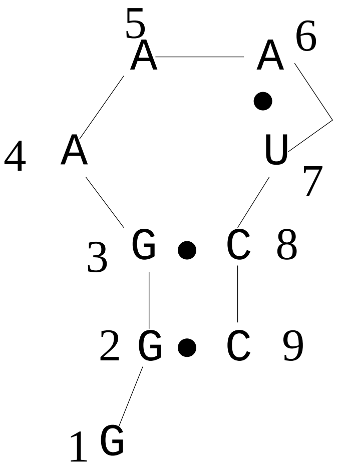


Nussinov Matrix

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	3
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

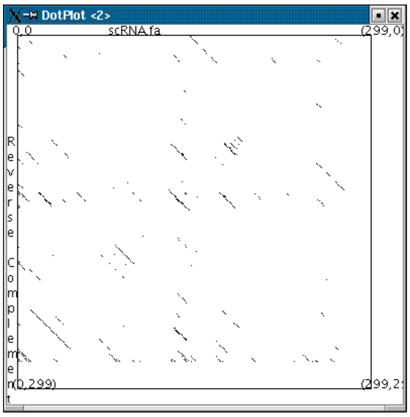
(There is a slight error in the traceback shown in Durbin et al. page 271)

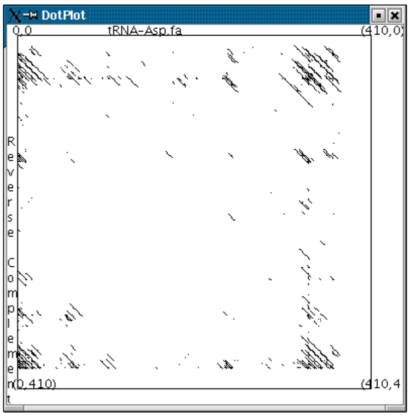
The resulting secondary structure is:



4.10 Application of dot plots

Given an RNA sequence x. Self complementary regions may be found by performing a dot matrix analysis of x with its reverse complement $\bar{x}$. Here are two examples:





4.11 Simple energy minimization

Maximizing the number of base pairs as described above does not lead to good structure predictions. Better predictions can be obtained by minimizing the following *energy function*

$$E((x,P)) = \sum_{(i,j)\in P} e(x_i, x_j),$$

where $e(x_i, x_j)$ is the amount of *free energy* associated with the base pair (x_i, x_j).

Reasonable values for e at 37^oC are -3, -2 and -1 kcal/mole for base pairs $\texttt{C}-\texttt{G}$, $\texttt{A}-\texttt{U}$ and $\texttt{G}-\texttt{U}$, respectively.

Obviously, a few simple changes to the Nussinov algorithm will produce a new algorithm that can solve this energy minimization problem.

Namely, we need to change ____________, ____________, and ____________.

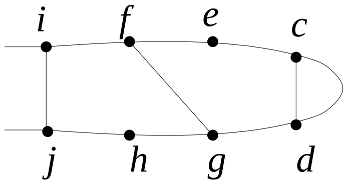
Unfortunately, this approach does not produce good structure predictions because it does not take into account that helical stacks of base pairs have a stabilizing effect whereas loops have a destabilizing effect on the structure. A more sophisticated approach is required.

The most sophisticated algorithm for folding single RNAs is the *Zuker* algorithm, an energy minimization algorithm which assumes that the correct structure is the one with the lowest equilibrium free energy ΔG.

The ΔG of an RNA secondary structure is approximated as the sum of individual contributions from loops, base pairs and other secondary structure elements. as we will see, an important difference to the Nussinov calculation is that the energies are computed from loops rather than from base pairs. This provides a better fit to experimentally observed data.

4.12 The k-loop decomposition

If (i, j) is a base pair in P and $i < h < j$, then we say that h is *accessible* from (i, j) if there is no base pair $(i', j') \in P$ such that $i < i' < h < j' < j$. Similarly, we say that (f, g) is *accessible* from (i, j), if both f and g are.



The set s of all $k-1$ base pairs and k' unpaired bases that are accessible from (i, j) is called the *k-loop* closed by (i, j).

The *null* k-loop consists of all *free* base pairs and unpaired bases that are accessible from no base pair.

Fact Given $x = (x_1, x_2, \ldots, x_L)$. Any secondary structure P on x partitions the set $\{1, 2, \ldots, L\}$ into k-loops $s_0, s_1, \ldots, s_m$, where $m > 0$ iff $P \neq \emptyset$.

We can now give a formal definition of names introduced earlier:

1. A 1-loop is called a *hairpin* loop.
2. Assume that there is precisely one base pair (i', j') accessible from (i, j). Then this 2-loop is called
   (a) a *stacked pair*, if $i' - i = 1$ and $j - j' = 1$,
   (b) a *bulge loop*, if $i' - i > 1$ or $j - j' > 1$, but not both, and
   (c) an *interior loop*, if both $i' - i > 1$ and $i - j' > 1$.
3. A k-loop with $k \geq 3$ is called a *multi-loop*.

The following is a consequence of nestedness:

Fact The number of non-null k-loops equals the number of base pairs.

The *size* of a k-loop is the number k' of unpaired bases that it contains.

Each k-loop is assigned an energy $e(s_i)$ and the energy of a structure P is given by:

$$E(p) = \sum_{i=0}^{m} e(s_i).$$

So now the energy is a function of k-loops instead of a function of base pairs.

4.13 Zuker's algorithm for folding RNA

We will now develop a more involved dynamic program that uses loop-dependent rules. It is due to M. Zuker (Zuker & Stiegler 1981, Zuker 1989). We will use two matrices, W and V.

For $i < j$, let $W(i,j)$ denote the minimum folding energy of all non-empty foldings of the subsequence $x_i, \ldots, x_j$.

Additionally, let $V(i,j)$ denote the minimum folding energy of all non-empty foldings of the subsequence $x_i, \ldots, x_j$, *containing the base pair* (i,j). The following obvious fact is crucial:

$$W(i,j) \leq V(i,j) \text{ for all } i,j.$$

These matrices are initialized as follows:

$$W(i,j) = V(i,j) = \infty \text{ for all } i,j \text{ with } j-4 < i < j.$$

(Note that we are now going to enforce that two paired bases are at least 3 positions away from each other).

4.14 Loop-dependent energies

We define different energy functions for the different types of loops:

- Let $eh(i,j)$ be the energy of the hairpin loop closed by the base pair (i,j),
- let $es(i,j)$ be the energy of the stacked pair (i,j) and $(i+1, j-1)$,
- let $ebi(i,j,i',j')$ be the energy of the bulge or interior loop that is closed by (i,j), with (i',j') accessible from (i,j), and
- let a denote a constant energy term associated with a multi-loop (a more general function for this case will be discussed later).

Predicted free-energy values (kcal/mole at 37^oC) for base pair stacking:

	A/U	C/G	G/C	U/A	G/U	U/G
A/U	-0.9	-1.8	-2.3	-1.1	-1.1	-0.8
C/G	-1.7	-2.9	-3.4	-2.3	-2.1	-1.4
G/C	-2.1	-2.0	-2.9	-1.8	-1.9	-1.2
U/A	-0.9	-1.7	-2.1	-0.9	-1.0	-0.5
G/U	-0.5	-1.2	-1.4	-0.8	-0.4	-0.2
U/G	-1.0	-1.9	-2.1	-1.1	-1.5	-0.4

Predicted free-energy values (kcal/mole at 37^oC) for features of predicted RNA secondary structures, by size of loop:

size	internal loop	bulge	hairpin
1	.	3.9	.
2	4.1	3.1	.
3	5.1	3.5	4.1
4	4.9	4.2	4.9
5	5.3	4.8	4.4
10	6.3	5.5	5.3
15	6.7	6.0	5.8
20	7.0	6.3	6.1
25	7.2	6.5	6.3
30	7.4	6.7	6.5

4.15 The main recursion

For all i, j with $1 \leq i < j \leq L$:

$$W(i,j) = \min \begin{cases} W(i+1,j) \\ W(i,j-1) \\ V(i,j) \\ \min_{i<k<j}\{W(i,k)+W(k+1,j)\}, \end{cases} \tag{4.1}$$

and

$$V(i,j) = \min \begin{cases} eh(i,j) \\ es(i,j)+V(i+1,j-1) \\ VBI(i,j), \\ VM(i,j), \end{cases} \tag{4.2}$$

where

$$VBI(i,j) = \min_{\substack{i<i'<j'<j \\ i'-i+j-j'>2}} \{ebi(i,j,i',j')+V(i',j')\}, \tag{4.3}$$

and

$$VM(i,j) = \min_{i<k<j-1}\{W(i+1,k)+W(k+1,j-1)\}+a. \tag{4.4}$$

Equation 4.1 considers the four cases in which (a) i is unpaired, (b) j is unpaired, (c) i and j are paired to each other and (d) i and j are paired, but not to each other. In case (c) we reference the auxiliary matrix V.

Equation 4.2 considers the different situations that arise when bases i and j are paired, closing (a) a hairpin loop, (b) a stacked pair, (c) a bulge or interior loop or (d) a multi-loop. The two latter cases are more complicated and are obtained from equations 4.3 and 4.4.

Equation 4.3 takes into account all possible ways to define a bulge or interior loop that involves a base pair (i',j') and is closed by (i,j). In each situation, we have a contribution from the bulge or interior loop and a contribution from the structure that is on the opposite side of (i',j').

Equation 4.4 considers the different ways to obtain a multi-loop from two smaller structures and adds a constant contribution of a to close the loop.

4.16 Time analysis

The minimum folding energy E_{min} is given by $W(1, L)$.

There are $O(L^2)$ pairs (i, j) satisfying $1 \leq i < j \leq L$.

The computation of

1. W takes $O(L^3)$ steps,
2. V takes $O(L^2)$ steps,
3. VBI takes $O(L^4)$ steps, and
4. VM takes $O(L^3)$ steps,

and so the total run time is $O(L^4)$.

The most practical way to reduce the run time to $O(L^3)$ is to limit the size of a bulge or interior loop to some fixed number d, usually about 30. This is achieved by limiting the search in Equation 4.3 to $2 < i' - i + j - j' - 2 \leq d$.

4.17 Modification of multi-loop energy

In Equation 4.4 we used a constant energy function for multi-loops. More generally, we can use the following function

$$e(\text{multi} - \text{loop}) = a + b \times k' + c \times k,$$

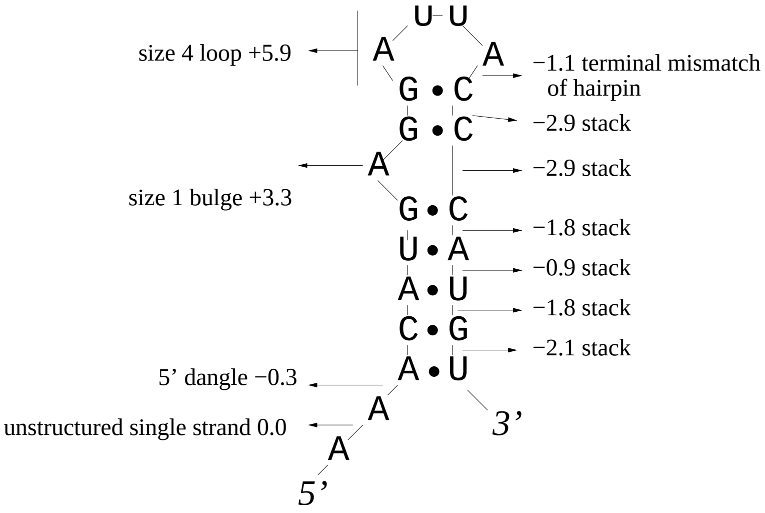
where a, b and c are constants and k' is the number of unpaired bases in the multi-loop.

This is a convenient function to use because, similar to the introduction of affine gap penalties in sequence alignment, a cubic order algorithm remains possible.

A number of additional modifications to the algorithm can be made to handle the stacking of single bases. These modifications lead to better predictions, but are beyond the scope of our lecture.

4.18 Example of energy calculation

Here is any example of the full energy calculation for an RNA stem loop (the wild type $R17$ coat protein binding site):



Overall energy value: -4.6 kcal/mol

4.19 RNA folding via comparative analysis

Although energy minimization techniques are attractive, almost all trusted RNA secondary structures to date where determined using comparative analysis. However, comparative methods require many diverse sequences and highly accurate multiple alignments to work well.

The key idea is to identify Watson-Crick correlated positions in a multiple alignment, e.g.:

seq1 G**C**CUUCGG**G**C
seq2 G**A**CUUCGG**U**C
seq3 G**G**CUUCGG**C**C

The amount of correlation of two positions can be computed as the *mutual information* content measure: *if you tell me the identity of position i, how much do I learn about the identity of position j?*

A method used to locate covariant positions in a multiple sequence alignment is the mutual information content of two columns.

First, for each column i of the alignment, the frequency $f_i(x)$ of each base $x \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ is calculated.

Second, the 16 joint frequencies $f_{ij}(x, y)$ of two nucleotides, x in column i and y in column j, are calculated.

If the base frequencies of any two columns i and j are *independent* of each other, then the ratio of $\frac{f_{ij}(x,y)}{f_i(x) \times f_j(y)} \approx 1$.

If these frequencies are *correlated*, then this ratio will be greater than 1.

To calculate the *mutual information content* $H(i, j)$ in bits between the two columns i and j,the logarithm of this ratio is calculated and summed over all possible 16 base-pair combinations:

$$H_{ij} = \sum_{xy} f_{ij}(x, y) \log_2 \frac{f_{ij}(x, y)}{f_i(x) f_j(y)}.$$

This measure is maximum at 2 bits, representing perfect correlation.

If either site is conserved, there is less mutual information: for example, if all bases at site i are A, then the mutual information is 0, even if site j is always U, because there is no covariance.

The main problem with the comparative approach is that we need an accurate multiple alignment to get good structures and we need accurate structures to get a good alignment!

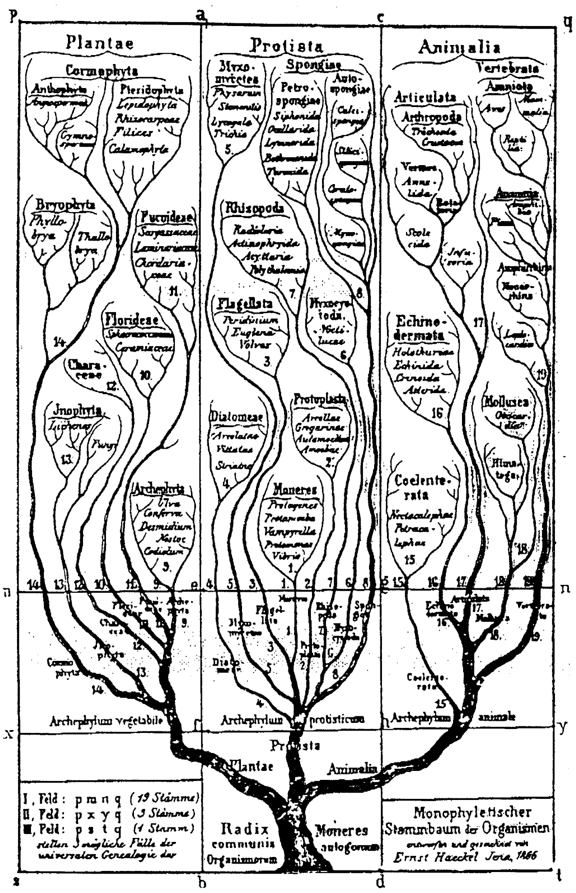
4.20 Mutual information content

Examples of how to compute the mutual information content:

1	2	3	4	5	6
C	G	C	G	A	U
C	G	G	C	C	G
C	G	C	G	G	C
C	G	G	C	U	A

Compute: $H_{12} =$ ________
$H_{34} =$ ________
$H_{56} =$ ________

5 Phylogeny



Evolutionary tree of organisms Ernst Haeckel, 1866

Sources for parts of this Chapter:

- R. Durbin, S. Eddy, A. Krogh & G. Mitchison, Biological sequence analysis, Cambridge, 1998
- J. Setubal & J. Meidanis, Introduction to computational molecular biology, 1997.
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, 2001.
- D.L. Swofford, G.J. Olsen, P.J.Waddell & D.M. Hillis, Phylogenetic Inference, in: D.M. Hillis (ed.), Molecular Systematics, 2 ed., Sunderland Mass., 1996.

5.1 Phylogenetic analysis

Given a collection of extant species. The goal of phylogenetic analysis is to determine and describe the *evolutionary relationship* between the species. In particular, this involves determining the order of speciation events and their approximate timing.

It is generally assumed that *speciation* is a branching process: a population of organisms becomes separated into two sub-populations. Over time, these evolve into separate species that do not cross-breed.

Because of this assumption, a *tree* is often used to represent a proposed phylogeny for a set of species, showing how the species evolved from a common ancestor. We will study this in detail.

However, there are a number of situations in which a tree is less appropriate than a phylogenetic *network*. We will study this in a later Chapter.

5.2 Phylogenetic trees

In the following, we will use $X = \{x_1, x_2, \dots, x_n\}$ to denote a set of *taxa*, where a *taxon* x_i is simply a representative of a group of individuals defined in some way.

A *phylogenetic tree* (on X) is a system $T = (V, E, \lambda)$ consisting of a connected graph (V, E) without cycles, together with a *labeling* λ of the leaves by elements of X, such that:

1. every leaf is labeled by exactly one taxon, and
2. every taxon appears exactly once as a label.

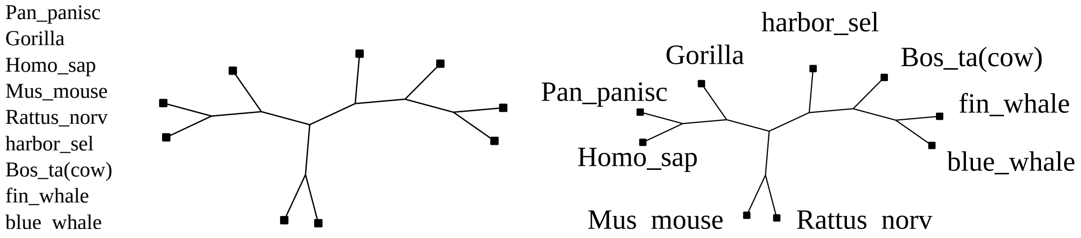
Further, we require that either all internal nodes have degree ≥ 3, in which case T is called *unrooted*, or there exists precisely one internal *root* node ρ of degree 2, and T is called *rooted.*

A phylogenetic tree T is called *binary*, if every internal node v has degree 3, except ρ, if T is rooted.

Sometimes we will relax the definition to allow labels to be placed on internal nodes, or nodes to carry multiple labels. Occasionally we will allow an arbitrary node to be root.

5.3 Unrooted trees

An unrooted phylogenetic tree is obtained by placing a set of taxa on the leaves of a tree:

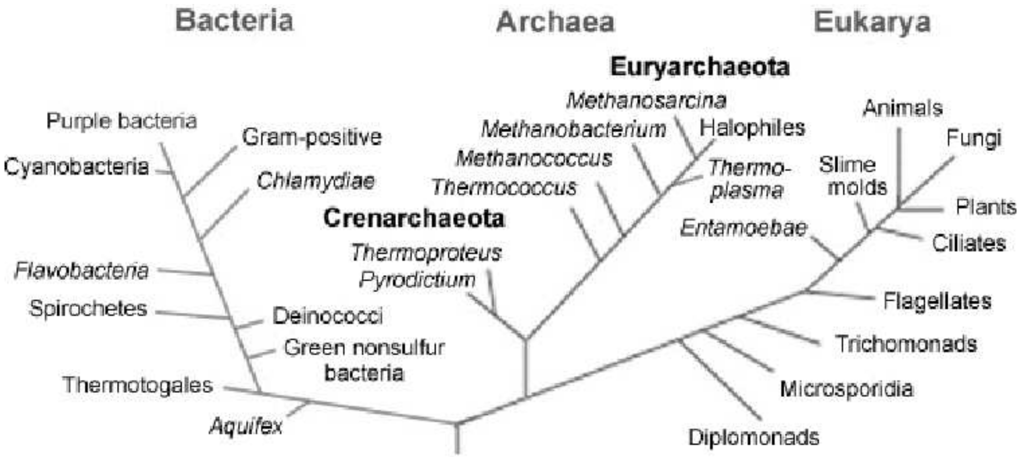


Taxa X + tree ⇒ phylogenetic tree T on X

Unrooted trees are often displayed using this type of *circular* layout.

5.4 Rooted trees

A rooted tree is usually drawn with the root placed at the bottom, top or left of the figure:



Modern version of the tree of life.

5.5 Edge lengths

Consider a phylogenetic tree T on X. The *topology* of the tree describes the putative order of speciation events that gave rise to the extant taxa.

Additionally, we can assign *lengths* to the edges of the tree. Ideally, these lengths should represent the amount of time that lies between two speciation events. However, in practice the edge lengths usually represent quantities obtained by some given computation and only correspond very indirectly to time.

In the following, we will use $\omega : E \to \mathbb{R}_{\geq 0}$ to specify edge lengths and will use $T = (V, E, \lambda, \omega)$ to denote a phylogenetic tree with edge lengths.

5.6 Computer representation of a phylogenetic tree

Let X be a set of taxa and T a phylogenetic tree on X.

To represent a phylogenetic tree in a computer we maintain a set of nodes V and a set of edges E. Each edge $e \in E$ maintains a reference to its source node $s(e)$ and target node $t(e)$. Each node $v \in V$ maintains a list of references to all adjacent edges.

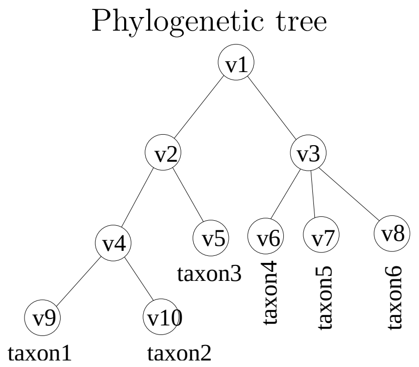
Each node $v \in V$ maintains a reference $\lambda(v)$ to the taxon that it is labeled by, and, vice versa, $\nu(x)$ maps a taxon x to the node that it labels.

Each edge $e \in E$ maintains its length $\omega(e)$.

5.7 Nested structure

A rooted phylogenetic tree $T = (V, E, \lambda)$ is a *nested* structure: Consider any node $v \in V$. Let T_v denote the subtree rooted at v. Let $v_1, v_2, \ldots, v_k$ denote the children of v. Then T_v is obtainable from its subtrees $T_{v_1}, T_{v_2}, \ldots, T_{v_k}$ by connecting v to each of their roots.

Such a nested structure can be written using nested brackets:



nested description

$$v_1$$
$$\downarrow$$
$$(v_2, v_3)$$
$$\downarrow$$
$$((v_4, v_5), (v_6, v_7, v_8))$$
$$\downarrow$$
$$(((v_9, v_{10}), v_5), (v_6, v_7, v_8))$$

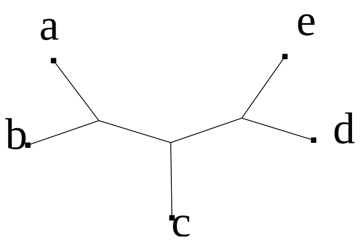
Description: $(((taxon_1, taxon_2), taxon_3), (taxon_4, taxon_5, taxon_6))$

5.8 Printing a phylogenetic tree

Phylogenetic trees are usually printed using the so-called *Newick* format which uses pairs of brackets to indicate the hierarchical structure of a tree. For example,

`((a,b),c,(d,e));`

describes the following unrooted phylogenetic tree, without edge lengths:

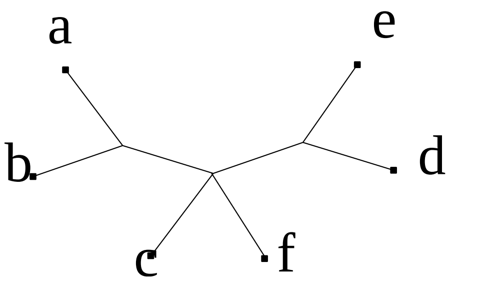


The following algorithm recursively prints a tree in Newick format. It is initially called with $e = null$ and v set to ρ, if the tree is rooted, or v set to an arbitrary internal node, else.

```
**Algorithm** toNewick($e, v$)
Input: Phylogenetic tree $T = (V, E)$ on $X$, with labeling $\lambda : V \to X$
Output: Newick description of tree
begin
if $v$ is a leaf then
        print $\lambda(v)$
else // $v$ is an internal node
        print '('
        for each edge $f \neq e$ adjacent to $v$ do
                If this is not the first pass of the loop, print ','
                Let $w \neq v$ be the other node adjacent to $f$
                call toNewick($f, w$)
        print ')'
end
```

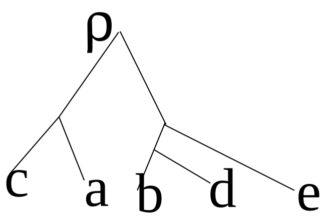
Here are two examples:

unrooted tree



Newick string:
`((a,b),c,f,(e,d))`
or e.g.:
`((a,b),(c,f,(e,d)))`

rooted tree



Newick string:
`((c,a),((c,d),e))`
...

5.9 Parsing a phylogenetic tree

We need to be able to read a phylogenetic tree into a program. The following algorithm parses a tree in Newick format from a (space-free) string $s = s_1 s_2 \dots s_m$. The initial call is parseNewick$(1, m, null)$.

```
Algorithm parseNewick($i$,$j$,$v$)
Input: a substring $s_i \dots s_j$ and a root node $v$
Output: a phylogenetic tree $T = (V, E, \lambda)$
begin
while $i \leq j$ do
    Add a new node $w$ to $V$
    if $v \neq null$ then add a new edge $\{v, w\}$ to $E$

    if $s_i$ = '(' then // recurse on subtree
        Set $k$ to the position of the balancing close-bracket for $i$
        call parseNewick($i + 1, k - 1, w$) // strip brackets and recurse

    else // hit a leaf
        Set $k = \min\{k \geq i \mid s_{k+1} = \text{','} \text{ or } k + 1 = j\}$
        Set $\lambda(w) = s_i \dots s_k$ // grab label for leaf

    Set $i = k + 1$ // advance to next ',' or $j$
    if $i < j$ then
        Check that $i + 1 \leq j$ and $s_{i+1}$ = ','
        Increment $i$ // advance to next token
end
```

In the Newick format, each pair of matching brackets corresponds to an internal node and each taxon label corresponds to an internal node. To add edge lengths to the format, specify the length *len* of the edge along which we visited a given node by adding :*len* behind the closing bracket, in the case of an internal node, and behind the label, in the case of a leaf.

For example, consider the tree `(a,b,(c,d))`. If all edges have the same length 1, then this tree is written as `(a:1,b:1,(c:1,d:1):1)`.

5.10 Drawing a phylogenetic tree

An *embedding* of a phylogenetic tree is given by an assignment of coordinates $(x(v), y(v))$ to each node v. Any edge e is represented by the line segment connecting the points $(x(v), y(v))$ and $(x(w), y(w))$ associated with the two nodes v and w adjacent to e. We will require that following additional conditions are satisfied:

1. no two line segments representing edges cross, and

2. for every edge e, the length of the line segment representing e is $\omega(e)$.

We will now discuss how to compute an embedding for an arbitrary unrooted phylogenetic tree. The algorithm proceeds in two stages. In the first stage, a direction is recursively computed for each edge. Then, in the second stage, all nodes are initially placed at the same point and from there they they are moved in the directions computed for the edges.

5.11 Circular tree embedding

Let T be a phylogenetic tree, together with an embedding $\epsilon : V \to \mathbb{R}^2$. Choose any leaf r and call it the *reference leaf.* Starting at r, circumnavigate the tree in anti-clockwise order and give each leaf an *index* $h(v) = 0, 1, \dots, n-1$, starting with $h(r) = 0$.

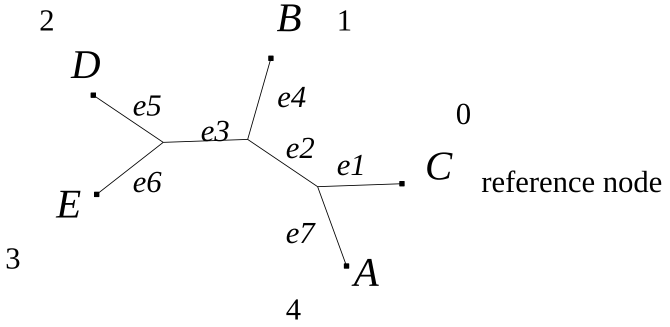
Direct all edges e of T away from r and let $V(e)$ denote the set of all leaves reachable from e in the directed graph.

We say that the embedding ϵ is a *circular embedding* with reference leaf r, if for any directed edge e we have that its angle equals

$$\alpha(e) = \frac{2\pi}{n|V(e)|} \sum_{v \in V(e)} h(v).$$

In other words, the leaves are placed around the unit circle at positions $\frac{2\pi}{n}, 1\frac{2\pi}{n}, \dots, n-1\frac{2\pi}{n}$ and a edge e points in the average direction of the set of leaves that it separates the reference node r from.

Example:



This is a circular layout with reference node C, which is verified as follows:

- $\alpha(e_1) = \frac{2\pi}{5\cdot 4}(1+2+3+4) = \pi$,

- $\alpha(e_2) = \frac{2\pi}{5\cdot 3}(1+2+3) = \frac{4}{5}\pi$,

- $\alpha(e_3) = \frac{2\pi}{5\cdot 2}(2+3) = \pi$,

- $\dots$

5.12 Computing the edge angles

Now, given a phylogenetic tree T, how do we obtain a circular embedding for it? We first assign an angle to each edge. Then, using these angles and the given edge lengths, we will assign coordinates to each node.

The following algorithm visits all edges and assign an angle to each. The initial call is setAngles$(0, null, r)$, where r is the reference leaf.

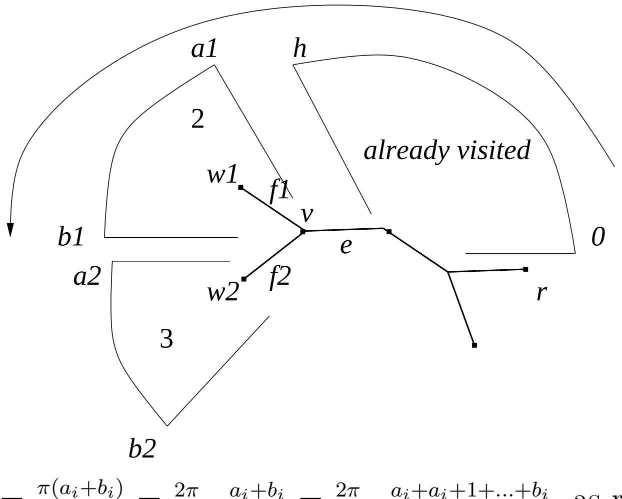
```
**Algorithm** setAngles$(h, e, v)$
Input: Number $h$ of nodes placed, arrival edge $e$, current node $v$
Output: Angle $\alpha(e)$ for every edge $e \in E$.
**begin**
**if** $v$ is a leaf **then**
    **return** $h + 1$
Set $a = h$ // number of nodes placed before recursion
Set $b = 0$ // number of nodes placed after recursion
**for all** edges $f \neq e$ adjacent to $v$ **do**
    Let $w \neq v$ be the other node adjacent to $f$
    Set $b = $ setAngles$(a, f, w)$
    Set $\alpha(f) = \frac{\pi(a+b)}{n}$
    Set $a = b$
**return** $b$.
```

Lemma The algorithm computes the edge angles of a a circular embedding in linear time.

Proof Every edge is visited exactly once, hence the runtime is linear in the number of edges $|E|$ (which is linear in the number of leaves).

To see that the algorithm produces the correct result, consider the situation for some arbitrary edge e and node v. The parameter h corresponds to the highest index assigned so far. Let $f_1, \ldots, f_k$ be the list of edges adjacent to v that are considered by the algorithm, let w_i denote the corresponding other node, and let a_i and b_i be the values of a and b directly after processing f_i. The latter two numbers denote the range of indices recursively assigned to the set of leaves in the subtree rooted at w_i.

The situation when processing node v:

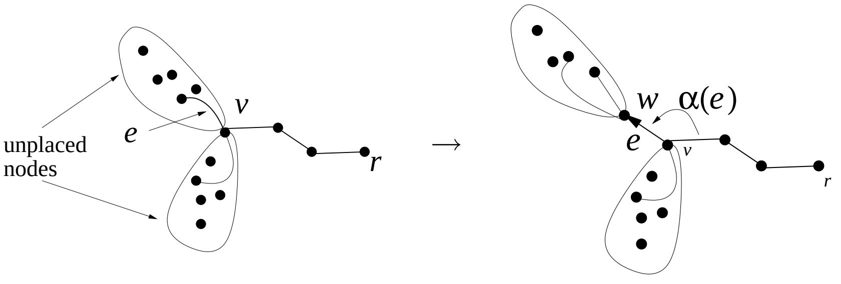


We then compute $\alpha(f_i) = \frac{\pi(a_i+b_i)}{n} = \frac{2\pi}{n} \cdot \frac{a_i+b_i}{2} = \frac{2\pi}{n} \cdot \frac{a_i+a_i+1+\ldots+b_i}{b_i-a_i+1}$, as required in the definition of a circular embedding. □

5.13 Determining coordinates

Given a phylogenetic tree T and an angle function $\alpha : E \to [0, 2\pi]$, how do we obtain coordinates for the nodes?

Idea: In a depth-first traversal of the tree starting at the reference node r, for every edge e, simply push the opposite node away from the current node in the direction specified by $\alpha(e)$ by the amount specified by the length $\omega(e)$:



before processing e after processing e

The following algorithm does the pushing. Initially, we set $\epsilon(r) = (0, 0)$ and invoke setCoordinates$(null, r)$.

```
Algorithm setCoordinates$(e, v)$
Input: Phylogenetic tree $T$ and edge angles $\alpha$
Output: Circular embedding $\epsilon$ of $T$
begin
Set $p = \epsilon(v)$
for each edge $f \neq e$ adjacent to $v$ do
        Let $w \neq v$ be the other node adjacent to $f$
        Obtain $p'$ by translating $p$ in direction $\alpha(e)$ by amount $\omega(e)$
        Set $\epsilon(w) = p'$
        call setCoordinates$(f, w)$
end
```

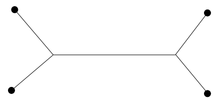
Theorem A circular embedding is computable in linear time.

Challenge: prove that the resulting configuration is indeed an embedding, that is, that no two edges can cross.

5.14 The number of edges and nodes of an unrooted phylogenetic tree

Let T be an unrooted phylogenetic tree on n taxa, i.e., with n leaves. How many nodes and edges does T have? Let us assume that T is binary. Any non-binary tree on n taxa will have less nodes and edges.

Consider a tree for $n = 4$, it has 6 nodes and 5 edges:



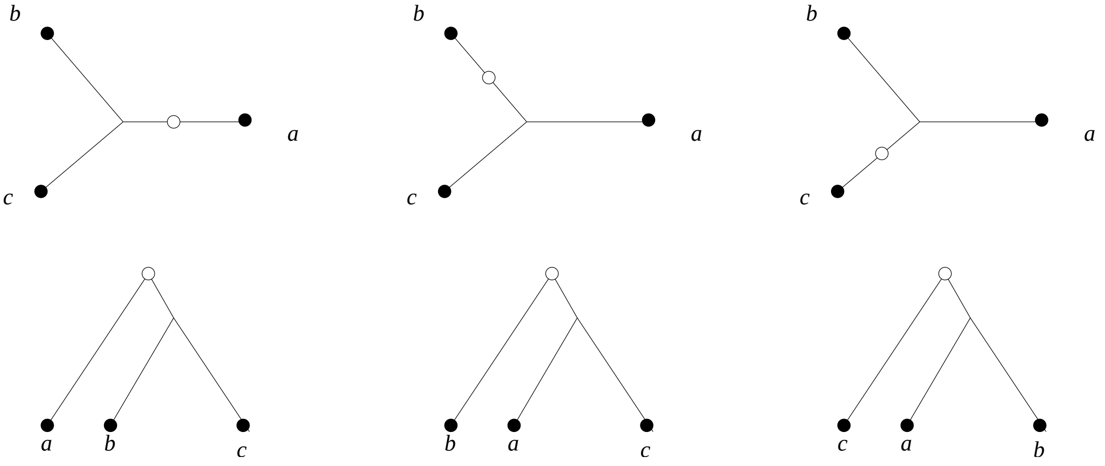
Now inductively, assume $n > 4$. Any tree T' with $n + 1$ leaves can be obtained from some tree

T with n leaves by inserting a new node v into some edge e of T and connecting v to a new leaf w. This increases both the number of nodes and the number of edges by 2.

Putting this together, we see that the number of nodes is $2n-2$ and the number of edges is $2n-3$.

5.15 The number of phylogenetic trees

An unrooted tree T with n leaves has $2n-2$ nodes and $2n-3$ edges. A root can be added in any of the $2n-3$ edges, thus producing $2n-3$ different rooted trees from T:



For $n = 3$ there are three ways of adding a root. Similarly, there are 3 different ways of adding an extra edge with a new leaf to obtain an unrooted tree on 4 leaves. This new tree has $(2n-3) = 5$ edges and there are 5 ways to obtain a new tree with 5 leaves etc.

Continuing this, we see that there are

$$U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2n-5)$$

unrooted trees on n leaves. Similarly, there are

$$R(n) = (2n-3)!! = U(n) \cdot (2n-3) = 3 \cdot 5 \cdot \ldots \cdot (2n-3)$$

rooted trees.

These numbers grows very rapidly with n, for example, $U(10) \approx 2$ million and $U(20) \approx 2.2 \times 10^{20}$.

5.16 Constructing phylogenetic trees

There are three main approaches to constructing phylogenetic trees from molecular data.

1. Using a *distance method*, one first computes a distance matrix from a given set of biological data and then one computes a tree that represents these distances as closely as possible.
2. *Maximum parsimony* takes as input a set of aligned sequences and attempts to find a tree and a labeling of its internal nodes by auxiliary sequences such that the number of mutations along the tree is minimum.
3. Given a probabilistic model of evolution, *maximum likelihood* approaches aim at finding a phylogenetic tree that maximizes the likelihood of obtaining the given sequences.

5.17 Distances

Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of taxa. The input to a distance method is a dissimilarity matrix $D : X \times X \to \mathbb{R}_{\geq 0}$ that associates a *distance* $d(x_i, x_j)$ with every pair of taxa $x_i, x_j \in X$. Sometimes we will abbreviate $d_{ij} := d(x_i, x_j)$ or $D_{ij} := d(x_i, x_j)$.

We usually require that

1. the matrix is *symmetric*, that is, $d(x, y) = d(y, x)$ for all $x, y \in X$, and
2. $d(x, x) = 0$ for all $x \in X$.

We call D a *pseudo metric*, if the *triangle inequalities* are satisfied:

$$d(x, z) \leq (x, y) + d(y, z) \text{ for all } x, y, z \in X,$$

and a *metric*, if additionally we have $d(x, y) > 0$ for all $x \neq y$.

5.18 Hamming distance

Let a collection of taxa be given by a set of distinct sequences $A = \{a_1, a_2, \ldots, a_n\}$ and assume we are given a multiple sequence alignment A^* of the sequences.

We define *sequence dissimilarity* as the (normalized) *Hamming distance* $Ham(a_i, a_j)$ between two taxa a_i and a_j as the number of mismatch positions in a_i^* and a_j^*, divided by the number of comparisons.

We ignore any column in which both sequences contain a gap. If only one sequence has a gap in a column then we can either ignore the column, or treat it as a match, or as a mismatch, depending on the type of data. Usually, one ignores *all* columns in which any of the n sequences contains the gap.

Lemma If the alignment is gap-less, then the corresponding distance matrix is a metric on A.

Proof: Consider three distinct sequences $a_i, a_j, a_k \in A$. If $a_i^* \neq a_k^*$, then either $a_i^* \neq a_j^*$, or $a_k^* \neq a_j^*$, and hence $Ham(a_i, a_k) < Ham(a_i, a_j) + Ham(a_j, a_k)$. □

For protein data, it makes sense to relax the definition of *sequence dissimilarity* to the number of "non-synonymous" residues divided by the number of sequence positions compared.

For example, we may choose to ignore "conservative substitutions" by pooling amino acids with similar properties into six groups: acidic (D,E), aromatic (F,W,Y), basic (H,K,R), cysteine, non-polar (A,G,I,L,P,V), and polar (M,N,Q,S,T). Two residues are considered synonymous, if the are contained in the same group, and non-synonymous, otherwise.

Example:

$$\begin{array}{l l}
a_1 & \texttt{C A A C C C C C A A A A A} \\
a_2 & \texttt{T A A T T T - C A A A A A} \\
a_3 & \texttt{C G G T T T - - A A A A A}
\end{array}$$

Distances:

$$Ham(a_1, a_2) = \frac{4}{12} = 0.\overline{33}$$

$$Ham(a_1, a_3) = \frac{5}{11} = 0.\overline{45}$$

$$Ham(a_2, a_3) = \frac{3}{11} = 0.\overline{27}$$

Hamming distances are only suitable for closely related sequences. Below we will discuss more sophisticated distances.

Question: What is the average Hamming distance between two random gap-free DNA sequences of the same length?

5.19 UPGMA

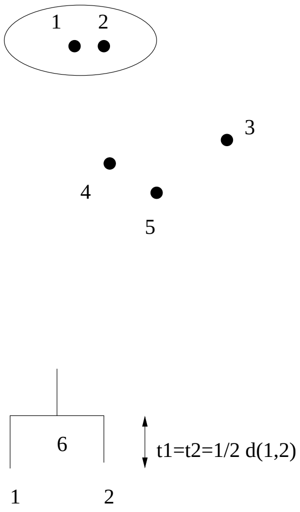
We will now discuss a simple distance method called UPGMA which stands for *unweighted pair group method using arithmetic averages* (Sokal & Michener 1958).

Given a set of taxa X and a distance matrix D, UPGMA produces a rooted phylogenetic tree T with edge lengths.

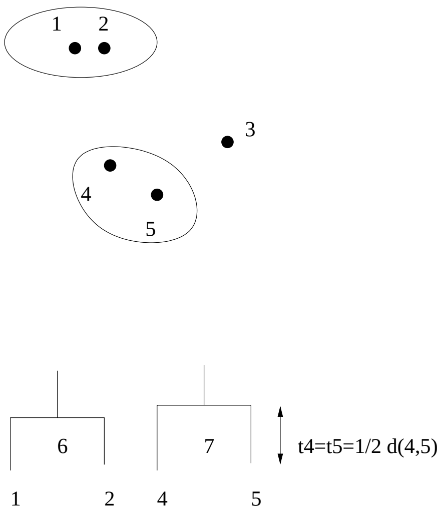
It operates by clustering the given taxa, at each stage merging two clusters and at the same time creating a new node in the tree. The tree is assembled "upwards", first clustering pairs of leaves, then pairs of clustered leaves etc. Each node is given a height and the edge lengths are obtained as the difference of heights of its two end nodes.

5.20 UPGMA example

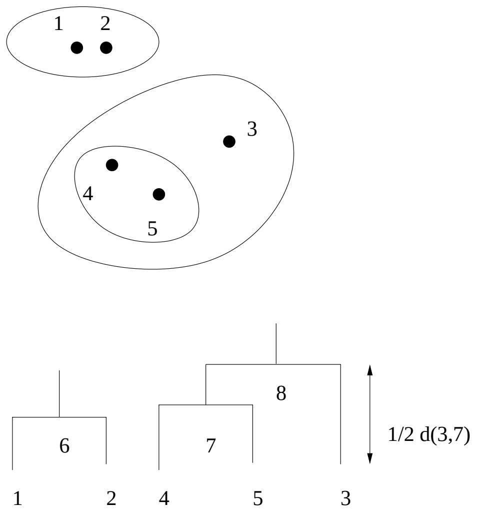
Example $X = \{1, 2, 3, 4, 5\}$, distances given by distance in the plane:



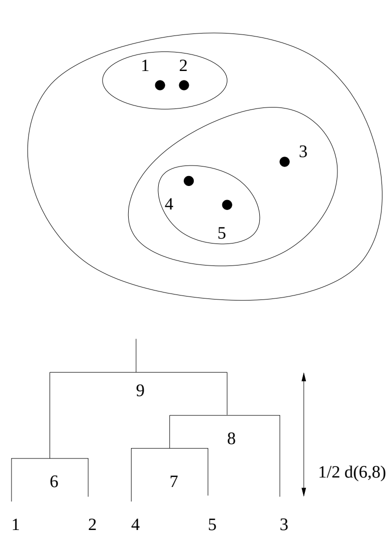
cluster 1 and 2



cluster 4 and 5



cluster 7 and 3



cluster 6 and 8

UPGMA produces a rooted, binary phylogenetic tree.

5.21 The distance between two clusters

Initially, we are given a distance $d(x, y)$ between any two taxa, i.e. leaves, x and y.

We define the distance $d(i, j) := d(C_i, C_j)$ between two clusters $C_i \subseteq X$ and $C_j \subseteq X$ to be the average distance between pairs of taxa from each cluster:

$$d(i, j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d(x, y).$$

Note that, if C_k is the union of two clusters C_i and C_j, and C_l is any other cluster, then

$$d(k, l) = \frac{d(i, l)|C_i| + d(j, l)|C_j|}{|C_i| + |C_j|}.$$

This is a useful *update* formula, because using it in the algorithm, we can obtain the distance between two clusters in constant time.

5.22 The UPGMA algorithm

The UPGMA algorithm is very straight-forward:

```
Algorithm UPGMA

Input: A set of taxa $X$ and a corresponding distance matrix $D$
Output: A binary, rooted phylogenetic UPGMA tree on $T$

Initialization
        Assign each taxon $x_i$ to its own cluster $C_i$
        Define one leaf of $T$ for each taxon, placed at height zero

Iteration
        Determine two clusters $C_i$ and $C_j$ for which $d(i, j)$ is minimal
        Define a new cluster $k$ by $C_k = C_i \cup C_j$
```

Define $d(k, l)$ for all existing clusters l using the update formula
Define a node k with daughter nodes i and j, and place it at height $\frac{d(i,j)}{2}$
Add C_k to the set of current clusters and remove C_i and C_j.

Termination

When only two clusters C_i and C_j remain, place the root at height $\frac{d(i,j)}{2}$.

(Problem: show that this algorithm produces well-defined edge lengths, i.e. a parent node always lies above its daughters.)

Finally, for each edge e, set the edge length $\omega(e)$ equal to the difference of the heights of the two incident nodes.

Example of UPGMA applied to 5S rRNA data:

Original distances:

Abbreviations:
Bsu: *Bacillus subtilis*
Bst: *Baclillus stearothermophilus*
Lvi: *Lactobacillus viridescens*
Amo: *Acholeplasma modicum*
Mlu: *Micrococcus luteus*

	Bsu	Bst	Lvi	Amo	Mlu
Bsu	–	0.1715	0.2147	0.3091	0.2326
Bst		–	0.2991	0.3399	0.2058
Lvi			–	0.2795	0.3943
Amo				–	0.4289
Mlu					

→

	Bsu + Bst	Lvi	Amo	Mlu
Bsu + Bst	–	0.2569	0.3245	0.2192
Lvi		–	0.2795	0.3943
Amo			–	0.4289
Mlu				–

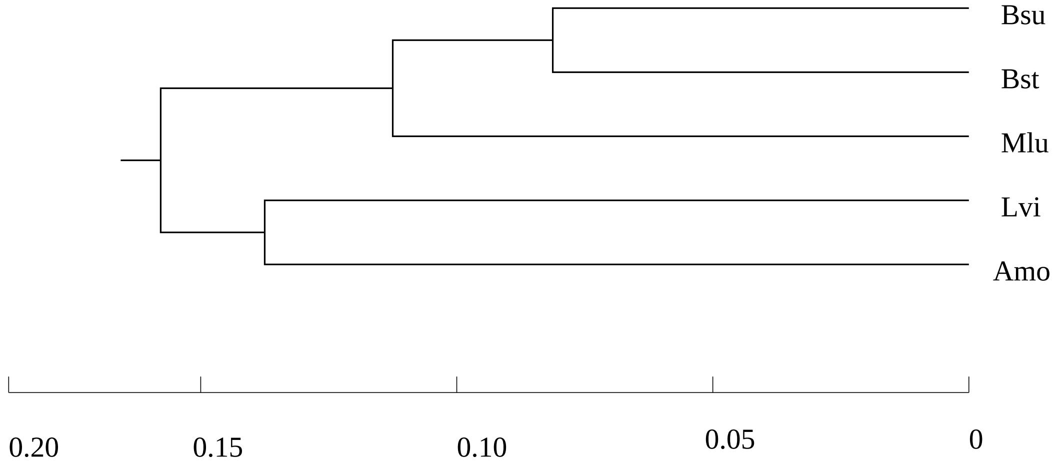
→

	Bsu + Bst + Mlu	Lvi	Amo
Bsu + Bst + Mlu	–	0.3027	0.3593
Lvi		–	0.2795
Amo			–

→

	Bsu + Bst + Mlu	Lvi + Amo
Bsu + Bst + Mlu	–	0.3310
Lvi + Amo		–

The resulting tree:



This tree is biologically incorrect, as we will see later.

5.23 Tree reconstruction

The goal of phylogenetic analysis is usually to *reconstruct* a phylogenetic tree from data, such as distances or sequences, that was produced by some *generating* or *true* tree.

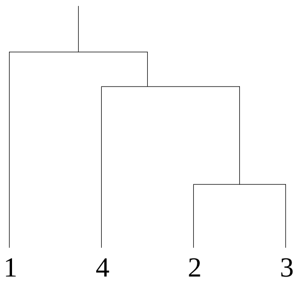
When reconstructing trees from real data, the generating tree is the path of events that evolution actually took. In this case, the true tree is unknown and the objective is, of course, to try and reconstruct it.

In contrast, in simulation studies, a *known* tree T_0 is used to generate artificial sequences and/or distances, under some specified model of evolution. A tree reconstruction method is then applied and its performance can be evaluated by comparing the resulting tree T with the true tree T_0.

Definition Given a phylogenetic tree T. We say that a distance matrix D is *directly obtainable* from T, if it was obtained by adding up edge lengths on paths between leaves.

5.24 The molecular clock hypothesis

Given a distance matrix D, the UPGMA method aims at building a rooted tree T with the property that all leaves have the same distance from the root ρ:

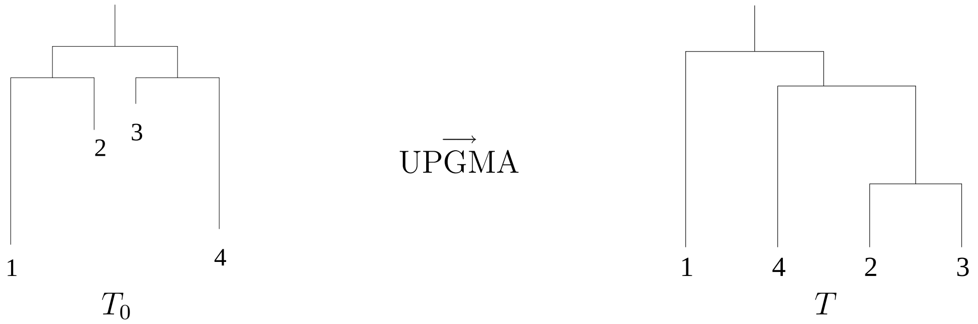


This approach is suitable for sequence data that has evolved under circumstances in which the rate of mutations of sequences is constant over time and for all lineages in the tree.

Definition The assumption that evolutionary events happen at a constant rate is called the *molecular clock* hypothesis.

5.25 UPGMA and the molecular clock

If the input distance matrix D was directly obtained from a generating phylogenetic tree T_0 that adheres to the molecular clock assumption, then the tree T reconstructed by UPGMA from D will equal T_0. Otherwise, if T_0 does not do so, then UPGMA may fail to reconstruct the tree correctly, for example:



The problem here is that the closest leaves in T_0 are not neighboring leaves: they do not have a common parent node.

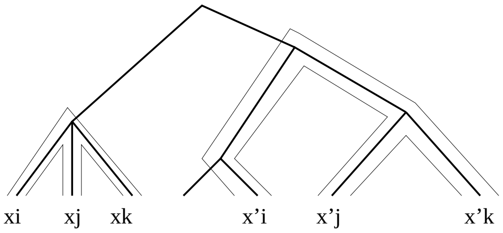
5.26 The ultrametric property

A distance matrix D is called an *ultrametric*, if for any triplet of taxa $x_i, x_j, x_k \in X$, the three distances $d(x_i, x_j)$, $d(x_i, x_k)$ and $d(x_j, x_k)$ have the property that either:

1. all three distances are equal, or

2. two are equal and the remaining one is smaller.

Note that if D was directly obtained from some tree T that satisfies the molecular clock hypothesis, then D is an ultrametric:



condition (1) condition (2)

We say that a rooted phylogenetic tree T is *ultrametric*, if every leaf has the same distance from the root.

One can show the following result:

Theorem A distance matrix D is directly obtainable from some ultrametric tree T, if and only if D is ultrametric.

5.27 Estimating the deviation from a molecular clock

Given a distance matrix D obtained by comparison of sequences generated along some unknown tree T_0. Biologically, it may be of interest to know how well the molecular clock hypothesis holds. In other words, how close is D to being an ultrametric?

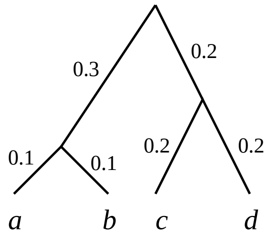
To answer this question, we define the *stretch of an ultrametric U* with respect to D as follows:

$$stretch_D(U) = \max\left\{\frac{D_{ij}}{U_{ij}}, \frac{U_{ij}}{D_{ij}} \mid i, j \in X\right\}.$$

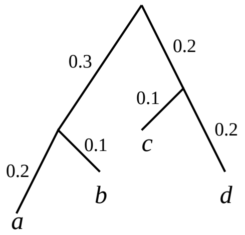
The *stretch of D* is defined as the minimum stretch over all possible ultrametrics: $stretch(D) = \min_U\{stretch_D(U)\}$ and gives a lower bound for the stretch of any tree obtained from D.

This value can be computed in $O(n^2)$ time, see L. Nakhleh, U. Roshan, L. Vawter and T. Warnow, LNCS 2452:287-299 (2002).

Example of an ultrametric U and a non-ultrametric D:



$$\rightarrow \quad U = \left\{ \begin{array}{ccccc} & a & b & c & d \\ a & - & 0.2 & 0.8 & 0.8 \\ b & & - & 0.8 & 0.8 \\ c & & & - & 0.4 \\ d & & & & - \end{array} \right.$$



$$\rightarrow \quad D = \left\{ \begin{array}{ccccc} & a & b & c & d \\ a & - & 0.3 & 0.8 & 0.9 \\ b & & - & 0.7 & 0.8 \\ c & & & - & 0.3 \\ d & & & & - \end{array} \right.$$

The stretch of U w.r.t. D is:

$$stretch_D(U) = \max\left\{\frac{0.3}{0.2}, \frac{0.9}{0.8}, \frac{0.8}{0.7}, \frac{0.4}{0.3}\right\} = \frac{3}{2}.$$

5.28 Additivity and the four-point condition

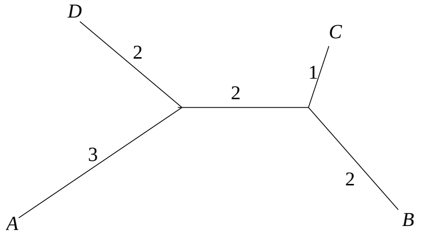
Given a set of taxa X. A distance matrix D on X is called *additive*, if D is directly obtainable from some phylogenetic tree T.

Given an arbitrary distance matrix D. Can we determine whether D is additive without attempting to compute an appropriate tree? The answer is yes, using the following result due to Peter Buneman (1971):

Theorem A distance matrix D on X is additive, iff for any four (not necessarily distinct) taxa $x_i, x_j, x_k, x_l \in X$ the so-called *four-point condition* holds:

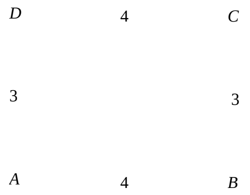
$$d(x_i,x_j) + d(x_k,x_l) \leq \max\left(d(x_i,x_k) + d(x_j,x_l), d(x_i,x_l) + d(x_j,x_k)\right).$$

Distances obtained directly from a phylogenetic tree:



Check the four-point condition with:
$d(A,B) + d(C,D) = 7 + 5 = 12$
$d(A,C) + d(B,D) = 6 + 6 = 12$
$d(A,D) + d(B,C) = 5 + 3 = 8$
$\Rightarrow$ the four-point condition holds.

Euclidean distances in the plane:



Check the four-point condition with:
$d(A,B) + d(C,D) = 4 + 4 = 8$
$d(A,C) + d(B,D) = 5 + 5 = 10$
$d(A,D) + d(B,C) = 3 + 3 = 6$
$\Rightarrow d(A,C) + d(B,D) \not\leq$
$\max\left(d(A,B) + d(C,D), d(A,D) + d(B,C)\right)$
$\Rightarrow$ 4-point condition doesn't hold.

5.29 Neighbor-joining on a known tree

The most widely used distance method is the *neighbor-joining (NJ)* method, originally introduced by Saitou and Nei (1987). Given a distance matrix D, neighbor-joining produces an unrooted phylogenetic tree T with edge lengths. It is especially suitable, when the rate of evolution of the separate lineages under consideration varies.

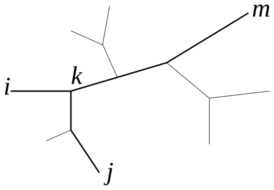
First, consider a tree T and let D be the distance matrix directly obtainable from T, i.e. D is a distance matrix defined on the leaves of T, obtained by adding edge lengths. In preparation of introducing the neighbor-joining algorithm, let us first understand how one could reconstruct T from D.

The following step reduces the number of leaves by one and we can repeatedly apply it until we arrive at a single pair of leaves:

Find a pair of *neighboring leaves*, i.e. leaves that have the same parent node, k. Suppose their numbers are i, j. Remove i, j from the list of nodes and add k to the current list of nodes, defining its distance to leaf m by

$$d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij}).$$

By additivity of D, the distances d_{km} defined in this way are precisely those between equivalent nodes in the original tree:



For any three leaves i, j, m there is a node k where the paths to them meet. By additivity,

$$d_{im} = d_{ik} + d_{km},\ d_{jm} = d_{jk} + d_{km} \text{ and } d_{ij} = d_{ik} + d_{jk},$$

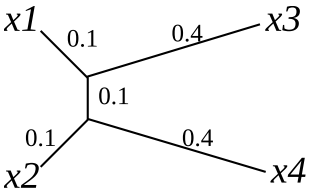
which implies $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$.

5.30 Neighbor-joining

In the above discussion, we assumed that the tree T is known and used it to determine which leaves are neighbors.

The neighbor-joining method is based on the fact that we can decide which nodes are neighboring *without* knowing the tree, but only using the distance matrix.

However, it does *not* suffice simply to pick the two closest leaves, i.e. a pair i, j with d_{ij} minimal, for example:



Given distances generated by this tree. Leaves x_1 and x_2 have minimal distance, but are not neighbors.

To avoid this problem, the trick is to subtract the averaged distances to all other leaves, thus compensating for long edges. We define:

$$N_{ij} := d_{ij} - (r_i + r_j),$$

where

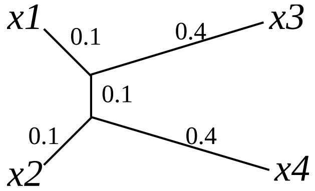
$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik},$$

and L denotes the set of leaves.

Theorem If D is directly obtainable from some tree T, then the two leaves x_i and x_i for which N_{ij} is minimal are neighbors in T.

This result ensures that the neighbor-joining algorithm will correctly reconstruct a tree from its additive distances.

Let us illustrate this result using the previous example:



Here, $r_1 = 0.7$, $r_2 = 0.7$, $r_3 = 1.0$ and $r_4 = 1.0$. And so,

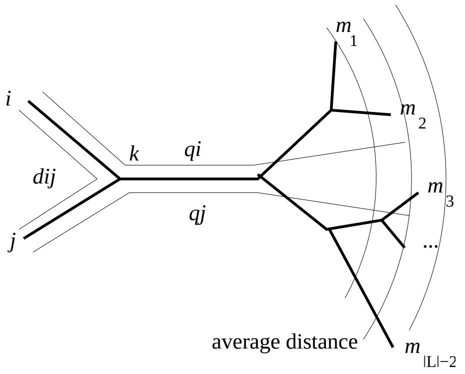
$$N = \left\{ \begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 \\ x_1 & - & -1.1 & -\mathbf{1.2} & -1.1 \\ x_2 & & - & -1.1 & -\mathbf{1.2} \\ x_3 & & & - & -1.1 \\ x_4 & & & & - \end{array} \right.$$

The matrix N attains a minimum value for the pair $i = 1$ and $j = 3$ and for the pair $i = 2$ and $j = 4$, as required.

5.31 The neighbor-joining algorithm

```
Algorithm (Neighbor-joining)
Input: Distance matrix $D$
Output: Phylogenetic tree $T$
Initialization:
    Define $T$ to be the set of leaf nodes, one for each taxon.
    Set $L = T$.
Iteration:
    Pick a pair $i, j \in L$ for which $N_{ij}$ is minimal.
    Define a new node $k$ and
        set $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$, for all $m \in L$.
    Add $k$ to $T$ with edges of lengths $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$ and
        $d_{jk} = d_{ij} - d_{ik}$, joining $k$ to $i$ and $j$, respectively.
    Remove $i$ and $j$ from $L$ and add $k$.
Termination:
    When $L$ consists of two leaves $i$ and $j$, add the remaining
        edge between $i$ and $j$, with length $d_{ij}$.
```

Why do we use $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$ to update distances? By definition, $r_i = \frac{1}{|L|-2}\sum_{m\in L} d_{im} = \frac{1}{|L|-2}\sum_{m\neq i,j} d_{im} + \frac{d_{ij}}{|L|-2}$. In other words, r_i is the average distance $q_i = \frac{1}{|L|-2}\sum_{m\neq i,j} d_{im}$ to all other nodes $m \neq i, j$, plus $\frac{d_{ij}}{|L|-2}$, as indicated here:



As we see from this figure: $2d_{ik} = d_{ij} + q_i - q_j$. This is equivalent to the update formula.

5.32 Application of neighbor-joining

Given an additive distance matrix D directly obtained from a phylogenetic tree T, neighbor-joining is guaranteed to reconstruct T correctly.

However, in practice we are never given a matrix that was "directly obtained" from the generating tree, but rather the distance matrix is usually obtained very indirectly by a comparison of finite sequence data generated along the tree. Such data is rarely additive. Nevertheless, the neighboring-joining method is often applied to such data and has proved to be a fast, useful and robust tree reconstruction method.

5.33 Example

Original data:

$$D_0 = \left\{ \begin{array}{ccccc|c} & x_1 & x_2 & x_3 & x_4 & r \\ x_1 & - & 3 & 5 & 6 & 7 \\ x_2 & 3 & - & 6 & 5 & 7 \\ x_3 & 5 & 6 & - & 9 & 10 \\ x_4 & 6 & 5 & 9 & - & 10 \end{array} \right. \quad \rightarrow \quad N_0 = \left\{ \begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 \\ x_1 & - & -11 & -12 & -11 \\ x_2 & & - & -11 & -12 \\ x_3 & & & - & -11 \\ x_4 & & & & - \end{array} \right.$$

Data after one merge of neighbors:

$$D_1 = \left\{ \begin{array}{cccc|c} & x_1+x_3 & x_2 & x_4 & r \\ x_1+x_3 & - & 2 & 5 & 7 \\ x_2 & & - & 5 & 7 \\ x_4 & & & - & 10 \end{array} \right. \quad \rightarrow \quad N_1 = \left\{ \begin{array}{cccc} & x_1+x_3 & x_2 & x_4 \\ x_1+x_3 & - & -12 & -12 \\ x_2 & & - & -12 \\ x_4 & & & - \end{array} \right.$$

Data after two merges of neighbors:

$$D_2 = \left\{ \begin{array}{ccc|c} & x_1+x_3 & x_2+x_4 & r \\ x_1+x_3 & - & 1 & \\ x_2+x_4 & & - & \end{array} \right. \quad \rightarrow \quad N_2 = \left\{ \begin{array}{ccc} & x_1+x_3 & x_2+x_4 \\ x_1+x_3 & - & quad \\ x_2+x_4 & & - \end{array} \right.$$

5.34 Example

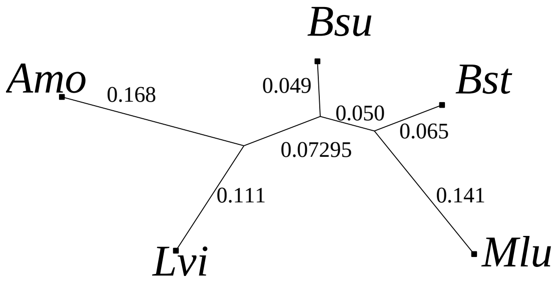
Example of neighbor-joining applied to 5S rRNA data:

Original distances:

	Bsu	Bst	Lvi	Amo	Mlu
Bsu	–	0.1715	0.2147	0.3091	0.2326
Bst		–	0.2991	0.3399	0.2058
Lvi			–	0.2795	0.3943
Amo				–	0.4289
Mlu					

Abbreviations:
Bsu: *Bacillus subtilis*
Bst: *Baclillus stearothermophilus*
Lvi: *Lactobacillus viridescens*
Amo: *Acholeplasma modicum*
Mlu: *Micrococcus luteus*

The resulting tree:



5.35 Rooting unrooted trees

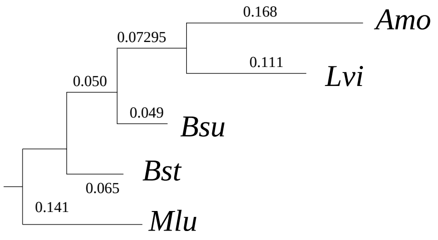
In contrast to UPGMA, most tree reconstruction methods produce an *unrooted* tree. Indeed, determining the root of a tree using computational methods is very difficult.

In practice, the question of rooting a tree is addressed by adding an *outgroup* to the set of taxa under consideration. This is a taxon that is more distantly related to all other taxa then any other of the taxa.

The root is then assumed to be on the branch attaching the outgroup taxon to the rest of the tree.

In practice, selecting an appropriate outgroup can be difficult: if it is too similar to the other taxa, then it might be more related to some than to others. If it is too distant, then there might not be enough similarity to the other taxa to perform meaningful comparisons.

In the above neighboring tree, Mlu is the outgroup, Hence, the rooted version of this tree looks something like this:



This tree is believed to be closer to the correct tree than the one produced earlier using UPGMA. In particular, the UPGMA tree does not separate the outgroup from all other taxa by the root node. The reason why UPGMA produces an incorrect tree is that two of the sequences, those of *L.viridescens* (Lvi) and *A.modicum* (Amo) are very much more diverged than the others.

5.36 Models of evolution

In phylogenetic analysis, a *model of evolution* is given by a rooted tree T, called the *model-*, *true-* or *generating* tree, together with a procedure for generating sequences along the model tree.

Usually, the procedure must determine how to generate an initial sequence at the root of the tree and specify how to "evolve" sequences along the edges of the tree. This involves obtaining intermediate sequences for all internal nodes of the tree, and producing a set of aligned sequences A at the leaves of the tree.

5.37 The Jukes-Cantor model of evolution

T. Jukes and C. Cantor (1969) introduced a very simple model of DNA sequence evolution:

Definition Let T be a rooted phylogenetic tree. The *Jukes-Cantor* model of evolution makes the following assumptions:

1. The possible states for each site are `A`, `C`, `G` and `T`.
2. The initial sequence length is an input parameter and for each site the state at the root is drawn from a given distribution (typically uniform).
3. The sites evolve identically and independently (i.i.d.) down the edges of the tree from the root at a fixed rate u.
4. With each edge $e \in E$ we associate a duration $t = t(e)$ and the expected number of mutations per site along e is $u\tau(e)$. The probabilities of change to each of the 3 remaining states are equal.

How do we "evolve" a sequence down an edge e under the Jukes-Cantor model?

Let $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$ denote the source and target sequences associated with e. We assume that v has already been determined and we want to determine w.

Under the Jukes-Cantor model, the evolutionary event $C_3 =$ *nucleotide changes to one of the other three bases* occurs at a fixed rate of u.

Now consider the event $C_4 =$ *nucleotide changes to any base*. The rate of occurrences of this event is taken to be $\frac{4}{3}u = 4\frac{u}{3}$.

The probability that event C_4 *does not* occur within time t is: $e^{-\frac{4}{3}ut}$ (Poisson distribution with $k = 0$).

Given that the event C_4 occurs, the probability of ending in any particular base is $\frac{1}{4}$.

Let P and Q denote the base pair present at a given site before and after a given time period t. The probability that an event of type C_4 occurs and changes $P \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ to base Q within time t is:

$$\text{Prob}(Q \mid P, t) = \frac{1}{4}\left(1 - e^{-\frac{4}{3}ut}\right).$$

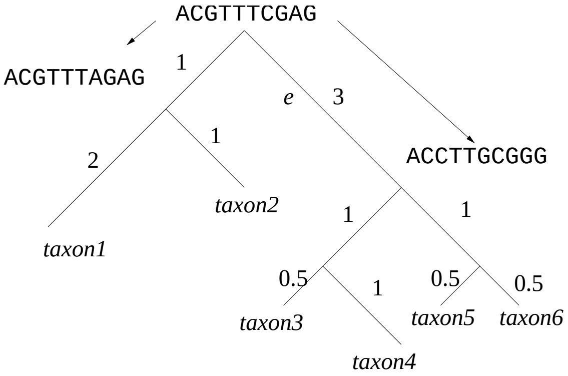
Now, adding the probability that the event C_4 does not occur, we obtain the probability that the state does not change, i.e. $P = Q$:

$$\text{Prob}(Q = P \mid P, t) = e^{-\frac{4}{3}ut} + \frac{1}{4}\left(1 - e^{-\frac{4}{3}ut}\right) = \frac{1}{4}\left(1 + 3e^{-\frac{4}{3}ut}\right).$$

Thus, we obtain a *probability-of-change formula* for the probability of an *observable* change occurring at any given site in time t:

$$\begin{aligned}\text{Prob}(\text{change} \mid t) &= 1 - \text{Prob}(Q = P \mid P, t) = 1 - \tfrac{1}{4}\left(1 + 3e^{-\frac{4}{3}ut}\right) \\ &= \tfrac{3}{4}\left(1 - e^{-\frac{4}{3}ut}\right).\end{aligned}$$

We can use this model to generate artificial sequences along a model tree. E.g., set the sequence length to 10 and the mutation rate to $u = 0.1$. Initially, the root node is assigned a random sequence. Then the sequences are evolved down the tree, at each edge using the probability-of-change formula to decide whether a given base is to change:



The probability of change along edge e is $0.75(1 - e^{-\frac{4}{3}\times 0.1\times 3}) = 0.75(1 - e^{-0.4}) = 0.247$.

5.38 The Jukes-Cantor distance transformation

Given sequences generated under the Jukes-Cantor model of evolution. We would like to calculate the expected number of mutations for any given site on the path in a Jukes-Cantor tree between the leaves a_i and a_j.

Lemma The maximum likelihood distance between a pair of sequences a_i, a_j (that is, the most likely ut to have generated the observed sequences) is given by the following formula:

$$JC(a_i, a_j) = -\frac{3}{4}\ln\left(1 - \frac{4}{3}Ham(a_i, a_j)\right).$$

This is called the *Jukes-Cantor distance transformation.*

Proof Asymptotically, for longer and longer sequences, $Ham(a_i, a_j)$ converges to the probability that any given site will have a different value at a_i than at a_j.

Based on this, we obtain the Jukes-Cantor estimation by setting:

$$Ham(a_i, a_j) = \frac{3}{4}\left(1 - e^{-\frac{4}{3}ut}\right),$$

thus,

$$e^{-\frac{4}{3}ut} = 1 - \frac{4}{3}Ham(a_i, a_j),$$

and so:

$$ut = -\frac{3}{4}\ln\left(1 - \frac{4}{3}Ham(a_i, a_j)\right).$$

□

5.39 Accounting for superimposed events

Hamming distances are suitable for inferring phylogenies between closely related species, given long sequences. For more distantly related sequences, the problem arises that mutation events will take place more than once at the same site. These *superimposed events* will not contribute to the Hamming distances and thus will go undetected.

Note that the expected Hamming distance between two random sequences is $\frac{3}{4}$. Any two sequences a_i, a_j for which $Ham(a_i, a_j) \geq \frac{3}{4}$ holds are called *saturated* w.r.t. each other.

Note that the Jukes-Cantor transformation is undefined for any pair of saturated sequences. In practice, when a saturated pair of taxa is encountered, their JC value is simply set to a large number which may be a fixed-factor times the largest value obtained between any two non-saturated taxa, for example.

5.40 More general transformations

The Jukes-Cantor model assumes that all nucleotides occur with the same frequency. This assumption is relaxed under the *Felsenstein 81* model introduced by Joe Felsenstein (1981), which has the following transformation:

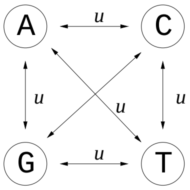
$$F81(a_i, a_j) = -B \ln(1 - Ham(a_i, a_j)/B),$$

where $B = 1 - (\pi_{\mathtt{A}}^2 + \pi_{\mathtt{C}}^2 + \pi_{\mathtt{G}}^2 + \pi_{\mathtt{T}}^2)$ and π_c is the frequency of base c. The base frequency is obtained from the pair of sequences to be compared, or better, from the complete set of given sequences.

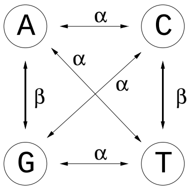
Note that this contains the Jukes-Cantor transformation as a special case with $B = \frac{3}{4}$.

Unlike the Jukes-Cantor transformation, this transformation can also be applied to protein sequences, setting $B = \frac{19}{20}$ if all amino acids are generated with the same frequency, and $B = \sum_{i=1}^{20} \pi_i^2$, in the proportional case.

Both of these transformations assume that all changes of states are equally likely:



However, this is a very unrealistic assumption. For example, in DNA sequences, we observe many more *transitions*, which are purine-to-purine or pyramindine-to-pyramindine substitutions, than *transversions*, which change the type of the nucleotide. We need to model two separate substitution rates α and β:



(Transitions: A ↔ G, C ↔ T, transversions: A ↔ T, G ↔ T, A ↔ C, and C ↔ G.)

Given equal base frequencies, but different proportions P and Q of transitions and transversions between a_i and a_j, the distance for the *Kimura 2 parameter* model is computed as:

$$K2P(a_i, a_j) = \frac{1}{2}\ln\left(\frac{1}{1-2P-Q}\right) + \frac{1}{4}\ln\left(\frac{1}{1-2Q}\right).$$

If we drop the assumption of equal base frequencies, then we obtain the *Felsenstein 84* transformation:

$$F84(a_i, a_j) = -2A\ln\left(1 - \frac{P}{2A} - \frac{(A-B)Q}{2AC}\right) + 2(A-B-C)\ln\left(1 - \frac{Q}{2C}\right),$$

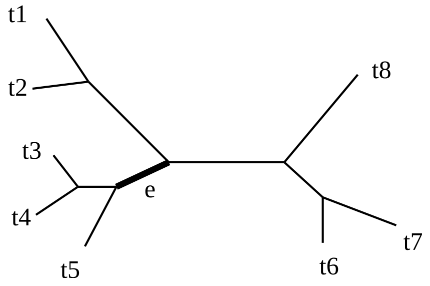
where $\pi_Y = \pi_C + \pi_T$, $\pi_R = \pi_A + \pi_G$, $A = \pi_C\pi_T/\pi_Y + \pi_A\pi_G/\pi_R$, $B = \pi_C\pi_T + \pi_A\pi_G$, and $C = \pi_R\pi_Y$.

Even more general models exist, but we will skip them...

5.41 Trees and splits

Any edge e of T defines a *split* $S = \{A, \bar{A}\}$ of X, that is, a partitioning of X into two non-empty sets A and $\bar{A}$, consisting of all taxa on the one side and other side of e, respectively.

For example:



Here, $A = \{t_3, t_4, t_5\}$ and $\bar{A} = \{t_1, t_2, t_6, t_7, t_8\}$.

We will use $\Sigma(T)$ to denote the *split encoding of* T, i.e. the set of all splits obtained from T. If $x \in X$ and $S \in \Sigma$, then we use $S(x)$ or $\bar{S}(x)$ to denote the split part that contains x, or doesn't contain x, respectively. We define the *size* of a split $S = \{A, \bar{A}\}$ as $\text{size}(S) = \min(|A|, |\bar{A}|)$. A split of size 1 is called a *trivial* split.

5.42 Compatible splits

Given a set of taxa X. Let Σ be a set of splits of X. Two splits $S_1 = \{A_1, \bar{A}_1\}$ and $S_2 = \{A_2, \bar{A}_2\}$ are called *compatible*, if one of the four following intersections

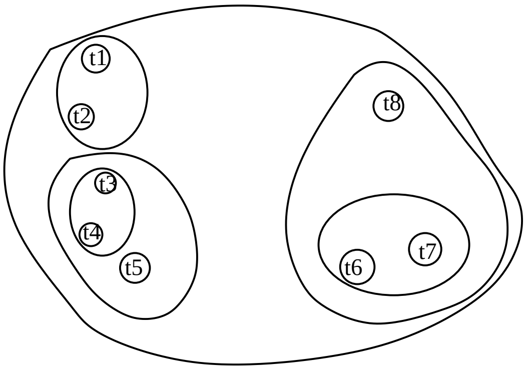
$$A_1 \cap A_2, \quad A_1 \cap \bar{A}_2, \quad \bar{A}_1 \cap A_2, \quad \text{or} \quad \bar{A}_1 \cap \bar{A}_2,$$

is empty.

A set Σ of splits of X is called *compatible*, if every pair of splits in Σ is compatible.

Example
Given the taxa set $X = \{a, b, c, d, e\}$. The splits $S_1 = \{\{a, b\}, \{c, d, e\}\}$, $S_2 = \{\{a, b, c\}, \{d, e\}\}$ and $S_3 = \{\{e\}, \{a, b, c, d\}\}$ are all compatible with each other. However, $S_4 = \{\{a, c\}, \{b, d, e\}\}$ is not compatible with the first one. Hence, the set $\Sigma = \{S_1, S_2, S_3\}$ is compatible, but $\Sigma' = \{S_1, S_2, S_3, S_4\}$ is not.

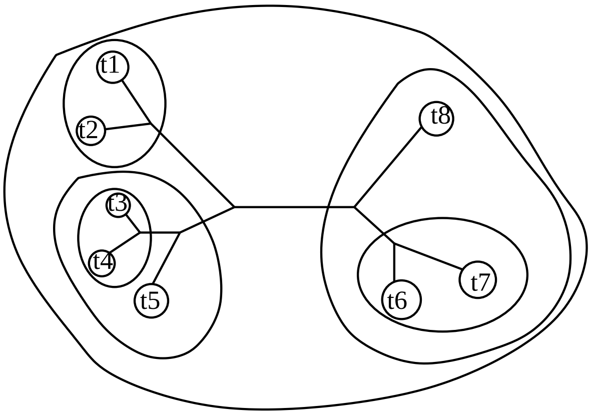
The compatibility condition states that any split S subdivides either the one side, or the other side, of any other split S', but not both sides. Hence, any set of compatible splits can be drawn as follows, without crossing lines:



This figure also shows the relationship between compatible splits and a *hierarchical clustering*. (A *hierarchical clustering* of a set X is a system $\mathcal{H}$ of subsets of X such that $\bigcup_{A \in \mathcal{H}} A = X$ and $A, B \in \mathcal{H} \Rightarrow$ either $A \cap B = \emptyset$, or $A \subset B$ or $B \subset A$.)

5.43 Compatible splits and trees

Any compatible set of splits Σ gives rise to a phylogenetic tree T, for example:



Note: To obtain a one-to-one correspondence between trees and compatible split systems, we now use the relaxed definition of a phylogenetic tree that allows any leaf to carry more than one label and also allows labeling of internal nodes, too.

Vice versa, any tree T gives rise to a compatible set of splits.

(Proof: Consider two edges e and e'. Because T is a tree, e and e' are connected by a unique path P:

$$v \overset{e}{\longleftrightarrow} w \longleftrightarrow P \longleftrightarrow w' \overset{e'}{\longleftrightarrow} v'$$

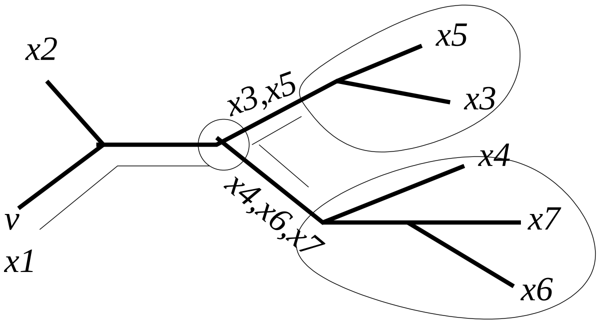
Because T is a tree, the set A, consisting of all nodes reachable from node v not using edge e, is disjoint from A', the set of all nodes reachable from node v' not using edge e'. Hence, the corresponding splits $S = \{A, \bar{A} = X \setminus A\}$ and $S' = \{A', \bar{A}' = X \setminus A'\}$ are compatible.)

In summary:

Theorem A set of splits Σ is compatible, iff there exists a phylogenetic tree T such that $\Sigma = \Sigma(T)$.

5.44 Splits from a tree

Given an unrooted phylogenetic tree T on $X = \{x_1, \ldots, x_n\}$. We can easily compute the set $\Sigma(T)$ in $O(n)$, where n is the number of taxa:



Starting at the leaf labeled x_1, visit all nodes in a post-order traversal, for each edge maintaining and reporting the set of leaves reached after crossing the edge.

The following algorithm recursively visits all nodes and prints out a split after visiting all nodes reachable over a particular edge. Initially, the algorithm is called with $e = null$ and v equal to the node labeled x_1.

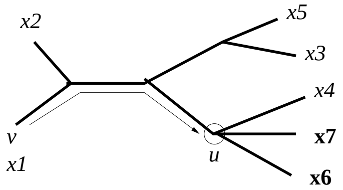
```
**Algorithm** TreeToSplits($e, v$)
Input: A phylogenetic tree $T$ on $X$
Output: The corresponding compatible splits $\Sigma(T)$
Returns: The set $\lambda(e)$ of all taxa separated from $x_1$ by $e$

begin Set $\lambda(e) = \emptyset$
for each edge $f \neq e$ adjacent to $v$ do
    Let $w$ be the opposite node adjacent to $f$
    call TreeToSplits($f, w$) and add the returned labels to $\lambda(e)$
print the split $\{\lambda(e), \overline{\lambda(e)}\}$
return $\lambda(e)$
```

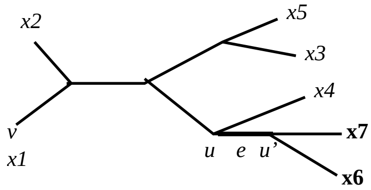
5.45 A tree from splits

Given a compatible set of splits $\Sigma = \{S_1, S_2, \ldots, S_m\}$ of X.

Assume we have already processed the first i splits and have obtained a tree T_i. To incorporate a new edge e to represent the next split $S_{i+1} \in \Sigma$, starting at the node v labeled x_1, we follow a path through the tree until we reach the node w at which e is to be inserted:

Example: insert split $\{x_1, x_2, x_3, x_4, x_5\}$ vs $\{x_6, x_7\}$:





This node is found as follows: if there is only one edge leaving the current node u that *separates* x_1 from elements in $\bar{S}(x_1)$, then we follow this edge. If more than one separating edges exist, then $u = w$ and we create a new edge e from u to a new node u' and all separating edges are moved from u to u'.

```
Algorithm (Splits to tree)
Input: A set $\Sigma = \{S_1, \ldots, S_m\}$ of compatible splits of $X$,
(including all trivial ones)
Output: The corresponding phylogenetic tree $T = (V, E)$ on $X$
Initialization: Let $T$ be a star tree with $n$ leaves labeled $x_1, \ldots, x_n$
Orient all edges away from the node $v$ labeled $x_1$
For $e \in E$, maintain the set $\tau(e)$ of all taxa separated from $x_1$ by $e$

begin
for each non-trivial split $S \in \Sigma$ do
    Set $u = v$
    while there is only edge $e$ leaving $u$ with $\tau(e) \cap \bar{S}(x_1) \neq \emptyset$ do
        Set $u$ equal to the opposite node of $e$
    Let $F$ be the set of all edges $f$ leaving $u$ with $\tau(f) \cap \bar{S}(x_1) \neq \emptyset$
    Create a new edge $e$ from $u$ to a new node $u'$
    Set $\tau(e) = \bigcup_{f \in F} \tau(f)$
    Make all edges in $F$ leave from $w'$ instead of $w$
end
```

Lemma The algorithm constructs the correct tree T in $O(n \log n)$ expected steps.

Proof Assume that we have correctly processed splits $S_1 \ldots, S_p$ and that $\tau(e_i) = \bar{S}_i(x_1)$ for all $i \leq p$. By compatibility of Σ, for every edge e_i we must have either

$$\bar{S}_{p+1}(x_1) \subsetneq \bar{S}_i(x_1) \text{ or } \bar{S}_i(x_1) \subsetneq \bar{S}_{p+1}(x_1).$$

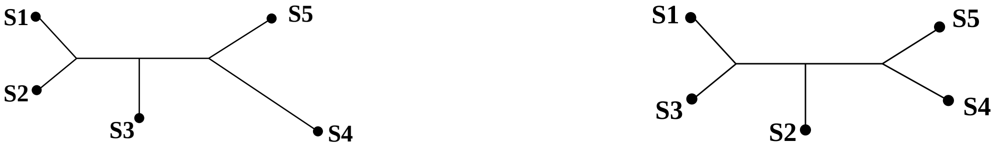
In the former case, the new edge e_{p+1} for S_{p+1} must be inserted on the other side of e_i and this is ensured by the while loop. In the latter case, the while loop is terminated and the edge e_{p+1} is placed between x_1 and all such edges e_i, as required.

There are $O(n)$ splits and the average path distance between any edge and leaf is $\log n$. In summary, the algorithm takes $O(n \log n)$ steps. □

5.46 Comparison of two trees

Given two unrooted phylogenetic trees T_1 and T_2 on the same set of taxa X. We will call the first tree T_1 the *model* tree and T_2 the *reconstructed* tree. How can we measure how similar their topologies are?

Let $\Sigma_1 = \Sigma(T_1)$ and $\Sigma_2 = \Sigma(T_2)$ be the split encodings of the two trees. We define the set of all *false positives (FP)* as $\Sigma_2 \setminus \Sigma_1$. These are all splits contained in the estimated tree that are not present in the model tree. Similarly, we define the set of *false negatives (FN)* as $\Sigma_1 \setminus \Sigma_2$ as the set of all splits missing in the estimated tree, that are present in the model tree.



In this example, there is one false positive $\{S_1, S_3\}$ vs $\{S_2, S_4, S_5\}$ and one false negative $\{S_1, S_2\}$ vs $\{S_3, S_4, S_5\}$.

5.47 Simulation studies

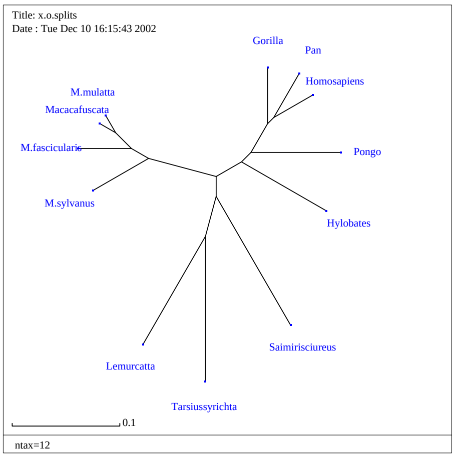
Simulation studies play an important role in bioinformatics. They are used to evaluate the performance of new algorithms on simulated datasets for which the correct answers are know.

For example, to evaluate the performance of a tree construction method, M, we can proceed as follows:

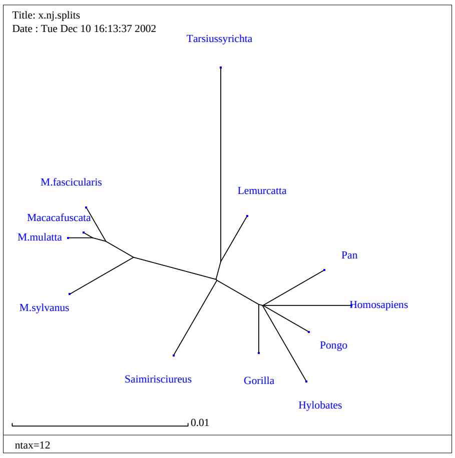
1. Select a model tree $T = (V, E, \omega)$ on X.
2. For a specified mutation rate u, generate a set of aligned sequences A on T under the Jukes-Cantor model.
3. Apply the method M to A to obtain a tree $M(A)$.
4. Compute the number of false positives and false negatives.
5. Repeat many times for many different parameters and report the average performance of the method.

Example Under the Jukes-Cantor model with $L = 1000$ and $u = 0.1$, sequences were generated on a model tree, then Hamming distances were computed, then neighbor-joining was applied:

Model tree



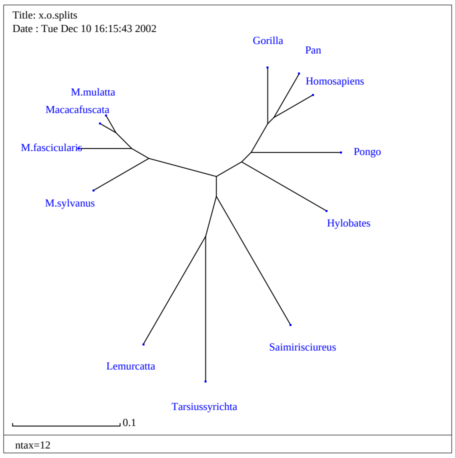
neighbor-joining tree



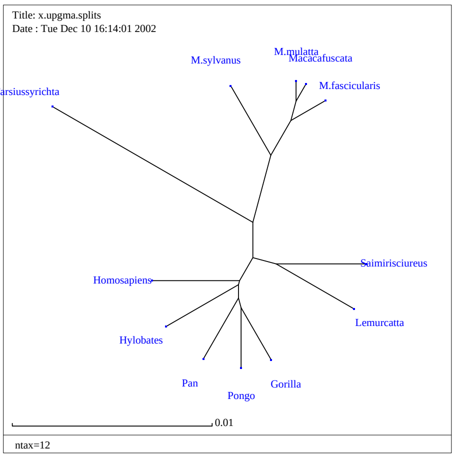
$$FP = FN = 3$$

Example Under the Jukes-Cantor model with $L = 1000$ and $u = 0.1$, sequences were generated on a model tree, then Hamming distances were computed, then UPGMA was applied:

Model tree

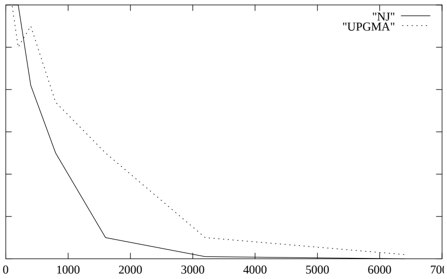


UPGMA tree

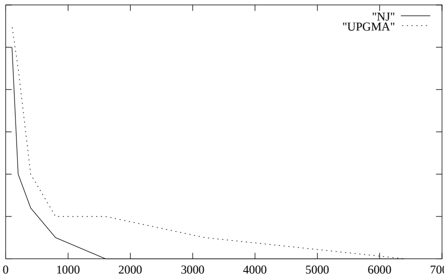


$$FP = FN = 5$$

Using the same tree and computations, here we plot the number of false positives obtaining for $u = 0.1, 0.5$ and different sequence lengths L ranging from $100 - 6400$. Each point plotted was obtained as the average of five independent simulations:



L vs FP, $u = 0.1$



L vs FP, $u = 0.4$

For this particular tree, neighbor-joining displays a slightly better performance than UPGMA. However, to obtain significant results, many trees and many different parameters must be considered.

5.48 Algorithms vs optimality criteria

In phylogenetics, the goal is to *estimate* the true evolutionary tree that generated a set of extant taxa. Tree reconstruction methods attempt to accomplish this goal by *selecting* one of the many different possible topologies.

This selection process is performed in one of two ways:

1. The tree selected by an *algorithmic* method such as neighbor-joining or UPGMA is defined by the sequence of steps that make up the algorithm. This tree does not solve any explicitly formulated optimization problem.
2. The tree selected by an *optimization* method is a solution to an explicitly formulated optimization problem such as "maximum parsimony" or "maximum likelihood". Here, algorithms play a secondary role as a means of obtaining or approximating a solution to the optimization problem.

Distance methods are usually *algorithmically* defined and have polynomial run time. Sequence methods usually involve solving an *optimization* problem by finding an optimal tree with respect to some criterion and are usually NP-hard.

Algorithmic methods combine the computation and definition of the preferred tree into a single statement.

Optimization methods involve formulating and solving two problems:

- computing the cost for a given tree T, and
- searching through all trees to find a tree that minimizes the cost.

5.49 Maximum parsimony

The maximum parsimony method is by far the most used sequence-based tree reconstruction method.

In science, the principal of *maximum parsimony* is well known: always use the simplest, most parsimonious explanation of an observation, until new observations force one to adopt a more complex theory.

In phylogenetic analysis, the *maximum parsimony problem* is to find a phylogenetic tree that explains a given set of aligned sequences using the minimum number of substitutions.

5.50 The parsimony score of a tree

The *difference* between two sequences $x = (x_1, \dots, x_L)$ and $y = (y_1, \dots, y_L)$ is simply their non-normalized Hamming distance

$$\text{diff}(x, y) = |\{k \mid x_k \neq y_k\}|.$$

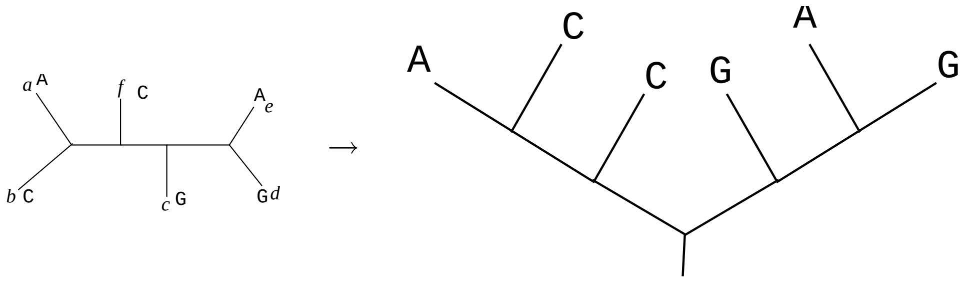
Given a multiple alignment of sequences $A = \{a_1, a_2, \dots, a_n\}$ and a corresponding phylogenetic tree T, leaf-labeled by A.

If we assign a hypothetical ancestor sequence to every internal node in T, then we can obtain a score for T, together with this assignment, by summing over all differences $\text{diff}(x, y)$, where x and y are any two sequences labeling two nodes that are joined by an edge in T.

The minimum value obtainable in this way is called the *parsimony score* $PS(T, A)$ of T and A.

Can the parsimony score of a tree T be computed efficiently?

As the parsimony score is obtained by summing over all columns, the columns are independent and so it suffices to discuss how to obtain an optimal assignment for one position:



unrooted tree rooted tree

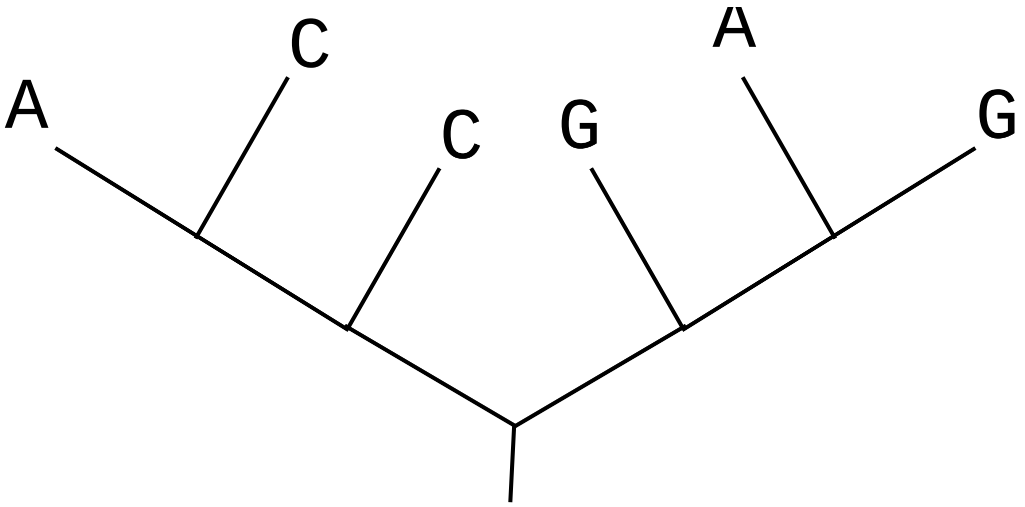
5.51 The Fitch algorithm

The following algorithm computes the parsimony score for T and a fixed column in the sequence alignment. It modifies a global score variable and is repeatedly run to obtain the total score. Initially it is called with $e = null$ and v the root node.

```
**Algorithm** ParsimonyScore($e, v$) (Walter Fitch 1971)
Input: A phylogenetic tree $T$ and a character $c(v)$ for each leaf $v$
Output: The parsimony score for $T$ and $c$
**if** $v$ is a leaf node **then**
  Set $R(v) = \{c(v)\}$
**else**
  **for each** edge $f_1, f_2 \neq e$ adjacent to $v$ **do**
      Let $w_i$ be the opposite node of $f_i$
```

```
        Call ParsimonyScore($f_i, w_i$) // to compute $R(w_i)$
    if $R(w_1) \cap R(w_2) \neq \emptyset$ then
        Set $R(v) = R(w_1) \cap R(w_2)$
    else
        Set $R(v) = R(w_1) \cup R(w_2)$ and increment the global score
end
```

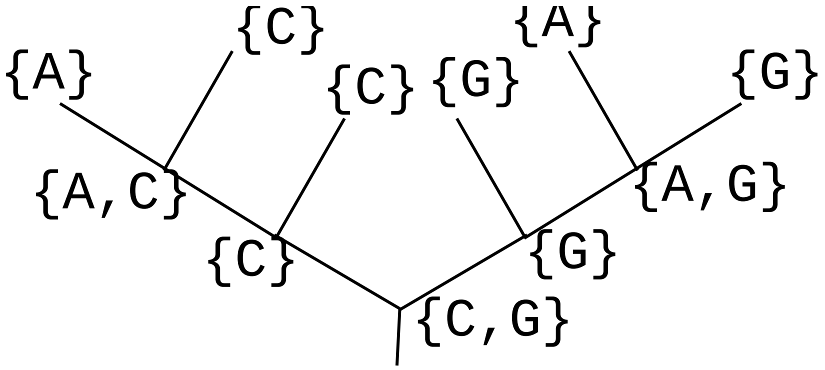
We can use the Fitch algorithm to show that the parsimony score for the following labeled tree is 3:



In total, the algorithm requires $O(nL)$ steps.

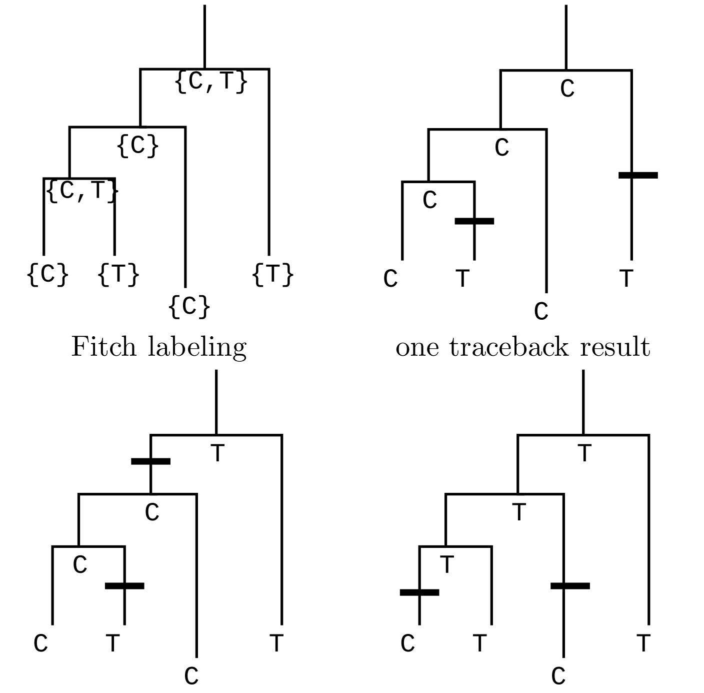
5.52 Traceback

The above algorithm computes the parsimony score. An optimal labeling of the internal nodes is obtained via traceback: starting at the root node r, we label r using any character in $R(r)$. Then, for each child w, we us the same letter, if it is contained in $R(w)$, otherwise we us any letter in $R(w)$ as label for w. We then visit the children von w etc:



Again, the algorithm requires $O(nL)$ steps, in total.

Example:



Fitch labeling one traceback result

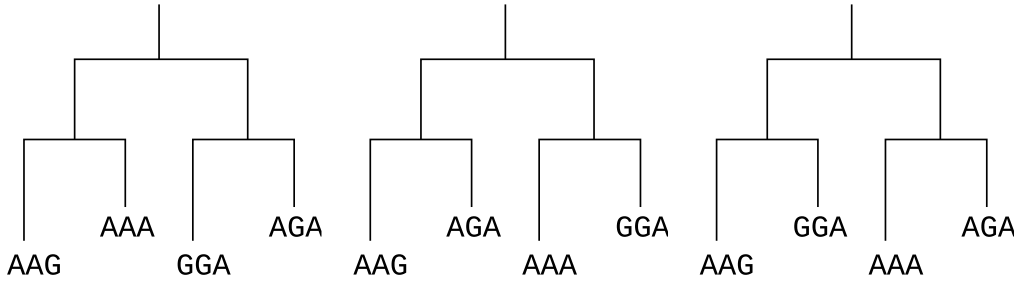
another traceback result not obtainable by traceback

5.53 A simple example

Assume we are given the following four aligned sequences:

```
A A G
A A A
G G A
A G A
```

There are three possible binary topologies on four taxa:



In each tree, label all internal nodes with sequences so as to minimize the score obtained by summing all mismatches along edges. Which tree minimizes this score?

5.54 The Fitch algorithm on unrooted trees

Above, f.e.o.e. we formulated the Fitch algorithm for rooted trees. However, the minimum parsimony cost is independent of where the root is located in the tree T:

To see this, consider a root node ρ connected to two nodes v and w, and let $c(\cdot)$ denote the character assigned to a node in the Fitch algorithm:

1. If $c(v) = c(w)$, then $c(\rho) = c(v) = c(w)$, because any other choice would add 1 to the score.

2. If $c(v) \neq c(w)$, then either $c(\rho) = c(v)$ or $c(\rho) = c(w)$, and both choices contribute 1 to the score.

In both cases, we could replace ρ by a new edge e that connects v and w without changing the parsimony score of T.

Thus, we can run the Fitch algorithm on the unrooted version of T, using *any* internal node in the initial call.

5.55 The parsimony score

Given a multiple alignment $A = \{a_1, \ldots, a_n\}$, it's *parsimony score* is defined as

$$PS(A) = \min\{PS(T, A) \mid T \text{ is a phylogenetic tree on } A\}.$$

The *maximum parsimony problem* is to compute $PS(A)$.

Potentially, we need to consider all $(n-5)!!$ possible trees. Unfortunately, in general this can't be avoided and the maximum parsimony problem is known to be NP-hard.

Exhaustive enumeration of all possible tree topologies will only work for $n \leq 10$ or 11, say.

Thus, we need more efficient strategies that either solve the problem exactly, such as the branch and bound technique, or return good approximations, such as heuristic searches.

Remark: As with most biological problems, we are not only interested in the optimal solution, but would also like to know something about other near optimal solutions as well.

5.56 Branch and bound

Recall how we obtained an expression for the number $U(n)$ of unrooted phylogenetic tree topologies on n taxa:

For $n = 3$ there are three ways of adding an extra edge with a new leaf to obtain an unrooted tree on 4 leaves. This new tree has $(2n-3) = 5$ edges and there are 5 ways to obtain a new tree with 5 leaves etc.

To be precise, one can obtain any tree T_{i+1} on $\{a_1, \ldots, a_i, a_{i+1}\}$ by adding an extra edge with a leaf labeled a_{i+1} to some (unique) tree T_i on $\{a_1, \ldots, a_i\}$.

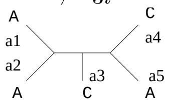
In other words, we can produce the set of *all* possible trees on n taxa by adding one leaf at a time in all possible ways, thus systematically generating a complete *enumeration tree.*

A simple, but crucial observation is that adding a new sequence a_{i+1} to a tree T_i to obtain a new tree T_{i+1} cannot lead to a smaller parsimony score.

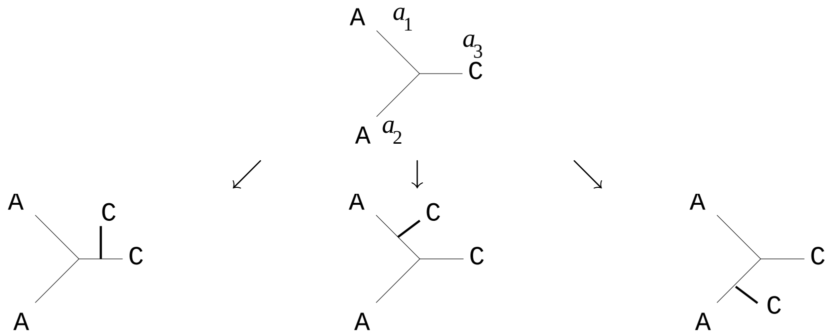
This gives rise to the following bound criterion when generating the enumeration tree: if the *local* parsimony score of the current incomplete tree T' is larger or equal to the best *global* score for any complete tree seen so far, then we do not generate or search the enumeration subtree below T'.

In practice, using branch and bound one can obtain exact solutions for data sets of twenty or more sequences, depending on the sequence length and the messiness of the data.

A good starting strategy is to first compute a tree T_0 for the data, e.g. using neighbor-joining, and then to initialize the *global* bound to the parsimony score of T_0.

Example Assume we are given an msa $A = \{a_1, a_2, \ldots, a_5\}$ and at a given position i the characters are: $a_{1i} = \texttt{A}$, $a_{2i} = \texttt{A}$, $a_{3i} = \texttt{C}$, $a_{4i} = \texttt{C}$ and $a_{5i} = \texttt{A}$. Assume that the neighbor-joining tree on A looks like this: 

and thus gives a global upper bound of $global = 2$ for position i. The first step in generating the enumeration tree is this:

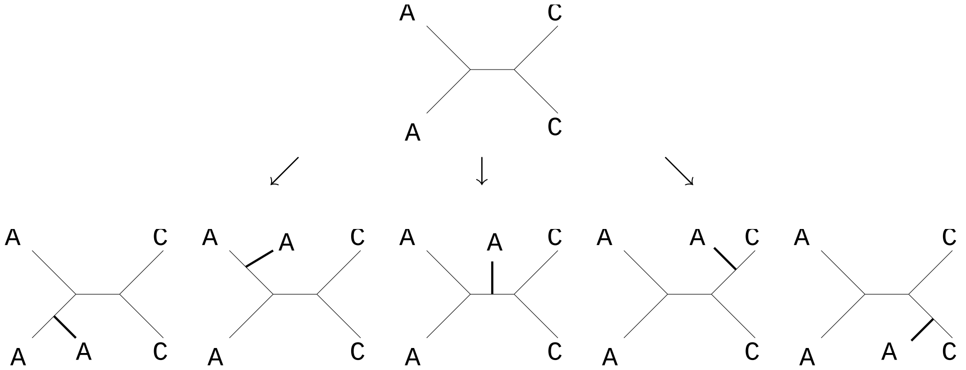


$local = 1 < global$ continue

$local = 2 \not< global$ don't continue

$local = 2 \not< global$ don't continue

Only the first of the three trees fulfills the bound criterion and we do not pursue the other two trees. The second step in generating the enumeration tree looks like this:



$local = 1$ $local = 1$ $local = 1$ $local = 2$ $local = 2$

The first three trees are optimal. Note that the bound criterion in the first step reduced the number of full trees to be considered by two thirds.

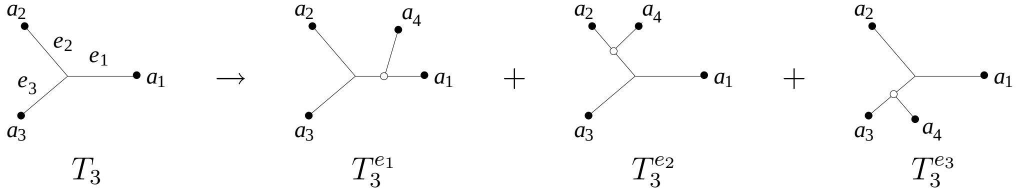
Application of branch-and-bound to evolutionary trees was first suggested by Mike Hendy and Dave Penny (1982).

5.57 The stepwise-additional heuristic

Now we discuss a simple greedy heuristic (Felsenstein 1981) for approximating the optimal tree or score. We build the tree T by adding one leaf after the other, in each step choosing the optimal position for the new leaf-edge:

Given a multiple sequence alignment $A = \{a_1, a_2, \ldots, a_n\}$. Start with a tree T_2 consisting of two leaves labeled a_1 and a_2.

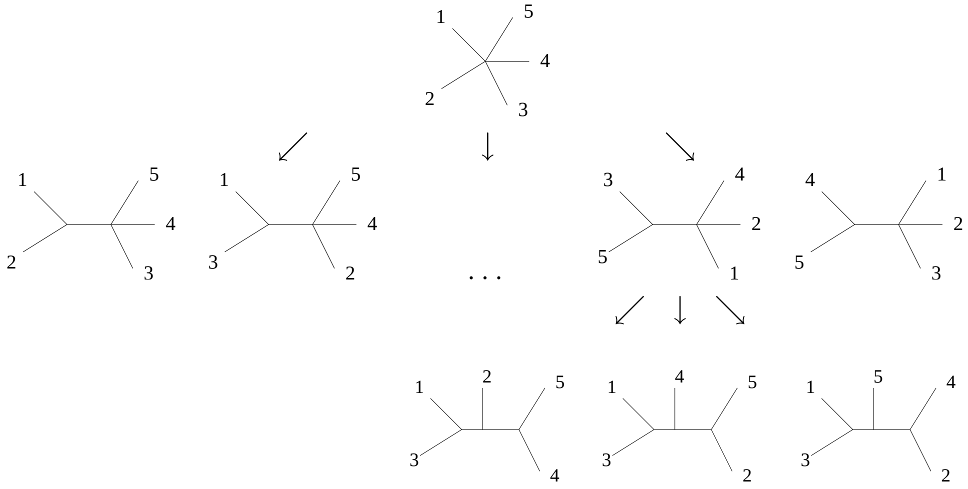
Given T_i. For each edge e in T_i, obtain a new tree T_i^e as follows: Insert a new node v in e and join it via a new edge f to a new leaf w with label a_{i+1}. Set $T_{i+1} = \arg\min\{P(T_i^e, A)\}$.



Obviously, this approach is not guaranteed to obtain an optimal result. Moreover, the result obtained will depend on the order in which sequence are processed.

5.58 The star-decomposition heuristic

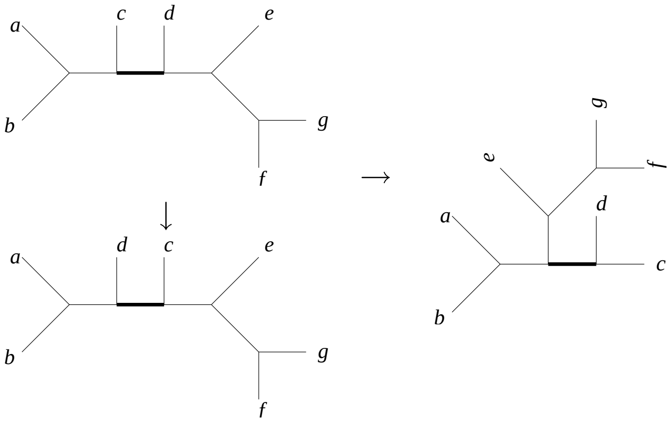
This employs a similar strategy to neighbor-joining. We start with a star tree on all n taxa. At each step, the optimality criterion is evaluated for every possible joining of a pair of lineages incident to the central node. The best tree found at one step is then used as the basis of the next step:



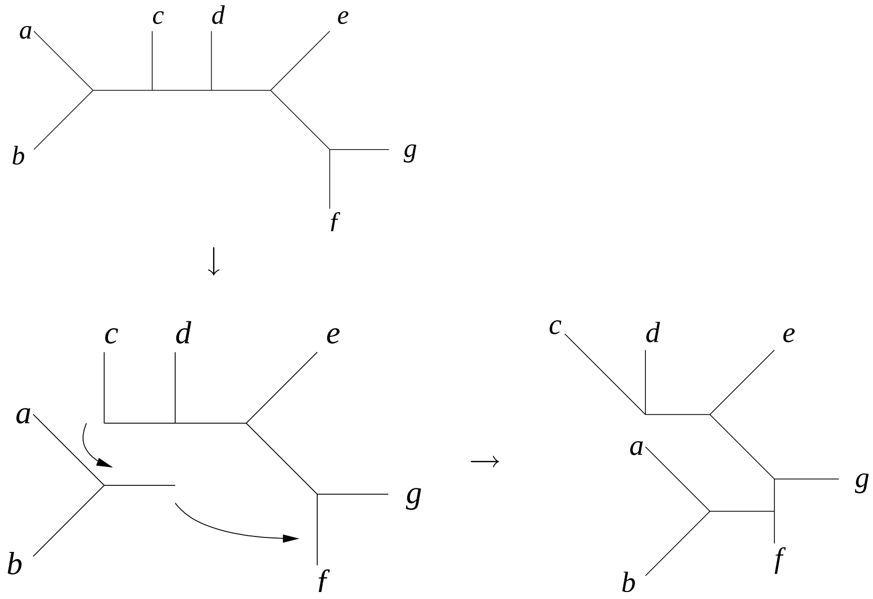
5.59 Branch swapping methods

The two heuristics just described are both very susceptible to entrapment in local optima. We now discuss a number of *branch-swapping operations* that one can use to move through the space of all trees, hopefully jumping far enough to escape from local optima.

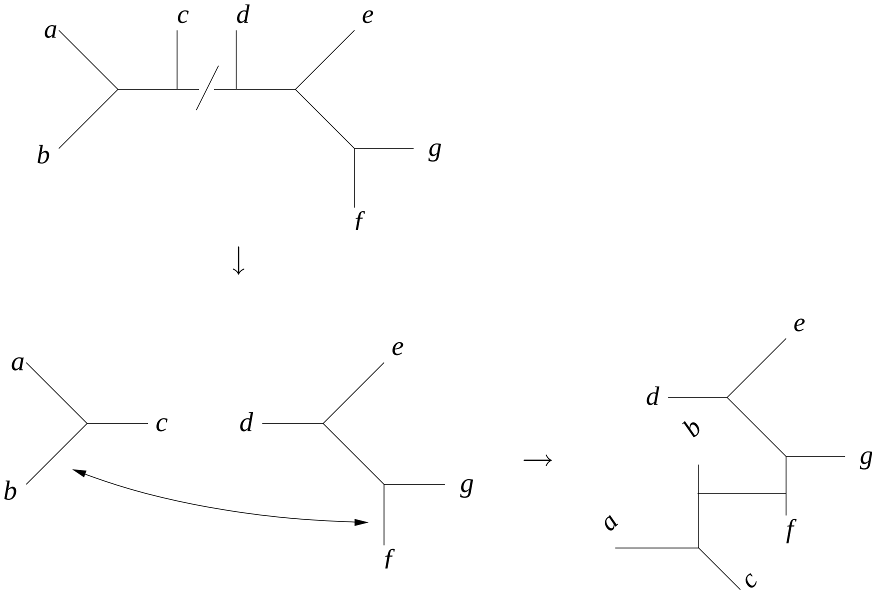
In a *nearest-neighbor interchange (NNI)*, two of the four subtrees around an edge are swapped, in two different ways:



In branch swapping by *subtree pruning and re-grafting*, a subtree is pruned from the tree and re-grafted to a different location of the tree:



In branch swapping by *tree bisection and reattachment*, the tree is bisected at an edge, yielding two subtrees. The two subtrees are then reconnected at two new positions:



5.60 Heuristic search

If the data set $A = \{a_1, \dots, a_n\}$ is too big to be solved exactly via branch and bound, then we can use a heuristic search method in an attempt to find or approximate the optimal solution.

This involves searching through the space of all unrooted phylogenetic trees on n labels and trying to proceed toward a globally optimal one. We "move" through tree space using one or more branching-swapping techniques.

Heuristic searches employ *hill-climbing techniques*: we imagine that the "goodness" $-PS(T)$ of the solution as a landscape along which we move during the search. The general strategy is to always move upwards in the hope of reaching the top of the highest peak.

Even using the above described branch-swapping techniques, any heuristic search is in danger of "climbing the wrong mountain" and getting stuck in a local optimum. Different strategies have been developed to avoid this problem.

5.61 Simulated annealing

The *simulated annealing* method employs a *temperature* that cools over time (Van Laarhoven and Aarts, 1987). At high temperatures the search can move more easily to trees whose score is less optimal than the score of the current tree. As the temperature decreases, the search becomes more and more directed toward better trees.

I.e., let T_i denote the current tree at step i and let $z(T_i)$ denote the goodness of T_i (e.g., $-PS(T)$). In hill climbing, a move to T_{i+1} is acceptable, if $z(T_{i+1}) \geq z(T_i)$. In simulated annealing, *any* new solution is accepted with a certain probability:

$$Prob(\text{accepting solution } T_{i+1})$$

$$= \begin{cases} 1 & \text{if } z(T_{i+1}) \geq z(T_i) \\ e^{-t_i(z(T_{i+1})-z(T_i))} & \text{otherwise,} \end{cases}$$

where t_i is called the *temperature* and decreases over time.

5.62 The Great Deluge method

The *Great Deluge* method, introduced by Guenter Dueck and Tobias Scheuer (1990), employs a slowly rising water level and the search accepts any move that stays above the water level.

The probability of accepting a new solution T_{i+1} is 1, if $z(T_{i+1}) > w_i$, where w_i is a bound that increases slowly with time.

If T_{i+1} is accepted, then we update the water level by setting

$$w_{i+1} = c \times (z(T_{i+1}) - z(T_i))\,.$$

Typically, the constant c is usually about 0.01 to 0.05.

Another of the many heuristics is *tabu search* method (Glover, 1989) that maintains a *tabu list* of $5 - 10$ solutions recently visited and prevents the search from revisiting any solutions it has just tried.

5.63 A 2-approximation for maximum parsimony

Given a connected graph $G = (V, E)$ with edge weights $\omega(\cdot)$. A *minimum spanning tree* $ST = (V', E')$ on G is a tree with $V' = V$ and $E' \subset E$ such that $\sum_{e \in E'} \omega(e)$ is minimum.

Given a multiple alignment of sequences $A = \{a_1, \ldots, a_n\}$. Consider the complete graph $G(A)$ on n nodes whose nodes are labeled by the sequences and whose edge lengths $\omega(\cdot)$ are set to the non-normalized Hamming distances between the sequences.

Theorem Let ST be a minimum spanning (phylogenetic) tree on $G(A)$. Then ST is a 2-approximation to the most parsimonious tree on A. In other words. the parsimony score $PS(ST, A)$ is at most twice as large the optimal parsimony score $PS(A)$.

Note that a minimum spanning tree is easily computed in polynomial time. This gives us an upper bound on the parsimony length of the most parsimonious tree for a given input.

Proof Let T^* be a most parsimonious tree on A and let $2T^*$ denote the graph obtained by duplicating all edges in T^*.

By construction, each node of $2T^*$ has even degree. Hence, there exists a *Eulerian* tour C of $2T^*$ that uses every edge exactly once (as shown by Euler). From C we obtain a (shorter) tour C' of all nodes in $G(A)$ by simply visiting all nodes in the order that they appear in C.

Let $\omega(C)$ denote the sum of weights of edges in C and define $\omega(C')$ similarly. Then, clearly

$$\omega(C) = \omega(2T^*) = 2\omega(T^*),$$

since C is Eulerian, and also

$$\omega(C') \leq \omega(C),$$

since Hamming distances satisfy the triangle inequality.

Note that if we delete any single edge in C', then we obtain a path P' with

$$\omega(P') \leq \omega(C').$$

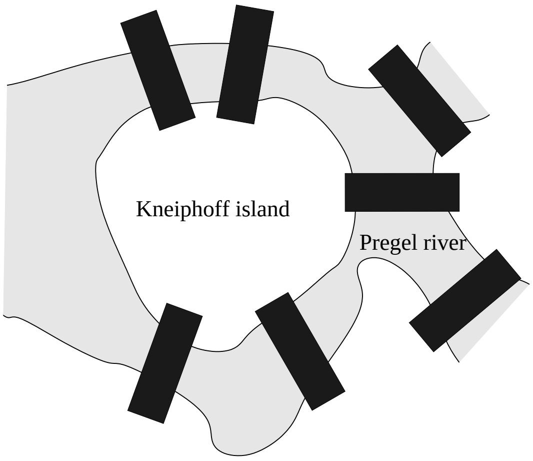
Let ST be a minimum spanning tree for $G(A)$. Since P' is also a spanning tree, we have

$$\omega(ST) \leq \omega(P') \leq \omega(C') \leq \omega(C') = 2\omega(T^*),$$

proving the claim. □

5.64 Euler Path

Leonard Euler wanted to know whether there exists a path that uses all seven bridges in Königsberg exactly once:



Birth of graph theory...

5.65 Maximum likelihood estimation (MLE)

Given a multiple alignment $A = \{a_1, \ldots, a_n\}$. Assuming a specific model of evolution M, one may attempt to *estimate* a phylogenetic tree T with edge lengths ω that *maximizes the likelihood*

$$P(A \mid T)$$

of generating the sequences $a_1, \ldots, a_n$ at the leaves of T.

A main attraction of maximum likelihood estimation (MLE) is that it provides a systematic frame-work for explicitly incorporating assumptions and knowledge about the process that generated the given data.

One potential draw-back is that any given model of evolution is only a rough estimation of real-world biological evolution. Fortunately, in practice, maximum likelihood methods have proved to be quite robust to many violations of the assumptions formulated in the models. MLE methods work very well on small data sets.

Similar to maximum parsimony, an optimal MLE tree is determined by a search in tree space. One can attempt to find an exact solution, using branch and bound techniques, or one can attempt to find a good approximate solution using a heuristic search technique...

A main draw-back of MLE is that, like maximum parsimony, finding an optimal tree is NP-hard.

In the case of maximum parsimony, we are able to compute the parsimony score for any given tree in polynomial time using the Fitch algorithm.

In the case of MLE, no such fast method of evaluating a single tree exists (unless $P = NP$), as the evaluation of a single tree is known to be NP-hard.

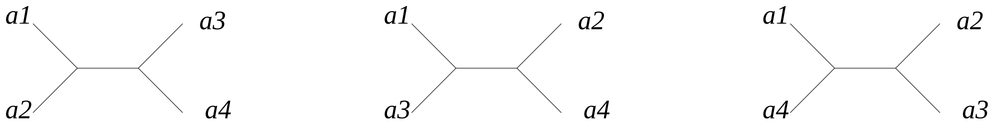
Given a multiple alignment $A = \{a_1, \dots, a_n\}$, MLE seeks to determine the topology (evolutionary branching order) and branch lengths of the true or generating tree.

This is done under the assumption of a model of evolution, such as the Jukes-Cantor model, described above, or more general models. Such models for biological sequences are usually *time reversible* and thus the likelihood of a tree is generally independent of the location of the root.

Example Assume that we are given the following msa $A = \{a_1, \dots, a_4\}$:

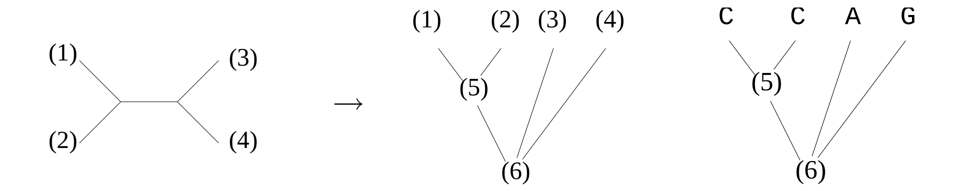
	1	2			i			N
sequence a_1	A	C	...	G	C	G	...	A
sequence a_2	A	C	...	G	C	C	...	G
sequence a_3	A	C	...	T	A	C	...	G
sequence a_4	A	C	...	T	G	G	...	A

There are three possible unrooted tree topologies on four taxa:



We now discuss how to compute the maximum likelihood for the first tree. The other trees are processed similarly.

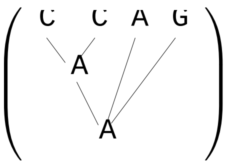
For simplicity, we root the tree at an arbitrary internal node and then consider each position $i = 1, 2, \dots, N$ in the sequence:



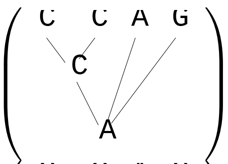
original tree topology rooted version labeled by characters at position i

Schematically, we obtain the likelihood that *this* tree generated the characters seen at position i of the multiple alignment by summing over *all possible* labellings of the internal nodes by characters:

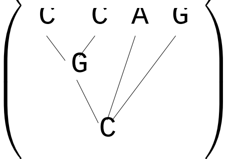
$L(i) =$ Prob



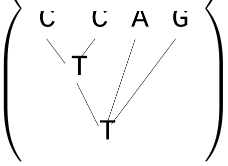
$+$ Prob



$+ \quad \dots \quad +$ Prob



$+ \quad \dots \quad +$ Prob



We then multiple the likelihoods obtained for each position:

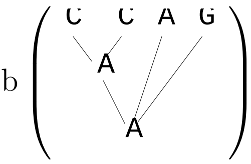
$$L = L(1) \cdot L(2) \cdot \ldots \cdot L(N) = \prod_{i=1}^{N} L(i).$$

Obviously, the individual probabilities will often be very small and so we add their logarithms instead of using multiplication:

$$\ln L = \ln L(1) + \ln L(2) + \ldots + \ln L(N) = \sum_{i=1}^{N} \ln L(i).$$

Actually, the situation is more complicated as we must determine the choice of edge lengths for the given tree that produces the highest likelihood.

How do we compute e.g. Prob



?

Let us look at this under the Jukes-Cantor model of evolution with a fixed mutation rate u, as discussed above. For any edge e, let P and Q denote the labels at the two opposite ends of e. The probability of $P = Q$ is given by

$$\text{Prob}(Q = P \mid T) = \frac{1}{4}(1 + 3e^{-\frac{4}{3}ut}),$$

and the probability that the two characters differ is

$$1 - \text{Prob}(Q = P \mid T) = \frac{3}{4}(1 - e^{-\frac{4}{3}ut}),$$

where $t = \omega(e)$.

The total probability that *this* tree with *this* labeling of internal nodes generated the observed data at the leaves of the tree is obtained by multiplication of the probabilities for each edge.

5.66 A MLE heuristic: Quartet puzzling

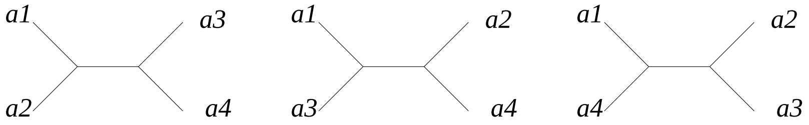
Korbinian Strimmer and Arndt von Haeseler (1996) proposed a straight-forward heuristic for approximating the MLE tree that is based on the idea of computing MLE trees on quartets of taxa and then using many rounds of combination steps in which one attempts to fit a random set of quartet trees together so as to obtain a tree on the whole dataset.

Given a multiple alignment of sequences $A = \{a_1, \dots, a_n\}$. The quartet puzzling heuristic proceeds in three steps:

1. Compute a maximum likelihood tree on each quartet of distinct taxa $a_i, a_j, a_k, a_l \in A$.
2. Choose a random order $a_{i_1}, a_{i_2}, \dots, a_{i_n}$ of the taxa and then use it to guide a combining or *puzzle* step that produces a tree on the whole data set from quartet trees.
3. Repeat step (2) many times to obtain trees $T_1, T_2, \dots, T_R$. Report the *majority consensus tree* T.

5.67 Quartets

Given four taxa $\{a_1, a_2, a_3, a_4\}$, There are three possible binary topologies, each characterized by the one non-trivial split that they contain:

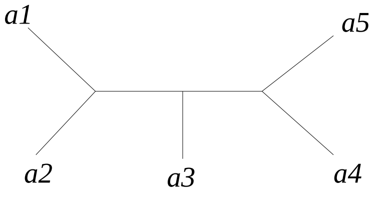


$\{\{a_1, a_2\}, \{a_3, a_4\}\}$ $\{\{a_1, a_3\}, \{a_2, a_4\}\}$ $\{\{a_1, a_4\}, \{a_2, a_3\}\}$

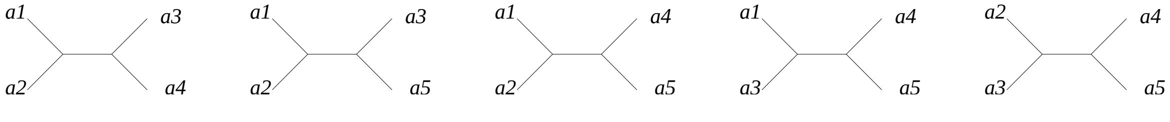
Now, let T be any phylogenetic tree that contains leaves labeled a_1, a_2, a_3, a_4, We say that T *induces* the split or tree topology $\{\{a_1, a_2\}, \{a_3, a_4\}\}$, if the exists edges in T that separates the nodes labeled a_1 and a_2 from the nodes a_3 and a_4, in which case we say that a_1 and a_2 (or a_3 and a_4) are *neighbors* (respectively).

We will sometimes abbreviate $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ to $a_1, a_2 \mid a_3, a_4$.

Given this tree T:



It induces the following tree topologies on four taxa:

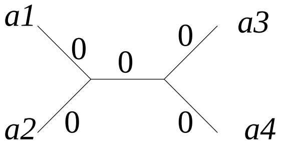


From the set of induced quartets one can reconstruct the original tree. However, in the presence of false quartet topologies as produced by any phylogenetic reconstruction method, obviously the reconstruction of the original tree can become difficult.

5.68 The initialization

In quartet puzzling, quartets are greedily combined to produce a tree for the whole data set. In an attempt to avoid the usual problems of a greedy approach, the combining step is run many times using different random orderings of the taxa.

The algorithm is initialized with the quartet tree T_4 produced on $\{a_1, a_2, a_3, a_4\}$, e.g.:



Each edge e is labeled with a *number of conflicts* $c(e)$ that is originally set to 0.

5.69 The puzzle step

In the puzzle step, the task is to decide how to insert the next taxon a_{i+1} in to the tree T_i already constructed for taxa $\{a_1, \dots, a_i\}$. We proceed as follows:

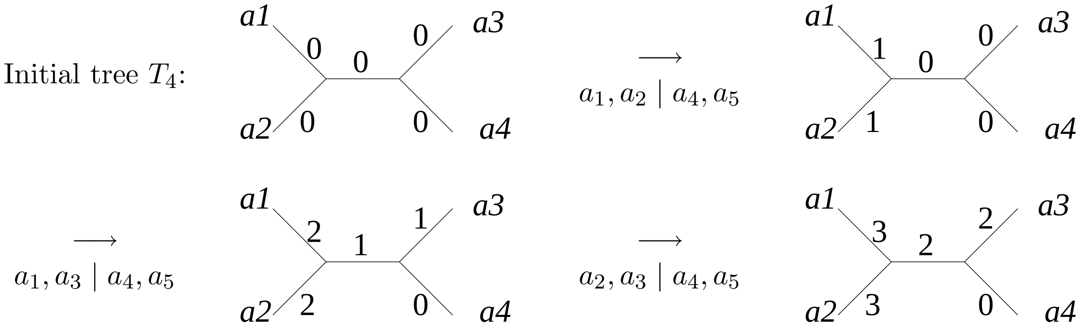
Set $c(e) = 0$ for all edges in T_i.

For each edge e in T_i we consider what would happen if we were to attach taxon a_{i+1} into the middle of e by a new edge:

We consider all quartets consisting of the taxon a_{i+1} and three taxa from $\{a_1, a_2, \dots, a_i\}$. For each such quartet q we compare the precomputed maximum likelihood tree on q with the tree induced on q by T_i. If these two trees differ, then we increment $c(e)$ by one.

Finally, we attach e to an edge f in T_i that has a minimum number of conflicts $c(f)$, thereby obtaining T_{i+1}.

Example: We illustrate the puzzle step using the five taxon tree shown above and assume that MLE indeed produced the five listed correct quartet topologies. The puzzle step for a_5 proceeds as follows:



Hence, the puzzle step suggests that taxon a_5 should be attached in the edge adjacent to the leaf labeled a_4.

5.70 The quartet puzzling algorithm

Algorithm Quartet puzzling
Input: Aligned sequences $A = \{a_1, \dots, a_n\}$, number of trees R
Output: Phylogenetic trees $T^1, T^2, \dots, T^R$ on A
Initialization: Compute an MLE tree t_q for each quartet $q \subseteq A$

```
for $r = 1$ to $R$ do
    Randomly permute the order of the sequences in $A$
    Set $T_4$ equal to the precomputed MLE tree on $\{a_1, a_2, a_3, a_4\}$
    for $i = 4$ to $n$ do
        Set $c(e) = 0$ for all edges in $T_i$
        for each quartet $q \subseteq \{a_1, \ldots, a_{i+1}\}$ with $a_{i+1} \in q$ do
            Let $x, y \mid z, a_{i+1}$ be the MLE tree topology on $q$
            Let $p$ denote the path from $x$ to $y$ in $T_i$
            Increment $c(e)$ by 1 for each edge $e$ that is contained
            in $p$ or that is separated from $z$ by $p$
        Choose any edge $f$ for which $c(f)$ is minimal
        Obtain $T_{i+1}$ by attaching a new leaf with label $a_{i+1}$ to $f$
    Output $T^r := T_n$
end
```

5.71 Majority consensus

Running this algorithm produces a whole collection $T^1, \ldots, T^R$ of trees on A. To obtain a single output tree T, the quartet puzzling method computes the *majority consenus tree* for the collection of trees:

Let $\Sigma(T^i)$ denote the split encoding of T^i. The *majority-consensus splits set* for $T^1, \ldots, T^R$ is defined as the set of all splits that occur in at least half of all trees $T^1, \ldots, T^R$:

$$MC := MC(T^1, \ldots, T^R) := \left\{ S \text{ split on } A \;\mid\; \#\{i : S \in \Sigma(T^i)\} > \frac{r}{2} \right\}.$$

Lemma The majority-consensus splits set is compatible and the corresponding tree is called the *majority consenus tree.*

Proof: Assume there exist two splits $S_1, S_2 \in MC$ that are not compatible with each other. Because each split occurs in more than half of all trees, there must exist a tree in which both occur, a contradiction. □

6 Phylogenetic Networks

Real evolutionary data often contains a number of different and sometimes conflicting phylogenetic signals, and thus do not always clearly support a unique tree. To address this problem, Hans-Jürgen Bandelt and Andreas Dress developed the method of *split decomposition.*

For ideal data, this method gives rise to a tree, whereas less ideal data are represented by a tree-like network that may indicate evidence for different and conflicting phylogenies.

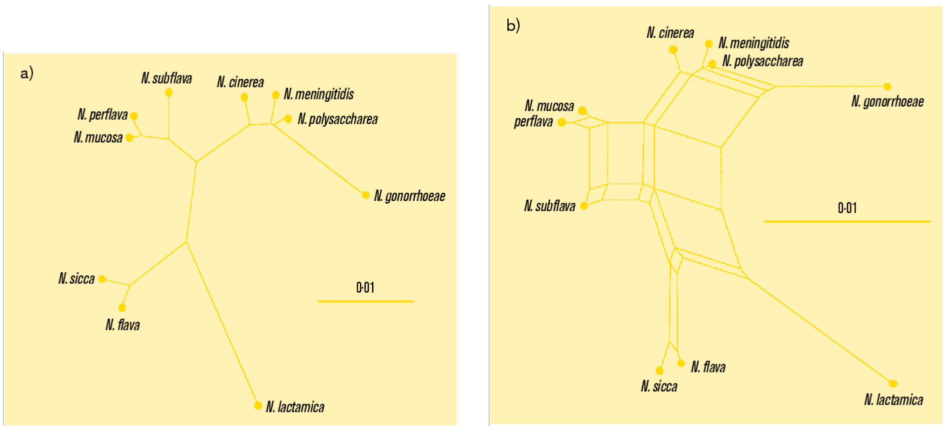
The following lectures are based on:

Hans-Jürgen Bandelt and Andreas W. M. Dress. *A canonical decomposition theory for metrics on a finite set*, Advances in Mathematics, 92(1):47-105 (1992)

Daniel H. Huson, *SplitsTree: analyzing and visualizing evolutionary data*, Bioinformatics, 14(10):68-73 (1998).

6.1 Trees vs networks

Here is (a) the unrooted neighbor-joining tree for 16S rRNA sequences (1355 bp) from ten species of *Neisseria* and (b) a splits graph computed from the same distance matrix:



(a) (b)

(Source: Eddie C. Holmes. Genomics, phylogenetics and epidemiology, Microbiology Today, 26:162-163 (1999).)

6.2 Representing distances using trees

Given a set X of taxa. For our purposes, a *phylogenetic tree* T is a tree such that:

- all leaves, and perhaps some of the internal nodes, too, are (multi-)labeled by elements of X, such that each taxon appears exactly once, and
- every edge e has a weight d_e associated with it.

Given a set of taxa X and a distance matrix $\{d_{ab}\}$ (i.e., a *dissimilarity function* or *pseudo metric*) describing "evolutionary distances" between the different taxa, obtained in some way, e.g. Hamming or Jukes-Cantor distances.

Any distance-based tree building method attempts to represent a given distance matrix d as well as possible using a phylogenetic tree T, i.e. for any two taxa $a, b \in X$ we approximate $d_{ab} \approx \sum_{e \in P} d_e$, where P is the unique path of edges in T that connects the nodes with labels a and b.

6.3 Main goal

Given a distance matrix d, a tree building method such as neighbor-joining will compute a phylogenetic tree T for d, no matter how "untree-like" the distance matrix d may be.

(Recall that the four-point condition determines whether a given distance matrix d is *additive* or not, i.e. whether it has an exact representation as by a phylogenetic tree, or not.)

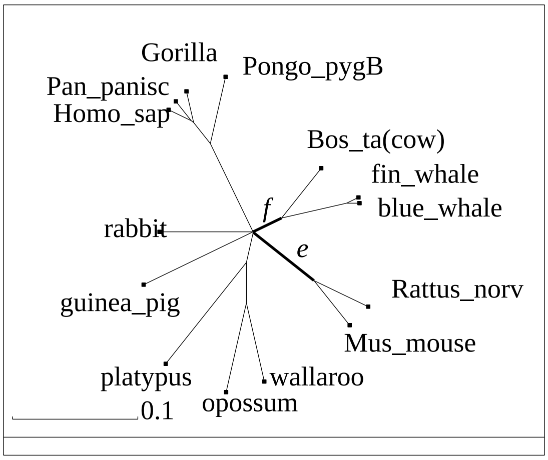
Our goal is to use more general graphs to represent distances, so-called *splits graphs*. As we will see, the graph will be a tree, whenever the given distances are tree-like (i.e., additive, or close to additive).

To reach this goal, we proceed indirectly by discussing sets of splits and introducing the notation of *weak-compatibility*.

Just as a set of compatible splits can be represented by a phylogenetic tree, we will see that a weakly-compatible set of splits can be represented by a splits graph.

6.4 Tree and splits

Here is an example of a phylogenetic tree T:



Each edge in T defines a split of the set of taxa X. For example, the edge labeled e separates rat and mouse from all other taxa, and the edge f separates cow, fin whale and blue whale from all others.

6.5 Tree distance and Σ-distance

Given a phylogenetic tree T with edge weights. We define the *tree distance* between two taxa a and b as

$$d_T(a, b) := \sum_{e \in P} d_e,$$

where P denotes the set of edges along the unique simple path from the node labeled a to the node labeled b.

We set $d_S := d_e$, if S is the split corresponding to e. We define the Σ-*distance* between two taxa a and b as

$$d_\Sigma(a,b) := \sum_{S \in \Sigma(a,b)} d_S,$$

where $\Sigma(a,b)$ is the set of all splits in Σ that separate a and b.

These definitions imply

$$d_T(a,b) = d_\Sigma(a,b)$$

for all taxa $a, b \in X$.

6.6 Weak compatibility

Compatibility is a requirement defined on any *two* splits. A relaxed concept is that of *weak compatibility*, which is a condition placed on any *three* splits.

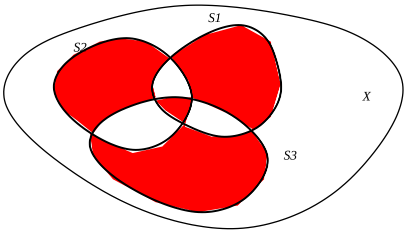
A triplet $S_1 = \{A_1, \bar{A}_1\}$, $S_2 = \{A_2, \bar{A}_2\}$ and $S_3 = \{A_3, \bar{A}_3\}$ is called *weakly compatible*, if at least one of the four intersections of

$$A_1 \cap A_2 \cap A_3,\ A_1 \cap \bar{A}_2 \cap \bar{A}_3,\ \bar{A}_1 \cap A_2 \cap \bar{A}_3,\ \text{or } \bar{A}_1 \cap \bar{A}_2 \cap A_3,$$

and of

$$\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3,\ \bar{A}_1 \cap A_2 \cap A_3,\ A_1 \cap \bar{A}_2 \cap A_3,\ \text{or } A_1 \cap A_2 \cap \bar{A}_3,$$

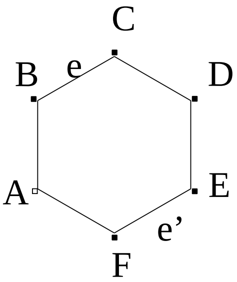
is empty. This means that at least one shaded and one unshaded region of the following diagram must be empty:



Note that if any pair of the three splits is compatible, then all three are weakly-compatible:

E.g., if for $S_1 = \{A_1, \bar{A}_1\}$ and $S_2 = \{A_2, \bar{A}_2\}$ we have $A_1 \cap A_2 = \emptyset$, then $A_1 \cap A_2 \cap A_3 = \emptyset$ and $A_1 \cap A_2 \cap \bar{A}_3 = \emptyset$.

On the other hand, it is possible that *every* pair of the three weakly-compatible splits is incompatible:



Here, a split of $X = \{A, B, C, D, E, F\}$ if given by each pair of parallel edges, e.g. edges e and e' define the split $\{\{A, B, F\}, \{C, D, E\}\}$.

6.7 Weak compatibility and splits graphs

As discussed above, any given set of splits Σ can be represented by a tree $T(\Sigma)$, if and only if Σ is compatible.

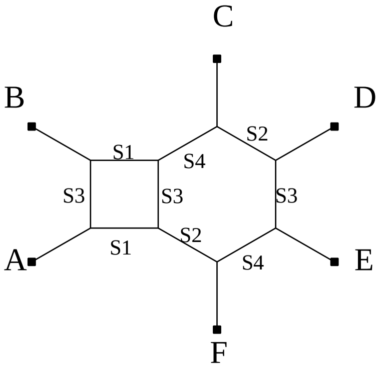
A weakly compatible split system $\mathcal{S}$ can be represented by a *splits graph* $G(\Sigma)$ that has the following properties:

- all leaves (and, additionally, some internal nodes, perhaps) are multi-labeled by taxa so that each taxon appears exactly once,
- edges are labeled by splits such that each split appears at least once,
- deleting all edges labeled by any given split $S = \{A, \bar{A}\}$ produces precisely two components, one containing all nodes with labels in A and the other containing all nodes with labels in $\bar{A}$, and
- the graph is minimal with these properties.

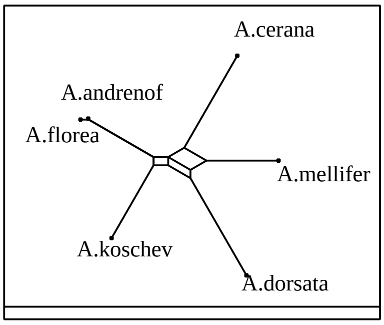
Given the following splits:

$$S_1 = \{\{A, B\}, \{C, D, E, F\}\}, \quad S_2 = \{\{A, B, C\}, \{D, E, F\}\},$$
$$S_3 = \{\{A, F, E\}, \{B, C, D\}\}, \quad S_4 = \{\{A, B, F\}, \{C, D, E\}\},$$
and all *trivial* splits A vs $B - F$, etc.

They can be represented as follows:



Here is another example of a splits graph:



This graph is based on DNA obtained from bees. It indicates that is there is some evidence that groups A.cerana and A.meillifer together, and conflicting evidence that groups A.mellifer with A.dorsata, for example.

6.8 Splits graphs and distances

Given a set of taxa X, a set of weakly compatible splits Σ of X and a value $d_S \geq 0$ for each split S.

As above, we define the Σ-distance between taxa a and b simply as $d_\Sigma(a,b) := \sum_{S \in \Sigma(a,b)} d_S$, where $\Sigma(a,b)$ is the set of all splits that separate a and b.

Assume we are given a corresponding splits graph G. In G, each split S is represented by a band of parallel edges and each such edge e has weight $d_e = d_S$.

Consider any two taxa $a, b \in X$. We define

$$d_G(a,b) := \min\{\sum_{e \in P} d_e \mid P \text{ is a simple path from } a \text{ to } b\}.$$

Lemma We have $d_\Sigma(a,b) = d_G(a,b)$ for all $a, b \in X$.

(Proof: need to show that a minimum path from a to b uses precisely one edge for every split that separates a and b.)

6.9 Two main questions

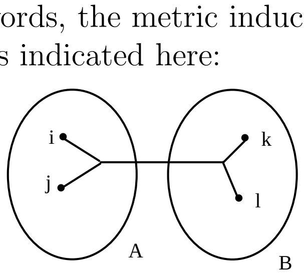
- First, given a set of taxa X and a distance matrix d. How do we compute a set of weakly compatible system of splits Σ and values d_S, such that $\sum_{S \in \Sigma(a,b)} d_S$ is a useful approximation of d_{ab}?
- Second, given a weakly compatible set of splits, how do we compute the corresponding splits graph?

6.10 Distance matrices and d-splits

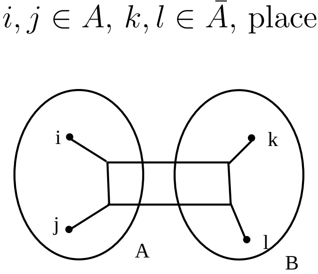
Given a distance matrix d on X. We call a split $S = \{A, \bar{A}\}$ a *d-split*, if for all $i, j \in A$ and $k, l \in \bar{A}$ we have

$$d_{ij} + d_{kl} < \max(d_{ik} + d_{jl}, d_{il} + d_{jk}).$$

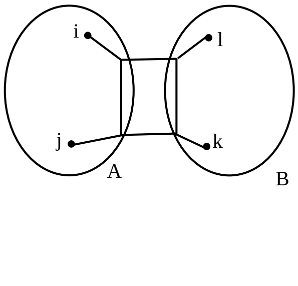
In other words, the metric induced by d on any four taxa $i, j \in A$, $k, l \in \bar{A}$, places i, j and k, l together as indicated here:



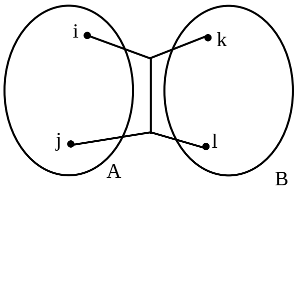
or



or



but NOT:

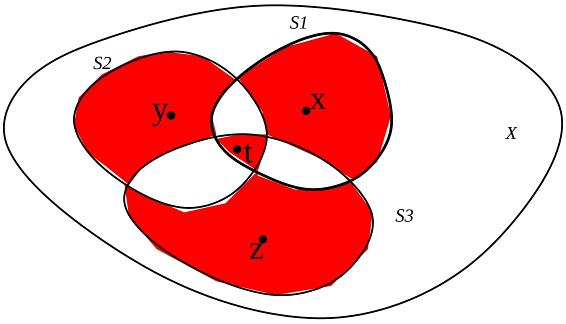


.

6.11 d-splits are weakly compatible

Lemma (Bandelt & Dress 1992) Let d be a distance matrix on X. Then the set of all d-splits is weakly compatible.

Proof Consider three d-splits $S_1 = \{A_1, \bar{A}_1\}$, $S_2 = \{A_2, \bar{A}_2\}$ and $S_3 = \{A_3, \bar{A}_3\}$ and *assume that they are not weakly-compatible*. Then there exist four taxa x, y, z, t contained in $A_1 \cap \bar{A}_2 \cap \bar{A}_3$, $\bar{A}_1 \cap A_2 \cap \bar{A}_3$, $\bar{A}_1 \cap \bar{A}_2 \cap A_3$ and $A_1 \cap A_2 \cap A_3$, respectively:



The definition of a d split implies the following three inequalities:

For S_1 : $d_{xt} + d_{yz} < \max(d_{xy} + d_{tz}, d_{xz} + d_{ty})$,
for S_2: $d_{yt} + d_{xz} < \max(d_{yx} + d_{tz}, d_{yz} + d_{tx})$, and
for S_3: $d_{zt} + d_{xy} < \max(d_{zx} + d_{ty}, d_{zy} + d_{tx})$.

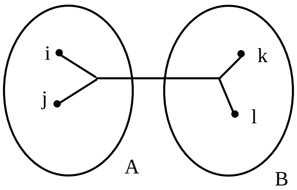
Note that these three inequalities cannot be fulfilled simutaneously, contradicting our assumptions and thus the three splits must be weakly-compatible. □.

6.12 The isolation index of a split

We give any d-split $S = \{A, \bar{A}\}$ a positive weight, namely the quantity

$$\alpha_{A,B} := \alpha^d_{A,B} := \frac{1}{2} \min_{i,j \in A, k,l \in B} \{\max(d_{ik} + d_{jl}, d_{il} + d_{jk}) - (d_{ij} + d_{kl})\},$$

called the *isolation index* of S.



We can easily modify this definition to apply to *any* split $S = \{A, \bar{A}\}$, whether d-split or not:

$$\alpha_{A,B} := \alpha^d_{A,B} := \frac{1}{2} \min_{i,j \in A, k,l \in B} \{\max(d_{ik} + d_{jl}, d_{il} + d_{jk}, d_{ij} + d_{kl}) - (d_{ij} + d_{kl}))\},$$

thus obtaining a value ≥ 0 that equals the previously defined isolation index, if S is a d-split, and 0, if not.

6.13 The split decomposition

For any split $S = \{A, \bar{A}\}$ of X, the *split metric* δ_S is given by

$$\delta_S(i,j) := \begin{cases} 0, & \text{if } i,j \in A \text{ or } i,j \in \bar{A}, \\ 1, & \text{else.} \end{cases}$$

Theorem (Bandelt & Dress) Any given distance matrix d on X possesses the following unique decomposition:

$$d_{ij} = \left(\sum_S \alpha_S \delta_S(i,j)\right) + d^0_{ij},$$

for all $i, j \in X$. Here, the sum runs over all possible splits S and the map $d^0 : X \times X \longrightarrow \mathbb{R}^{\geq 0}$ is a (pseudo-)metric that does not admit any further splits with positive isolation index, i.e. there exist no d^0-splits.

Hence, we have $\sum_S \alpha_S \delta_S(i,j) \leq d_{ij}$ for any pair of taxa $i, j \in X$ and the Σ-distance α_S approximates d_{ij} from below.

One can prove that the number of d-splits is $\leq \binom{|X|}{2}$.

6.14 Computing the set of d-splits

Given a distance matrix d on X. The set of all d-splits can be computed iteratively in $O(n^6)$ steps:

Algorithm
Input: Distance matrix d, taxon set $X = \{x_1, x_2, \ldots, x_n\}$.
Output: Set $\Sigma = \Sigma_n$ of all d-splits

```
Initialization: $\Sigma_0 := \emptyset$

for each $k = 1, 2, \ldots, n$ do:
    Set $\Sigma_k := \emptyset$
    for each split $S = \{A, \bar{A}\} \in \Sigma_{k-1}$:
        if $\{A \cup \{x_k\}, \bar{A}\}$ has positive isolation index then
            Add $\{A \cup \{x_k\}, \bar{A}\}$ to $\Sigma_k$
        if $\{A, \bar{A} \cup \{x_k\}\}$ has positive isolation index then
            Add $\{A, \bar{A} \cup \{x_k\}\}$ to $\Sigma_k$
    If $\{\{x_1, x_2, \ldots, x_{k-1}\}, \{x_k\}\}$ has positive isolation index then
        Add $\{\{x_1, x_2, \ldots, x_{k-1}\}, \{x_k\}\}$ to $\Sigma_k$.
end
```

Lemma This algorithm computes all d-splits.

Proof First note that in the k-iteration of the algorithm the new *partial* trivial split $\{\{x_1, \ldots, x_{k-1}\}, \{x_k\}\}$ is evaluated and then added to the current set of splits, if $\alpha_{\{\{x_1,\ldots,x_{k-1}\},\{x_k\}\}} > 0$. Additionally, the algorithm attempts to extend all existing partial splits by adding x_k to the one side, or the other side, of them. By definition of the isolation index as the *minimum* of certain sums involving quartets of taxa, adding a taxon to either side of a partial split can only decrease the isolation index. Hence, any split of X is obtainable as a partial trivial split for some k, followed by successive addition of the remaining taxa to the split. □

6.15 Computing the splits graph

Given a compatible system of splits, it is easy to construct the corresponding tree and to compute coordinates for the tree.

The problem of computing a splits graph for a given set of weakly compatible splits is more difficult. In practice, one distinguishes between *circular* split systems, which correspond to planar split graphs, and non-circular ones. A nice algorithm exists for circular split systems that produces a planar graph.

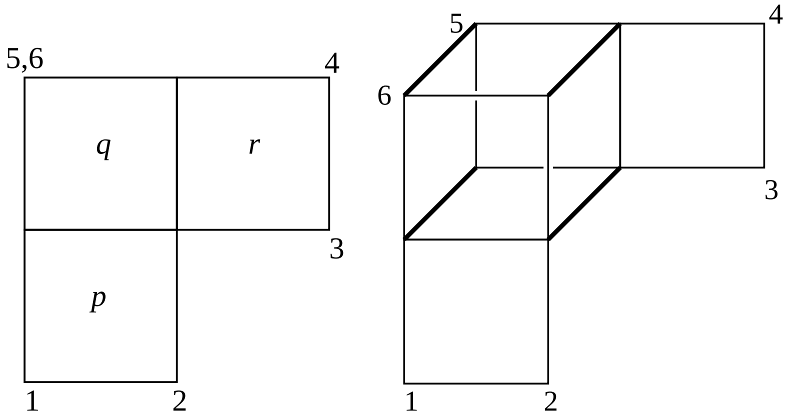
Here we discuss the *convex hull* approach that applies to any set of weakly compatible splits, whether circular or not. This method is easy to describe. Its main draw-back is that it usually produces redundant nodes and edges and so the resulting graph is not always *minimal* in the sense postulated above.

6.16 Convex-hull construction method

Given a splits graph G. For a given set of taxa $A \subset X$, let G_A denote the set of all nodes labeled by taxa in A. The *convex hull* $\overline{G_A}$ of G_A is obtained by first setting $\overline{G_A} = G_A$ and then repeatedly adding any node v to $\overline{G_A}$, if there exist two nodes a, b already in $\overline{G_A}$ such that $d_G(a, v) + d_G(v, b) \leq d_G(a, b)$.

Given a weakly compatible set of splits $\Sigma = \{S_1, S_2, \ldots, S_k\}$. The convex-hull construction method constructs a graph by adding one split at time. For each split, the convex hull for both sides of the split is computed. The intersection of the two is duplicated, one copy is connected to one side of the graph corresponding to one side of the split, the other to the other and then the two duplicated subgraphs are connected by a set of new edges that represent the new split.

Given the graph shown in (a). How do we add the split $S = \{\{1, 2, 6\}, \{3, 4, 5\}\}$ to it? The convex hull of $A = \{1, 2, 6\}$ consists of all nodes in squares p and q, and the convex hull of $\bar{A}$ of all nodes in q and r. The intersection H of both is q.



The graph (b) is obtained by duplicating H as described in the algorithm.

Assume that G is the graph constructed for splits $S_1 \ldots, S_i$. To add the next split $S_{i+1} = \{A, \bar{A}\}$:

- Determine the two convex hulls $\overline{G_A}$ and $\overline{G_{\bar{A}}}$.
- Let $H := \overline{G_A} \cap \overline{G_{\bar{A}}}$ denote their intersection.
- For each node $v \in H$, produce two new nodes v^+ and v^- and connect them by an edge labeled S_{i+1}.
- If $v \in H$ is labeled by a taxon $x \in A$, or $x \in \bar{A}$, then attach this label to node v^+, or v^-, respectively.
- Connect any two nodes v^+ and w^+, and v^- and w^-, respectively, by an edge, if v and w are connected by an edge in G.

- If $v \in H$ is connected to some node $w \in \overline{G_A} \setminus \overline{G_{\bar{A}}}$, then connect v^+ and w by an edge.
- If $v \in H$ is connected to some node $w \in \overline{G_{\bar{A}}} \setminus \overline{G_A}$, then connect v^- and w by an edge.
- Delete H.

6.17 Computing the splits graph

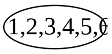
Given the following set Σ of splits:

$$\begin{array}{lll} S_1 = \{ & \{1,5,6\} & ,\{2,3,4\} \quad \} \\ S_2 = \{ & \{1,2,3\} & ,\{4,5,6\} \quad \} \\ S_3 = \{ & \{1,2,5,6\} & ,\{3,4\} \qquad \} \\ S_4 = \{ & \{1,2\} & ,\{3,4,5,6\}\} \\ S_5 = \{ & \{1,6\} & ,\{3,4,5,6\}\} \end{array}$$

We will demonstrate how to generate G_Σ.

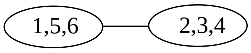
Initially, start with a single node labeled by all of $X = \{1,2,3,4,5,6\}$:

G_1



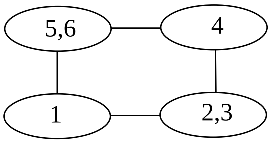
Then add the first split S_1. Note that H consists of the single node present in G_1:

G_2



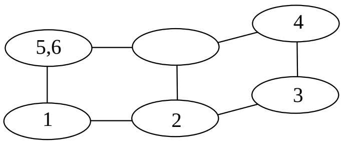
Add the second split $S_2 = \{\{1,2,3\},\{4,5,6\}\}$. Note that H consists of both nodes in G_2:

G_3



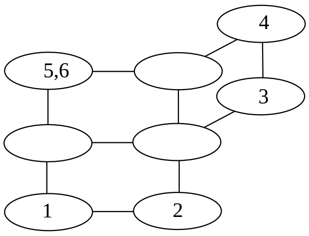
Add the third split $S_3 = \{\{1,2,5,6\},\{3,4\}\}$. Note that H consists of the two nodes labeled $2,3$ and 4 in G_3:

G_4

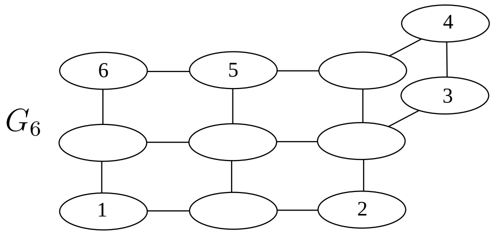


Add the fourth split $S_4 = \{\{1,2\},\{3,4,5,6\}\}$. Note that H consists of the two nodes labeled 1 and 2 in G_4:

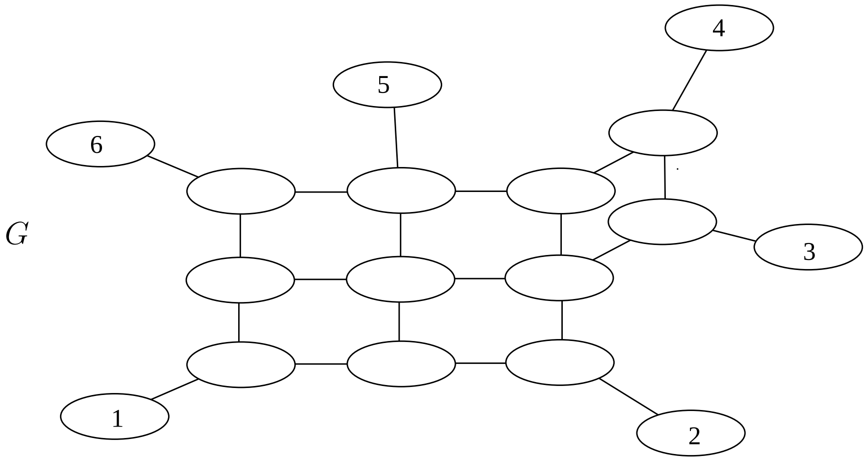
G_5



Add the fifth split $S_5 = \{\{1,6\},\{2,3,4,5\}\}$. Note that H consists of the two nodes labeled 1 and $5,6$, plus the node lying between these two in G_5:

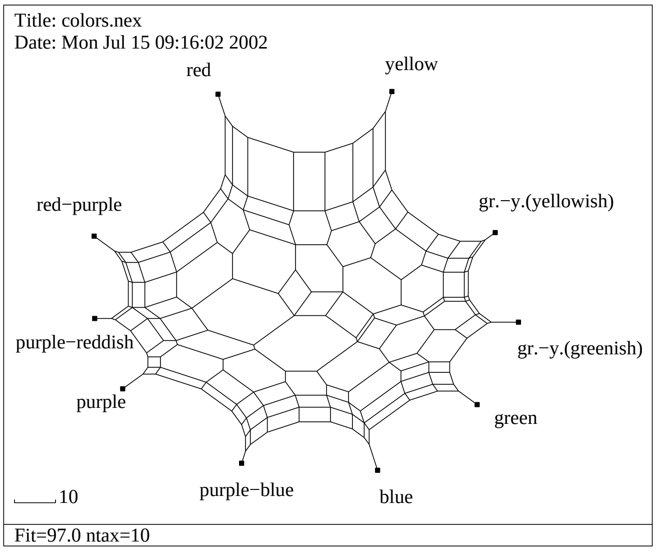


Finally, add all trivial splits to G_6 to obtain the final graph G:



6.18 Example of splits graph

The distance matrix for the following example was produced in a psychology experiment in which people where asked to estimate the distance between different colors:

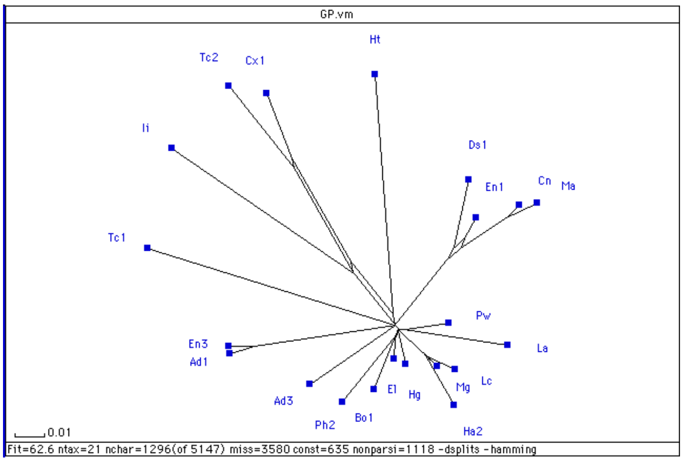


6.19 Manuscript analysis

The Canterbury Tales text was left unfinished in 1400 and survives in some 88 versions dating before 1500. Many questions remain open, including *which* version is closest to the original one written by Geoffrey Chaucer?

To address such questions, the spelling of a part of the manuscripts was transcribed and then compared, giving rise to “sequence data” that can be analyzed using methods such as split decomposition.

The main use of such programs in this context is seen as a way to suggest possible relationships that can then be further investigated, confirmed or rejected, using conventional tools.



GP.vm

Fit=62.6 ntax=21 nchar=1296(of 5147) miss=3580 const=635 nonparsi=1118 -dsplits -hamming

For more details of the application of phylogenetic methods to the Canterbury Tales, see `http://www.cta.dmu.ac.uk/projects/ctp/desc2.html`.

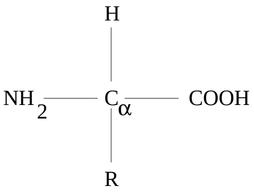
7 Protein secondary structure

Sources for this chapter, which are all recommended reading:

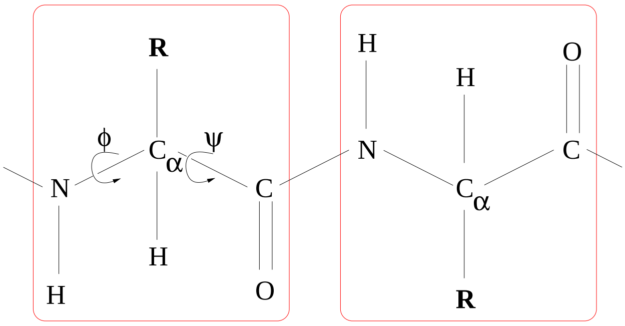
- V.V. Solovyev and I.N. Shindyalov. Properties and prediction of protein secondary structure. In *Current Topics in Computational Molecular Biology*, T. Jiang, Y. Xu and M.Q. Zhang (editors), MIT press, chapter 15, pages 366-401, 2002.
- D.W. Mount. *Bioinformatics: Sequences and Genome analysis*, Cold Spring Harbor Press, Chapter 9: Protein classification and structure prediction. pages 381-478, 2001.

7.1 Proteins

A *protein* is a chain of amino acids joined by peptide bonds. It is produced by a ribosome that moves along an mRNA and adds amino acids according to the codons that it observes.

The structure of an amino acid: 

Here are two amino acids within a polypeptide chain:



The R group is different for each of the twenty amino acids. The R groups of a protein are called its *side chains*. Neighboring amino acids are joined by a peptide bond between the C=O and NH groups. A chain of repeated N-C_α-C's make up the *backbone* of the protein.

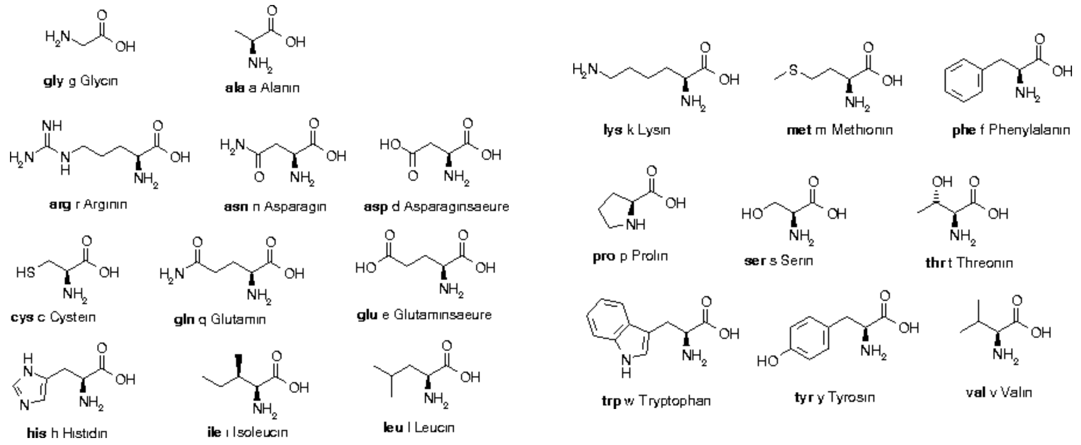
In such a polypeptide chain, each amino acid has two rotational degrees of freedom:

- the rotational angle ϕ of the bond between N and C_α, and
- the rotational angle ψ of the bond between C_α and C.

Both bonds are free to rotate, subject to spatial constraints posed by adjacent R groups. The third angle Ω of the peptide bond between the C=O and NH groups is nearly always 180°.

A protein starts with a free NH group (the *N-terminus*) and ends with a free COOH group (the *C-terminus*).

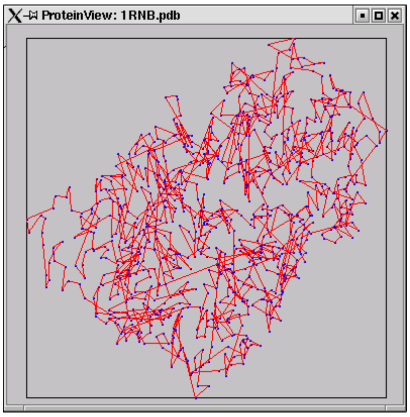
The twenty common amino acids:



(Figure from: `http://www.chemie.fu-berlin.de/chemistry/bio/amino-acids_en.html`)

7.2 Visualization of proteins

The aim of assignment sheet 12 is that you familiarize yourselves with PDB files and the data associated with three-dimensional protein structures. You are requested to write a simple viewing program that displays the three-dimension structure like this:



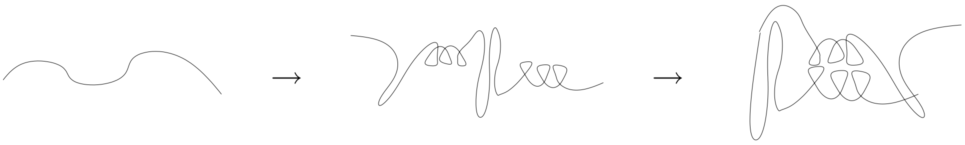
(Note that of the three given pdb files, only `2BOP.pdb` contains secondary structure information (`HELIX`, `SHEET` and `TURN` lines).)

7.3 Hierarchy of protein structure

K.U. Linderstrom-Lang (Linderstrom-Lang & Schnellman 1959) proposed to distinguish four levels or protein structure:

- The *primary structure* is the chemical structure of the polypeptide chain(s) in a given protein, i.e. its sequence of amino acid residues that are linked by peptide bonds.
- The *secondary structure* is folding of the molecule that arises by linking the C=O and NH groups of the backbone together by means of hydrogen bonds.
- The *tertiary structure* is the three dimension structure of the molecule consisting of secondary structures linked by "looser segments" of the polypeptide chain stabilized (primarily) by side-chain interactions.
- The *quaternary structure* is the aggregation of separate polypeptide chains into the functional protein.

Pathway for folding a linear chain of amino acids into a three-dimensional protein structure:



linear polypeptide chain — secondary structures — secondary structures folded into 3D structure

The tertiary structure of proteins is of great interest, as the shape of a protein determines much, if not all, of its function.

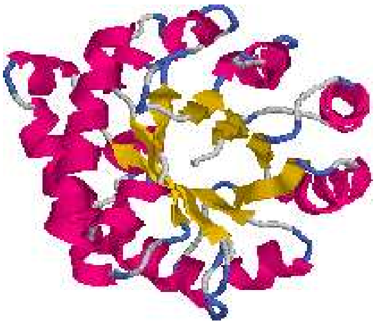
At present, the experimental determination of protein structure via x-ray crystallography is difficult and time-consuming. Hence, we would like to be able to determine the structure of a protein from its sequence. Determining the secondary structure is an important first step.

7.4 The "Holy Grail" of bioinformatics

The *holy grail* of bioinformatics: develop and algorithm that can reliably predict the structure (and thus function) of a protein from its amino-acid sequence!

```
...
IIFIATTNLLGLLPHSFTPTTQLSMNLAMAIPLWA
GAVILAHFLPQGTPTPLIPMLVIIETISLLIQPAL
AVRLTANITAGHLLMGSATLAMTLIIFTILILLTI
LEIAVALIQAYVFTLLVSLYLHDNTPQLNTTVWPT
MITPMLLTLFLITQLKMLPWEPKWADRWLFSTNHK
DIGTLYLLFGAWAGVLGTALSLLIRAELGQPGNLL
GNDHIYNVIVTAHAFVMIFFMVMPIMIGGFGNWLV
PLMIGAPDMAFPRMNNMSFWLLPPSLLLLLASAMV
...
```

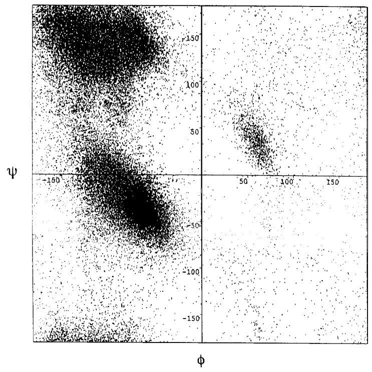
$\xrightarrow{?}$



This is not easy...

7.5 Secondary structure of proteins

Regular features of the main chain of a protein give rise to a secondary structure. The two most common regular structures are called α *helix* and β *sheet* (L. Pauling 1951), corresponding to specific choices of the ϕ and ψ angles along the chain. This can be seen in a *Ramachandran plot* of observed pairs of angles in a collection of known protein structures:



The pairs near $\phi = -60°$ and $\psi = -60°$ correspond to α helices. The pairs near $(-90°, 120°)$

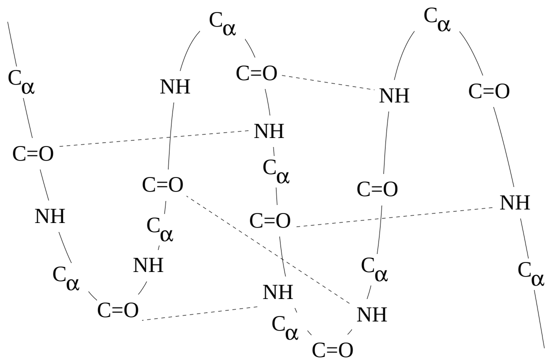
correspond to β strands.

7.6 α helices and β sheets

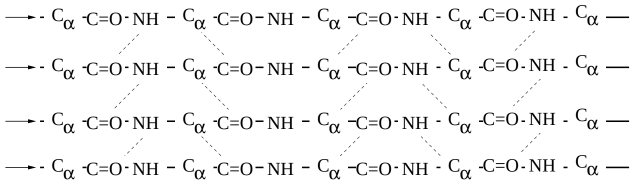
Helices arise when hydrogen bonds occur between (the C=O group of) the amino acid at position i and (the NH group of) the amino acid at position $i+k$ (with $k = 3, 4$ or 5), for a run of consecutive values of i.

Most often, $k = 4$ or 5 and the resulting structure is called an *α helix*, whereas $k = 3$ gives rise to a 3_{10} *helix*.

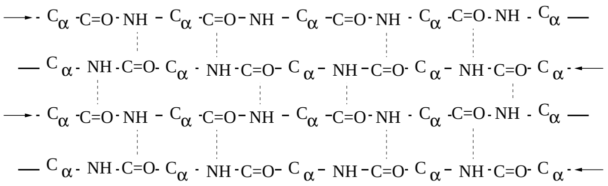
Here is the bonding pattern of an α helix:

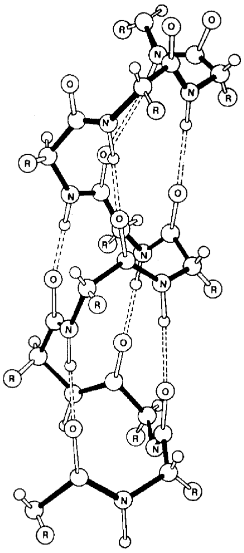


So-called *β sheets* are formed by H bonds between a run of 5-10 consecutive amino acids in one portion of the chain and a another 5-10 consecutive amino acids further down the chain. There are two possible configurations. In a parallel β sheet, all chains run in the same direction:

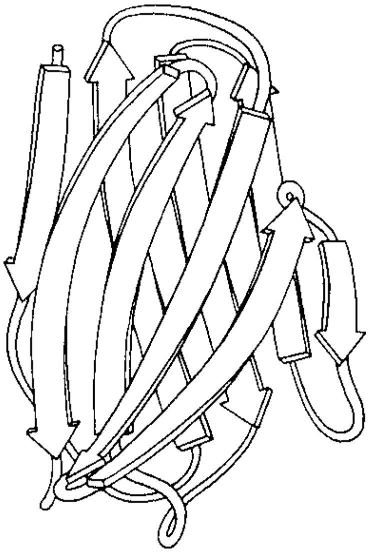

Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα
Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα
Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα
Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα

In an anti-parallel sheet, chains run in alternating directions:


Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα
Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα
Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα -C=O-NH - Cα
Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα



α helix
3.6 residues per turn



anti-parallel β sheet
(V2 domain of an immunoglobulin)

(Images from: `http://www.tau.ac.il/~becker/course/secondary.html`)

7.7 Loops and coils

Loops or *turns* are regions of a protein chain that lie between α helices and β sheets. The lengths and three-dimensional structure of loops can vary. Hairpin loops that constitute a complete turn joining two anti-parallel β-strands may be as short as two amino acids. They lie on the surface of the structure.

Because they are not spatially constrained and do not affect the arrangement of the secondary structure elements, more substitutions, insertions and deletions can occur than inside the *core* of the protein that is made up of the α helices and β sheets.

A region of secondary structure that is not a helix, a sheet, or recognizable turn is called a *coil.*

7.8 Secondary structure of proteins

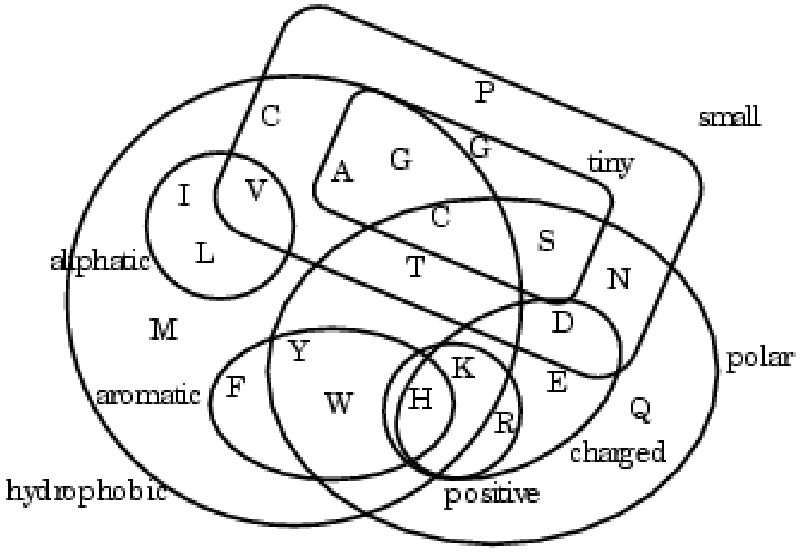
About 35% of residues in proteins lie in α helices. The amino acids Ala, Glu, Leu and Met are often found in α helices, whereas Pro, Gly, Ser, Thr and Val occur rarely in them.

Most α helices are immersed in the protein interior from one side and form a exterior protein surface from the other side. Non polar residues are usually located on one side of an α helix (forming a hydrophobic cluster) and polar and charged residues are on the other side.

About 36% of all residues of proteins are contained in β sheets. The C_α atoms and side chains alternate above and below the sheet in a pleated structure. If one side of a β sheet is exposed to solvent, then this results in a characteristic interchange of hydrophobic and polar amino acids. Parallel β sheets are usually found in the interior of a protein, often protected from solvents by α helices on both sides. They usually incorperate at least 5 strands.

Amino acids Val, Ile, Try and Thr prefer β sheets, whereas Glu, Gln, Lys, Asp, Pro and Cys are seldomly found in them.

7.9 Classification of amino acids

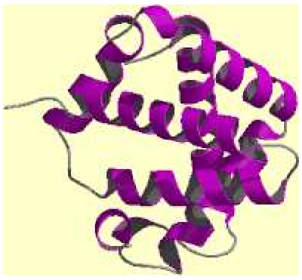


7.10 Classification of protein structure

D.W. Mount describes six principal classes of protein structures, four from Levitt and Chothia (1976), and two additional ones taken from the SCOP database (Murzin et al., 1995):

(1) A member of class α consists of a bundle of α helices connected by loops on the surface of the proteins.

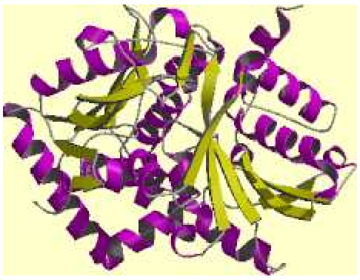
(2) A member of class β consists of β sheets, usually two sheets in close contact forming a sandwich. Examples are enzymes, transport proteins and antibodies.

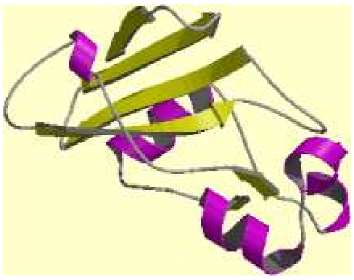

(1) Hemoglobin (3hhb)

(2) T-cell receptor CD8 (1cd8)

(3) A member of class α/β consists mainly of β sheets with intervening α helices. This class contains many metabolic enzymes.

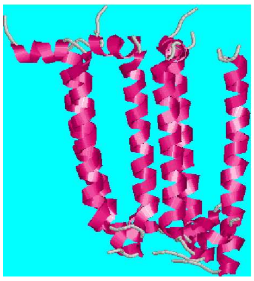
(4) A member of class $\alpha + \beta$ consists of segregated α helices and β sheets.


(3) Tryptophan synthase β subunit (2tsy)


(4) G-specific endonuclease (1rnb)

(5) This class consists of all multi-domain (α and β) proteins with domains from more than one of the above four classes.

(6) Membrane and cell-surface proteins and peptides excluding proteins of the immune system.



(6) Integral membrane light-harvesting complex (1kzu)

Figures from `http://www.biochem.ucl.ac.uk/bsm/cath_new/`.

The three letter codes are PDB codes.

7.11 Detecting the secondary structure of a known 3D structure

Given the positions of the main chain atoms of a protein. For each amino acid in the protein, we want to determine whether it belongs to an α helix or a β sheet, or neither. The *DSSP (definition of secondary structure of proteins)* algorithm (Kabsch, Sander, 1983) proceeds as follows:

First determine which C=O and NH groups in the main chain are joined by hydrogen bonds. This decision is based on an electrostatic model using the following energy calculation:

$$E = q_1 q_2 \left(\frac{1}{r(ON)} + \frac{1}{r(CH)} - \frac{1}{r(OH)} - \frac{1}{r(CN)} \right) \cdot f \cdot N_A,$$

with $q_1 = 0.42e$ and $q_2 = 0.20e$, where $e = 4.80325 \times 10^{-10} gr^{\frac{1}{2}} cm^{\frac{3}{2}} s^{-1}$ is the unit electron charge, $r(AB)$ is the inter-atomic distance from atom A in the first amino acid and atom B in the second in Angstroms, $f = 332$ is a constant called the *dimensionality factor*, and E is the energy in kcal/mol.

7.12 Detecting α helices

Recall that Avogadro's constant $N_A = 6.02214199 \times 10^{23} \text{mol}^{-1}$ equals the number of molecules in one mole of substance.

For any pair of amino acids, for which $E < -0.5$ kcal/mol, an H-bond is assigned.

Any H-bond detected in this way is called

- an *n-turn*, if it connects the O=C group of amino acid i to the NH group of amino acid $i + k$, where $k = 3, 4$ or 5, and
- a *bridge*, if it connects residues that are not near to each other.

An α helix is identified as a consecutive sequence of (at least two) n-turns, Any two helices that are offset by two or three residues are concatenated into a single helix.

7.13 Detecting β sheets

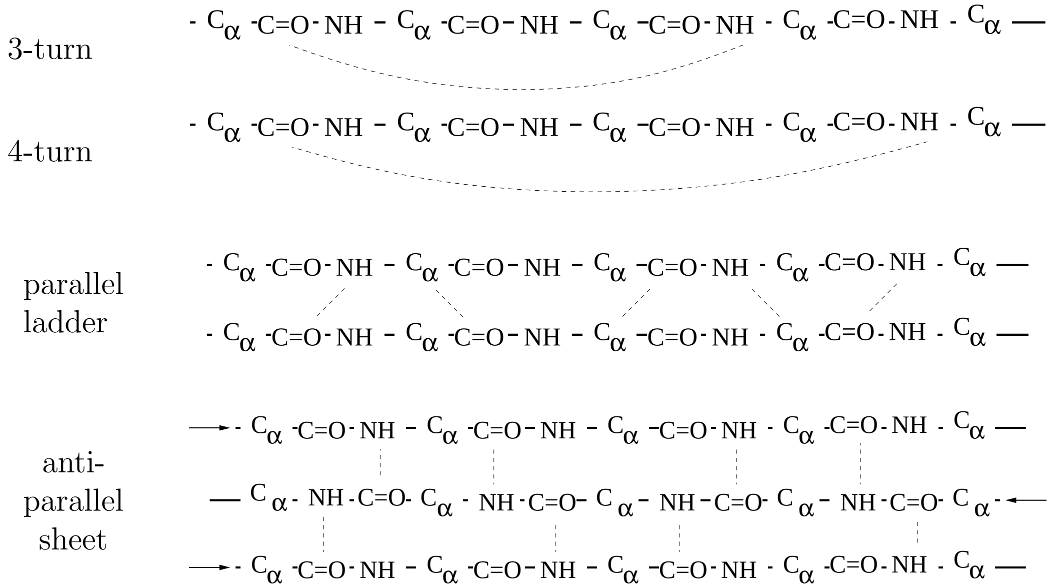
A β-sheet corresponds to a sequence of bridges between consecutive residues in two different regions of the chain. More precisely, we need to introduce two types of patterns:

- a *ladder* is a set of one or more consecutive bridges of the same type, and
- a *sheet* is one or more ladders connected by shared residues.

To allow for irregularities, *β-bulges* are introduced, in which two perfect ladders or bridges can be connected through a gap of one residue in on one side and four on the other.

7.14 Types of H bond patterns

In summary, here are some of the patterns that are used to identify secondary structure elements:


3-turn
4-turn
parallel ladder
anti-parallel sheet
- Cα -C=O- NH - Cα -C=O- NH - Cα -C=O- NH - Cα -C=O- NH - Cα —
— Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα - NH-C=O- Cα -

7.15 DSSP vs STRIDE

The STRIDE algorithm (Frishman and Argos, 1995) extends the DSSP algorithm by adding information on backbone torsion angles. The H-bonding criterion uses a refined energy function and experimental data on H-bond geometry is also taken into account.

In an experimental study of 226 proteins, STRIDE assigned 58% of the proteins closer to the experimental assignment compared to DSSP, whereas DSSP assigns only 31% percent closer than STRIDE. For 11% of the proteins, both computational assignments are in the same proximity of the experimental ones.

7.16 Secondary structure prediction from sequence

Above we discussed how to determine the secondary structure from the coordinates of a known tertiary structure. Now we discuss the more difficult problem of predicting the secondary structure of a protein from its sequence of amino acids. This problem has been studied for more than 30 years and still is an area of active research.

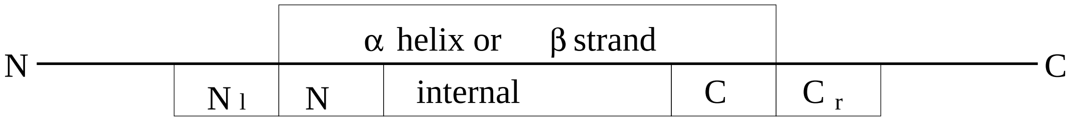
Many different types of approaches have been considered, namely ones: (1) based on stereochemistry rules, (2) using a combination of such rules and statistics, (3) that apply physical models of secondary structure formation, (4) based on statistical information and information theory, (5) that analyze sequence patterns, *(6) based on discriminant analysis*, *(7) using a neural net approach*, (8) analyzing evolutionary conservation in multiple alignments, that apply a nearest neighbor analysis, and (10) based on consensus prediction.

7.17 SSP and discriminant-analysis

The definition of secondary structure is a somewhat imprecise and different approaches will determine slightly different structures. For the purpose of modeling tertiary structures, it is less important to assign each individual residue to the correct type of secondary structure than it is to get the location of entire α helices and β strands correct.

The *secondary structure prediction program (SSP)* developed by Solovyev and Salamov (1991, 1994) is aimed at solving this problem.

The SSP algorithm is based on the assumption that secondary structures can be identified by statistical properties of five regions associated with an α-helix or β-strand, namely the N$_l$ region, N-terminal, internal, C-terminal and C$_r$ regions, respectively, as indicated here:



This subdivision into five regions was suggested by experimental data. Additionally, it makes sense to distinguish between short and long structures. I.e., the goal is to predict short and long α helices and β strands.

The SSP algorithm uses a linear combination of three discriminant functions (LDF) to determine whether any given segment of a protein sequence has a preference for being a short or long α helix, or not, and whether it has a preference for being a short or long β strand, or not. If two α helix and β strand predictions overlap, then the one with the larger LDF scores is chosen.

7.18 The singleton characteristic

The *singleton characteristic* is an average of single-residue preferences. Using a database of known protein structures, for every amino acid a the *preference* of being in a specific segment of type k (e.g., the N$_l$ segment of a short α helix or the internal segment of a long β strand) is calculated as

$$S^k(a) = \frac{P^k(a)}{P(a)},$$

where $P(a)$ and $P^k(a)$ are the proportions of amino acids of type a that are contained in the whole database and in segments of type k, respectively (see P.Y. Chou and G.D. Fasman 1978).

Given an amino acid sequence $x = (x_1, \ldots, x_L)$.

Choose start and end positions p and q in the sequence, and a structure type k (e.g, short

α-helix). The singleton characteristic $\mathcal{S}^k(p,q)$ is defined as:

$$\mathcal{S}^k(p,q) = \frac{1}{L}\left(\sum_{i=p-m}^{p-1} S_i^{N_l} + \sum_{i=p}^{p+m-1} S_i^{N} + \sum_{i=p+m}^{q-m} S_i^{internal} \right.$$
$$\left. + \sum_{i=q-m+1}^{q} S_i^{C} + \sum_{i=q+1}^{q+m} S_i^{C_r}\right),$$

where $S_i^k := S^k(x_i)$ denotes the preference of amino acid x_i to be contained in a segment of type k.

Here, m is a pre-chosen parameter that determines the size of the non-internal segments N$_l$, N, C and C$_r$. It is usually equals 4 or 3 for long α helices and β strands, and $\lfloor \frac{q-p+1}{2} \rfloor$ for short ones.

7.19 Using the singleton characteristic

How do we obtain predictions using the function $\mathcal{S}^k(p,q)$?

Training Present databases contain over 500 independent secondary structures. These can be used as a training set.

We compute the value of $\mathcal{S}^k(p,q)$ for each annotated secondary structure k associated with a subsequence $(x_p,\dots,x_q)$. This gives us a range of values corresponding to correct predictions.

Similarly, for each type of secondary structure k we compute $\mathcal{S}^k(p,q)$ for pairs of positions that do not correspond to annotated structures, obtaining a range of values for incorrect predictions.

Discrimination For each type of secondary structure k a threshold c^k is determined that is used to discriminate between true predictions and false ones. For each pair of positions p and q a prediction of type k for the corresponding subsequence $(x_p, x_{p+1},\dots,x_q)$ is reported, if and only if $\mathcal{S}^k(p,q) > c^k$.

7.20 The doublet characteristic

The *doublet characteristic* is similar to the singlet characteristic. The hope is to obtain a better discrimination by considering *pairs* of amino acids separated by $d = 0, 1, 2$ or 3 other residues.

The *preference* for a particular type of secondary segment k for a pair of amino acids of type a and b, separated by d other residues, is defined as:

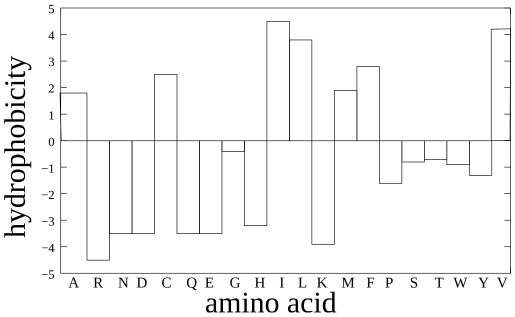
$$D^k(a,b,d) = \frac{P^k(a,b,d)}{P(a,b,d)},$$

where $P(a,b,d)$ and $P^k(a,b,d)$ are the proportions of amino acids of type a and b separated by d others, that are contained in the whole database and in segments of type k, respectively.

The average preference of a segment $(x_p,\dots,x_q)$ to be in a particular secondary structure k is denoted by $\mathcal{D}^k(p,q,d)$ and is obtained as the sum of all the pair characteristics occurring in the N$_l$, N, internal, C and C$_r$ segments.

7.21 The hydrophobic moment

Secondary structure prediction can be aided by examining the periodicity of amino acids with hydrophobic side chains in the protein chain. Tables assigning a *hydrophobicity* value $h(a)$ to each amino acid a (Kyte and Doolittle 1982) are used to the determine the hydrophobicity of different regions of a protein:



For example, there is a tendency of hydrophobic residues located in α helices on the surface of a protein to face the core of the protein and for polar and charged amino acids to face the aqueous environment on the outside of the α helix.

This is captured using the concept of *hydrophobic moment*, which is the hydrophobicity of a peptide measured for different angles of rotation per residue (from $0 - 180^o$), giving a measure of the probability that the peptide separates hydrophobic and hydrophilic regions (Eisenberg *et al*, 1984).

For any given segment $(x_p, \ldots, x_q)$ of sequence, the hydrophobic moment for the angle ω is defined as:

$$\mathcal{M}^{\omega}(p,q) = \left[\left(\sum_{i=p}^{q} h(x_i) \cos(i\omega) \right)^2 + \left(\sum_{i=p}^{q} h(x_i) \sin(i\omega) \right)^2 \right]^{\frac{1}{2}},$$

where $h(a)$ denotes the hydrophobicity of the amino acid a.

In the context of predicting α helices and β sheets, the angles of interest are $\omega = 100°$ and $\omega = 160°$, respectively. We use $\omega(k)$ to denote the angle associated with the structure type k.

7.22 Combining the discriminant functions

The SSP method for secondary structure prediction uses a linear combination of all three described discriminant functions (*LDF*, linear discriminating function):

$$\mathcal{Z}^k(p,q) = \alpha_1^k \times \mathcal{S}^k(p,q) + \alpha_2^k \times \mathcal{D}^k(p,q,d) + \alpha_3^k \times \mathcal{M}^{\omega(k)}(p,q).$$

Given a threshold c^k, this function classifies a segment of sequence (x_p, x_q) into *class 1* (i.e., is structure of type k), if $\mathcal{Z}^k(p,q) > c^k$, or *class 2* (i.e., is not structure of type k), if $\mathcal{Z}^k(p,q) \leq c^k$. (Moreover, d is given).

For each type of structure k, the method of *linear discriminant analysis* is used to to determine the coefficients $(\alpha_1^k, \alpha_2^k, \alpha_3^k)$ and the threshold constant c^k. Given a training set, the goal is to maximize the ratio of the between-class variation of $\mathcal{Z}^k$ to within-class variation. (We will skip the details.)

7.23 The SSP algorithm

Given a protein sequence $x = (x_1, \ldots, x_L)$, the SSP algorithm predicts secondary structures in the following way:

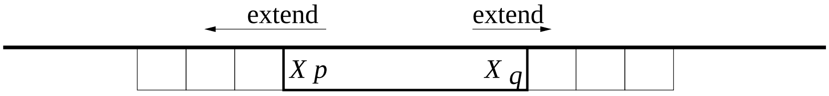
First, a nucleus of a potential α helix is searched for as a segment $(x_p, \ldots, x_q)$ of five residues with an average singleton characteristic higher than a pre-given threshold.

Then, the value of the LDF $\mathcal{Z}^k(p, q)$ for $k =$"short α helix" is compared with the corresponding threshold c^k. If the threshold is not exceeded, then the given segment is deemed unpromising and a new segment is considered.

In the case that the threshold is exceeded, the segment is repeatedly extended in either direction by one residue per step, up to a maximal extension of 15 residues in either direction.

After each extension, the value of the appropriate LDF is computed. The extended segment that gives rise to the highest LDF score is then considered a potential α helix.

A similar *seed-and-extend* strategy is used to determine potential β strand segments:



The result is a collection of potential α helices and β strands.

To obtain a final prediction, overlapping pieces are assigned to the secondary structure types that have the highest LDF value in the region of the overlap. Non-overlapping remainders of such pieces with lower LDF values are retained as predictions, if they are still long enough.

The minimum length for assigning an α helix or β strand is five or three residues, respectively.

SSP server (possibly dead?):
`http://dot.imgen.bcm.tmc.edu:9331/seq-search/struc-predict.html`

7.24 Measuring prediction accuracy

The accuracy of computational methods that need to be trained on a database of solved structures is often assessed using the *jackknife* procedure: The method is trained on *all but one* data set and then applied to the left out data set D, comparing the result to the true structure associated with D.

To evaluate the performance of a secondary structure prediction, one possibility is to assess the level of single-residue accuracy. For example, this can be measured as the percentage of correct residue predictions Q, or the sensitivity and specificity of the method.

However, this may be problematic, for example, a clearly wrong prediction such as $\alpha\beta\alpha\beta\alpha\ldots$ in an α helix region will still give rise to a score of 50% correct residue predictions.

Thus, in practice one also evaluates the number of correctly predicted α helices and β strands, considering a structure to be *correctly predicted*, if it contains more than a pre-defined number of correctly predicted residues, often just 2.

7.25 Performance of different characteristics

An experimental evaluation of secondary structure predictions was performed on 126 non-homologous proteins with known three-dimensional structures (Rost and Sander 1993), the secondary structure of which was assigned using the DSSP program.

Different combinations of characteristics were compared with each other, giving rise to the following results:

Characteristics used	Q (%)
Singleton characteristic ($\mathcal{S}$)	58.5
$\mathcal{S}$ + hydrophobic moment $\mathcal{M}$	61.4
Doublet characteristic ($\mathcal{D}$)	62.2
$\mathcal{D} + \mathcal{M}$	64.8
$\mathcal{S} + \mathcal{D} + \mathcal{M}$	65.1

(Source: T. Jiang, Y. Xu and M.Q. Zhang (editors), 2002, page 383)

7.26 Segment prediction accuracy

As stated above, single-residue accuracy measures sometimes give a misleading impression of the usefulness of a prediction.

For example, assigning the coil state to all positions in the protein $4sgb$ produces a score of $Q = 76.7\%$, although this protein contains several (missed) β structures.

On the other hand, for the protein $3b5c$, SSP correctly predicts four out of five existing α helices and three out of five existing β strands, although the attained Q value is only 56.5%.

The segment prediction accuracy can be estimated as follows: a structure is considered correctly predicted if it has at least two amino acids in common with the correct one.

Under this measure, it was observed that long structures are better predicted than short ones: 89% of α helices of length ≥ 8 and 71% of β strands of length ≥ 6 were correctly predicted, with a specificity of 0.82 and 0.78, respectively.

7.27 A more elaborate discriminant approach

The slightly more recent DSC method (King and Sternberg 1996) is also based on linear discriminant functions and involve from 10 to 27 protein features. Additional post-processing steps are performed to account for higher order properties that cannot be captured by a linear discriminant function. The performance of this method was measured to be 70.1% correctly classified residues.

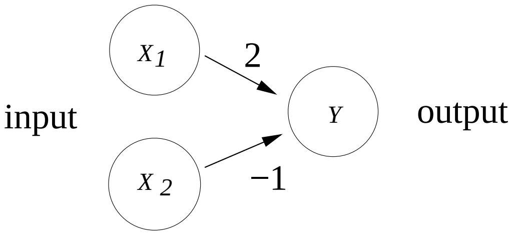
7.28 Neural networks

Several algorithms using neural networks for secondary structures have been developed. One of the most popular is the PHD algorithm by Rost and Sander (1993), which we discuss in detail below.

A *neural network* is a graph whose nodes represent simple processing units and whose (directed) edges define interconnections between the processing elements. Each connection has a *strength*, or *weight*, and the processing ability of the network resides in the weights of its connections. These weights are usually set in a process of adaptation to, or learning from, a given training set.

Neural networks are inspired by neurons in the brain. Neural networks are used for classification problems for which there exist a good supply of training data and little understanding of the structure of the problem at hand.

Here is a very simple example of a neural net whose task it is to determine whether $x_1 > \frac{1}{2}x_2$:

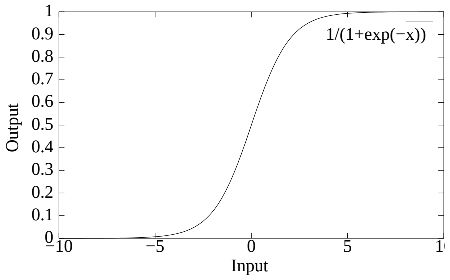


It takes two numbers x_1 and x_2 as input and produces a signal $y = 2x_1 + (-1)x_2$ as output that is positive, if $2x_1 > x_2$, and negative, if $2x_1 < x_2$.

To mimic the *firing* of a neuron, we would like the output of the node labeled y to be 1, if $2x_1 > x_2$ and 0, if $2x_1 < x_2$. This could be realized using a simple step function $y = \begin{cases} 1 & \text{if } 2x_1 > x_2 \\ 0 & \text{else.} \end{cases}$

However, it is better to use a continuous function for this purpose, such as a step-like sigmoidal function of the form:

$$y = \frac{1}{1 + \exp(-x)}.$$



In a neural net, a node y that is fed from r nodes $x_1, \ldots, x_r$ by edges (x_i, y) with weights w_i "processes" these inputs and fires a signal of strength $\frac{1}{1+\exp(-x)}$, where $x = \sum_{i=1}^{r} w_i x_i$.

(This is all in the simple case of a feed-forward network, which is what we will consider here.)

7.29 Constructing a neural network

There are two steps to constructing a neural network.

The first step is to design the topology of the network. This involves determining the number of *input nodes* and *output nodes* and how they are associated with external variables. Additionally, the number of internal or *hidden* (layers of) nodes must be determined. Finally, nodes have to be connected using edges.

The second step is called *training. Supervised training* requires a training set consisting of input data points for which the desired output is known. Each such data point is presented to the neural net and then the weights in the net are slightly modified using a gradient descent method so as to increase the performance of the network (as discussed below).

The goal is to set the weights of the edges so that the number of correct results produced for a given training data set is maximized.

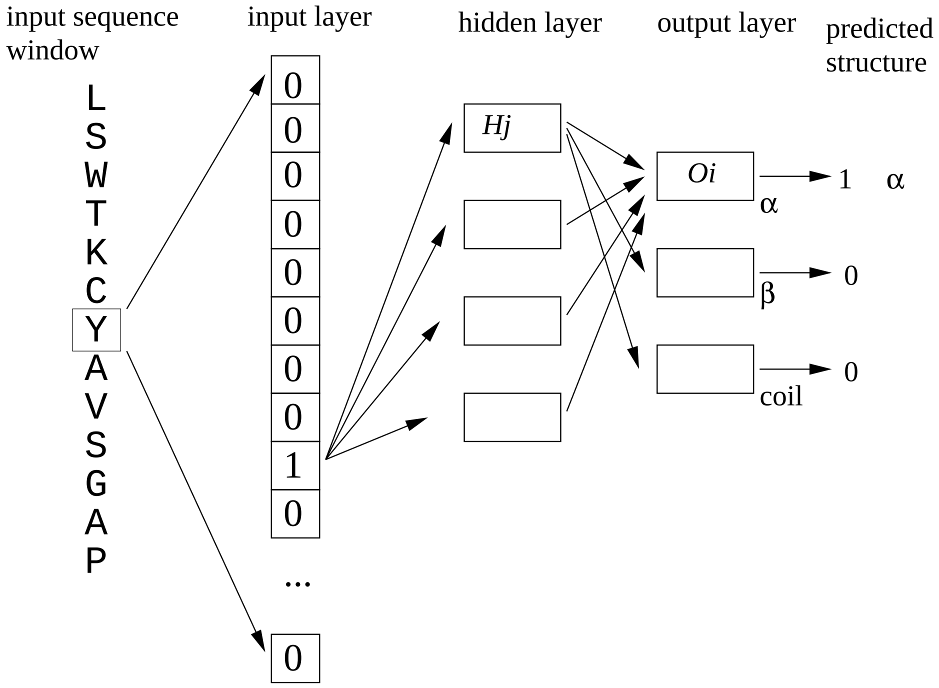
7.30 The PHD neural network

The PHD algorithm by Rost and Sander (1993) uses a neural network to predict the secondary structure of a given residue.

The model consists of three processing units, the input layer, the output layer and a hidden layer. The units of the input layer the amino acids read a small segment (13-17 residues) of sequence around the position of interest, obtained using a *sliding window*.

There are 21 input units per sequence position, namely one per amino acid and one for padding at the beginning and end of the sequence. Given a single sequence, the input unit corresponding to a given amino acid at a given position is set to 1.

Then signals are sent to units in the hidden layer, which process them and pass them on to the units of the output layer. The final output determines which of the three types of secondary structure is assigned to the central residue.



(Adapted from Mount, 2001)

If the input to the neural net consists of a sequence profile, then each input unit is set to the frequency of the associated amino acid at the given position. Additionally, two input units are used to count insertions and deletions.

The predictions obtained for adjacent windows are then post-processed by applying rules or additional neural nets to obtain a final prediction.

Experimental studies show that the PHD method applied to sequences obtains a single-residue accuracy of 70.8%. Application to sequence profiles gives rise to an accuracy of 72% (Rost and Sand 1994).

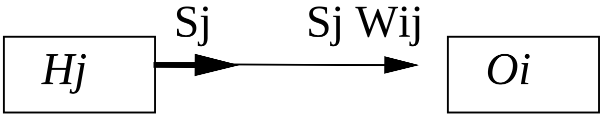
The PHD algorithm uses sequences from the HSSP (homology-derived secondary structure of proteins) database for training (Sander and Schneider, 1991).

7.31 Training the PHD neural network

A method called *back-propagation* can be used to train such neural networks. For example, consider the output node O_i shown in the network above and assume that it predicts whether the central residue lies in an α helix. The output signal s_i predicts an α helix, if it is close to 1, or not, if it is close to 0.

Presented with a training data point, we know whether or not the central residue actually lies in an α helix, and thus, what the desired output d_i of O_i should be.

Consider one of the hidden units H_j that is connected to O_i and emits a signal s_j that is modified by the weight w_{ij}. The signal arriving at O_i is $s_j \times w_{ij}$:



When training the network, the main question is how do we alter w_{ij} so as to bring the value s_i of O_i closer to the desired value d_i? The *gradient descent* method specifies that we modify w_{ij} by the following amount:

$$\Delta w_{ij} = -n\delta E/\delta w_{ij} + m,$$

where the partial derivative of the error E with respect to w_{ij} is given by

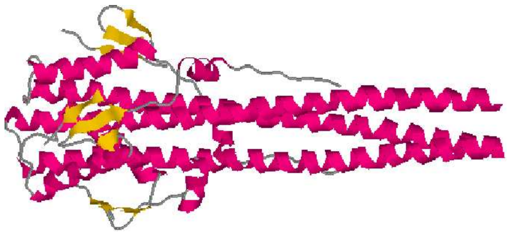
$$\delta E/\delta w_{ij} = (s_i - d_i)s_i(1 - s_i)s_j,$$

and where n is the *training rate* ($\approx$ 0.03) and m is a *smoothing factor* that allows a carryover of a fraction of previous values of w_{ij} ($\approx$ 0.2).

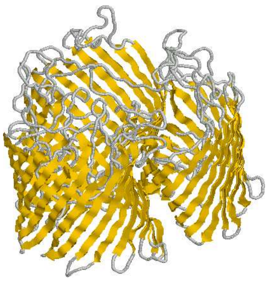
PHD web server:
`http://www.embl-heidelberg.de/predictprotein/predictprotein.html`

7.32 Configurations of secondary structure elements

Secondary structure elements such as α helices and β sheets can sometimes group into larger structures such as (a) a *coiled coil*, a configuration in which α helices (originally called "coils") are wound into a superhelix, or (b) a *parallel β helix* in which β strands wrap around to form a helix.



(a) coiled coil
Hemagglutinin at pH 4.0
1HTM



(b) parallel β helices
Maltoporin Sucrose Complex
1AF6

7.33 Coiled coils

In the following we discuss how to predict coiled coils from protein sequence. This is mainly based on the following two papers:

- Andrei Lupas, Marc van Dyke and Jeff Stock, *Predicting coiled coils from protein sequences*, Science, 252:1162-64 (1991), and

- Andrei Lupas, *Coiled coils: new structures and new functions*, TIBS 21:375-382 (1996).

Coiled coils were first described in 1953 by Pauling and Corey, and, independently, by Crick, as the main structural element of a large class of fibrous proteins that included keratin, myosin and fibrinogen.

About three percent of all proteins are thought to contain a coiled coil domain. Hence, this type of configuration is probably important for many cellular processes.

7.34 Description of coiled coils

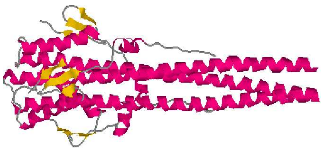
By definition, *coiled coils*

- are formed by two or three α helices in parallel in and register that cross at an angle of $\approx 20°$,
- are strongly amphipathic and display a pattern of hydrophilic and hydrophobic residues that is repeated every seven residues, and
- their sequences exhibit common patterns of amino acid distribution that appear to be distinct from those of other proteins.

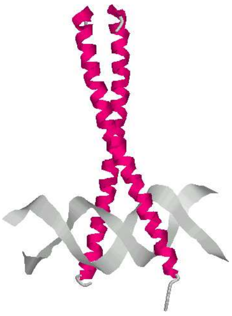
The prediction of coiled coil domains from protein sequence is based on the two latter observations. Based on them, it can be predicted with significant reliability which α helices participate in a coiled coil. However, if more than two such α helices are present, it is usually very difficult to predict *which ones will match up* to form a specific coiled coil.

Here are some examples of coiled coils:

Tropomyosin (2TMA)



Hemagglutinin
at pH 4.0 (1HTM)



GCN4 with
DNA-binding domain (1YSA)

The *leucine zipper* domain is typically made of two anti parallel α helices held together by interactions between hydrophobic leucine residues located at every seventh position in each helix, see 1YSA above. The zipper holds protein subunits together.

In the transcription factors Gcn4, Fos, Myc, and Jun, the binding of the subunits form a scissor-like structure with ends that lie on the major groove of DNA.

Coiled coils fulfill a variety of functions: they can form large, mechanically rigid structures, e.g. hair or feathers (keratin), or blood clots (fibrin), the cellular skeleton (intermediate filaments), provide a scaffold for regulatory complexes (tropmyosin), form spacers that separate the outer membrane from the cell wall in bacteria (murein lipoprotein), and provide a protective surface for pathogens (the M proteins of staphylococci). (Lupas 1996)

7.35 The heptad repeat

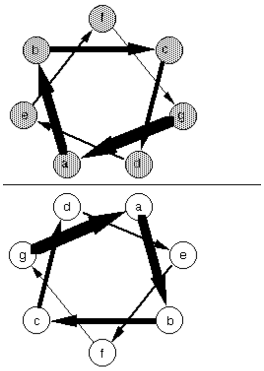
In an α helix, it takes about 3.62 amino acids to complete one turn of the helix and so the positions of the residues around the central axis of an α helix do not display a short periodicity.

However, if a right-handed α helix is given a slight left-handed twist, then the number of residues per turn can be reduced to 3.5 and the positions will display a periodicity of 7.

(By twisting the helix in the other direction to about 3.7 residues per turn, a periodicity of 11 can be achieved, but it is unclear whether such right-handed coiled coils actually exist.)

In a coiled coil configuration, the participating α helices do indeed give each other a slight left-handed twist, thus enabling themselves to line up along a periodic subset of amino acids.

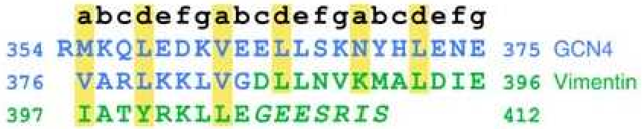
Viewed from above, the configuration of two α helices forming a coiled coil can be displayed using a *helical-wheel* plot:



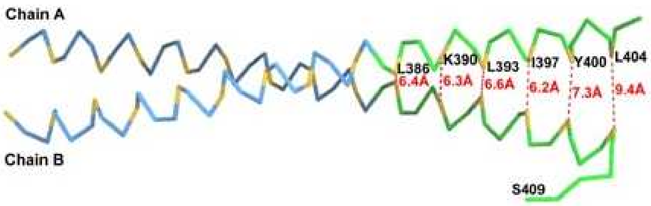
http://www.cryst.bbk.ac.uk/PPS95/course/6_super_sec/cc.html

The seven periodic classes of amino acid positions are called a, b, c, d, e, f and g. The positions a and d are filled by hydrophobic residues and form the helix interface. The other residues are hydrophilic and form the solvent exposed part of the coiled core.

Here is another illustration of the periodic nature of the distribution of hydrophobic residues along α helices participating in coiled coils:



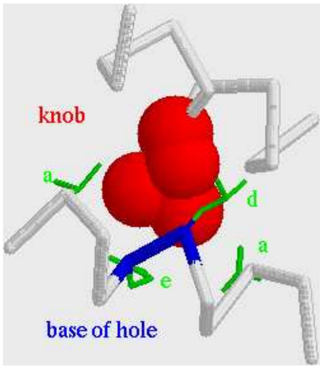
```
     abcdefgabcdefgabcdefg
354 RMKQLEDKVEELLSKNYHLENE 375 GCN4
376  VARLKKLVGDLLNVKMALDIE 396 Vimentin
397  IATYRKLLEGEESRIS      412
```



http://www.biozentrum.unibas.ch/personal/burkhard/

7.36 Knobs into holes

In this configuration, the residues of the two helices mesh nicely in what is called a *knobs-into-holes* packing:



http://www.cryst.bbk.ac.uk/PPS95/course/6_super_sec/cc.html

Here, a residue from one helix (knob) packs into a space surrounded by four side chains of the facing helix (hole).

7.37 Obtaining coiled-coil statistics

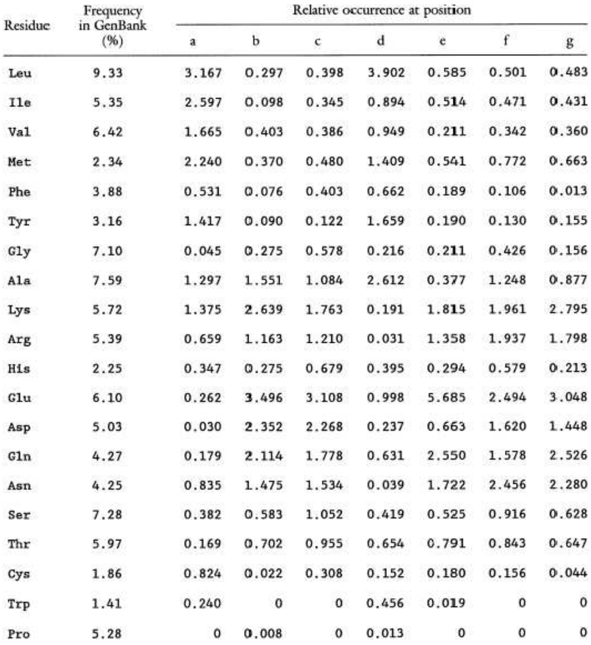
To be able to predict coiled coil forming α helices from sequence, a database of training sets is needed.

The sequences of coiled-coil domains from tropmyosins, myosins, and keratins deposited in GenBank provide a coiled-coil database.

For each of the twenty amino acids A, one can determine the frequency $P_i(A)$ with which it occurs at position $i \in \{a, \ldots, g\}$ of the heptad repeat and the frequency $P(A)$ with which A occurs anywhere, in any sequence in GenBank. This then gives rise to the *relative frequency* with which A occurs at position i:

$$F_i(A) = \frac{P_i(A)}{P(A)}.$$

Relative frequencies reported by Lupas et al (1991):



Residue	Frequency in GenBank (%)	Relative occurrence at position						
		a	b	c	d	e	f	g
Leu	9.33	3.167	0.297	0.398	3.902	0.585	0.501	0.483
Ile	5.35	2.597	0.098	0.345	0.894	0.514	0.471	0.431
Val	6.42	1.665	0.403	0.386	0.949	0.211	0.342	0.360
Met	2.34	2.240	0.370	0.480	1.409	0.541	0.772	0.663
Phe	3.88	0.531	0.076	0.403	0.662	0.189	0.106	0.013
Tyr	3.16	1.417	0.090	0.122	1.659	0.190	0.130	0.155
Gly	7.10	0.045	0.275	0.578	0.216	0.211	0.426	0.156
Ala	7.59	1.297	1.551	1.084	2.612	0.377	1.248	0.877
Lys	5.72	1.375	2.639	1.763	0.191	1.815	1.961	2.795
Arg	5.39	0.659	1.163	1.210	0.031	1.358	1.937	1.798
His	2.25	0.347	0.275	0.679	0.395	0.294	0.579	0.213
Glu	6.10	0.262	3.496	3.108	0.998	5.685	2.494	3.048
Asp	5.03	0.030	2.352	2.268	0.237	0.663	1.620	1.448
Gln	4.27	0.179	2.114	1.778	0.631	2.550	1.578	2.526
Asn	4.25	0.835	1.475	1.534	0.039	1.722	2.456	2.280
Ser	7.28	0.382	0.583	1.052	0.419	0.525	0.916	0.628
Thr	5.97	0.169	0.702	0.955	0.654	0.791	0.843	0.647
Cys	1.86	0.824	0.022	0.308	0.152	0.180	0.156	0.044
Trp	1.41	0.240	0	0	0.456	0.019	0	0
Pro	5.28	0	0.008	0	0.013	0	0	0

7.38 Sliding window evaluation

Given a protein sequence $x = (x_1, \ldots, x_L)$. These relative frequencies $F_i(A)$ are used for prediction as follows:

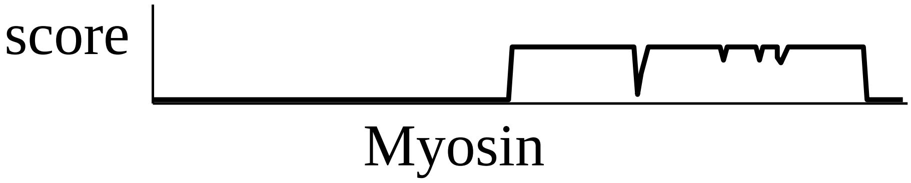
A *sliding window* of length 28 residues is moved along the sequence and for each start position $p = 1, 2, \ldots, L - 27$ of the window, the following steps are performed:

- the window is assigned a heptad repeat frame,
- each residue in the window is assigned the appropriate frequency f_i obtained from the above table, and then
- the geometric mean G all these values $f_1, f_2, \ldots, f_{28}$ is computed, $G = \left(\prod_{i=1}^{28} f_i\right)^{\frac{1}{28}}$.

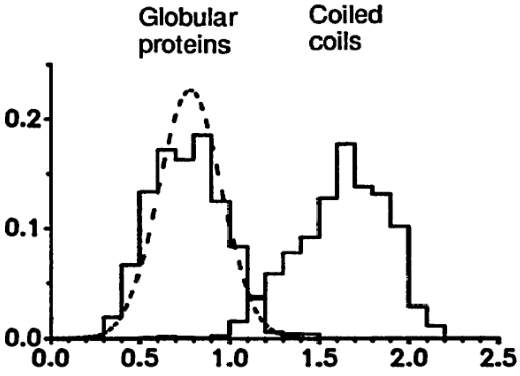
Thus, each choice of window and heptad repeat frame is given a score.

Consider a fixed residue x_i: it is contained in 28 different windows and for each such window, there are 7 different choices for heptad frames. (If the residue is close to one end of the sequence, then this number is smaller, of course.) The residue x_i is assigned a score that is simply the largest score assigned to any window (and heptad repeat frame) that contains it.

Because the maximal score over all windows is taken, we obtain a step-like score function along the sequence, e.g.:



Running the algorithm on a collection of globular proteins and then on a collection (of roughly the same size) of known coiled coils produces the following distribution of scores:



Lupas et al. 1991

For globular proteins, the mean score is 0.77 with a standard deviation of 0.20. For coiled-coil sequences, the mean score is 1.63 and the standard deviation is 0.24.

7.39 Estimation of probability of being a coiled coil

The above score distributions allows an estimate of the probability that a residue with a given score would be in a coiled score. The ratio of globular to coiled-coil proteins is estimated to be approximately 1 : 30. The probability P of forming a coiled coil of a given score S is then:

$$P(S) = \frac{G_{cc}(S)}{30G_g(S) + G_{cc}(S)},$$

where G_g and G_{cc} are two Gaussian curves that approximate the distribution of globular and coiled-coil sequences, respectively. This probability is then used to predict coiled coils.

7.40 Implementation

An implementation of the approach described here can be run at: `http://www.ch.embnet.org/software/COILS_form.html`.

8 Protein tertiary structure

Sources for this chapter, which are all recommended reading:

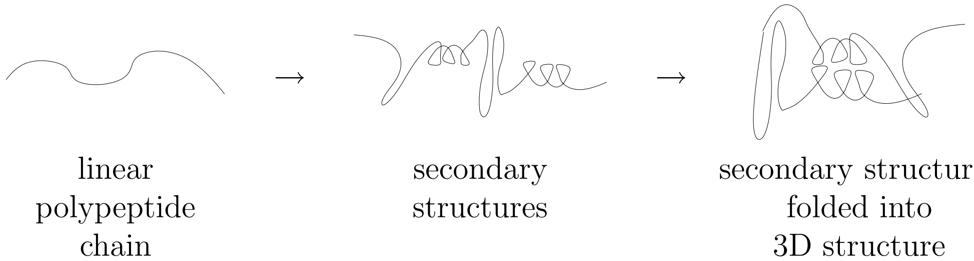
- D.W. Mount. *Bioinformatics: Sequences and Genome analysis*, Cold Spring Harbor Press, Chapter 9: Protein classification and structure prediction. pages 381-478, 2001.
- Nick Alexandrov, Ruth Nussinov, and Ralf Zimmer. *Fast protein fold recognition via sequence to structure alignment and contact capacity potentials.* In Lawrence Hunter and Teri E. Klein, editors, Pacific Symposium on Biocomputing'96, World scientific publishing company, pages 53-72, 1996.
- K.T. Simons, C. Kooperberg, E. Huang and D. Baker, *Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions*, JMB 268:209-225 (1997).

8.1 Hierarchy of protein structure

K.U. Linderstrom-Lang (Linderstrom-Lang & Schnellman 1959) proposed to distinguish four levels or protein structure:

- The *primary structure* is the chemical structure of the polypeptide chain(s) in a given protein, i.e. its sequence of amino acid residues that are linked by peptide bonds.
- The *secondary structure* is folding of the molecule that arises by linking the C=O and NH groups of the backbone together by means of hydrogen bonds.
- The *tertiary structure* is the three dimension structure of the molecule consisting of secondary structures linked by "looser segments" of the polypeptide chain stabilized (primarily) by side-chain interactions.
- The *quaternary structure* is the aggregation of separate polypeptide chains into the functional protein.

Pathway for folding a linear chain of amino acids into a three-dimensional protein structure:



The tertiary structure of proteins is of great interest, as the shape of a protein determines much, if not all, of its function.

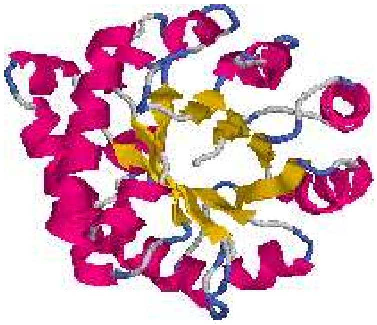
At present, the experimental determination of protein structure via x-ray crystallography is difficult and time-consuming. Hence, we would like to be able to determine the structure of a protein from its sequence. Determining the secondary structure is an important first step.

8.2 The "Holy Grail" of bioinformatics

The *holy grail* of bioinformatics: develop and algorithm that can reliably predict the structure (and thus function) of a protein from its amino-acid sequence!

```
...
IIFIATTNLLGLLPHSFTPTTQLSMNLAMAIPLWA
GAVILAHFLPQGTPTPLIPMLVIIETISLLIQPAL
AVRLTANITAGHLLMGSATLAMTLIIFTILILLTI
LEIAVALIQAYVFTLLVSLYLHDNTPQLNTTVWPT
MITPMLLTLFLITQLKMLPWEPKWADRWLFSTNHK
DIGTLYLLFGAWAGVLGTALSLLIRAELGQPGNLL
GNDHIYNVIVTAHAFVMIFFMVMPIMIGGFGNWLV
PLMIGAPDMAFPRMNNMSFWLLPPSLLLLLASAMV
...
```

$\xrightarrow{?}$



Protein structure prediction consists of three main areas: *fold recognition*, *comparative modeling* and *de novo fold prediction*. We will look at each in the following lectures.

8.3 The fold recognition problem

As of June 2000, more than 12500 protein structures had been deposited in the Brookhaven Protein Data Bank (PDB), and 86500 protein sequence entries were contained in the SWISSProt protein sequence database.

Structural alignments have revealed that there are more than 500 common structural folds that are found in the domains of these protein structures.

The **fold recognition problem** can be formulated as follows:

> Given a protein sequence of unknown structure and a database of representative folds, identify the most plausible fold for the sequence, if there is one, and assess the quality or reliability of the proposed structure.

A mapping of the target sequence on to one of the known structures is called a *sequence-structure* alignment or a *threading*.

8.4 The 123D fold recognition method

In the following, we will discuss one such *threading* method called *123D*, as described in the paper by Alexandrov, Nussinov and Zimmer (1996).

- Using simple empirical potentials, this approach optimizes mappings of residues in the target sequence x onto structural positions of any of the proposed folds.
- The resulting alignments are then evaluated and ranked according to the potential.
- Finally, the statistical significance of the best alignment is estimated in comparison with the other alignments.

8.5 Potentials

The method uses statistically derived *potentials* computed from a non-redundant set of approx. 150 representative protein structures.

An empirical free energy function is used that is given by the sum of three terms:

- *secondary structure* preferences,
- *pairwise contact* potentials, and
- *contact capacity* potentials.

Given a target sequence $x = (x_1, x_2, \ldots, x_m)$ and a proposed fold $y = (y_1, y_2, \ldots, y_n)$, dynamic programming is used to find an optimal threading of x through y, i.e. an an alignment of x to y that minimizes the free energy function.

8.6 Secondary structure preference

Each position $j = 1, \ldots, n$ in the fold y is assigned a secondary structure class $s(j) \in \{\texttt{alpha}, \texttt{beta}, \texttt{other}\}$.

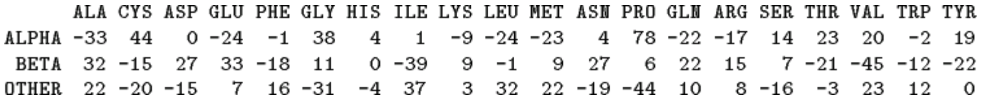
The *secondary structure preference (SSP)* of an amino acid a to be found in a secondary structure of class k is calculated for each of the 20 amino acids as follows:

$$SSP(a,k) := -\log \frac{P^k(a)}{\langle N(a,k)\rangle}, \text{ with } \langle N(a,k)\rangle = \frac{N(a) \times N(k)}{N}.$$

These numbers are obtained by counting occurrences in the given database of known structures:

- $N(a,k)$ is the number of amino acids of type a contained in a secondary structure of class k,
- $\langle N(a,k)\rangle$ is the expected number of residues of type a to be contained in a secondary structure of class k,
- $N(a)$ is the number residues of type a,
- $N(k)$ is the number of residues contained in a secondary structure of class k, and
- N is the total number of amino acids.

This is very similar to the the singleton characteristic used in the context of secondary structure prediction. Typical values obtained for the SSP potential are shown here (multiplied by 100 for clarity):



	ALA	CYS	ASP	GLU	PHE	GLY	HIS	ILE	LYS	LEU	MET	ASN	PRO	GLN	ARG	SER	THR	VAL	TRP	TYR
ALPHA	-33	44	0	-24	-1	38	4	1	-9	-24	-23	4	78	-22	-17	14	23	20	-2	19
BETA	32	-15	27	33	-18	11	0	-39	9	-1	9	27	6	22	15	7	-21	-45	-12	-22
OTHER	22	-20	-15	7	16	-31	-4	37	3	32	22	-19	-44	10	8	-16	-3	23	12	0

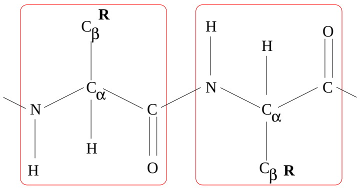
(Alexandrov, Nussinov and Zimmer, 1996)

This confirms the observation that amino acids Ala, Glu, Leu and Met are often found in α helices, whereas Pro, Gly, Ser, Thr and Val occur rarely in them.

Similarly, amino acids Val, Ile, Try and Thr prefer β sheets, whereas Glu, Gln, Lys, Asp, Pro and Cys are seldomly found in them.

8.7 Pairwise contact potentials

Two residues x_i and x_j are defined in be *in contact*, if the distance between their C_β atoms is less than 7.0 Angstrom (S. Miyazawa and R.L. Jernigan 1985).



Note that Glycine has no C_β atom, so fake coordinates must be calculated from the backbone for amino acids of type Gly.

The contact potentials for amino acids a and b are calculated as follows:

$$CP(a,b) := -\log \frac{N(a,b)}{\langle N(a,b)\rangle}, \text{ with } \langle N(a,b)\rangle := \frac{N(a)\times N(b)}{N}.$$

These numbers are obtained by counting occurrences in the given database of known structures:

- $N(a,b)$ is the actual number of residues a and b in contact,
- $\langle N(a,b)\rangle$ is the expected number of contacts between an amino acid of type a and one of type b,
- $N(a) = \sum_z N(a,z)$ and $N(b) = \sum_z N(z,b)$ are the total number of contacts of an amino acid a or b, respectively, and
- $N = \sum_{a,b} N(a,b)$ is the total number of pairs of amino acids in contact.

8.8 Contact capacity potentials

The *contact capacity potential (CCP)* characterizes the ability of a residue to make a certain number of contacts with other residues. Its role is to account for the hydrophobic contribution to the free energy, as obviously, hydrophobic residues will prefer to have more contacts than hydrophilic ones.

For each type a of amino acid, its ability to form r contacts is given by:

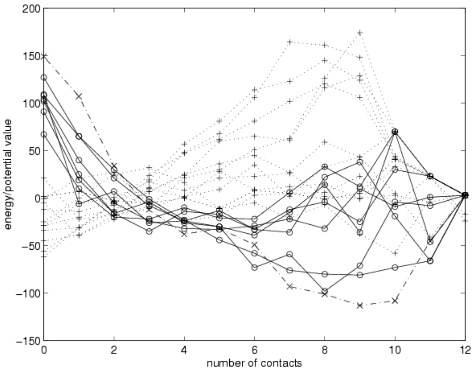
$$CCP(a,r) := -\log \frac{N(a,r)}{\langle N(a,r)\rangle}, \text{ with } \langle N(a,r)\rangle := \frac{N(a)\times NC(r)}{N}.$$

Again, these numbers are obtained by counting occurrences in the given database of known structures:

- $N(a,r)$ is the number of residues of type a having precisely r contacts,
- $\langle N(a,r)\rangle$ is the expected number of residues having r contacts,
- $N(a)$ is the number of residues of type a,
- $NC(r)$ is the number of residues having r contacts, and
- N is the total number of residues.

A contact is called *local*, if less than five residues lie in the sequence between the two residues in contact. In practice, it makes sense to distinguish between *local* and *long-range* contact capacity potentials:

There is a clear correlation between hydrophobicity and long-range contact capacity potentials, indicated here for hydrophobic (circled), polar (dotted) and Cysteine residues (dashed):



(Alexandrov, Nussinov and Zimmer, 1996)

Local contact capacity shows some correlation with secondary structure preferences: obviously, those amino acids that have a preference to be in an α helix tend to have more local contacts.

The contact capacity potential can be refined further in a number of ways, e.g. by considering the secondary structure, the actual distances of contacts, or even the angles of contacts, thus obtaining up to 648 different contact capacity potentials.

8.9 Secondary structure dependent CCP

The ability of residues to make contacts may depend on the secondary structure: residues in α helices have less vacant surrounding space for contacting residues than ones in β strands.

Thus, we obtain six types of SS-contact capacity potentials:

$$\left\{ \begin{array}{c} \text{local} \\ \text{long} - \text{range} \end{array} \right\} \times \left\{ \begin{array}{c} \text{alpha} \\ \text{beta} \\ \text{other} \end{array} \right\}.$$

A significant different between the long-range CCP for α helices and the one for β strands (values ×100) can be observed:

#contacts:	0	1	2	3	4	5	6	7	8	9
ALA	8	35	39	-10	-40	-36	-22	-3	0	5
CYS	95	78	26	-36	-64	-24	-73	-111	-133	5
ASP	-61	-20	16	72	104	115	79	53	40	5
GLU	-60	-20	22	54	95	104	134	99	40	5
PHE	107	11	-44	-49	-19	20	11	54	1	5
GLY	-14	13	32	52	-3	-35	-59	-85	-31	-47
HIS	31	-43	-46	14	35	77	59	-34	40	5
ILE	108	38	-7	-34	-44	-46	-60	-33	-34	-26
LYS	-49	-38	-8	60	134	165	130	122	40	5
LEU	109	27	-21	-41	-41	-34	-12	-22	22	5
MET	82	2	-18	-34	-13	-42	-18	63	-17	5
ASN	-36	-1	6	21	39	50	40	-13	1	5
PRO	-28	10	2	31	37	19	-37	-14	-96	5
GLN	-25	-25	-10	30	33	65	73	85	40	5
ARG	-6	-44	-23	5	80	96	70	87	40	5
SER	-26	1	18	31	8	-5	17	-5	-14	5
THR	-9	13	7	10	12	-31	-18	-16	-21	5
VAL	106	48	21	-38	-48	-61	-79	-62	9	5
TRP	99	-39	-25	-20	-7	-35	81	122	40	5
TYR	100	-5	-33	-31	-33	0	11	70	-11	5

long-range CCP for α helices

#contacts:	0	1	2	3	4	5	6	7	8	9	10	11	12
ALA	9	26	23	10	-5	0	-9	-19	-31	-3	-3	9	-24
CYS	197	124	71	8	-1	-4	-18	-51	-67	-77	-47	-48	3
ASP	-59	-59	-38	-4	-6	19	68	104	89	58	47	9	-31
GLU	-55	-41	-58	-28	3	26	92	91	123	103	9	9	3
PHE	91	35	24	7	-4	-19	-33	-21	18	-14	47	9	3
GLY	-74	-21	9	17	33	0	25	4	10	-8	-67	-14	3
HIS	7	-1	13	-43	-21	-6	45	22	27	76	47	9	3
ILE	153	105	55	30	19	1	-57	-40	-73	-39	24	-13	3
LYS	-40	-50	-56	-28	2	9	89	125	119	141	18	9	3
LEU	101	90	55	21	-3	-33	-36	-22	-35	4	-7	-12	3
MET	86	50	12	7	-9	-17	-45	-2	10	66	47	9	3
ASN	-41	-63	-42	0	5	34	48	54	52	18	47	9	3
PRO	-17	-54	-19	-6	-10	43	24	29	39	-3	9	9	3
GLN	-36	-48	-25	-41	-12	50	47	53	114	87	47	9	3
ARG	-24	-9	-5	-45	-27	-2	66	51	108	141	47	9	3
SER	-42	-42	-28	1	-3	36	40	27	17	29	-23	9	3
THR	-24	-6	-26	-23	0	5	42	27	20	16	47	9	3
VAL	81	77	64	33	8	-14	-28	-54	-43	-64	-37	-7	3
TRP	127	65	35	-13	-20	20	-40	-35	10	13	-36	9	3
TYR	103	52	23	41	-10	-35	-41	-10	17	-28	47	9	3

long-range CCP for β strands

(Alexandrov, Nussinov and Zimmer, 1996)

8.10 Alignment using CCPs and sequence information

Given a target protein sequence $x = (x_1, \dots, x_m)$ and a sequence $y = (y_1, \dots, y_m)$ whose fold is known, we use dynamic programming to determine the best scoring alignment between the two sequences.

Recall that slightly different algorithms are used, depending one whether one is interested in the best global-, local or e.g. overlap alignment. All three are variations of the following recurrence:

$$\begin{aligned} M(i,j) &= \max\left(M(i-1,j-1), I_x(i-1,j-1), I_y(i-1,j-1)\right) \\ &\quad + \text{match}(x_i, y_j), \\ I_x(i,j) &= \max\nolimits_{k<i}\left(M(i-k,j) - g_x(i,j,k)\right), \\ I_y(i,j) &= \max\nolimits_{k<i}\left(M(i,j-k) - g_y(i,j,k)\right). \end{aligned}$$

This recursion defines the maximal score $M(i,j)$ of the alignments of the i- and j-prefixes of the two sequences x and y.

To take the introduced potentials into account, the term for single matches can be modified as follows:

$$\text{match}(i,j) = \alpha \times s(i,j) + \beta \times l(i,j) + \gamma \times cc(i,j),$$

where

- $s(i,j)$ is the sequence score of substituting y_j by x_i using an appropriate BLOSUM or PAM matrix,
- $l(i,j) = SSP(x_i, s(j))$ scores the local preference of the i-th amino acid x_i to be in the secondary structure class $s(j)$ of the amino acid at site j of the folded sequence y, and
- $cc(i,j) = CCP(x_i, nc(j))$ denotes the contact capacity score achieved when mapping x_i to position j, i.e. the energy assigned to amino acid x_i to have $nc(j)$ contacts, where $nc(j)$ is the number of contacts that position j in the folded sequence actually has.

The variables α, β and γ are weighting factors relating the different contributions, usually set to 0 or 1 to turn different contributions off and on.

For smoothing, an averaged match score can be obtained by averaging over a window of length $2w+1$ centered at the match in question, usually with $w = 3$:

$$\overline{\text{match}}(i,j) := \frac{1}{2w+1} \sum_{k=-w}^{w} \text{match}(i+k, j+k).$$

Gaps in the alignment are penalized using an affine gap score with gap open penalty d and gap extension penalty e:

$$g(i,j,k) = \begin{cases} \sigma(d+ke) & \text{if } s(j) \in \{\texttt{alpha}, \texttt{beta}\}, \\ (d+ke) & \text{otherwise.} \end{cases}$$

Typical values used are $d = 10$ to 80 and $e = \frac{d}{10}$. The value of σ is either 1 or 10, depending on whether gaps for secondary structures are to be weighted.

8.11 Assessing potentials and threading methods

There are a number of tests that can be used to assess the performance of the potentials and associated optimization procedures for fold recognition:

The *shuffle* test is a simple statistical test: given a *native* score for an optimization problem, consider many permutations of the input problem and then express the native score in terms of standard deviations of the randomized scores.

In the *Sippl* test (M. Sippl 1990), the given target sequence is aligned against all sequences in a given fold database in all possible gap-free ways. The score for all these evaluations are computed and then the native score in expressed in terms of standard deviations. In effect, the Sippl test performs threading without gaps.

The *threading test* is to use the respective method and potential, try align a given sequences as well as possible to any fold of a fold database, evaluate, score and rank the resulting alignments. The native (identity) threading should be the best alignment of the sequence onto its native fold, and the score of this combination should be better than the score of all non-native combinations.

To perform the shuffle test or Sippl test, all that is needed is a program that can evaluate a given threading. The threading test is the most realistic test, but requires a full implementation of the proposed method.

The Sippl test was applied to 167 sequences, of which 139 were longer than 60 residues, yielding the following percentages correct fold recognitions:

	using SSCCP+ SSP	DCCP	ACCP
all 167 sequences	86.8%	83.8%	89.2%
139 (> 60 residues)	94.2%	93.5%	97.1%

Here, SSCCP+SSP means using the secondary-structure dependent contact capacity potential, DCCP means using a potential taking contact distances into account and ACCP means using a potential taking contact angles into account.

Addition experiments using the Sippl, shuffle and threading test show that the approach indicated here produces useful results.

The program 123D runs on the web at:

`http://cartan.gmd.de/ToPLign.html`

8.12 De novo structure prediction

As we have seen, threading methods can be used to identify a suitable structure for a given protein sequence in a database.

Alternatively, sequence alignment can be used to identify a family of homologous proteins that have similar sequences ($\geq 30\%$ sequence identity, say) and thus similar three-dimensional structures. The structure of members of a family or even super family can often be determined in this way.

However, because $40 - 60\%$ of proteins in newly sequenced genomes do not have significant sequence homology to proteins of known structure, the problem of determining new structures is important.

The **de novo protein folding problem** can be formulated as follows:

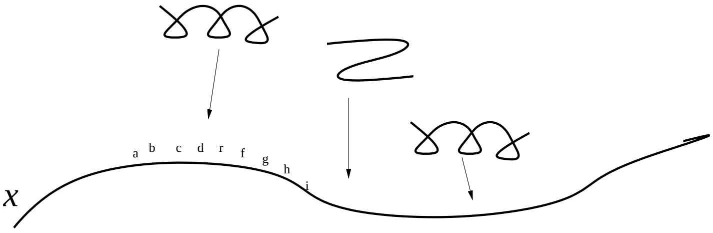
> Given a protein sequence of unknown tertiary structure, determine its structure *in the absence* of close homologs of known structure.

8.13 De novo prediction using ROSETTA

The most successful *de novo* fold prediction program at present is probably ROSETTA, due to David Baker and others (see `http://depts.washington.edu/bakerpg`).

ROSETTA is based on the assumption that the distribution of conformations sampled for a given nine residue segment of the target sequence x is reasonably well approximated by the distribution of structures adopted by the segment (and closely related sequences) in known protein structures.

The predicted structure is pieced together from the structures corresponding to the segments:



In the following we will discuss the general approach of ROSETTA as described in K.T. Simons, C. Kooperberg, E. Huang and D. Baker (1997). Much work has been done to improve the algorithm in the last five years.

As in the case of threading, structures are represented using a simplified model consisting of the heavy atoms of the main-chain and the C_β atoms of the side-chain. (For glycine residues, a fake C_β atom is used). All bond lengths and angles are held constant according to the ideal geometry of alanine, the only remaining variables are the backbone torsion angles ϕ and ψ.

Initial studies showed that there is a stronger correlation between local sequence and local structure for nine residue fragments than for other fragment lengths less than 15.

Hence, ROSETTA attempts to build structures from segments of 9 residues.

8.14 Nearest neighbors

ROSETTA uses a database of sequences with known structures selected from PDB. This is used to compute frequency distributions for all 20 amino acids at each position i of any nine-residue segment of each of the given sequences, additionally using multiple alignments from other databases and pseudo counts.

Every nine-residue segment x' in the given sequence $x = (x_1, x_2, \dots, x_L)$ is compared with every nine-residue segment y' of every sequence y in the database using the amino-acid frequency distributions at each position in the two segments:

$$DIST := \sum_{i=1}^{9} \sum_{a} |y'(a,i) - x'(a,i)|,$$

where $x'(a,i)$ or $y'(a,i)$ is the frequency of amino acid a at position i of x' or y', respectively.

Using this distance measure, for each nine-residue segment x' in the target sequence x, the set $N(x')$ of 25 *nearest neighbor* segments in the database are identified.

By applying this computation to target sequences with known structures, it can be verified that the structural similarity of the 25 neighbors to the structure of the target segment is higher than would be expected by chance alone:

Sequence information used to find fragments	% similar
Multiple sequence alignment	20.8
Single sequence	17.5
Random sequence	8.0

So, for example, in the case that the frequency distributions are obtained using multiple alignments, on average, 20.8% of the 25 nearest neighbors determined for a nine-residue segment x' lie within 1 Angstrom "distance matrix error" from the native structure of the segment.

8.15 Ideal bonds and torsion angles

The conformation of each nine-residue segment x' in the target sequence x is chosen from the list of *template* structures adopted by the 25 nearest neighbors.

As the template structures come from PDB, they are not based on ideal bonds and angles. Hence, the torsion angles found in the template structures can not be used directly, as this would lead to significant inaccuracies.

To address this problem, for each such template structure t, a random torsion angle search is performed to find a new conformation t' that assumes ideal bond lengths and angles and has a low *rmsd* (root mean squared distance) to t.

Torsion angles for the nearest neighbors were taken from these idealized structures.

8.16 The consistency of structure

Consider a nine-residue segment x' of x. If all 25 nearest neighbor sequences in $N(x')$ have very similar structures, then one can expect that a structure prediction for x' based on these similar structures will be more reliable than in the case when these structures are very varied.

Indeed, Such a correlation between the amount of structure variation in a given set of templates and the reliability of the prediction has been shown to hold.

This observation could be used to choose optimal fragment sets for building up a structure from the many local segments with different boundaries and lengths which cover each position in the sequence.

8.17 Simulated annealing

The *simulated annealing* method employs a *temperature* that cools over time (Van Laarhoven and Aarts, 1987). At high temperatures the search can move more easily to solutions whose score is less optimal than the score of the current solution. As the temperature decreases, the search becomes more and more directed toward better solutions.

I.e., let T_i denote the current solution at step i and let $z(T_i)$ denote the goodness of T_i (e.g., $-PS(T)$). In hill climbing, a move to T_{i+1} is acceptable, if $z(T_{i+1}) \geq z(T_i)$. In simulated annealing, *any* new solution is accepted with a certain probability:

$$\begin{aligned} & Prob(\text{accepting solution } T_{i+1}) \\ = & \left\{ \begin{array}{cc} 1 & \text{if } z(T_{i+1}) \geq z(T_i) \\ e^{-t_i(z(T_{i+1})-z(T_i))} & \text{otherwise,} \end{array} \right. \end{aligned}$$

where t_i is called the *temperature* and decreases over time.

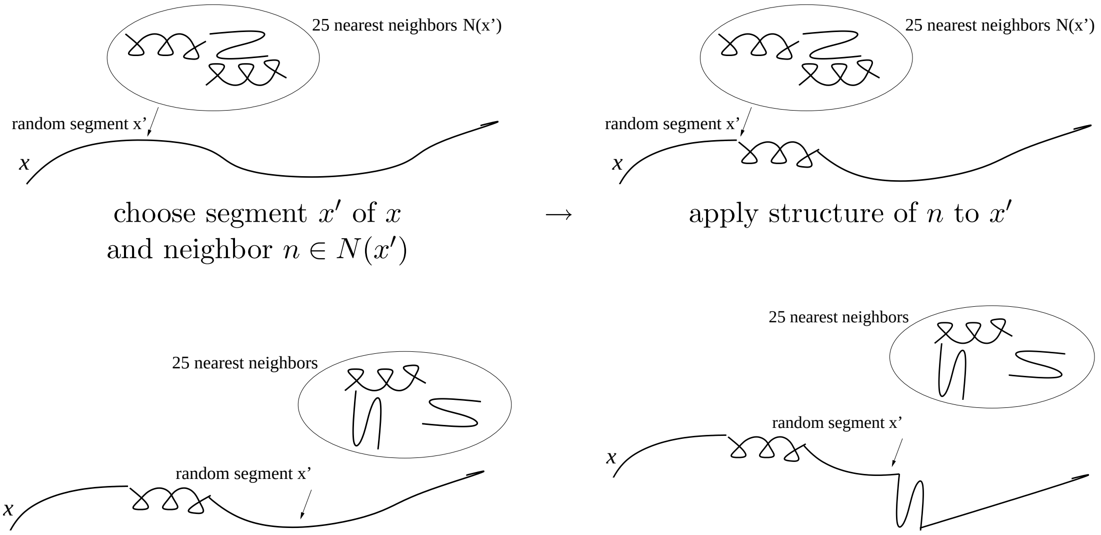
8.18 The ROSETTA algorithm

Structures are generated using a simulated annealing algorithm, employing a temperature that decreases linearly from 2500 to 10 over 10000 iterations (i.e. attempted moves).

Given a protein sequence x. The starting configuration is a fully extended chain. Repeat the following *move* 10000 times:

1. Randomly choose a nine-residue segment x' of x.
2. Randomly choose a nearest neighbor sequence $y' \in N(x')$.
3. Replace the torsion angles associated with x' by the ones associated with y'.
4. If the move puts two atoms closer than 2.5 Angstrom together, reject it.
5. If the move results in a score that does not meet the simulated annealing criterion, reject it.

Here is an illustration of the main move used in the algorithm:



choose segment x' of x and neighbor $n \in N(x')$ → apply structure of n to x'

choose segment x' of x and neighbor $n \in N(x')$ → apply structure of n to x'

8.19 Scoring

Obviously, the reliability of the results obtained using the ROSETTA algorithm is crucially dependent on the score function. We will discuss a simple form of it.

In probabilistic terms, we would like to maximize the probability of the predicted structure, given the sequence, which, using Bayes theorem, is:

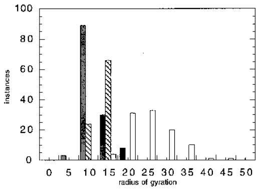
$$P(\mathit{structure} \mid \mathit{sequence}) = P(\mathit{structure}) \times \frac{P(\mathit{sequence} \mid \mathit{structure})}{P(\mathit{sequence})}.$$

In the comparison of different structures for the same sequence, $P(\mathit{sequence})$ is constant and can be ignored.

In the context of threading, it is simplest to assume $P(\mathit{structure}) = \frac{1}{(\text{number of structures})}$.

In the context of de novo folding, not all generated structures are equally likely to be proteins, for example, conformations with unpaired β strands are quite unlikely. The term $P(\mathit{structure})$ could be used to capture all features that distinguish protein structures from random chain configurations.

A simple choice is the following: set $P(\mathit{structure}) = 0$, for configurations in which atoms overlap. Otherwise, set $P(\mathit{structure}) = e^{-\kappa^2}$, where κ is the *radius of gyration* of the structure, which is small for compact structures.



(Simons et al. (1997))

Here we see the radius of gyration of obtained for structures generated using no scoring function (open bars), or κ^2 (hatched bars), and, for comparison, the values from structures selected from PDB (solid bars).

Now, let us consider the term $P(\mathit{sequence} \mid \mathit{structure})$. In a very simple model, the conditional probability of seeing a particular amino acid a at a particular position i in a sequence will depend on the *environment* $E(i)$ of the position in the structure. In this case:

$$P(\mathit{sequence} \mid \mathit{structure}) \approx \prod_i P(a_i \mid E_i).$$

Another approach assumes the independence of pairs rather than individual amino acids:

$$P(\mathit{sequence} \mid \mathit{structure}) \approx \prod_{i<j} P(a_i, a_j \mid r_{ij}),$$

where r_{ij} denotes the distance between residues i and j. Applying Bayes we get:

$$P(a_i, a_j \mid r_{ij}) = P(a_i, a_j) \times \frac{P(r_{ij} \mid a_i, a_j)}{P(r_{ij})}.$$

The first factor in the previous term is independent of structure. Putting these results together, we get:

$$P(\mathit{structure} \mid \mathit{sequence}) \approx e^{-\kappa^2} \times \prod_{i<j} \frac{P(r_{ij} \mid a_i, a_j)}{P(r_{ij})},$$

and a *scoring function* is given by $-\log$ of this expression.

A more detailed scoring function can be based on the following equation:

$$P(a_1, a_2, \ldots, a_n \mid \mathit{structure}) \approx$$

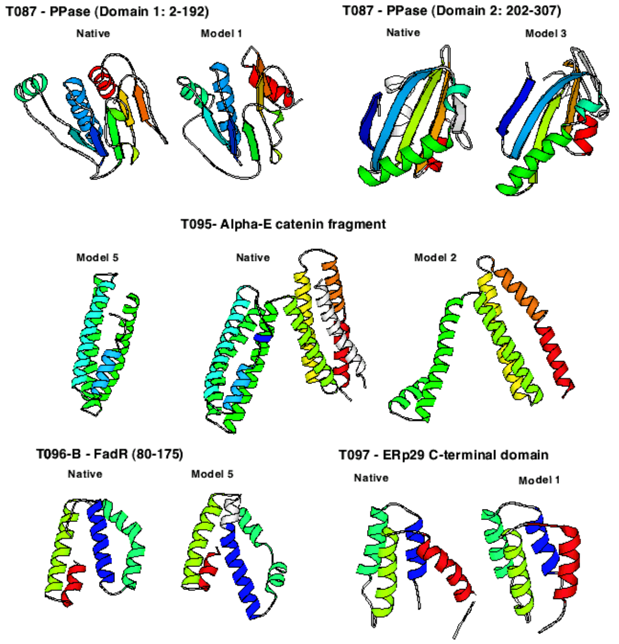
$$\prod_i P(a_i, \mid E_i) \times \frac{P(a_i, a_j \mid r_{ij}, E_i, E_j)}{P(a_i, \mid r_{ij}, E_i, E_j) P(a_j \mid r_{ij}, E_i, E_i)},$$

where E_i can represent a variety of features of the local structural environment around residue i. There are many more details that could be mentioned here, but we will skip them.

8.20 Performance

Every couple of years, a competition takes place called CASP (critical assessment of techniques for protein structure prediction, see `http://predictioncenter.llnl.gov/` for details) in which protein sequences are provided as target sequences for protein folding algorithms. These are sequences for which the three-dimensional structure has already been determined experimentally, but has not yet been published. Many groups working on protein folding submit models for the target sequences.

In the latest such competition, CASP4 in 2000, the ROSETTA method performed very well: Large segments were correctly predicted (more than 50 residues superimposed within an rmsd of 6.5 Angstrom) for 16 of 21 domains under 300 residues for which models were submitted. Models with the global fold largely correct were produced for several targets with new folds, and for several difficult fold recognition targets, the Rosetta models were more accurate than those produced with traditional recognition models.


T087 - PPase (Domain 1: 2-192)
Native
Model 1
T087 - PPase (Domain 2: 202-307)
Native
Model 3
T095- Alpha-E catenin fragment
Model 5
Native
Model 2
T096-B - FadR (80-175)
Native
Model 5
T097 - ERp29 C-terminal domain
Native
Model 1

(R. Bonneau, J. Tsai, I. Ruczinski, D. Chivian, C. Rohl, C.E.M. Strauss and D. Baker, Rosetta in CASP4: Progress in *ab initio* protein structure prediction, manuscript.)

8.21 Classification of protein structures

There are a number of databases that classify protein structures, each based on slightly different concepts. These include:

- SCOP: Structural Classification Of Proteins (`http://scop.mrc-lmb.cam.ac.uk/scop/`),
- CATH: Class, Architecture, Topology and Homologous superfamily (`http://www.biochem.ucl.ac.uk/bsm/cath_new/index.html`),
- FSSP: Fold Classification based on Structure-Structure alignment of Proteins (`http://www2.ebi.ac.uk/dali/fssp/`), and
- DALI domain dictionary. (`http://www2.ebi.ac.uk/dali/domain/`).

8.22 SCOP

This description of the SCOP database closely follows the original paper:

- A.G. Murzin, S.E. Brenner, T. Hubbard and C. Chothia, *SCOP: A structural classification of proteins database for the investigation of sequences and structures*, J. Mol. Biol. 247:536-540 (1995).

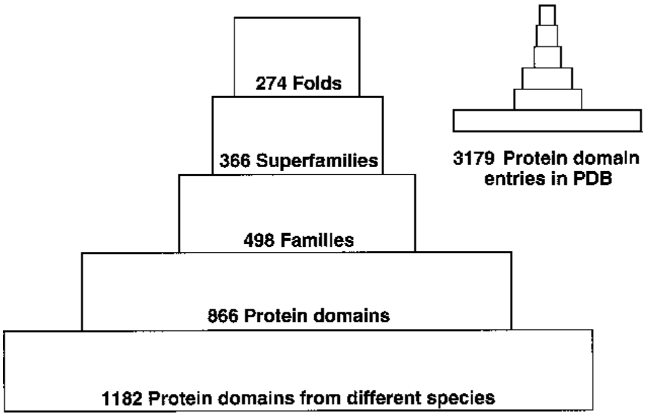
The goal of the SCOP database is to provide a detailed and comprehensive description of the *structural* and *evolutionary* relationships of the proteins of known structure. It also provides for each entry links to coordinates, images of the structure, interactive viewers, sequence data and literature references.

The *homology search* permits users to enter a sequence and obtain a list of any structures to which it has significant levels of sequence similarity.

The *key word search* finds matches from both the text of the SCOP database and the headers of Brookhaven Protein Database structure (PDB) files.

In SCOP, the unit of classification is usually the protein domain. Small and most medium size proteins are treated as a whole. The domains in large proteins are usually treated separately.

When SCOP was introduced in 1995, PDB contained 3179 domains. The SCOP classification of proteins is hierarchical:


274 Folds
366 Superfamilies
498 Families
866 Protein domains
1182 Protein domains from different species
3179 Protein domain entries in PDB

(Source: Murzin *et al.* (1995).)

In the following we list the different units of classification.

Family: Proteins are clustered together into families if they appear to have a common evolutionary origin, i.e. if they have residues identities of 30% or more, or they have lower identities, but their functions and structures are very similar, e.g. globins with sequence identities of 15%. Families are subclassed by species.

Superfamily: Families, whose proteins have low sequence identities, but whose structures and functional features suggest a common evolutionary origin, are placed together in superfamilies.

Common fold: Superfamilies and families are defined as having a common fold if their proteins have the same major secondary structures in the same arrangement with the same topological connections.

Class: For the convenience of users, the different folds are currently grouped into 11 classes:

1. All alpha proteins (151).
2. All beta proteins (111).
3. Alpha and beta proteins (α/β) (117). Mainly parallel beta sheets (β-α-β units).

4. Alpha and beta proteins ($\alpha + \beta$) (212). Mainly anti-parallel beta sheets (segregated alpha and beta regions).

5. Multi-domain proteins (α and β) (39). Folds consisting of two or more domains belonging to different classes.

6. Membrane and cell surface proteins and peptides (12). Does not include proteins in the immune system.

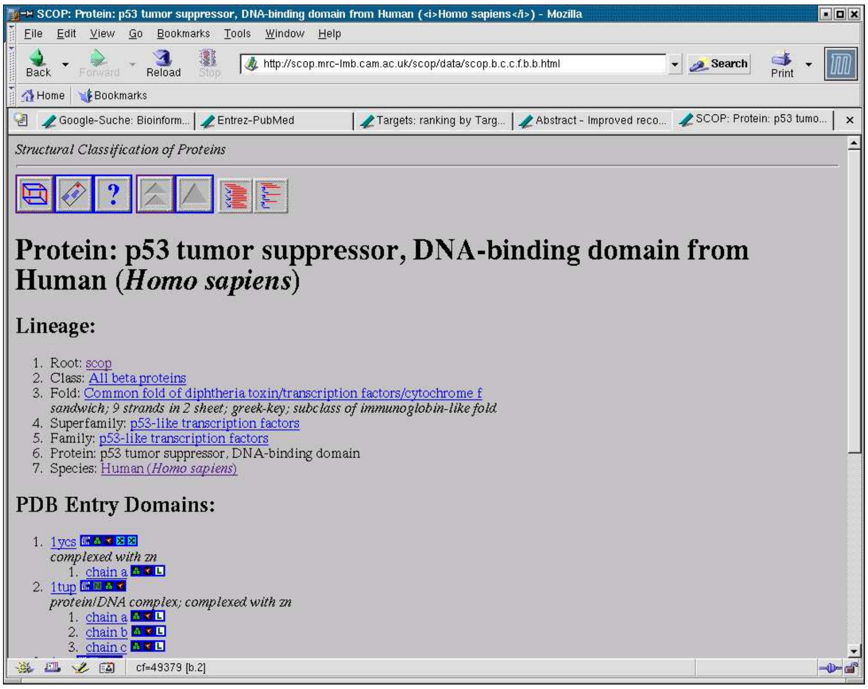
7. Small proteins (59). Usually dominated by metal ligand, heme, and/or disulfide bridges.

8. Coiled coil proteins (5). Not a true class.

9. Low resolution protein structures (17). Not a true class.

10. Peptides (95). Peptides and fragments. Not a true class.

11. Designed proteins (36). Experimental structures of proteins with essentially non-natural sequences. Not a true class.


SCOP: Protein: p53 tumor suppressor, DNA-binding domain from Human (<i>Homo sapiens</i>) - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://scop.mrc-lmb.cam.ac.uk/scop/data/scop.b.c.c.f.b.b.html
Search
Print
Home
Bookmarks
Google-Suche: Bioinform...
Entrez-PubMed
Targets: ranking by Targ...
Abstract - Improved reco...
SCOP: Protein: p53 tumo...
?
cf=49379 [b.2]

Structural Classification of Proteins

Protein: p53 tumor suppressor, DNA-binding domain from Human (*Homo sapiens*)

Lineage:

1. Root: scop
2. Class: All beta proteins
3. Fold: Common fold of diphtheria toxin/transcription factors/cytochrome f
   sandwich; 9 strands in 2 sheet; greek-key; subclass of immunoglobin-like fold
4. Superfamily: p53-like transcription factors
5. Family: p53-like transcription factors
6. Protein: p53 tumor suppressor, DNA-binding domain
7. Species: Human (*Homo sapiens*)

PDB Entry Domains:

1. 1ycs
   complexed with zn
   1. chain a
2. 1tup
   protein/DNA complex; complexed with zn
   1. chain a
   2. chain b
   3. chain c

8.23 CATH

CATH is a hierarchical classification of protein domain structures, which clusters proteins at four major levels, defined on the CATH website as follows:

- **Homologous superfamily:** This groups together protein domains which are thought to *share a common ancestor* and can therefore be described as homologous. Similarities are identified first by sequence comparisons and subsequently by structure comparison using SSAP. Structures are clustered into the same homologous superfamily if they satisfy one of the following criteria:
  - Sequence identity $\geq 35\%$, 60% of larger structure equivalent to smaller,
  - SSAP score ≥ 80.0 and sequence identity $\geq 20\%$, 60% of larger structure equivalent to smaller,

  - SSAP score ≥ 80.0, 60% of larger structure equivalent to smaller, and domains which have related functions.

- **Topology** or fold family: Structures are grouped into fold families depending on both the *overall shape and connectivity of the secondary structures.* This is done using the structure comparison algorithm SSAP. Parameters for clustering domains into the same fold family have been determined by empirical trials throughout the database. Structures which have a SSAP score of 70 and where at least 60% of the larger protein matches the smaller protein are assigned to the fold family. Some large families are subclassed using a higher score threshold.

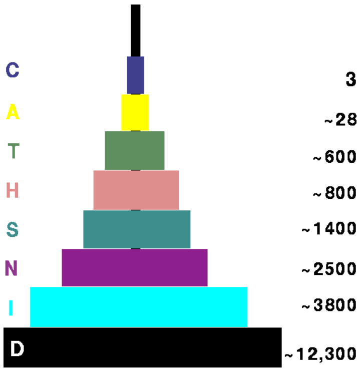
- **Architecture** describes the *overall shape of the domain structure* as determined by the orientations of the secondary structures but ignores the connectivity between the secondary structures. It is currently assigned manually using a simple description of the secondary structure arrangement e.g. barrel or 3-layer sandwich.

- **Class** is determined according to the *secondary structure composition* and packing within the structure. It can be assigned automatically for over 90% of the known structures. For the remainder, manual inspection is used and where necessary information from the literature taken into account. Three major classes are recognized;

  - mainly-alpha,
  - mainly-beta and
  - alpha-beta.

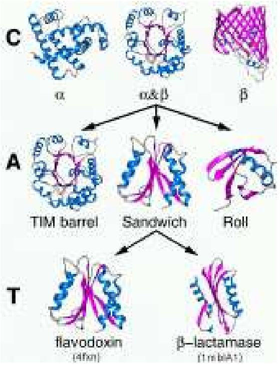
  The last class (alpha-beta) includes both alternating α/β structures and $\alpha+\beta$ structures, as originally defined by Levitt and Chothia (1976). A fourth class is also identified which contains protein domains which have low secondary structure content.

The following pyramid plot shows the number of groups identified at each level in the CATH database. Characters on the lefthand-side gives the CATH levels: Class; Architecture, Topology; Homologous superfamily; Sequence family - 35% sequence identity; Near-identical - 95% sequence identity; Identical - 100% sequence identity; Domain entry:



(http://wserv1.dl.ac.uk/CCP/CCP11/newsletter/vol2_3/orengo/)

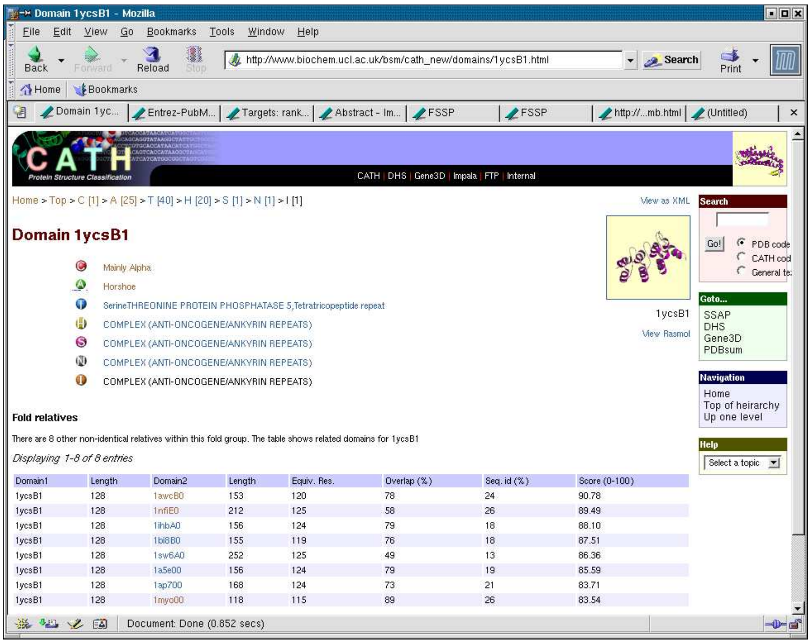
This figure illustrates the hierarchical nature of CATH:



(http://www.biochem.ucl.ac.uk/bsm/cath_new/cath_info.html)

References:

- Orengo, C.A., Michie, A.D., Jones, S., Jones, D.T., Swindells, M.B., and Thornton, J.M. *CATH- A Hierarchic Classification of Protein Domain Structures.* Structure. 5(8):1093-1108 (1997).
- Pearl, F.M.G, Lee, D., Bray, J.E, Sillitoe, I., Todd, A.E., Harrison, A.P., Thornton, J.M. and Orengo, C.A. (2000) Assigning genomic sequences to CATH Nucleic Acids Research. 28(1):277-282 (2000).



Home > Top > C [1] > A [25] > T [40] > H [20] > S [1] > N [1] > I [1]

Domain 1ycsB1

C Mainly Alpha
A Horshoe
T SerineTHREONINE PROTEIN PHOSPHATASE 5,Tetratricopeptide repeat
H COMPLEX (ANTI-ONCOGENE/ANKYRIN REPEATS)
S COMPLEX (ANTI-ONCOGENE/ANKYRIN REPEATS)
N COMPLEX (ANTI-ONCOGENE/ANKYRIN REPEATS)
I COMPLEX (ANTI-ONCOGENE/ANKYRIN REPEATS)

Fold relatives

There are 8 other non-identical relatives within this fold group. The table shows related domains for 1ycsB1

Displaying 1-8 of 8 entries

Domain1	Length	Domain2	Length	Equiv. Res.	Overlap (%)	Seq. id (%)	Score (0-100)
1ycsB1	128	1awcB0	153	120	78	24	90.78
1ycsB1	128	1nfiE0	212	125	58	26	89.49
1ycsB1	128	1ihbA0	156	124	79	18	88.10
1ycsB1	128	1bi8B0	155	119	76	18	87.51
1ycsB1	128	1sw6A0	252	125	49	13	86.36
1ycsB1	128	1a5e00	156	124	79	19	85.59
1ycsB1	128	1ap700	168	124	73	21	83.71
1ycsB1	128	1myo00	118	115	89	26	83.54

Document: Done (0.852 secs)

8.24 The FSSP database

The FSSP database includes all protein chains from the Protein Data Bank which are longer than 30 residues. The chains are divided into a *representative set* and *sequence homologs* of structures in the representative set.

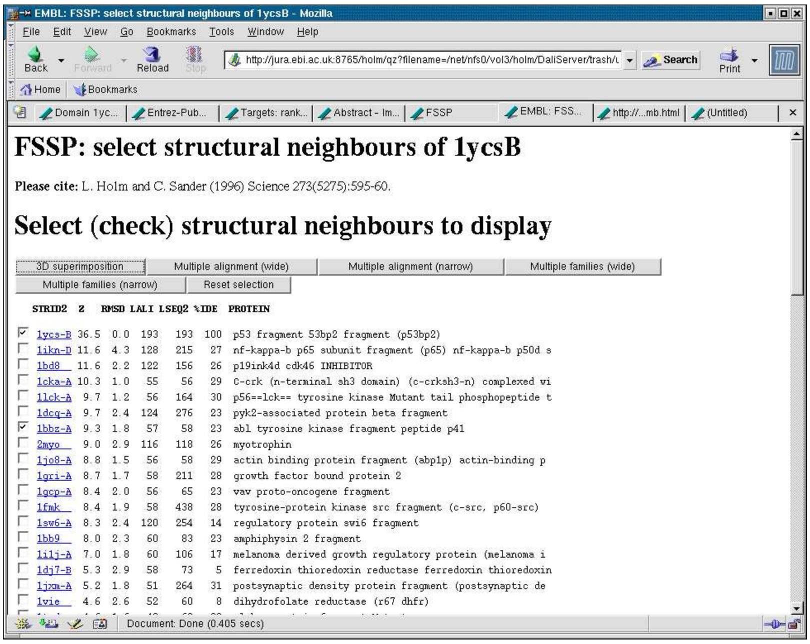
Sequence homologs have more than 25% sequence identity, and the representative set contains no pair of such sequence homologs. An all-against-all structure comparison is performed on the representative set.

The resulting alignments are reported in the FSSP entries for individual chains.

In addition, FSSP entries include the structure alignments of the search structure with its sequence homologs.

Reference:

- L. Holm and C. Sander, *Mapping the protein universe*, Science 273:595-602 (1996).


EMBL: FSSP: select structural neighbours of 1ycsB - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://jura.ebi.ac.uk:8765/holm/qz?filename=/net/nfs0/vol3/holm/DaliServer/trash/
Search
Print
Home
Bookmarks
Domain 1yc...
Entrez-Pub...
Targets: rank...
Abstract - Im...
FSSP
EMBL: FSS...
http://...mb.html
(Untitled)
3D superimposition
Multiple alignment (wide)
Multiple alignment (narrow)
Multiple families (wide)
Multiple families (narrow)
Reset selection
Document: Done (0.405 secs)

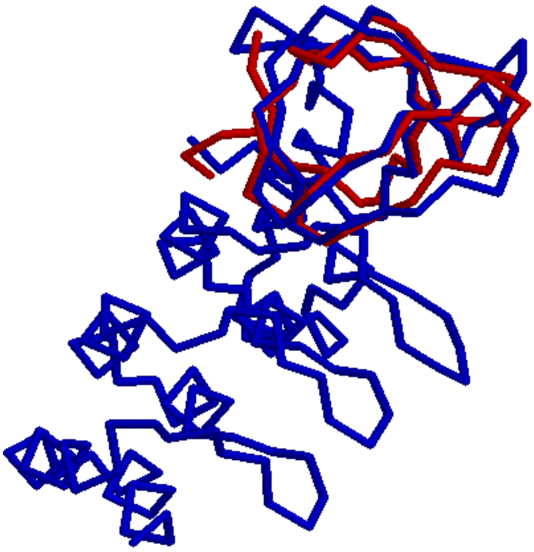
FSSP: select structural neighbours of 1ycsB

Please cite: L. Holm and C. Sander (1996) Science 273(5275):595-60.

Select (check) structural neighbours to display

	STRID2	Z	RMSD	LALI	LSEQ2	%IDE	PROTEIN
☑	1ycs-B	36.5	0.0	193	193	100	p53 fragment 53bp2 fragment (p53bp2)
□	1ikn-D	11.6	4.3	128	215	27	nf-kappa-b p65 subunit fragment (p65) nf-kappa-b p50d s
□	1bd8	11.6	2.2	122	156	26	p19ink4d cdk46 INHIBITOR
□	1cka-A	10.3	1.0	55	56	29	C-crk (n-terminal sh3 domain) (c-crksh3-n) complexed wi
□	1lck-A	9.7	1.2	56	164	30	p56==lck== tyrosine kinase Mutant tail phosphopeptide t
□	1dcq-A	9.7	2.4	124	276	23	pyk2-associated protein beta fragment
☑	1bbz-A	9.3	1.8	57	58	23	abl tyrosine kinase fragment peptide p41
□	2myo	9.0	2.9	116	118	26	myotrophin
□	1jo8-A	8.8	1.5	56	58	29	actin binding protein fragment (abp1p) actin-binding p
□	1gri-A	8.7	1.7	58	211	28	growth factor bound protein 2
□	1gcp-A	8.4	2.0	56	65	23	vav proto-oncogene fragment
□	1fmk	8.4	1.9	58	438	28	tyrosine-protein kinase src fragment (c-src, p60-src)
□	1sw6-A	8.3	2.4	120	254	14	regulatory protein swi6 fragment
□	1bb9	8.0	2.3	60	83	23	amphiphysin 2 fragment
□	1i1j-A	7.0	1.8	60	106	17	melanoma derived growth regulatory protein (melanoma i
□	1dj7-B	5.3	2.9	58	73	5	ferredoxin thioredoxin reductase ferredoxin thioredoxin
□	1jxm-A	5.2	1.8	51	264	31	postsynaptic density protein fragment (postsynaptic de
□	1vie	4.6	2.6	52	60	8	dihydrofolate reductase (r67 dhfr)

The structural alignment obtained from FSSP between 1ycs-B and 1bbz-A:



8.25 The DALI domain dictionary

In the "Dali domain dictionary", structural domains are delineated automatically. Each domain is assigned a *domain classification number DC_l_m_n_p* representing

- a fold space attractor region (l),
- a globular folding topology (m),
- a functional family (n) and
- a sequence family (p).

Based on the Dali website, the levels of the classification can be described as follows:

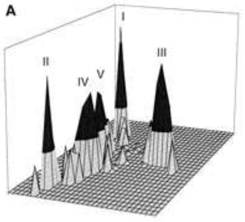
- **Sequence families:** The finest level of classification is a representative subset of the Protein Data Bank extracted using a 25% sequence identity threshold. All-against-all structure comparison are carried out within the set of representatives. Homologs are only shown aligned to their representative.

- **Functional families:** The next level of classification infers *plausible evolutionary relationships* from strong structural similarities which are accompanied by functional or sequence similarities. Functional families are branches of the fold dendrogram where all pairs have a high average neural network prediction for being homologous. The neural network weighs evidence coming from: overlapping sequence neighbors as detected by PSI-Blast, clusters of identically conserved functional residues, E.C. numbers, Swissprot keywords. The threshold for functional family unification was chosen empirically and is conservative; in some cases the automatic system finds insufficient numerical evidence to unify domains which are believed to be homologous by human experts.

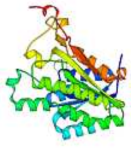
- **Fold type:** The next level of the classification is fold type. Fold types are defined as clusters of *structural neighbors in fold space* with average pairwise Z-scores (by Dali) above 2. The threshold has been chosen empirically and groups together structures which have topological similarity. Higher Z-scores corresponds to structures which agree more closely in architectural detail.

- **A fold space attractor region:** The highest level of the fold classification corresponds to secondary structure composition and super-secondary structural motifs. Five attractor regions in fold space have been identified. The fold space is partitioned so that each domain is assigned to one of attractors 1-5, which are represented by archetype structures, using a shortest-path criterion.

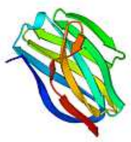
Density distribution of domains in fold space and attractors:



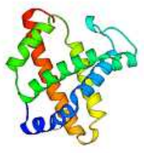
Five attractor regions in fold space have been identified, represented by archetype structures:



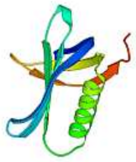
I
α/β



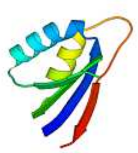
II
all-β



III
all-α



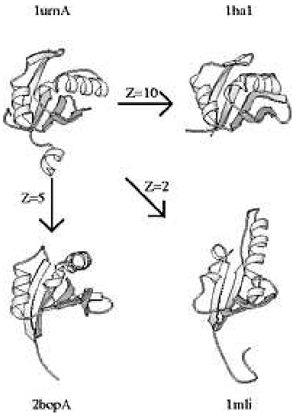
IV
antiparallel β-barrels



V
α-β meander

(Both figures: http://www.ebi.ac.uk/dali/domain/3.1beta/Help.html)

The fold types are defined via Z-scores. In the following example, some structural neighbors of 1urnA (top left) are displayed. Note that 1mli (bottom right) has the same fold type as 1urnA, even though there are shifts in the relative orientation of secondary structure elements.



(http://www.ebi.ac.uk/dali/domain/3.1beta/Help.html)

Additional to the five attractor regions in fold space, two further classes are defined. Domains which are not clearly closer to one attractor than another, are assigned to the mixed class 6. Currently, class 6 comprises about one sixth of the representative domain set. In the future, some of these may be assigned to emerging new attractors. Structures, which are disconnected from other structures (no connected path to any attractor), are assigned to class 7.

References:

- L. Holm, L. and C. Sander. *Dictionary of recurrent domains in protein structures*, Proteins, 33:88-96 (1998).
- L. Holm and C. Sander. *Mapping the protein universe*, Science, 273:595-603 (1996).

8.26 Summary of discussed classifications

SCOP	CATH	Dali dictionary
family common evolutionary origin	**sequence family** common evolutionary origin	**sequence family** common evolutionary origin
super family common evolutionary origin	**hom. super family** common evolutionary origin	**functional family** plausible common evolutionary origin
common fold same major secondary structure arrangement	**topology** same shape and connectivity of secondary structure	**fold type** structural neighbors in fold space
	architecture overall shape of domain structure	
class 11 classes of folds	**class** 3 classes of secondary structure composition and packing	**attractor region** 5 attractors in fold space, plus 2 additional classes

8.27 Structure comparison

Given two three-dimensional protein structures. There are a number of reasons why we would like to compare them, e.g.:

- for classification purposes, i.e. to determine whether they probably have a common evolutionary origin and thus belong in the same family etc.,
- to predict the function of one protein by determining structurally similar proteins of known function, or
- if one is the true structure and one is a predicted one, to measure how good the prediction is, e.g. in the CASP contests.

A number of different approaches exist, e.g. for alignments without indels, one can use rigid body superposition and distance map similarity, whereas in the general case, e.g. one can use double dynamic programming and contact map overlap.

8.28 Rigid body superposition

Given two structures x and y, each represented by a linear list $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_m)$, respectively, of three-dimensional coordinates, with the understanding that x_i is matched to y_i, for all i.

Naively, one might consider the following measure:

$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - y_i)^2}.$$

In practice, this is not useful, because it depends on the absolute locations of the two structures.

Let T denote any orientation- and distance preserving motion. The *root squared mean distance* (RSMD) between x and y is defined as:

$$RSMD(x,y) = \sqrt{\frac{1}{m}\min_T\sum_{i=1}^{m}(x_i - Ty_i)^2}.$$

For $RSMD$, we need to find the transformation T that minimizes the expression above. This is usually done in two steps: first, translate the center of mass of x and y to the origin. Then find the best rotation, e.g. by doing a step-wise search of rotational space.

8.29 Distance map comparison

Alternatively, if we only compare the *distances between pairs* of positions in x and y, then the resulting value will be independent of the absolute coordinates of the two structures:

$$RSMD_d(x,y) = \frac{1}{m}\sqrt{\min_T\sum_{i=1}^{m}\sum_{j=1}^{m}(d(x_i,x_j) - d(y_i,y_j))^2},$$

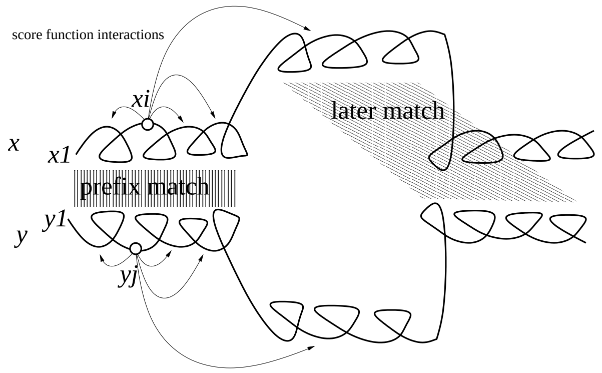
where $d(x_i, x_j)$ denotes the distance between the two points x_i and x_j.

There is a close relationship between $RSMD$ and $RSMD_d$, however $RSMD_d$ cannot distinguish between mirror images.

8.30 Structure alignment

Two *sequences* x and y can be aligned using dynamic programming, because the optimality of the alignment of two prefixes of x and y *does not depend* on how later positions of the two sequences are aligned.

This kind of independence generally does not hold for the alignment of structures, because the employed scoring functions usually take the matching of neighboring positions into account:



8.31 Double dynamic programming

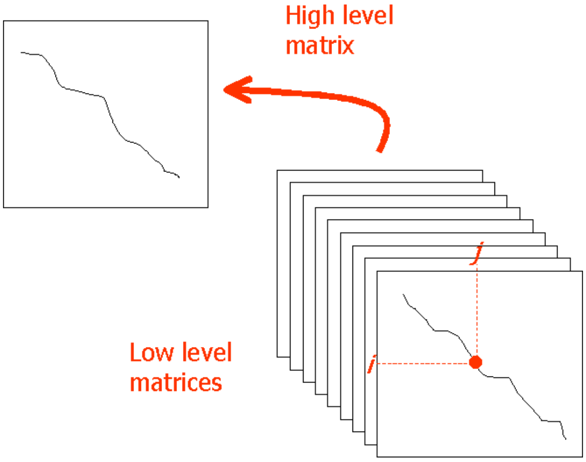
Double dynamic programming is a heuristic that attempts to extend the application of dynamic programming to the problem of aligning protein structures. It is the basis of the SSAP program, see C.O. Orengo and W.R. Taylor (1996) for details.

The following description of DDP is based on:
`http://www.ii.uib.no/~inge/talks/ismb-tutorial/`.

Given two folded sequences $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$. The main idea in double dynamic programming is to perform *two levels* of dynamic programming:

- Ordinary dynamic programming is performed on a *high level* scoring matrix R to obtain the final alignment.
- Each cell (i, j) of R should contain a value that expresses how likely it is that the pair (x_i, y_j) will be contained in an optimal alignment.
- For each cell (i, j), this value is determined by performing a *low level* dynamic program on a matrix ${}^{ij}R$, under the constraint that (x_i, y_j) must be part of the low-level alignment.
- The scores along each of these low-level alignments are accumulated in the high level scoring matrix R.

The high-level DP matrix R is obtained by computing a low-level matrix ${}^{ij}R$ for each cell (i, j) in R and then accumulating the values in R:



(http://www.ii.uib.no/~inge/talks/ismb-tutorial/)

8.32 The low-level matrix

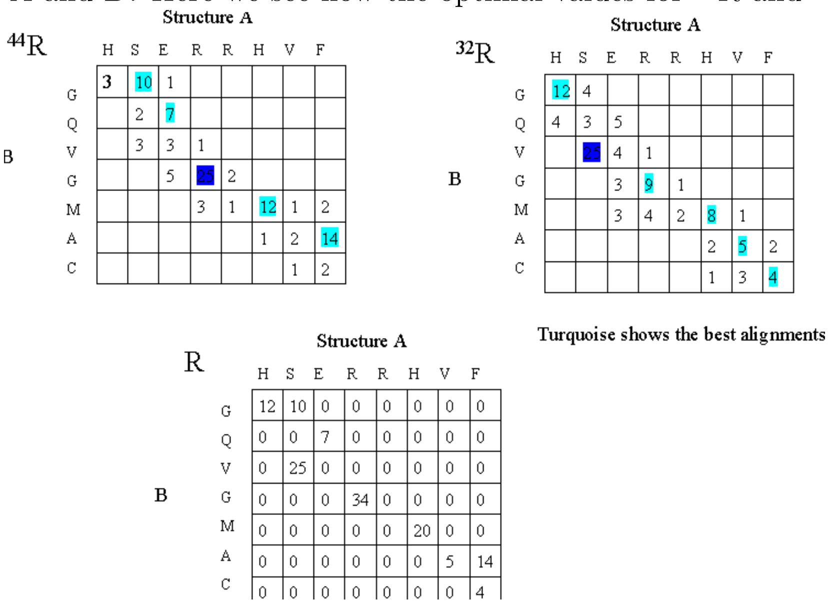
For each cell (i, j) in the high-level matrix R, we compute a matrix ${}^{ij}R$ such that ${}^{ij}R_{kl}$ is a score obtained for aligning x_k to y_l, under the constraint that x_i is aligned to y_j, using dynamic programming.

Then, given such a low-level matrix ${}^{ij}R$: For each cell (k, l) on the optimal path in ${}^{ij}R$ through (i, j), we increase the value of R_{kl} by ${}^{ij}R_{kl}$, if ${}^{ij}R_{kl}$ exceeds a pregiven threshold.

The overall aim is to give high scores to cells in R that occur in optimal paths for many of the low-level alignments.

Running a dynamic program using the values in the high-level matrix R produces the final alignment.

Given two folded sequences A and B. Here we see how the optimal values for ${}^{44}R$ and ${}^{32}R$ are added to the R matrix:



^{44}R

Structure A

B	H	S	E	R	R	H	V	F
G	3	10	1					
Q		2	7					
V		3	3	1				
G			5	25	2			
M				3	1	12	1	2
A						1	2	14
C							1	2

^{32}R

Structure A

B	H	S	E	R	R	H	V	F
G	12	4						
Q	4	3	5					
V		25	4	1				
G			3	9	1			
M			3	4	2	8	1	
A						2	5	2
C						1	3	4

Turquoise shows the best alignments

R

Structure A

B	H	S	E	R	R	H	V	F
G	12	10	0	0	0	0	0	0
Q	0	0	7	0	0	0	0	0
V	0	25	0	0	0	0	0	0
G	0	0	0	34	0	0	0	0
M	0	0	0	0	0	20	0	0
A	0	0	0	0	0	0	5	14
C	0	0	0	0	0	0	0	4

(http://www.ii.uib.no/~inge/talks/ismb-tutorial/)

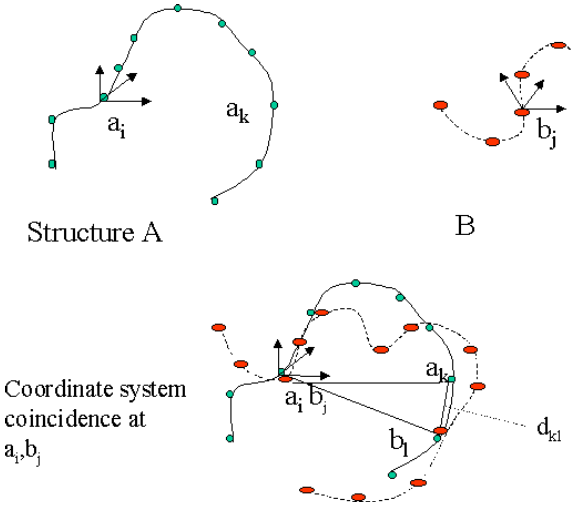
8.33 Scoring the low-level matrix

The value ${}^{ij}R_{kl}$ should capture how well x_k and y_l match up in three-dimensional space, under the assumption that x_i and y_j are superimposed upon each other.

To measure this,

- define two local reference systems for x_i and y_j, e.g. by using the C$_\alpha$ atoms of x_{i-1}, x_i, x_{i+1} and of y_{j-1}, y_j, y_{j+1}, respectively,
- transform the coordinates of the residues of x and y into the respective coordinate systems, and then
- compute a score for x_k and y_l based on the transformed coordinates. For example, simply use the distance between the two residues.

For sequences A and B, the simplest scoring scheme for scoring the low-level matrix ${}^{ij}R$ is obtained by superimposing residue a_i onto b_j and then setting ${}^{ij}R_{kl}$ using the distance between residues a_k and b_l:



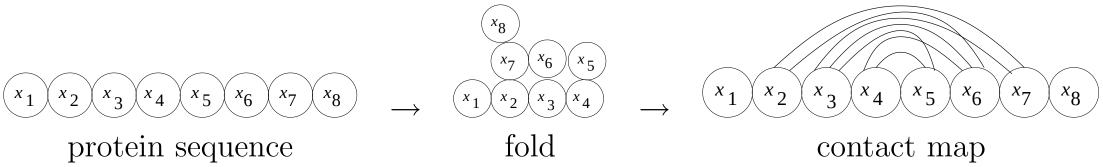
(http://www.ii.uib.no/~inge/talks/ismb-tutorial/)

8.34 Contact map overlap

Given a protein sequence $x = (x_1, \ldots, x_m)$ and assume we are given coordinates for each residue (e.g., for their C$_\beta$ atoms).

We say that two residues x_i and x_j are *neighbors*, or are *in contact*, if the distance between the two corresponding C$_\beta$ atoms is less than some given threshold, e.g. 7 Angstrom.

Here is an illustration:



The contact map of a folded protein is represented as a $0-1$, symmetric $m \times m$ matrix M, with $M_{ij} = 1 \Leftrightarrow x_i$ and x_j are neighbors.

It is convenient to think of M as the adjacency matrix of a graph $G = (V, E)$ with node set $V = \{1, 2, \ldots, m\}$.

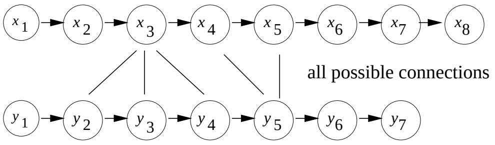
8.35 The return of the alignment graph...

Given two folded sequences x and y. Loosely speaking, the goal is to find an alignment of the two sequences that preserves as much of the contact map as possible. The idea here is that similar sites in a structure will make similar contacts.

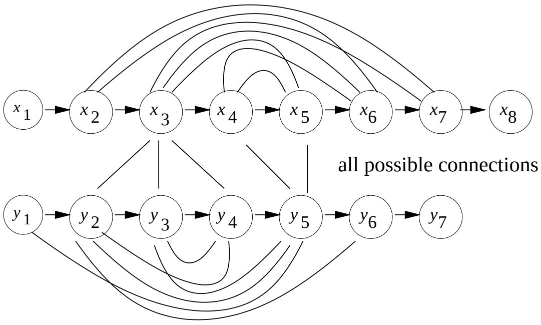
Recall the definition of an *alignment graph* (Chapter 3):

Given two sequences $x = (x_1, \dots, x_m)$ and $y = (y_1, \dots, y_n)$. The *alignment graph* $G = (V, E)$ is the complete bipartite graph on nodes $V = \{v_1, \dots, v_m\} \cup \{w_1, \dots, w_n\}$ such that v_i represents residue x_i and w_j represents residue y_j. Moreover, every node in the first subset is connected to every node in the second subset, and vice versa. We obtain the *extended* alignment graph by adding a set H of directed edges (v_i, v_{i+1}) and (w_j, w_{j+1}) for all $i = 1, \dots, m-1$ and $j = 1, \dots, n-1$.

Here is the extended alignment graph $G = (V, E, H)$ for two sequences x and y:



Then we add the set I of given contact edges for both sequences to obtain a *structural alignment* graph $G = (V, E, H, I)$:



8.36 Structural trace

Note that this graph is very similar to the one defined for the problem of RNA secondary structure alignment, where interactions were defined via base-pairing. In that problem, we formulated an integer linear program (ILP) for computing an optimal weight alignment subject to the constraint that any one node can participate in at most one interaction.

Given the structural alignment graph $G = (V, E, H, I)$ for two folded sequences x and y. Let $T \subseteq H$ be a trace, i.e. a non-crossing subset of H. Two contact edges p and q (that do not belong to the same sequence) are called *matched*, if the two alignment edges e_l and e_r joining the two left respectively right nodes of p and q are contained in the trace T.

Let B be the set of all matched contacted edges. We call (T, B) a *structural trace.*

8.37 Scoring a structural trace

Let (T, B) be a structural trace. As for conventional traces, each edge $e \in T$ is given a weight $\omega(e)$ which is simply the score for aligning the two corresponding symbols.

Any contact match $\{i_p, i_q, e_l, e_r\}$ is completely specified by the two edges e_l and e_r and so we can denote it by m_{lr} and assign a weight $\omega(l,r)$ to it. Let M denote the set of all interaction matches:

$$M = \{m_{lr} \mid m_{lr} = \{i_p, i_q, e_l, e_r\} \text{ is an interaction match in } G\}.$$

We define the *score of a structural trace* as:

$$S((T,B)) = \sum_{e \in T} \omega(e) + \sum_{\substack{i_p, i_q \in B, e_l, e_r \in T \\ \{i_p, i_q, e_l, e_r\} \in M}} \omega(l,r).$$

We can find the maximal scoring trace by solving an ILP.

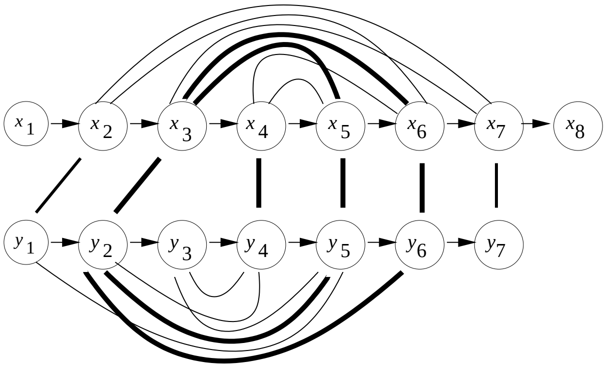
8.38 Maximal scoring structural trace

Given a structural alignment graph $G = (V, E, H, I)$, the score ω_i for realizing an edge $e_i \in E$ and the score ω_{lr} for realizing a contact match m_{lr}.

By dropping the constraint that any given node can interact with at most one other node, from the ILP formulated for RNA structure alignment we obtain an ILP for maximizing the score:

$$\begin{array}{rll} \text{Maximize} & \sum_{e_i \in E} \omega_i x_i + \sum_{m_{ij} \in M} \omega_{ij} x_{ij}, & \\ \text{subject to} & \sum_{e_i \in C \cap E} x_i \leq |C \cap E| - 1, & \text{for all critical} \\ & & \text{mixed cycles } C, \\ & x_{ij} \leq x_i \text{ and } x_{ij} \leq x_j, & \text{for all variables, and} \\ & x_i, x_{ij} \in \{0,1\} & \text{for all variables.} \end{array}$$

In this example, two matches are realized:



By appropriate application of the "Lagrangian relaxation method", such structural ILPs can be solved very efficiently. For details, see:

- Giuseppe Lancia, *Mathematical programming approachs for computational biology problems*, to appear in: Scuola CIRO.