Kevin Wallimann wkevin@student.ethz.ch 16.02.2014

Computational Biology

Disclaimer: Many parts of this document were written by Sander Schaffner. This document may be edited and republished, but only by keeping the names of all previous authors. No warranty is given on the correctness of the contents.

1 Mutation and Markovian Model

- Training data: Estimate parameters
- Testing data: Evaluate and validate models and parameters
- Training data and testing data must be independent

1.1 Markov model

- State change independent on former states.
- State changes defined by a **transition matrix**.
- Finite number of states.
- If a Markov model converges, the final state $M^{\infty}p =: f$ is called **natural frequency**
- For natural frequencies, the **detailed balance** condition holds: $f_i M_{ij} = f_j M_{ji}$

1.1.1 Continuous Markov Models

Discrete step: $\underline{p} \rightarrow M\underline{p}$

Continuous step: $\underline{p} \rightarrow \underline{p} + Q \cdot \underline{p} \cdot \Delta t \;\Rightarrow\; \underline{p}(t) = \underline{p}(0)e^{Qt}$

1.1.1.1 Relation between M and exp(Q)

Is the continuous step equivalent to the discrete step?

Set $\Delta t = 1 \Rightarrow p(t)|_{t=1} = e^{Qt}|_{t=1} = e^{Q} \Rightarrow M = e^{Q}$

So if we choose $\boldsymbol{M} = \boldsymbol{e^{Q}}$ then the two models are equivalent.

How can we achieve smaller time steps than 1?

Set $\Delta t = \frac{1}{p} \Rightarrow M = e^{\frac{Q}{p}}$ for p=2,3,...

1.2 Transition matrix M

1.2.1 Properties

- Its Eigenvalues are
  - $\lambda_1 = 1$
  - $|\lambda_i| < 1, i = 2, \dots, n$
- M is non-symmetric
- $0 \leq M_{ij} \leq 1$
- If M is the transition matrix for AA's it is usually diagonally dominant.

1.2.2 Estimation of M

Assume a sequence s1 has evolved to a sequence s2 without any indels, i.e. we have two sequences of the same length. We compare now the pairs of AA. Whenever encounter a match we count this into our count matrix C

$$C(AA_1, AA_2) = C(AA_1, AA_2) + 1$$

This matrix C should correspond to a multiple of Mutation matrix (distance d) times the natural appearance of AA for a sample of length k

$$C \approx M^d A$$

We can rewrite A as the multiplication of the frequency matrix times the total number of AA

$$A = FN, \qquad N = n_A + n_C + \dots + n_W \in \mathbb{N}, \qquad F = \begin{pmatrix} f_A & & \\ & \ddots & \\ & & f_W \end{pmatrix}$$

This leads to

$$C \approx M^d F N$$

Therefore we can derive M^d (d unknown)

$$M^d \approx C F^{-1} \frac{1}{N}$$

1.2.2.1 Computing M from M^d

$$M = \left(M^d\right)^{\frac{1}{d^*}}$$

For M we have a condition: $\sum f_i M_{ii} = 0.99$ (1-PAM unit)

Therefore, we can compute M and d^* by a numerical method (e.g. binary search).

1.2.2.2 Constructing a symmetric matrix

The counting rule

$$C(AA_1, AA_2) = C(AA_1, AA_2) + 1$$

Implies AA_2 has evolved to AA_1. We can remove this implication by counting equally to $C(AA_1, AA_2)$ and $C(AA_2, AA_1)$, thus creating a symmetric counting matrix.

$$C(AA_1, AA_2) = C(AA_1, AA_2) + \frac{1}{2}$$
$$C(AA_2, AA_1) = C(AA_2, AA_1) + \frac{1}{2}$$

1.2.2.3 Flaws

- From one evolutionary tree, maximal one sequence pair can be used for the estimation, otherwise some parts of the tree might be oversampled.
- Since the organisms are not the same, the mutation matrices vary:

$$\exists M_b, M_v, \dots$$

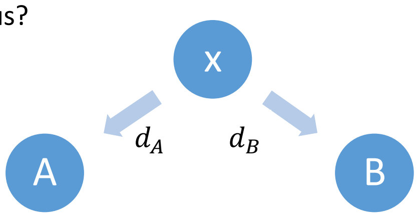
- Therefore, a maximum-likelihood estimation of all parameters needs to be done.
    - DB of pairs
    - Initialize parameters (find them numerically, directly impossible)
    - Align all sequences at their optimal distance
- ⇒Joint-score likelihood

1.3 Reversible evolution (Homology probability)

Definition: **Homologous** position are position which have a common ancestor.

What is the probability that A & B are homologous?

- A & B are known, x is unknown
- Mutation is a Markovian process
- M is the transition matrix
- d_A and d_B are integer (distances)



x can be any AA. Let's assume a frequency of each AA: $f_x = frequency\ of\ AA\ x$

$$\begin{aligned}
\Pr\{A, B\ are\ homologous\} &= \sum_x f_x \left(\Pr\left\{x \rightarrow A, d_A\right\} \cdot \Pr\left\{x \rightarrow B, d_B\right\}\right) \\
&= \sum_x f_x (M^{d_A})_{Ax} (M^{d_B})_{Bx} \\
&= f_B \sum_x (M^{d_A})_{Ax} (M^{d_B})_{xB} \\
&= f_B (M^{d_A} M^{d_B})_{AB} \\
&= f_B (M^{d_A+d_B})_{AB} = f_A (M^{d_A+d_B})_{BA}
\end{aligned}$$

We used the detailed balance condition in step 3 and 5. Step 4 is the matrix multiplication.

The probability if A and B are homologous is only dependent of the sum of d_A and d_B. Therefore, we can't know whether A evolved to B or B evolved to A. This is called **reversible evolution**.

1.3.1 Reversible evolution for multiple positions

$$\prod_{i=1}^{n} f_{B_i} (M^d)_{A_i B_i}$$

For multiple positions, the probability of all positions A_i and B_i to be homologous is just the product of the individual probabilities.

1.4 Score

The score is a measure for the relevance of the homology probability. The score relates the homology probability with the probability of two sequences being independent.

$$Score = \log\left(\frac{Pr\{homologous\}}{Pr\{independent\}}\right) = \log \frac{\prod_{i=1}^{n} f_{B_i} (M^d)_{A_i B_i}}{\prod_{i=1}^{n} f_{B_i} f_{A_i}} = \sum_{i=1}^{n} \log\left(\frac{(M^d)_{A_i B_i}}{f_{A_i}}\right)$$

1.5 Dayhoff Matrix D (Scoring matrix)

The Dayhoff matrix is a scoring matrix for all AA (or any other objects). It is defined as follows:

$$D_{AB} = 10 \log_{10}\left(\frac{(M^{250})_{AB}}{f_A}\right)$$

- For AA's the dimension of D is 20 x 20.
- D is symmetric, because of the detailed balance condition.
- In general, the diagonal of D is positive and largest, off-diagonals are mostly negative.
- AA with similar values in D have similar properties and are exchangeable.

- Here, the so called **PAM-Distance** is 250. (PAM = Point accepted mutations)
- M^d is also called d-PAM-Matrix

1.5.1 PAM-Unit

- 1-PAM, i.e. M^1 describes a mutation that effects an average change of 1%.
- 2-PAM is not exactly a 2% change, because some might have changed back. I.e. mutation is not equal to change!
- 250-PAM means that every position was mutated 2.5 times.
- The only strict relation between amount of mutation and change can be established for the 1-PAM

$$\sum_i f_i M_{ii} = 0.99$$

2 Sequence alignment / Dynamic Programming

2.1 Global alignment / Dynamic Programming algorithm (Needleman-Wunsch)

The **forward phase** of the dynamic programming algorithm for sequence alignment returns the alignment cost. The algorithm is listed below.

```
Function forwardPhase(x: sequence1, y:sequence2,
Del:indelCostFunction, match: matchCostFunction)
for i=0..m
      E(i,0)=0
for j=0..n
      E(0,j)=0
for i=1..m
      for j=1..n
            E(i,j)= min{E(i-1,j)    + Del(x[i]),
                        E(i,j-1)    + Del(y[j]),
                        E(i-1,j-1)  + match(x[i],y[j])}
return E(m,n)
```

x, y are the sequences to match, while Del is a cost function for indels, and match is a matching cost function (usually a lookup on the Dayhoff matrix)

The **backward phase** returns the effective alignment.

```
Function backwardPhase(x: sequence1, y:sequence2, Del:
indelCostFunction, match: matchCostFunction, E: DP-Table)
x_aln, y_aln;
while i>1 && j>1
      if E(i,j) – match(x[i],y[j]) == E(i-1, j-1)
            x_aln += x[i]
            y_aln += y[j]
            i-=1; j-=1;
      elseif E(i,j) – Del(y[j]) == E(i,j-1)
            x_aln += x[i]
            y_aln += '_'
            j-=1
      elseif E(i,j) – Del(x[j]) == E(i-1,j)
            x_aln += '_'
            y_aln += y[j]
            i-=1
```

```
endwhile
return x_aln, y_aln
```

An **example** tableau is given below

	_	E	N	O	U	G	H
_	0	1	2	3	4	5	6
G	1	1	2	3	4	4	5
E	2	1	2	3	4	5	5
N	3	2	1	2	3	4	5
U	4	3	2	2	2	3	4
G	5	4	3	3	3	2	3

The final alignment is:

```
_ENOUGH
 ||  ||
GEN_UG_
```

The deletion cost is 1, the matching cost is 0 for matches and 1 for non-matches.

Note that the zeroth row and column are not shown since they are only zeros.

2.1.1 Interpretation of backward phase

3	4
2	3

- Going from 3 to 2 means: Align H with a deletion (space character) at the Genug string.
- Going from 3 to 4 would mean: Align G with a deletion (space character) at the Enough string.
- Going from 3 to 3 would mean: Align H and G.

2.1.2 Symmetry of Dynamic Programming

Doing DP on A^T is the same as doing DP on A from bottom-right to top-left.

2.1.3 Complexity

Time: $O(n^2)$

Space: $O(n^2)$

2.1.4 Matching cost or Dayhoff matrix as scoring matrix

For multiple mutating positions, the Dayhoff matrix is defined as

$$\sum_i \mathrm{D}_{\mathrm{a_i b_i}} = 10\log_{10}\left(\frac{\prod_i \Pr\{\mathrm{a_i \text{ and } b_i \text{ have a common ancestor}}\}}{\prod_i \Pr\{\mathrm{a_i \text{ and } b_i \text{ are random}}\}}\right)$$

Under the assumption that the positions evolve independently, this product represents the joint probability of the two sequences evolving from a common ancestor. Hence if we maximize the score using Dayhoff matrices, we find the alignment which maximizes the probability of coming from a common ancestor, i.e. we have a **maximum likelihood alignment**.

Recall that maximum likelihood estimates are

- unbiased (with infinite information they give the correct results),
- asymptotically normal, and
- most powerful (in the sense of having the smallest variance of the estimator).

2.1.5 Indel cost

2.1.5.1 *Affine indels*

Motivation: Opening a gap (the first indel) should cost more than increasing an already open gap.

$$Cost(gap\ of\ length\ k) = c_0 + (k-1)c_1$$

- c_0 is the cost for opening a gap
- c_1 is the cost for continuing a gap ($c_0 > c_1$)

Note: This is just an intuitive idea! No science!!!

Affine indels increase the complexity of DP to $O(n^3)$, since the newest sub problem could score better with a k gap in any of the sequences 1 or 2.

To fix the problem we save 3 values per filed. Two for open gaps ($x_{1,ij}$& $x_{3,ij}$) and one for the actual value ($x_{2,ij}$).

(i) $x_{1,ij}$ (gap from column): take max out of $x_{2,i-1j} + c_0$ (opening new gap) or $x_{1,i-1j} + c_1$ (continuing a gap)
(ii) $x_{3,ij}$ (gap from row): take max out of $x_{2,ij-1} + c_0$ (opening new gap) or $x_{3,ij-1} + c_1$ (continuing a gap)
(iii) $x_{2,ij}$ (new value): take max out of the diagonal value or the two gap values computed in (i) and (ii): $\{x_{2,i-1j-1}, x_{1,ij}, x_{3,ij}\}$

This reduces the computational costs to $O(N^2)$ in space and time.

2.1.5.2 *Logarithmic indels*

Idea: Compute indel cost by score.

Example: Let AKPQLLT and ANILS be two sequences. They may have a common ancestor for which their alignment would be

```
AKPQLLT
AN__ILS
```

Using the definition of a score and including the indels into that score, we get a relevant expression of the indel cost.

$$Score = 10\log_{10}\frac{\Pr(homologous)}{\Pr(random)}$$
$$= 10\log_{10}\left(\frac{f_A M_{AA}}{f_A f_A}\cdot\frac{f_K M_{NK}}{f_K f_N}\cdot\frac{\Pr\big(indel(2)\big)\cdot f_P\cdot f_Q}{f_P f_Q}\cdot\frac{f_L M_{IL}}{f_L f_I}\cdot\frac{f_L M_{LL}}{f_L f_L}\cdot\frac{f_T M_{ST}}{f_T f_S}\right)$$

$$= D_{AA} + D_{NK} + \underbrace{10log_{10}\Pr(indel(2))}_{Cost(indel(2))} + D_{IL} + D_{LL} + D_{ST}$$

In general,

$$Cost(indel(k)) = 10log_{10}\Pr(indel(k))$$

$\Pr(indel(k))$ can be approximated by trusted alignments.

The above cost function can still be converted into the affine indel model by setting

$$\Pr(indel(k)) = e^{c_0-c_1}(e^{c_1})^k$$

In reality, this is not observed but Zipf's distribution.

2.1.5.3 *Estimate* probability *of an indel with length k using Zipf's Distribution*

A sequence following Zipf's distribution is defined by

$$p_i \propto \frac{1}{i^q}, q \in \mathbb{R}$$

For indels of length k, it can be observed that

$$\Pr(indel(k)) \propto \frac{1}{k^{1.7}}$$

For the cost function it follows that

$$Cost(indel(k)) = c_0 + c_1 \log_{10}(k) - "logarithmic\ deletion\ cost"$$

The Zipfian distribution can be justified by the probability of returning to the origin in k steps (random walk). For different dimensions, this probabilities are

$$1D: \propto \frac{1}{\sqrt{k}}$$
$$2D: \propto \frac{1}{k}$$
$$3D: \propto \frac{1}{k^{1.5}}$$

A random walk returning to the origin corresponds to a protein making a loop making itself closed. Therefore, such a protein is likely to be accepted for an indel.

2.2 Local alignment (Smith-Waterman)

Sometimes we are interested in finding the subsequences of the input sequences which align at the highest score. In other words we are ready to ignore any deletions which happen at the beginning or end of any of the two sequences. This is called local alignment or the Smith–Waterman algorithm. There are good biological reasons for using this type of alignment, in particular when we want to search a portion of a sequence which could be immersed in another. Local alignments can be computed with scores arising from Dayhoff matrices (similarity scores). The mechanism for computing a local alignment is quite simple:

(i) Compute the cost matrix as in Needleman–Wunsch (Dynamic programming with Dayhoff matrix).
(ii) Replace any negative entry (while computing it) with zero.

(iii) To start backtracking, choose the highest entry in the whole matrix. (This position will mark the end of the alignment.)

(iv) Backtrack up to the upper left corner or up to a zero, whichever comes first. (This position will mark the beginning of the alignment.)

How to get the best local alignment?

1. Store negative values and the overall maximum
2. Redo (start from end) DP from max until you get a negative value

2.3 Cost-Free End alignment (CFE)

A third variant of alignments is called cost-free end alignment (CFE).

- In CFE we do not penalize for a single insertion/deletion (indel) at the beginning or at the end of the sequences.
- If we penalized for end-indels in both sequences, this would be a global alignment.
- If we do not penalize for any end-indel then this would be a local alignment.
- Since CFE penalizes only for one end-indel on each side, it can be viewed as an intermediate algorithm between global and local.

This is computed easily with a small variation of the global alignment:

(i) Make the top row and leftmost column entries all zero.

(ii) Compute the cost matrix as in Needleman–Wunsch (Dynamic programming with Dayhoff matrix).

(iii) Choose the largest value in the bottom row or rightmost column to start the backtracking. (This position will mark the end of the alignment.)

(iv) Backtrack up to the top row or leftmost column. (This will mark the beginning of the alignment.)

2.4 Database search

We want to search **s** (sequence) in **d** (database). s is in most cases only a small part of d (s can be a interesting sequence, where d can be a whole genome ($10^6 - 10^8, human\ genome: 3 * 10^9$)).

If d doesn't fit into cache, a modified Dynamic programming algorithm is needed:

(i) d is the top row and s the right column → We get an dim(s) x dim(d) matrix

(ii) top row all zero → We want to start anywhere in the database d

(iii) search for maximum in bottom row → This is the end of the best alignment of s in d

(iv) compute column by column

2.4.1 Practical issues

1. A dim(s) x dim(d) matrix will be too huge. Since we compute column by column we only store the biggest end-value of a column while going through all columns. Then we have to store of course the latest column for the next one. But in the end we only need one value for the max (with its position) and a 2 x dim(s) array to save the latest and actual column.
2. After 1. we have the maximal value stored. From there we compute the reverse, that's how we find the best matching sequence s' in d.

2.5 Hirschberg's algorithm

Hirschberg's algorithm applies the Divide and Conquer Strategy to the Needleman-Wunsch algorithm. Hirschberg's algorithm needs $O(nm)$ time, but only $O(\min(n, m))$ space and is thus suited for database searches.

Observe

1. One can compute the optimal alignment score by only storing the current and previous row of the Needleman-Wunsch score matrix
2. If (Z,W) = NW(X,Y) is the optimal alignment of (X,Y) and X=XL+XR is an arbitrary partition of X, there exists a partition YL + YR of Y such that NW(X,Y) = NW(XL,YL) + NW(XR,YR)

```
function Hirschberg(X,Y)
   Z = ""
   W = ""
   if length(X) == 0 or length(Y) == 0
     if length(X) == 0
       for i=1 to length(Y)
         Z = Z + '-'
         W = W + $Y_i$
       end
     else if length(Y) == 0
       for i=1 to length(X)
         Z = Z + $X_i$
         W = W + '-'
       end
     end
   else if length(X) == 1 or length(Y) == 1
     (Z,W) = NeedlemanWunsch(X,Y)
   else
     xlen = length(X)
     xmid = length(X)/2
     ylen = length(Y)

     ScoreL = NWScore($X_{1:xmid}$, Y)
     ScoreR = NWScore(rev($X_{xmid+1:xlen}$), rev(Y))
     ymid = PartitionY(ScoreL, ScoreR)

     (Z,W) = Hirschberg($X_{1:xmid}$, $Y_{1:ymid}$) + Hirschberg($X_{xmid+1:xlen}$,
$Y_{ymid+1:ylen}$)
   end
   return (Z,W)


function NWScore(X,Y)
   Score(0,0) = 0
   for j=1 to length(Y)
     Score(0,j) = Score(0,j-1) + Ins($Y_j$)
   for i=1 to length(X)
     Score(i,0) = Score(i-1,0) + Del($X_i$)
     for j=1 to length(Y)
       scoreSub = Score(i-1,j-1) + Sub($X_i$, $Y_j$)
       scoreDel = Score(i-1,j) + Del($X_i$)
```

```
        scoreIns = Score(i,j-1) + Ins(Yj)
        Score(i,j) = max(scoreSub, scoreDel, scoreIns)
      end
    end
    for j=0 to length(Y)
      LastLine(j) = Score(length(X),j)
    return LastLine

function PartitionY(ScoreL, ScoreR)
    return arg max ScoreL + rev(ScoreR)
```

From: http://en.wikipedia.org/wiki/Hirschberg%27s_algorithm

2.6 Maximum likelihood

3 Evaluation of quality of matching

3.1 Entropy measure

Entropy is defined as

$$H(p) = -\sum_i p_i \log_2 p_i$$

p_i are the frequencies of the symbols in a code. The entropy yields the average number of bits with which each state can be encoded.

Low entropy ➔ high compressibility ➔ low information

For AA's the entropy is

$$H(f) = -\sum_i^{20} f_i \log_2 f_i \approx 4$$

- Therefore, repetitions are likely to happen.
- Repetitions are bad for alignment algorithms, as scores are multiplied and artificially increased.

3.2 Significance measure by random shuffling

A method to test the significance of an alignment is by shuffling the positions of one sequence and align the shuffled sequences with the other sequences.

The following Darwin-Code shows the procedure

```
CreateDayMatrices():
S1 := rand(protein(200)):
S2 := mutate(s1,100,ZipfGaps):
St := stat('score'):
Hist := CreateArray(1..100):
to 100 do
      al := Align(s1,shuffle(s2),Global):
      st + al[Score]:
      i := round(a[Score]):
      Hist[i] := Hist[i]+1:
```

```
od:
print(st);
View(Histogram(Hist));
```

3.3 Gumbel distribution

Scores of randomly shuffled sequences follow the Gumbel distribution. Its cumulative is defined by

$$F(x;\alpha,\beta) = \exp\left(-exp\left(\frac{\alpha - x}{\beta}\right)\right)$$
$$\sigma = \frac{\beta\pi}{\sqrt{6}}$$
$$\mu = \alpha + \gamma\beta$$

σ is the standard deviation and μ is the mean. γ is the Euler-Mascheroni constant ($\gamma \approx 0.5772156$)

To check the quality of an alignment, α and β need to be estimated by calculating σ and μ from the scores of the shuffled sequences.

The score in question x is then checked against a significance level φ (e.g. $\varphi = 0.05$)

$$P(x \geq c|H_0) = 1 - F(x;\alpha,\beta) \leq \varphi$$

H_0 is the Null-hypothesis, stating that the alignment is by chance. If $P \leq \varphi$ then the Null-hypothesis is rejected, i.e. the alignment is not by chance.

3.4 E-value

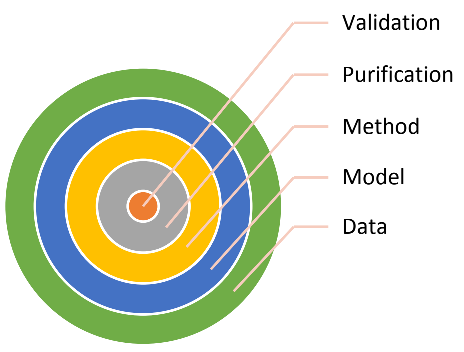
From the BLAST website
http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=FAQ

$$E = \frac{n \cdot m}{2^{S'}}$$

N = number of residues (AA's, nucleotides, etc.), m = length of sequence, S' = normalized score

- The Expect value (E) is a parameter that describes the number of hits one can "expect" to see by chance when searching a database of a particular size.
- For example, an E value of 1 assigned to a hit can be interpreted as one might expect to see 1 match with a similar score simply by chance.
- It decreases exponentially as the Score (S) of the match increases.

4 Phylogenetic trees



Algorithms to construct a phylogenetic usually follow the CHE procedure

1. **Construct** an initial tree (Method)
2. Use **heuristics** to search for better trees (Purification)
3. **Evaluate** and continue using heuristics until it no longer improves the score (Validation)

This procedure is justified, since the constructor yields only approximate solutions, because tree construction is known to be a NP-hard problem, thus all algorithms only yield approximate solutions.

Rooted phylogenetic trees have the following key numbers:

- $n = \# \, of \, species$
- n leaves
- $2n - 3$ branches $\rightarrow$ $n - 3$ internal branches
- $\frac{n(n-1)}{2}$ data points (One data point for every pair of species)

4.1 Data

Data source	Is there an ortho / paralogy problem?	Amount of available data	Additional pros & cons
Mitochondrial proteins / DNA	No	Small amount, only available in higher eukaryotes	
Ribosomal proteins / RNA	No	Very small amount	
Proteins, AA level	Yes	Plenty	- variable evolution rates
Proteins, DNA level (CodonPAM)	Yes	Plenty	- variable evolution rates + more precise for shorter evolution
Proteins, DNA level (SynPAM)	Yes	Plenty	+ more precise for shorter evolution
Non-coding DNA	Yes	Largest amount	- variable evolution rates
Indels	Yes		+ rather unique + mostly non-reversible + low frequency + => long distances trackable - requires quality MSA
Ordering of genes in genomes	Yes	Small amount	+ very unique - short evolution

4.1.1 Considerations on Data

- *The ortholog/paralog problem*: For a tree we have to find the orthologs! This is the only way of measuring distance.
- *Time vs amount of mutation*: is influenced by
  - variable reproduction rate
  - selection pressure (e.g. color of skin against enemies, ...)
  - quality of reproduction of DNA
- *Indels*: For a mutation to happen we have different probabilities with respect to AA, DNA, etc.
  - AA: $1 \rightarrow 19$
  - DNA: $1 \rightarrow 3$
  - Indel: $1 \rightarrow \binom{n}{2} \sim \frac{n^2}{2}$

  Indels are therefore quite rare (=unique) and trackable over long evolution distances.

- *Genome-Reordering*: Genomes of two species like human and chimp can for example simply differ in one place due to reversed reading.

4.2 Model

Model	Description
Markov	State change independent on former states, etc.
Extensions to Markov	Gamma correction for variable rates, corrections for invariants sites, etc.
Occam's razor principle (parsimony)	We assume that evolution proceeded in a way that needs the fewest changes possible to evolve from one sequence to another. Changes are meant in terms of changed positions (and not some fancy mutation operations)
Double cut and join	Assume that rearrangements always happen with DCJ operations. E.g. 1 transposition = 2 DCJ, 1 inversion = 1 DCJ, 1 Fusion = 1 DCJ
Transposons	Relatively short pieces of DNA have the property of self-replication, a non-reversible operation

4.3 Method

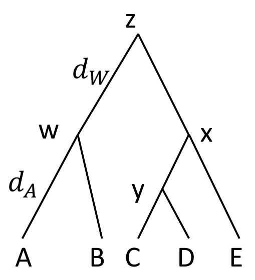
Tree building boils down to finding the right **topology** and the correct **branch lengths**.

There are rooted and unrooted trees.

	# internal nodes (at n leaves)	**# branches (at n leaves)**
Rooted tree	n-2	2n-3
Unrooted tree	n-1	2n-2

- Most methods produce unrooted trees.
- There are 2n-3 possibilities to convert an unrooted tree to a rooted tree (by declaring the root being on a specific branch)
- An unrooted tree can be converted to a rooted tree with an outgroup. An outgroup is related, but outside our set. Therefore, the root is at the position where the outgroup connects to our set. The outgroup and our set can be connected using an unrooted tree building algorithm.

4.3.1 Maximum Likelihood



As an example, we will compute the likelihood for a given tree using DNA sequences, but these procedures can be used for other characters as well. This example is borrowed from Felsenstein (1981).

We have a set of aligned sequences with m sites (MSA = Multiple Sequence Alignment). To compute the probability of a tree, we must have a phylogeny with branch lengths and an evolutionary model giving the probabilities of change along the tree. This model will allow us to compute the transition probabilities $P_{ij}(d_x)$, the probability that state j will exist at the end of a branch of length d_x where x is a node, if the state at the start of the branch is i. There are two central assumptions:

(1) Evolution at different sites on the tree is independent (and identically distributed).
(2) Evolution in different lineages is independent. This gives us the Markov property down trees (cf. pedigrees).

The first assumption allows us to express the probability as a product, with one term for each site:

$$L = \Pr(D|T) = \prod_{i=1}^{m} \Pr\left(D^{(i)}\middle|T\right)$$

Where $D^{(i)}$ is the data at the i-th site and $T = (d_A, \ldots, d_z)$. Therefore, we need to know how to compute the probability at a single site. If we have the tree and the data at a site, the probability is the sum, over all possible nucleotides that may have existed at the interior nodes of the tree, of the probabilities of each scenario.

$$\Pr\left(D^{(i)}\middle|T\right) = \sum_x \sum_y \sum_z \sum_w \Pr(A, B, C, D, E, x, y, z, w|T)$$

Based on the Markov property (2) above, we can decompose the probability on the right side of the equation into a product:

$$\begin{aligned}&\Pr(A, B, C, D, E, x, y, z, w|T)\\ &= \Pr(z)\Pr(w|z, d_w)\Pr(A|w, d_A)\Pr(B|w, d_B)\Pr(x|z, d_x)\Pr(E|x, d_E)\Pr(y|x, d_y)\Pr(C|y, d_C)\Pr(D|y, d_D)\end{aligned}$$

In the notation of the course this looks then like

$$\Pr\left(D^{(i)}\middle|T\right) = \sum_x \sum_y \sum_z \sum_w f_z \Pr\{z \to w, d_w\}\Pr\{w \to A, d_A\} \ldots$$

If we assume that evolution has been proceeding for a long time according to the model of substitution that we are using, then Pr(z) is the equilibrium probability of base z under that model.

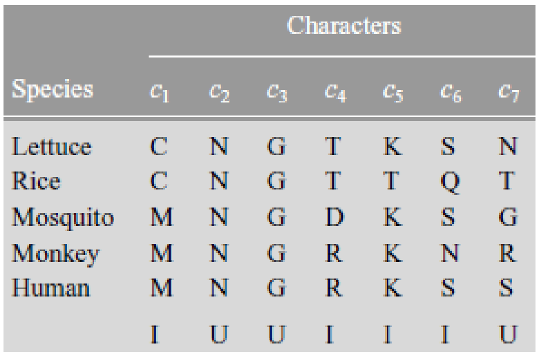
However, there are still a large number of terms in the previous equation, such that for each site we sum $4^4 = 256$ terms. The number of terms will rise exponentially with the number of species. On a tree with n species, there are $n - 1$ interior nodes, each of which can have one of 4 states. If we need 4^{n-1} terms, then for 10 taxa, this would be 262,144 terms!

4.3.2 Character-based (Parsimony)

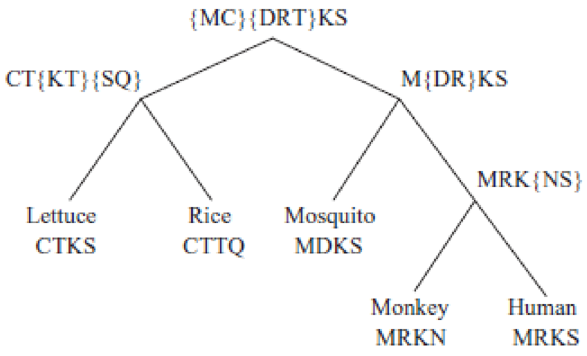
4.3.2.1 Characters

- A character is a heritable trait possessed by an organism. (E.g. eye color, vertebrate or not, DNA at position x, etc.)
- Characters can be derived from observable properties or can be derived from molecular sequences (amino acids of a protein or bases of DNA).
- For DNA and amino acids, a **multiple sequence alignment** (MSA) is normally computed between homologous sequences and each position (column of the MSA) is considered a character.
- A character has a **finite number of discrete states**. All state transitions are equally probable.
- Characters can be **informative or uninformative**. Characters, which are the same for all species is uninformative.

4.3.2.2 Union intersection algorithm



	Characters						
Species	c_1	c_2	c_3	c_4	c_5	c_6	c_7
Lettuce	C	N	G	T	K	S	N
Rice	C	N	G	T	T	Q	T
Mosquito	M	N	G	D	K	S	G
Monkey	M	N	G	R	K	N	R
Human	M	N	G	R	K	S	S
	I	U	U	I	I	I	U



The leaves are given from the informative characters of the MSA.

Forward phase (computing minimal number of changes)

From bottom to top, at each node, do

1. For each position, compute the intersection of both children's characters in that position.
2. If the intersection is non-empty, store the intersection in the node. (e.g. T & T → T)
3. If the intersection is empty, store the union of both children's characters in the node. (e.g. {T,G} & A → {T,G,A} and count++

```
cost = 0;
for each internal node i#(bottom up)
      for each character c
            tmp = intersect(child1[c], child2[c]);
            if(tmp == NULL)
                  cost++;
                  i[c] = union(child1[c], child2[c]);
            else
                  i[c] = tmp;
            endif;
      endfor;
endfor;
```

Backward phase (compute assignment)

1. Choose a root
2. Choose the same value for every child unless it is not in the set of possible assignments. (Whenever this happen we "lose" one edge. These losses sum up to the count we already have from the forward phase)

4.3.2.3 General union intersection algorithm (General parsimony)

In general parsimony, the cost of an empty intersection is not uniform anymore, but variable.

Chlorophyll		
from	to	
	true	false
true	0	1
false	3	0

Vertebrate		
from	to	
	true	false
true	0	4
false	4	0

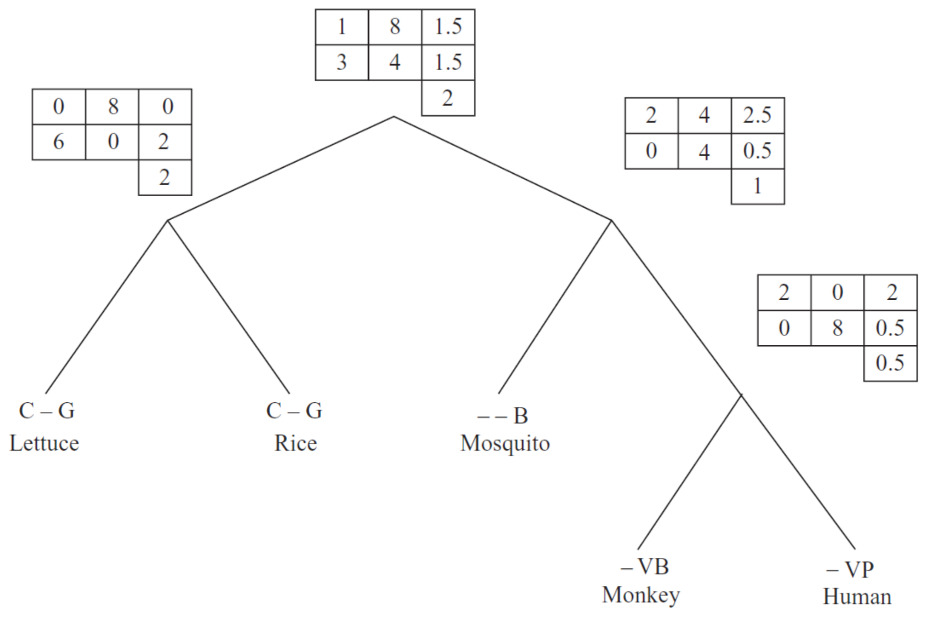
Color			
from	to		
	green	brown	pink
green	0	1	1
brown	1	0	$\frac{1}{2}$
pink	1	$\frac{1}{2}$	0

The cost (forward phase) is then computed as follows:

```
for each internal node i do #(bottom up)
     for each character c do
        for each state s of c do
          cost[i,c,s]:= sum( min( transit[c,s -> t] + cost[d,c,t]) )
        od;
     od;
od;
```

- The min is computed over all states t. The sum is computed for all descendants d of i.
- cost[i,c,s] represents the tables of optimal costs at each internal node i for character c having the state s.
- transit[c,s->t] represents the cost of character c going from state s to state t where typically the cost of not changing, transit(c,s->s), is zero.
- The above loop represents the forward phase of the dynamic programming. Once this is done we can, if desired, compute the internal node assignments. At the root we select for each character the state with the lowest cost.

An example for the cost tables is given below.



4.3.2.4 Remarks

- The position of the root is arbitrary and does not affect the minimal number of changes required for the tree for simple parsimony (all changes having the same weight) or if the transition cost matrices are symmetric.
- The resulting optimal tree is therefore **unrooted**.
- If the transition cost matrix is not symmetric, the resulting optimal tree is rooted

4.3.3 Distance

4.3.3.1 Computing a tree from a distance matrix

From measurements (like: Markovian, DCJ or Parsimony (# changes)) we have a distance Matrix D, where D_{AB} denotes the distance between two species A and B.

A given tree has branches, and the branch lengths between each node can be adjusted (with least squares) in such a way that the summed branch lengths between leaf A and leaf B coincide with the distance between A and B.

The tree construction problem is defined as finding a tree T (topology and branch lengths) such that $\|D - T\|$ is minimized. With the least square method this minimization for the distance becomes

$$\|error\|_{dist}^2 = \|D - T\|^2 = \sum_{i,j} (D_{ij} - T_{ij})^2$$

and for the variance

$$\|error\|_{var}^2 = \sum_{i,j} \frac{(D_{ij} - T_{ij})^2}{\sigma_{ij}^2}$$

For a fixed topology, finding the optimal branch lengths becomes a problem of linear least squares

$$\begin{array}{rcl} T_{AB} = & a + b & \cong dist[A,B] = D_{AB} \\ T_{AC} = & a + e + f + c & \cong dist[A,C] = D_{AC} \\ & \vdots & \\ T_{CD} = & c + d & \cong dist[C,D] = D_{CD} \end{array}$$

Finding the right topology is a very hard problem that is not discussed here.

There are some problems which can arise with the least square method:

4.3.3.1.1 Underdetermined system

The data could be such that some edges are undetermined. That way we are not able to place the root.

4.3.3.1.2 Incomplete data

If sequences are **too distant**, the alignment may be of very poor quality and it is better to discard the particular distance estimate (by setting its variance to infinite). In this case, the system may also become **underdetermined** or we may get a **disconnected tree**.

4.3.3.1.3 Negative edge length

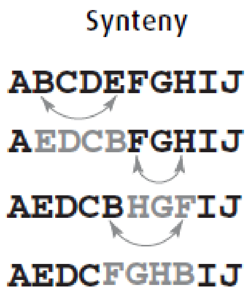
In general the least square method can give us **negative lengths**, which is **nonsense** in our model. Having negative edges is a clear indication that the data is not fitting a tree model, i.e. **poor data**. This could be due to the wrong topology, to very short sequences being aligned or very distant species being compared.

4.3.3.2 Distance measures

Properties of distance measure for tree building	
Identity	$d(A,A) = 0$
Symmetry	$d(A,B) = d(B,A)$
Additivity	$d(A,B) + d(B,C) = d(A,C)$

Note that additivity is a stronger form than the triangle inequality, which is mandatory for any distance measure.

Distance measures for tree building	Description	Additive?	Other info
Percentage of changed position (cf. Parsimony)	Percentage of changed positions and indels	No	+ Easy - Treats all changes equally
Synteny (cf. Double cut and join)	Measure of the relative order of genes	Almost	Only non-additive if a synteny operation exactly undoes another synteny operation. Very unlikely, therefore additive for practical purposes
PAM Distance (cf. Markovian model)	Measure of amino acid mutation	Yes	



Three inversions in a gene sequence

4.3.4 Comparison of methods

Method	Input	Pros / Facts	Cons
ML	MSA, Transition probabilities on tree	• Unbiased • Asymptotically normal • Most powerful (smallest variance in estimator)	• Needs MSA • Exponential growth of number of terms to evaluate. Exponential in number of species
Parsimony	MSA	Explicit computation with differences (characters)	
Distance	Distance matrix	Differences reduced to a single number	

All methods return the branch lengths of a given tree. None of the methods computes a topology of the tree.

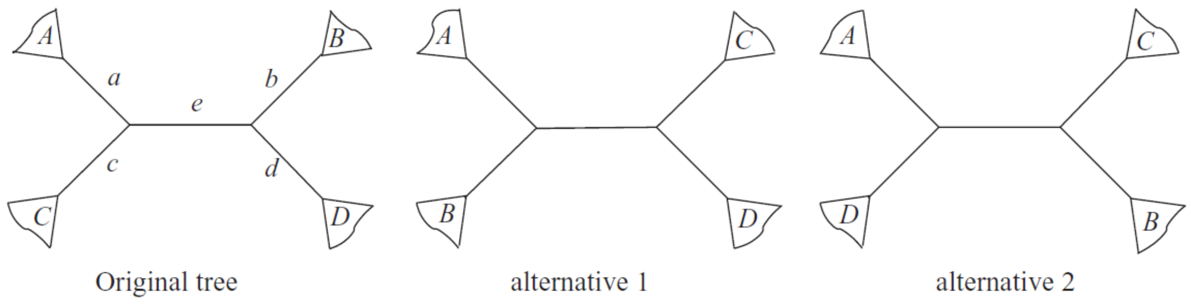
4.4 Purification

Since the constructor returns only an approximate solution, there might always be an even better solution. We explore neighbouring trees for two arguments

1. Saving computation: Neighbouring trees require only small changes on the tree, thus we expect to reuse previous results.
2. If our tree is a good tree, we should examine nearby trees whether they are better rather than any other distant tree.

4.4.1 4-optim

Inspects all alternative permutations that can be done for each **internal edge**, i.e. an edge which is not attached to a leaf. An internal edge induces a quartet. A topology is different if and only if the distance between two species changes.



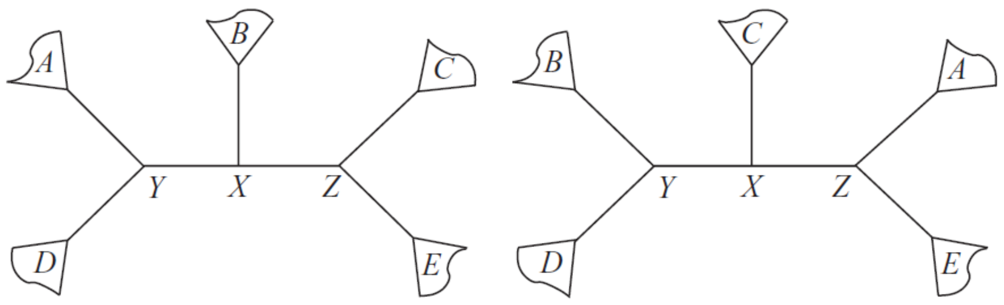
Quartets

```
curr_error := edge error(tree);
improved := true;
while improved do
      improved := false;
      for e in [internal edges not attached to leaves] do
            for t in [alternative topologies(e)] do
                  new error := edge error(t); %local least-squares fit
                  if new error < curr_error then
                        improved := true;
                        curr_error := new error;
                        tree := t
                  fi
            od
      od
od;
```

- There are **n-3 internal edges**
- Per quartet, there are 2 alternative topologies. (Fix one branch path, e.g. AB and do all possible permutations without order on this branch path, i.e. AB, AC and AD)
- **2(n-3)** quartets need to be looped through

4.4.2 5-optim

Inspects all alternative permutations that can be done for each **internal node**, with a configuration of five subtrees.

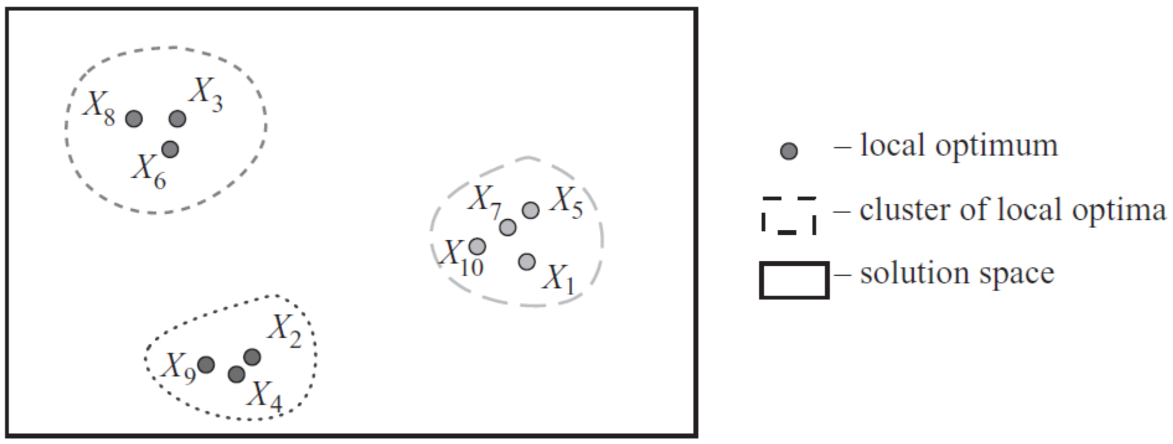


Quintets

- X must have two adjacent nodes Y and Z.
- Per quintet, there are 14 alternative topologies. (Fix one branch path, e.g. AB and do all possible permutations without order on this branch path, i.e. AB, AC, AD and AE. This gives 5 quintet. Now multiply by three, since X,Y and Z can be permutated as well.)

4.4.3 More than neighbours: Randomness

The solution space of the problem might consist of several clusters



If we restrict ourselves only to the neighbours it might be impossible to reach other clusters. Therefore, we would like to insert some portion of randomness into our solution. If we have are able to order the results by their quality, we will select the ith choice with probability $r^{i-1}(1-r)$, a geometrical distribution.

Choice	**Probability**	**4-optim, 5-optim**
1st	$1-r$	Best topology
2nd	$r(1-r)$	2nd best topology
3rd	$r^2(1-r)$	3rd best topology

4.5 Validation

4.5.1 Bootstrapping

In statistics it is a method of resampling. For our purposes it is about building a tree out of a subset of the data. By doing this many times we get different trees.

We can sample the subset of data in two ways:

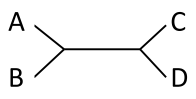
- With replacement: leads to duplications. 69% of the data are unique data points. We can cover all data points.

- Without replacement: no duplicates, but only $\frac{2}{3}$ of the data is used.

On a MSA (multi sequence alignment) bootstrapping means taking columns out of a MSA matrix.

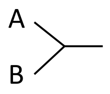
4.5.1.1 Example

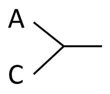
We have data which represents:



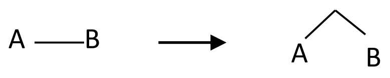
Bootstrapping supports this in that example with 98%. What does this mean?

When we perform bootstrapping, then out of 100 cases we get for example the following statistic

98 times


and 2 times


Bootstrapping is most used for branch-support. We want to split species such that we know where they evolved from:



4.5.2 Synthetic Evolution

4.6 Tree building

A tree with n species has $2n-3$ branches. The total number of possible unrooted trees is

$$T_n = 1 \times 3 \times 5 \times \ldots \times (2n-5) = (2n-5)!!$$

This is a large number of possibilities:

- $T_{10} \sim 3 \cdot 10^3$
- T_{20} possible, but very large
- T_{50} is bigger than the # particles in the universe

For rooted trees it is even a bit worse, since we can place the root at any internal branch:

$$R_n = (2n-3)T_n = (2n-3)!!$$

Most tree building is NP-complete. There are many different trees and also many methods to build them.

4.6.1 Ultrametric trees

Ultrametric because the distance from any leaf to the root is constant:

$$root - leaf_i = const = d$$

This makes the algorithm very simple:

1. Pick the two min distances and build first part
2. Take next min and add it to the growing tree
3. While there are distances which aren't used, repeat 2.

However, this needs the data to be ultrametric which the case is rarely.

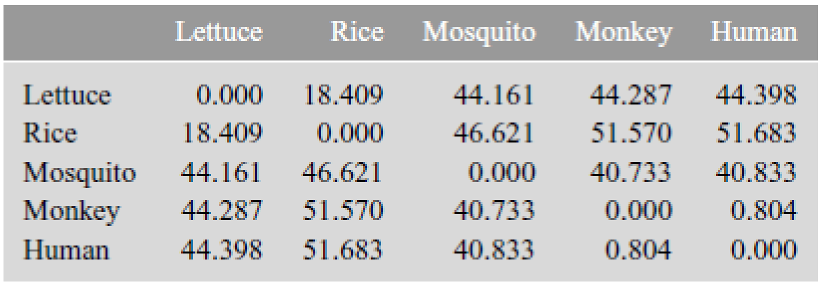
4.6.2 UPGMA

UPGMA stands for unweighted pair group method with arithmetic mean. It has a greedy heuristic which joins at each step the two closest leaves or subtrees which are no already joined.

The algorithm is:

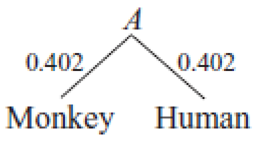
1. Take the minimal element D_{MN} (the shortest distance) of the distance matrix D
2. Connect M and N as children of a new parent A.
3. Set $d_{NA} = d_{MA} = \frac{D_{MN}}{2}$ (Ultrametricity condition). The children are equally distant from their parent.
4. Remove the columns and rows with N and M from D, and add a column and a row for A. A becomes the new leaf for the rest of the algorithm (Note: this subtree under A won't change!)
5. Calculate new distances from the remaining species to A of step 1. This by simple average of distances to N and M
$$d(X_i, A) = \frac{d(M, X_i) + d(N, X_i)}{2} - \frac{d(M, N)}{2}$$
6. Repeat step 1. & 2. with the updated matrix D till all species are in the tree.

4.6.2.1 Example

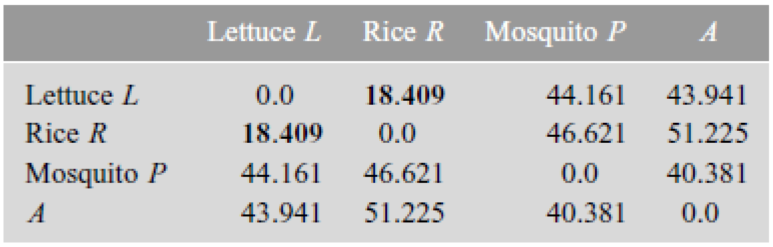


	Lettuce	Rice	Mosquito	Monkey	Human
Lettuce	0.000	18.409	44.161	44.287	44.398
Rice	18.409	0.000	46.621	51.570	51.683
Mosquito	44.161	46.621	0.000	40.733	40.833
Monkey	44.287	51.570	40.733	0.000	0.804
Human	44.398	51.683	40.833	0.804	0.000

1. The shortest distance is between Monkey and Human (0.804). Build initial subtree



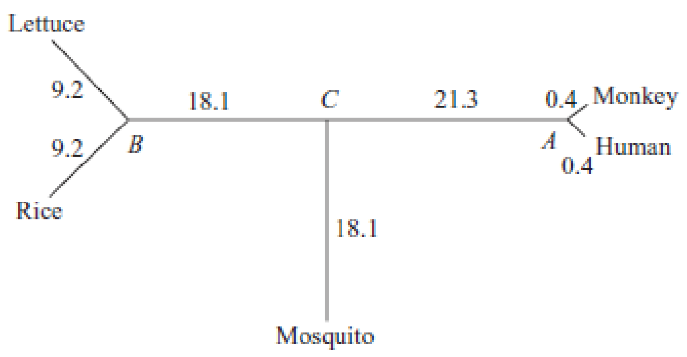
2. Calculate new distances
$$d(L, A) = \frac{d(M, L) + d(N, L)}{2} - \frac{d(M, N)}{2} = 43.941$$
And also with R and P. We get



	Lettuce L	Rice R	Mosquito P	A
Lettuce L	0.0	**18.409**	44.161	43.941
Rice R	**18.409**	0.0	46.621	51.225
Mosquito P	44.161	46.621	0.0	40.381
A	43.941	51.225	40.381	0.0

3. Repeat step 1. & 2.

We end up with the unrooted tree



4.6.3 WPGMA

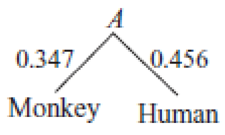
Stands for weighted pair group method with arithmetic mean. WPGMA has two improvements over UPGMA. The first one is to optimize the construction of the subtrees by choosing the branches of new subtrees according to the distances to the rest of the leaves instead of arbitrarily halving it. The second one is to use the inverses of the variances as weights.

1. We want to set d_i and d_j according to the distances to the rest of the leaves, denoted as y.
$$\forall y: d_i - d_j \approx d_{iy} - d_{jy}$$
Because there are too many unknowns, we take the average
$$d_i - d_j = \underset{y}{Avg}(d_{iy} - d_{jy})$$
With $d_i + d_j = d_{ij}$, this leads to
$$d_i = \frac{d_{ij} + Avg(\dots)}{2} \; \& \; d_j = \frac{d_{ij} - Avg(\dots)}{2}$$
Note: With UPGMA, $d_i - d_j = 0$
2. To compute the average, we can take the inverses of the variances as weights.
In general, a weighted average is computed as
$$\frac{\sum \omega_i X_i}{\sum \omega_i}$$
Recall that
$$\sigma^2(\alpha X + \beta Y) = \alpha^2\sigma^2(X) + \beta^2\sigma^2(Y)$$
Therefore, the variance of the average is
$$\sigma^2(Avg) = \frac{\sum w_i^2\, \sigma^2(X_i)}{(\sum w_i)^2}$$
The correct weight to choose is
$$w_i = \frac{1}{\sigma^2(X_i)}$$
Therefore, the average function for WPGMA becomes
$$Avg(x_1, \dots, x_n) = \frac{\sum \frac{x_i}{\sigma^2(x_i)}}{\sum \frac{1}{\sigma^2(x_i)}} = \frac{\sum w_i X_i}{\sum w_i}$$
The variance of this average is
$$\sigma^2(Avg) = \frac{1}{\sum \frac{1}{\sigma^2(x_i)}}$$
Noting the obvious: If $x_i = d_i - d_j \Rightarrow \sigma^2(d_i - d_j) = \sigma^2(d_i) - \sigma^2(d_j)$, assuming that d_i, d_j independent.

4.6.3.1.1 Example

We do the same example as for UPGMA. We now need the variance matrix too:

	Lettuce	Rice	Mosquito	Monkey	Human
Lettuce	0.000	10.621	35.460	32.296	32.450
Rice	10.621	0.000	33.808	30.669	30.799
Mosquito	35.460	33.808	0.000	28.614	28.748
Monkey	32.296	30.669	28.614	0.000	0.323
Human	32.450	30.799	28.748	0.323	0.000

1. Again take the shortest distance. We compute the weighted distance from monkey and human to the other three species
$$d_M = \frac{d(M,L)}{\sigma^2(M,L)} + \frac{d(M,R)}{\sigma^2(M,R)} + \frac{d(M,P)}{\sigma^2(M,P)} = 45.437, \qquad d_H = 45.546$$
The new subtree will have two leaves which are separated by 0.804. So the two branches from the root A to the leaves M and H should add to 0.804 while their difference should be $d_M - d_H = -0.109$. This leads to the following subtree:


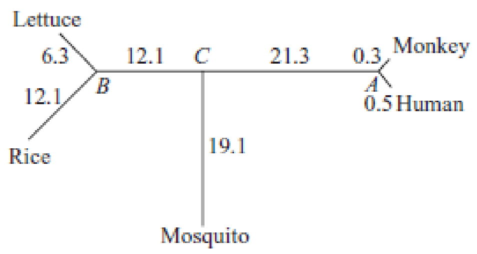
2. Now we calculate the new distance from A to the remaining species. In the case of lettuce L this becomes
$$d(L,A) = \frac{\frac{d(H,L) - d(H,A)}{\sigma^2(H,L) - \sigma^2(H,A)} + \frac{d(M,L) - d(M,A)}{\sigma^2(M,L) - \sigma^2(M,A)}}{\frac{1}{\sigma^2(H,L) - \sigma^2(H,A)} + \frac{1}{\sigma^2(M,L) - \sigma^2(M,A)}} = 43.941$$
And this has a variance of
$$\sigma^2(L,A) = \frac{1}{\frac{1}{\sigma^2(H,L) - \sigma^2(H,A)} + \frac{1}{\sigma^2(M,L) - \sigma^2(M,A)}} = 16.187$$

	Lettuce *L*	Rice *R*	Mosquito *P*	*A*
Lettuce *L*	0.0	**18.409**	44.161	43.941
Rice *R*	**18.409**	0.0	46.621	51.225
Mosquito *P*	44.161	46.621	0.0	40.381
A	43.941	51.225	40.381	0.0

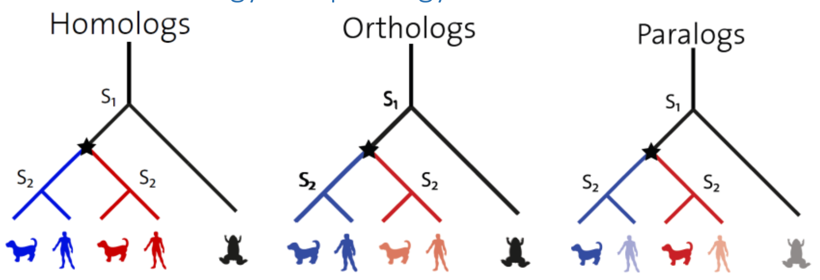
	Lettuce *L*	Rice *R*	Mosquito *P*	*A*
Lettuce *L*	0.0	10.621	35.460	16.187
Rice *R*	10.621	0.0	33.808	15.367
Mosquito *P*	35.460	33.808	0.0	14.340
A	16.187	15.367	14.340	0.0

3. Repeat step 1 & 2.

After performing this we end up in the unrooted tree



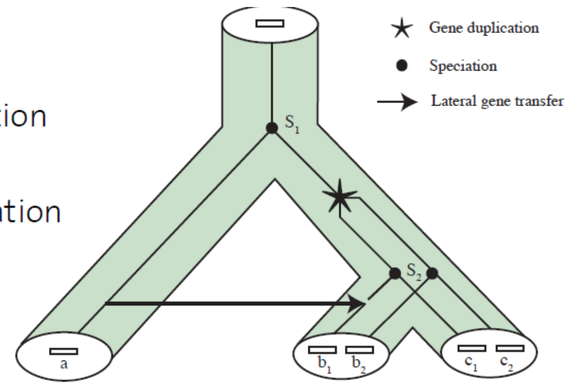
4.7 Model violations: orthology and paralogy



Homologs are species with shared ancestry between a pair of structures, or genes. Two segments of DNA can have shared ancestry because of either a speciation event (**orthologs**) or a duplication event (**paralogs**). Homologs resulting from horizontal gene transfer are termed **xenologs**. Orthologs are generally referred to a species tree, while paralogs are generally referred to a gene tree.

Main Subtypes

- *Orthologs*: through speciation
- *Paralogs*: through duplication
- *Xenologs*: through lateral transfer



4.7.1 Detecting orthology and paralogy

There are two main methods to distinguish between orthology and paralogy:

- Tree reconciliation
- Graph-based methods

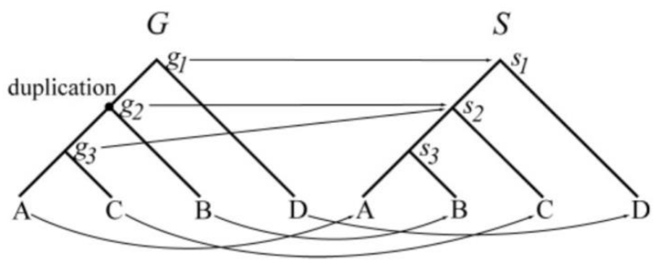
4.7.1.1 Tree reconciliation

Parsimony arguments:

- Duplications are rare events. The reconciliation that requires the fewest number of duplications is the preferred one
- The incongruences can be explained in terms of speciation, duplication and loss events on gene tree
- Trivial to establish orthology/paralogy from reconciled tree

The algorithm goes like that:

1. Map leaves of G to their species in S. Map inner node g_i in postfix order (from leaves to root)
2. Map g_i to the lowest node s_i such that the species below g_i are all included below s_i

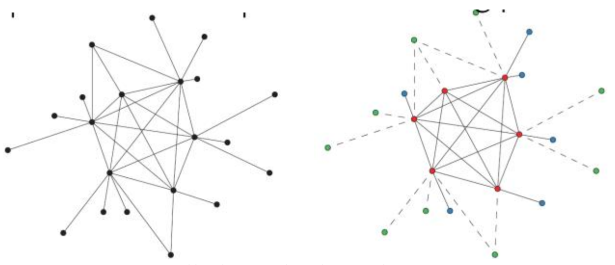


3. If g_i maps to the same node s_i in S as one of its children, g_i is a duplication node, otherwise a speciation node

4.7.1.2 Graph-based methods

Typically it runs in two stages:

1. Graph-construction phase
2. Clustering phase

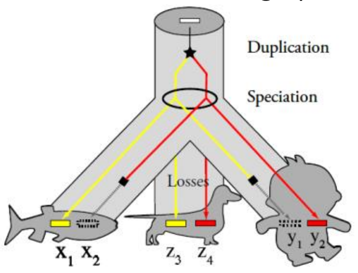


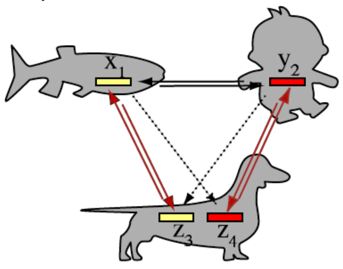
By observation we know that between two species, orthologs are closer than paralogs. Closer genes usually have higher alignment scores, so it is species-specific. A top scoring hit is likely to be an ortholog.
To improve the result one could:

- Instead of score, use evolutionary distance RSD (Reciprocal smallest distance)
- Relax the top/smallest requirement to include more than one orthologs
- Take into account statistical uncertainty of distance estimates
- Detect differential gene losses

4.7.1.2.1 Detecting Differential losses

Let's have a look on a graph example





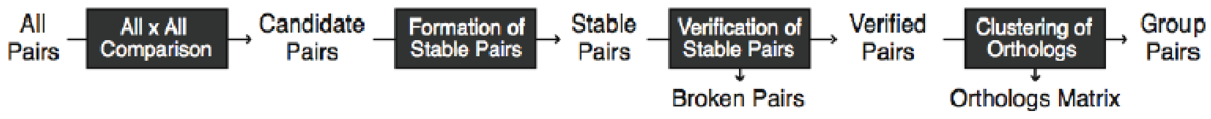
With (x_1, z_3) and (y_2, z_4) we have two stable pairs (orthologous).
(x_1, z_3) is significantly closer than (x_1, z_4) and also (y_2, z_4) is significant closer than (y_2, z_3).
But (x_1, z_4) and (y_2, z_4) are not significantly different (paralogous).

Therefore, x_2 and y_1 were lost.

The conditions exclude cases where X (respectively Y) speciated before the duplication event, in which cases x_1 (or y_2) would be orthologous to all three other genes (y_2, z_3, z_4). For example, if x_1 is closer to z_4 than y_2 to z_3, Y speciated before the duplication event.

4.7.1.3 OMA algorithm

The OMA algorithm uses as input the sequences from complete genomes and yields orthologous relations and groups of orthologs. The process works in 4 phases:



1. All-against-all: the pairwise alignments between every pair of proteins of all genomes ("all-against-all"). For significant ones, an evolutionary distance is estimated in PAM units.
2. Formation of "stable pairs"(SPs): the mutually closest pairs within a confidence interval, to avoid exclusion of n:m orthologs.
3. Detection of paralogs: among SPs using witnesses of non-orthology. Pairs surviving this step are named "verified pairs" (VPs). At this point, they are very likely to be orthologs.
4. Formation of OMA groups: made of a subset of proteins in which all pairs are VPs (clique).

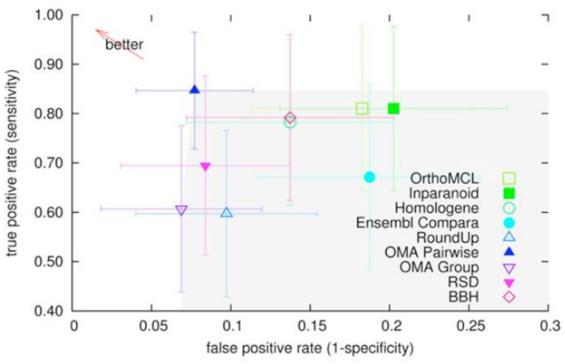
4.7.1.4 Tree-based vs. Graph-based methods

Gene trees provide more information than pairs of groups of orthologs. However, rooting of the trees is difficult.

Graph-based methods have lower computational complexity, that's why tree-based methods often start with a graph-based clustering step.

4.7.1.5 Benchmarking orthology

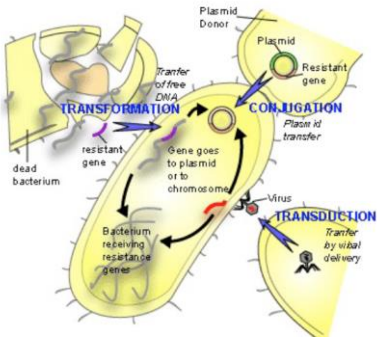
The main challenge is that the precise evolutionary history of genomes is unknown. An ideal measure for benchmarking should correlate with orthology/paralogy but should be independent from methods used to detect orthology/paralogy.



What are the candidates for measuring?

- Functional conservation
  pro: orthologs tend to have conserved function
  con: correlation of orthology/parology with function is often unclear
- Convervation of gene neighborhood
  pro: shared syntony implies common ancestry
  con: paralogs can also have conserved neighborhood
- Phylogeny
  pro: agreement of inferred orthologs with the undisputed species tree
  con: some methods rely on phylogenetic trees to infer orthologs. Measure not independent

4.7.2 Detecting xenology

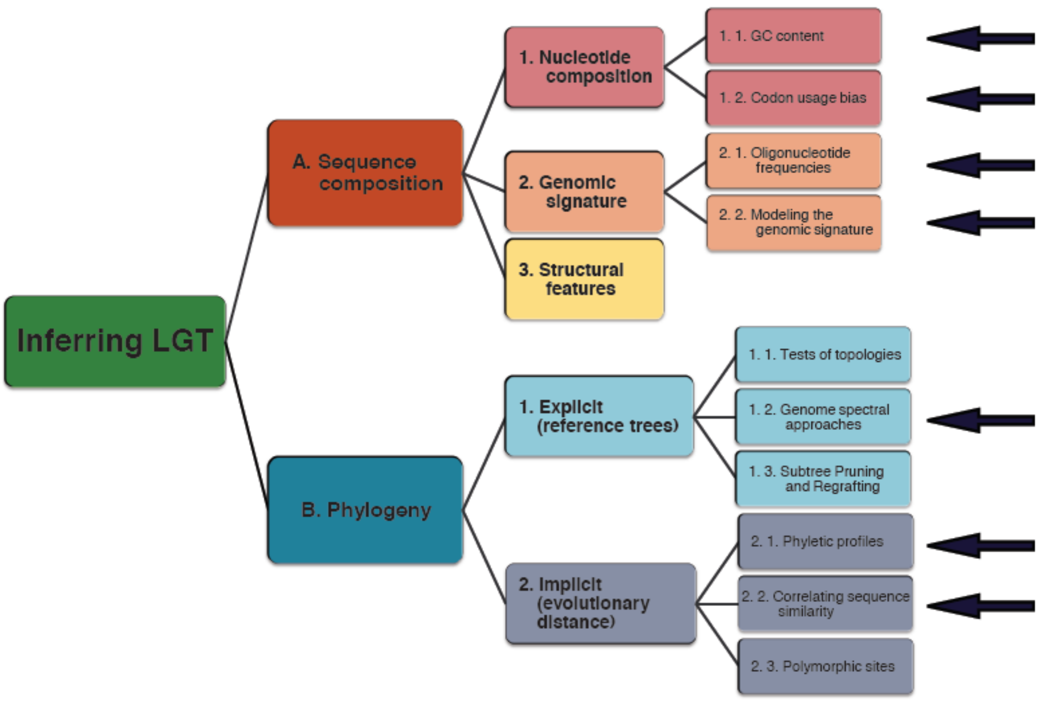


Two species are xenologs if there was a lateral gene transfer between them. This means the flow of genes between different species.

In bacteria this can happen in three different ways:

- Transformation
- Conjugation
- Transduction

There are many methods of LGT events, like GC content, codon usage bias, ...


Inferring LGT
A. Sequence composition
1. Nucleotide composition
1. 1. GC content
1. 2. Codon usage bias
2. Genomic signature
2. 1. Oligonucleotide frequencies
2. 2. Modeling the genomic signature
3. Structural features
B. Phylogeny
1. Explicit (reference trees)
1. 1. Tests of topologies
1. 2. Genome spectral approaches
1. 3. Subtree Pruning and Regrafting
2. Implicit (evolutionary distance)
2. 1. Phyletic profiles
2. 2. Correlating sequence similarity
2. 3. Polymorphic sites

As we can see we have 2 main groups:

- Sequence composition
- Phylogeny

5 Codon bias

There are 64 codons (61 without stop-codons) to encode 20 AA. Synonymous codons encode the same AA, but not necessarily with the same rate. That is, the codons are biased.

For example, AAA and AAG both encode Lysine. In Streptomyces venezuelae 98% of Lysine is encoded with AAG, in Lactobacillus acidophilus, the rate is 50% (as naively expected)

5.1 Main reason

The main factor is the selection for optimal translation at protein synthesis.

The other reasons are diverse, but there is a balance between **mutational biases** and **natural selection**

5.1.1 Mutational bias

Mutational biases are neutral (do not affect fitness) and typically act globally on all DNA sequences of a given organism.

Caused by underlying mechanisms that favour certain types of changes in DNA

- Chemical decay of nucleotide bases
- Non-uniform DNA repair
- Non-random replication errors

5.1.2 Selection bias

Selection biases can be specific to genes or even codon positions.

DNA level	Some codon patterns are avoided or preferred.

	Example: Codons TA* is avoided since an error in copying this codon could be catastrophic as TAA and TAG are stop codons.
RNA level	• Codon bias correlates well with mRNA levels
Translation level	• Codon usage shows positive correlation to tRNA abundance (Isotopenhäufigkeit)

5.1.3 Codon bias correlation (causality unclear)

Global codon bias has been shown to correlate with:

- GC content
- tRNA content
- organism growth temperature
- Significant correlation between CB and protein abundance

5.1.4 Applications

- Assess likelihood of being in a protein-coding region of the genome
- Detecting lateral gene transfer
- Deriving alternative (parameterized) substitution matrices which take care of the codon bias.

5.1.5 Measures (Codon Usage Indices)

Codon usage indices map codon frequencies to a single number

- Consider degree of codon degeneracy
- Exclude one-codon AAs Methionine and Tryptophan
- Consider start codons separately
- Avoid sequences with less than 80-100 codons because of problems with accuracy.

5.1.5.1 Frequency of Optimal Codons (Fop)

- Ratio of the number of optimal codons used to the total number of synonymous codons
- Optimal codons defined by nucleotide chemistry, codon usage bias or tRNA availability

5.1.5.2 Codon bias index

- Similar to Fop but uses a scaling factor and is normalized between -1 and 1
- 1: only preferred codons are used
- 0: random choice
- -1: only non-preferred codons are used

5.1.5.3 Codon adaptation index

Based on relative adaptiveness: Frequency of each synonymous codon is normalized by frequency of most frequent codon. Most frequent codon has adaptiveness = 1, others < 1

5.1.5.4 tRNA Pairing Index (TPI)

- Compares ordering of synonymous codons to a random distribution of existing codons
- Most uncorrelated sequences (ABABAB), TPI: -1
- Most correlated sequences (AAABBB), TPI: 1

6 Multiple Sequence Alignment (MSA)

Idea of MSA: Every aligned column has a common ancestor.

Exact MSA using multidimensional dynamic programming is too expensive. For m sequences of length n, the complexity is $O(n^m)$. Useful algorithms are therefore only approximative.

6.1 Applications

- Tree building
- Secondary structure prediction
- More accurate distances (We know better about a pairwise alignment by looking at several other alignments too)
- Ancestral sequences

6.2 Probabilistic ancestral sequences

- This type of MSA aligns a sequence to a probabilistic sequence (aka profile).
- The profile doesn't store symbols, but the probabilities of all symbols to be in the profile.
- The profile's dimension is therefore 20 x n (for AA) as opposed to 1 x n if it was a sequence.
- The profile is a probability matrix, i.e. all columns sum up to 1.

The probability of a symbol a being homologous to a symbol probability vector B is computed as follows:

Prenote: a is a scalar, B is a 1 x 20 matrix, where B is a column of the profile. Normally, we would have computed the homology probability of two scalars.

$$\Pr\{a, B \text{ are homologous}\} = \sum_x f_x \left(\Pr\left\{x \to a, d_a\right\} \cdot \sum_y \Pr\left\{x \to B, d_B\right\} B_y \right)$$
$$= \sum_x f_x \left(M^{d_A}\right)_{ax} \left(M^{d_B}\right)_{*x} B$$
$$= f_a \sum_x \left(M^{d_A}\right)_{xa} \left(M^{d_B}\right)_{*x} B$$
$$= f_a \left(M^{d_A+d_B}\right)_{*a} B$$

The score is then

$$Score = \log\left(\frac{Pr\{homologous\}}{Pr\{independent\}}\right) = \log \frac{f_a \left(M^{d_a+d_B}\right)_{*a} B}{f_a f \cdot B} = \log \frac{\left(M^{d_a+d_B}\right)_{*a} B}{f \cdot B}$$

Note: The term $f \cdot B$ poses a problem for computing the Dayhoff matrices, since it depends on B which is the input to the dynamic programming algorithm. Therefore, the scores need to be precomputed for every profile.

6.3 Circular tour algorithm

From: http://www.biorecipes.com/CompBioExercises/MSAConstrPaperEx.html

This algorithm aligns multiple sequences based on pairwise alignments. The algorithms goes like this:

1. Align all sequences pairwise with each other, and store the distances in a distance matrix D. D is a quadratic, symmetric matrix with dimension equal to the number of sequences.
2. Find a minimal circular tour that visits all sequences, e.g. with a travelling salesman solver.
3. Cut the tour at the maximal distance and order the sequences according to the tour.
4. Consider the case where the first k sequences are already aligned in the MSA. Now we want to add sequence k+1 using the alignment between sequences k and k+1. We call the kth aligned sequence in the MSA k' and the aligned sequence k in the pairwise alignment k''. The following cases can occur:
    - k' and k'' are the same: This is the easy case. The aligned sequence k+1 can directly be added to the MSA.

- If there are any gaps in k', not present in k'', then these gaps have to be added to the aligned sequence k+1, before it can be added to the MSA.
- Gaps in k'' but not in k' have to be added to all already aligned sequences 1 to k. Afterwards, sequence k+1 can be added to the MSA. This case makes the MSA longer (more characters).

6.4 Progressive alignment algorithm

From: http://www.cbrg.ethz.ch/education/CompBiol/Exercises/msa-consensus-sol and http://www.cbrg.ethz.ch/education/CompBiol/Exercises/msa-consensus

This algorithm builds a MSA using a guide tree.

For every node, do

1. Compute consensus sequences for the left and the right MSA. A consensus sequence reduces a MSA (n x m matrix) to a 1 x m matrix, where n is the number of sequences and m the length of the sequences. E.g. a consensus sequence could consist of the most frequent character at each position.
2. Align the consensus sequences
3. Merge the left and the right MSA according to this alignment. This boils down to inserting gaps at the right positions in the left and right MSA.

6.5 Heuristics

A problem of the above algorithms is gap propagation. There are two categories of gap fixing strategies, Ad hoc and recomputing.

6.5.1 Ad hoc

Align gaps. Gaps should normally be vertically aligned in a MSA.	`ABCDAB_____ABCD`
	`AABD_____ADDA`
	`  AABD_____ADDA //corrected`

Island elimination. An island is a short aligned portion which is between two large gaps. This is biologically unrealistic. The island should be replaced with a gap.	`_______**________`
	`_________________ //corrected`

Compress gaps. Sometimes we know that in a region there should be only 0 or 1 gap. Solution: Delete one gap and align	`*********____***** //1`
	`*****___*____***** //2`
	`*****___********** //3`
	
	`   ******____***** //corrected 2`
	`    *****____***** //corrected 3`

Delete bad matching endings. In the example, o stands for an ending.	`********_____****o //1`
	`************o     //2`
	`********__****o   //3`
	
	`********_____ //corrected 1`
	`********      //corrected 2`
	`********__    //corrected 3`

6.5.2 Recomputing

1. Identify essentially bad columns by some quality measurement
2. Remove bad part from MSA (shrink)
3. Recompute it and reinsert.

6.6 Evaluation

1. Build trees using two MSA's and use some tree evaluator. The MSA inducing the better tree wins. (Also depends on the tree-generator of course)
2. Sum of pairs. Compute sum of pairwise scores. Bad, since it induces a weighting on edges of the tree!
3. Circular tour. Compute a circular tour and sum scores of sequences connected by the tour. => All edges are weighted exactly twice.

7 Mass profiling

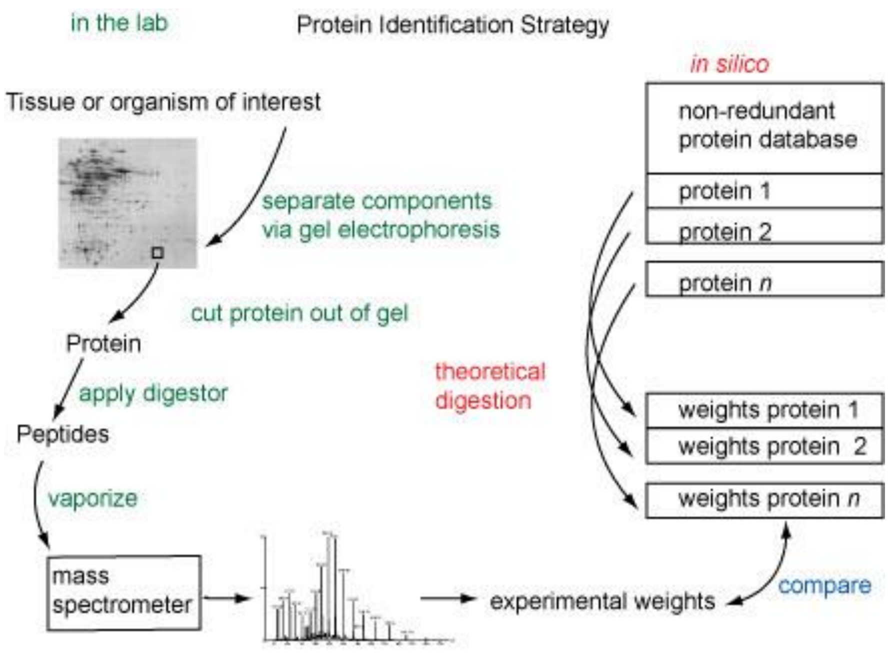
From: http://www.biorecipes.com/MolWeight/cbrg.html and http://www.cbrg.ethz.ch/biorecipes/biological/MolWeight

7.1 Goal

Identify a sample of a known protein using a protein database.

7.2 Idea

Use digesters to cut the problem to smaller pieces.



7.3 Algorithm (in silico part)

For all proteins in a database, do

1. Digest protein (digestion is an operation which cuts a protein sequence at some characteristic positions)
2. Compute Mol masses of protein fragments
3. Compare with Mol masses of experimentally digested protein

Best match wins!

The main problem is to come up with an error-tolerant matching strategy, since the theoretical and experimental data won't match or have ambiguous matches, due to the following reasons.

- The recording of molecular mass is subject to a **relative error**

- Two of the amino acids, leucine and isoleucine, are composed of the same atoms and thus have **identical weights**. In addition, Lysine (molecular weight, 146.1740) and Glutamine (molecular weight, 146.1310) have very close molecular weights.
- The searched sequence may **not be verbatim** in the database, although maybe a close relative of the sequence is.
- The **mutations** in the database sequence can cause the digestion to be different, splitting into more or fewer fragments. This will cause a complete mismatch of weights involving such fragments.
- **Impurities** in the sample and in the digestors may produce spurious data in the searched sample.
- The fragmentation (digestion) although in general accurate, is **not 100% deterministic**. Partial digestion or incorrect ones are also possible.
- The mass spectrometry measures are subject to **systematic (biased) errors** due to calibration.

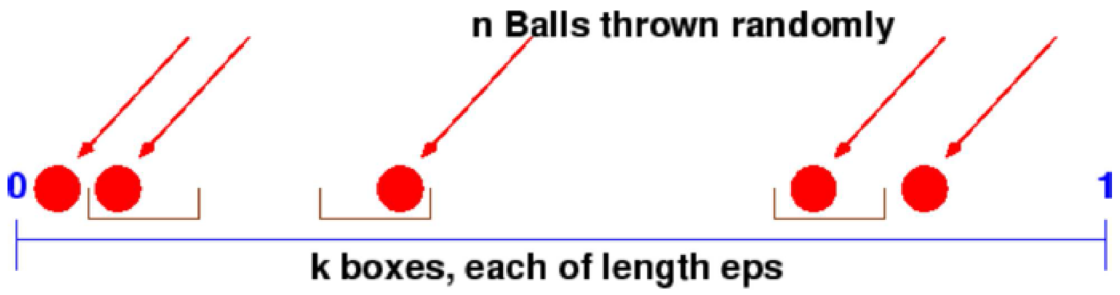
7.3.1 Comparison strategy

We replace step 3 of the above algorithm with a probability computation.

- Compute the probability that a match of the given weights against the computed ones happens at random.
- Record the m lowest probabilities.

We have k weights from a sampled protein and match them against an unrelated digested protein which splits into n fragments. For now, these fragments can be considered random. We further specify a tolerance ϵ .

Then the problem of matching the k sampled weights to the n random weights with tolerance ϵ is analogous to the problem of randomly throwing n balls into k boxes each with a width of ϵ. The probability of a match is the probability of each box having at least one ball.



This probability is given by

$$\Pr\{k, n, \epsilon\} = \sum_{i=0}^{k} (-1)^i \binom{k}{i} (1 - i\epsilon)^n$$

ϵ is chosen as the relative error of the nearest match of the k sample weights w_i and the n database weights of each sequence v_j, i.e.

$$\epsilon_i = \frac{2|\log w_i - \log v_j|}{\log w_{max} - \log w_{min}}$$

That means that i weights are within distance ϵ_i of database weights. The best match is considered the i, for which $\Pr\{i, n, \epsilon_i\}$ is minimal. Therefore, the score can be computed as

$$Score = -10 \log_{10} \min_i \Pr\{i, n, \epsilon_i\}$$

For every sequence, the result consists of the score and the corresponding i and n.

So, for every database protein, do

1. Digest protein to n fragments and compute their weights v_j (n is different for every protein)
2. For every sampled protein fragment w_i, compute the relative error ϵ_i to the nearest v_j
3. Compute the maximal score over all i (Minimal probability => Maximal score)
4. Return the score, i and n

The best matching database protein is the one with the highest score. The corresponding i and n are only for information.

8 Genome rearrangements

From: http://www.math.nus.edu.sg/~matzlx/ma3259/GB_Lecture21.pdf

8.1 Motivation

Sequence alignment aligns strictly vertically only. However, biology might be cleverer and do something like this

```
ABCDEFGH

EFGHABCD
```

Strict column-by-column sequence alignment can't really resolve what has happened here, so we need other distance measures.

8.2 Bullet points

- Genome rearrangements can serve as a distance measure. The distance is computed as the number of genome rearrangements needed to evolve from sequence A to sequence B.
- Genome rearrangements are inversion, translocation, double cut and joins, fission, fusion, etc.
- Rearrangements can be abstracted as signed permutations.

8.3 Bipartite matching

The solution to the problem in the motivation can be found using a bipartite matching algorithm. The branches need to be initialized with all candidate pairs, for which some threshold must be specified.

To preserve synteny (i.e. neighbours remain near), the threshold could penalize distant candidate pairs

$$S^*_{A,B} = S_{A,B} + \log \frac{n}{k}$$

With n the length of the string and k the distance between A and B.

8.4 Inversion

Inversion is inspired by reverse strand reading. In the below example, an inversion from 3 to 5 is shown. Minus signs stand for reverse strand coding order.

```
{1,2,3,4,5,6,7}

{1,2,-5,-4,-3,6,7}
```

1 Transposition = 3 Inversions

8.5 Transpositions

Transpositions is something like a translation. In the below example a transposition from B to C is shown.

```
*****AB*****CD***XY

*****AD***XB*****CY
```

8.6 Double cut and join

Double cut and join is only mildly biological. In the below example, a cut between A,B and C,D is shown. There are two possibilities for a join. Cut and join count as one operation

```
Cut:   ****AB******CD***

Join1: ****AC******BD*** //C and B in reverse order

Join2: ****AD***        ******CB***** //sequences splitted
```

1 Transposition = 2 Double cut and joins (DCJ)

1 Inversion = 1 DCJ

1 Fusion = 1 DCJ

8.6.1 Adjacency graph

Given two sequences, A and B an adjacency graph can be constructed from which the number of DCJ and signed inversions can be computed.

1. In A and B, make pairwise pairs, i.e. $\{A_0, A_1\}, \{A_1, A_2\}, \dots, \{A_{n-1}, A_n\}, \{B_0, B_1\}, \dots., \{B_{n-1}, B_n\}$
2. Make an adjacency graph with these pairs. Connect two pairs, if $A_l = B_l$ or $A_r = B_r$
3. Count the number of cycles C and the number of odd length sequences I

The number of DCJ is then given by

$$\#DCJ = n - \left(C + \frac{I}{2}\right)$$

The number of signed inversions is given by

$$\#signed\ inversions = n - \left(C + \frac{I}{2}\right) + t$$

Whatever t is.

9 Appendix

9.1 Matrix power

$$A^k x = TD^k T^{-1} x$$

Where T is the Eigenvector matrix and D is the Eigenvalue diagonal matrix of A.

9.2 Matrix exponential

$$\begin{aligned}\exp(A) &= I_n + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots \\ &= Te^D T^{-1}\end{aligned}$$

Where T is the Eigenvector matrix and D is the Eigenvalue diagonal matrix of A.

Note that $\exp(A)$ and A have the same eigenvectors.

Note that $e^D$is simply an element-wise exponential, since D is diagonal.

9.3 Matrix logarithm

$$\exp(A) = T \underbrace{e^D}_{D^*} T^{-1} =: B$$

$$\log(B) = T \log(D^*)\, T^{-1} = A$$

Where T is the Eigenvector matrix of A and B. D^* is the Eigenvalue diagonal matrix of B.

Note that $\log(D^*)$ is an element-wise logarithm since D^* is diagonal.

9.4 Maximum likelihood

The maximum likelihood method is a statistical method for estimating a parameter of a statistical model. Simplified we have a distribution which is dependent on a parameter θ (so we have a faimily of curves). We choose now the best θ according to the realization of the observed data. We call this θ^* maximum likelihood estimator (MLE).

Think about a probability mass function (dt: Wahrscheinlichkeitsfunktion) which depends on θ

$$p: \Omega \to [0,1], \qquad x \mapsto p(x|\theta)$$

To the observed data x we get the following likelihood-function

$$l: \Theta \to [0,1], \qquad \theta \mapsto p(x|\theta)$$

Ω is the sample space (dt: Ergebnisraum) and Θ the space of all possible parameters.

Now let's write it more explicit:

Thinks about a probability mass function p which depends on θ. Now we take samples (dt: Zufallsstichproben) whit n independent and identical realizations. We can therefore factorize the function

$$f(x_1, \dots, x_n|\theta) = \prod_{i=1}^{n} f_{x_i}(x_i|\theta)$$

Instead of interpreting the function as dependent on $x_1, \dots, x_n$ with a fixed θ, we can also interpret it as function dependent on θ with fixed $x_1, \dots, x_n$. This leads to the likelihood-function

$$l(\theta) = \prod_{i=1}^{n} f_{x_i}(x_i|\theta)$$

To get now the best θ, i.e. θ^*, we simply have to set the derivative of it to zero. Since this is cumbersome we do this on the log-likelihood-function, since the logarithm is monotonic and has its maximum at the same place

$$L(\theta) = ln\left(\prod_{i=1}^{n} f_{x_i}(x_i|\theta)\right) = \sum_{i=1}^{n} ln\left(f_{x_i}(x_i|\theta)\right)$$

9.4.1 Derivatives of L

We need the derivatives of L. $\theta = [\theta_1, \dots, \theta_n]$

1. $\frac{dL(\theta)}{d\theta} = 0$ leads us to θ^* and is a vector

2. $\frac{d^2L(\theta^*)}{d\theta^2} = -\frac{1}{\sigma^2\theta^*}$ where σ is useful to build the covariance matrix $\sigma_\theta^2 = \begin{pmatrix} \sigma_1^2 & \cdots & \sigma_{n,n}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{n,n}^2 & \cdots & \sigma_n^2 \end{pmatrix}$

   where $\sigma_{i,k}^2 = cov(\theta_i, \theta_j)$