Inhalt

Computational Biology

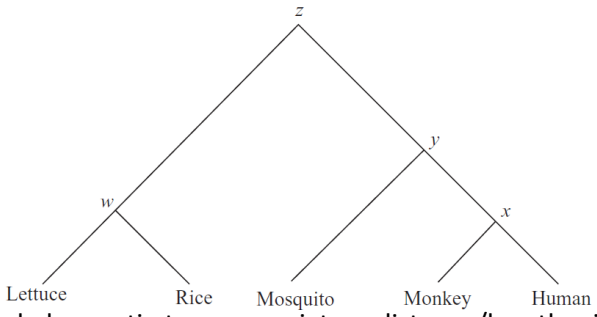
Modelling

3 steps to create a model:

1. Create a model with formulas that depend on parameters
2. Calibrate the model with trainings data -> estimate parameters
3. Validate the model with testing data

!!! Trainings data & testing data have to be independent!!!

Motivation: Phylogenetic tree construction



The problem in phylogenetic tree construction, is to find the tree which best describes the relationships between objects in a set. Usually and in our examples, the objects are species.

To build a phylogenetic tree we need to have some measurable criteria which allow us to decide which tree fits the data best. Most phylogenetic trees associate a distance/length with each branch of the tree, the time or amount of change that happened in that part of the evolution.

Building a phylogenetic tree always implies finding the topology (shape) of the tree, and usually also finding the branch lengths. As it turns out, determining the topology is much more difficult than estimating the branch lengths. As mentioned above, it is difficult or impossible to obtain information about the internal nodes x, y, z of the tree. We only know the leaves (lettuce, rice, mosquito, monkey, human) of the tree which represent present-day species. So we will restrict ourselves to this case.

Tree types

There are three main types of phylogenetic trees, which are bases on:

1. **distance information**: We need a definition of distance between the species. Then the tree (topology and branch length) is constructed such that distances between any pair of leaves can be mapped as accurately as possible.
2. **character information**: A character is a discrete property of a species (leaf). It can be assigned to each species. For example a species is either a mammal or not, has gray, blue, etc. eyes. Under this classification internal nodes are assigned characters to minimize the total number of character changes. The method which minimizes this is called parsimony.
3. **probabilistic methods** (max likelihood): The species are described by characteristics which evolve according to some probabilistic model. Mostly applied to genomic sequences. Will not be covered

Trees based on distance information

To construct a tree of n leaves based on distance information, the following input is necessary

- an n x n symmetric matrix with positive distances,
- optionally an n x n symmetric matrix of the variances of the distances (a measure

of the error that the distances may have).

The output is an unrooted tree with n leaves (present-day species) and n - 2 internal nodes (representing ancestral species) connected by 2n - 3 branches, each one with its length. In graph theory terms this is an unrooted tree with each edge weighted by its length and each internal node of degree 3. What we need are the following.

(i) A measure of the distance between species (and optionally their variance).
(ii) A method for constructing good initial topologies (constructor).
(iii) A method of fitting the distances between sequences to a tree topology. This needs branch lengths. The error of this approximation is what we want to optimize, hence computing this error is our evaluator.
(iv) A search strategy for better topologies (heuristics).

There are three common measures of distance:

- **Percentage of changed positions**: equal to 100% minus the percentage of identity, a measure which is not additive.
- **Synteny**: a measure of the relative order of genes in the genomes which is very close to additive for close species.
- **PAM distance**: a measure of amino acid mutation which is additive.

A distance is additive iff:

$$d(A,B) + d(B,C) = d(A,C)$$

Percentage of change

We could simply count the percentage of identity in the alignment. Our distance would be the percentage of mismatches:

- number of exact matches: 148
- number of amino acids: 245 and 243
- length of the alignment: 245
- percentage of identity: 148/245 ≈ 60.41%
- percentage of change: 97/245 ≈ 39.59%.

Then 1 − p (p stands for percent identity) is a measure of distance. This measure is very simple but has some problems. The main problem is that it does not satisfy the additivity property, since a point mutation during the change from amino acid sequence A → B which is reverted in the transition from amino acid sequence B →C will make d (A, B) + d (B,C) > d (A,C).

The second problem is that percentage of change is a very crude measure of distance. In particular, when comparing amino acid sequences there is more information hidden in the mutations to

different amino acids: some amino acids are chemically very similar, some are very different, so considering all the mutations as equal reduces the information.

Synteny

Synteny refers to the preservation of gene order in species descending from a common ancestor. Two species that are close will have the same or nearly the same order of genes. With time DNA rearrangements occur which will alter the order of genes in a genome. We model such a rearrangement with one or more inversions of a large portion of DNA, normally including many genes. The minimum number of inversions needed to reorder a genome of one species into the genome of another species can be used as a measure of distance, for example

- species 1 has its genes in the order `ABCDEFGHIJ`
- species 2 has its genes in the order `AEDCFGHBIJ`

Here each letter represents a gene – a long portion of DNA encoding one protein. These two sequences require three inversions to convert one into the other; as shown in Figure 8.7. Hence we will say that the synteny distance between these two genomes is three. Genomes may have thousands of genes, so when there are few rearrangements the chance of one rearrangement exactly undoing a previous one is very small. Intuitively we can argue that each inversion, which is identified by the two end points, is rather unique: there are $\binom{n}{2}$ possible inversions for a genome with n genes. This is the reason why the synteny distance can be considered additive for practical purposes.

PAM distance with the Markov Model

We will use a Markovian model of evolution. This is of course a simplification of what happens in nature. Yet it is quite powerful and the most widely accepted model of evolution.

It implies that amino acids (AA) mutate randomly and independently of each other. Each AA mutates with a probability which depends only on itself (and not on its history or its neighbours). Insertions or deletions also happen at random, independently of other events. Since there are 20 AA, the transition probabilities are described by a 20x20-mutation matrix, denoted by **M** with

$$M_{ji} = \Pr\{aminoacid_j \rightarrow aminoacid_i\}$$

While simple, this is one of the best methods for modelling evolution. Again:

- Finite number of **states**
- Transitions between states (only from to, NOT dependant on history!)
- A sequence of states is called a **Markov Chain**

Recall: Linear Algebra - Eigenvector & Eigenvalue

$$A\underline{x} = \lambda\underline{x}$$

λ = Eigenvalue, $\underline{x}$ = Eigenvector

Eigendecomposition

The eigendecomposition of A is a set of eigenvalues & eigenvectors:

- spectral decomposition of A

- Common: $\|\underline{x}\| = 1$
- Express any vector y as linear combination of eigenvectors:

$$\underline{y} = \sum_i a_i \underline{x_i}$$

Markov Matrix

The Markov Matrix **M** has Eigenvalues for which must hold:

- $\lambda_1 = 1$
- $|\lambda_i| < 1, i = 2, \dots, n$

For any vector y must hold:

$$\underline{y} = \sum_i a_i \underline{x_i} \;\&\; M\underline{x} = \lambda \underline{x} \;\Rightarrow\; M\underline{y} = \sum_i a_i \lambda_i \underline{x_i}$$

So we can write any vector which was multiplied by M as a linear combination of Eigenvectors and Eigenvalues. Of course we can apply M k-times:

$$M^k \underline{y} = \sum_i a_i {\lambda_i}^k \underline{x_i}$$

Note: $|\lambda_i|^k \leq 1$, i=1,...,n (kind of intuitive since the Markov Matrix is about probabilities)

Continuous Markov Models

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

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

Recall: exp(A)

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \Rightarrow e^A = \underline{1} + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

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 $M = e^Q$ then than 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,...

Example

Lets construct Q out of a given situation:

We have two states: A & B. The probability that we change from A to B in one time step is 0.2 and that to change from B to A is 0.1. This leads to:

$$Q = \begin{pmatrix} -0.1 & 0.2 \\ 0.1 & -0.2 \end{pmatrix}$$

Consequently in one time step we lose 0.1 of A and also 0.2 of B, that's why we have negative values on the diagonal.

Note: The columns in Q add to 0. !!! In M the columns add to 1!!!

Recall: Different notations for Matrix

Be aware: We encounter different notations depending on the paper we are reading:

$$\begin{matrix} & from \\ to & \begin{pmatrix} & \\ & \end{pmatrix} \end{matrix} \Rightarrow Qy = \cdots$$

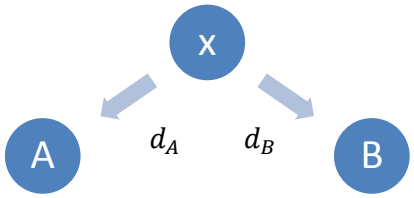
$$\begin{matrix} & to \\ from & \begin{pmatrix} & \\ & \end{pmatrix} \end{matrix} \Rightarrow yQ = \cdots$$

Example

Assume two equivalent sequences where only one position is different (A & B). Let's say these two had the same ancestor x. So x is the AA (=amino acid) in the **LCA** (=last common ancestor) .

Question: What is the probability of this event to happen that from the LCA we get two different species (or: What is the probability that A & B are homologous)?

- A & B are known AA in sequence 1 & 2
- x is the unknown AA in the LCA
- mutation is a Markovian process
- d_A and d_B are integer



Probability that x changes to A:

$$\Pr\{x \to A, d_A\}$$

where the process for an selected element Ax can be written as:

$$\left(x \to A\right)_{Ax} = \left(M^{d_A}\right)_{Ax}$$

Same holds true for B.

The probability that we get out of the LCA two different species is simply the multiplication of both probabilities:

$$\Pr\{x \to A, d_A\} \times \Pr\{x \to B, d_B\}$$

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

$\Pr\{A, B\ are\ homologous\}$

$$= \sum_x f_x \left(\Pr\{x \to A, d_A\} \times \Pr\{x \to B, d_B\}\right)$$

$$= f_B \sum_x \left(M^{d_A}\right)_{Ax} \left(M^{d_B}\right)_{xB}$$

$$= f_B \left(M^{d_A} M^{d_B}\right)_{AB}$$

$$= f_B \left(M^{d_A+d_B}\right)_{AB} = f_A \left(M^{d_A+d_B}\right)_{BA}$$

d_A & d_B can be equal or not. In case one of them is zero, so that x is A or B, we talk about **reversible evolution**: A evolved to B or was it B evolved to A? We cannot tell what was first: A or B?

PAM-Distance

A 1-PAM (Point accepted mutation) matrix describes an amount of evolution which will change, on the average 1% of the amino acids. In mathematical terms this is expressed as a matrix **M** such that

$$\sum_{i=1}^{20} f_i(1 - M_{ii}) = 1 - \sum_{i=1}^{20} f_i M_{ii} = 0.01$$

where f_i is the frequency of the i^{th} amino acid (The vector $\underline{f} = (f_1, f_2, \dots, f_{20})$ describes the natural frequencies of the amino acids in nature, consequently $\sum f_i = 1$). The M_{ii}, the diagonal elements of **M**, are the probabilities that a given AA does not change.

If we have a probability vector $\underline{p}$, the product $M\underline{p}$ gives the probability vector of $\underline{p}$ after a random evolution equivalent to 1-PAM unit. Or, if we start with a given AA i (i.e. a probability vector which contains a 1 in position i and zeros in all other positions) M_{*i} (the i^{th} column of M) is the corresponding probability vector after one unit of random evolution. Similarly: after k units of evolution (what is sometimes called d-PAM evolution) a frequency vector $\underline{p}$ will be changed into the frequency vector $M^d\underline{p}$.

By changing d we change the PAM value. Bigger d corresponds to more mutations. Consequently $M^\infty \underline{p} = \underline{f}$. (After infinite evolution, any probability distribution converges to the distribution in nature.) From this we also see that $M\underline{f} = \underline{f}$ or that $\underline{f}$ is the eigenvector with eigenvalue 1 of M.

A d-PAM mutation matrix describes a d-PAM mutation and is identical to M^d. Two sequences which are 250 PAM units distant from each other (PAM distance = 250) are expected to have approximately 17% identity, since

$$\sum_{i=1}^{20} f_i \, M_{ii} := 0.99$$

for 1-PAM, and

$$\sum_{i=1}^{20} f_i \, (M^{250})_{ii} \approx 0.17$$

for 250-PAM. PAM distances measure the amount of change. The rate of change depends on many other important factors. E.g.:

- the time needed for a reproduction cycle (from minutes for bacteria to hundreds of years in redwood-trees)
- the importance of the protein
- the degree of optimization
- changes in the environment etc.

This model of evolution is symmetric, i.e. we cannot distinguish the probability of amino acid i evolving from amino acid j from the probability of j evolving to i.

Some remarks on PAM matrices:

- 250-PAM is only weakly diagonal dominant
- In general, the diagonal of D is positive and largest off-diagonals are mostly negative
- AA with similar values for transition have similar properties and are replacable

Dayhoff matrix

The example before the PAM-Distance should give us a intuition about homology of two sequences. Now we do it more formal with the likelihood method which leads us to the Dayhoff matrices. Dayhoff Matrices D are derived from the 250-PAM mutation matrix, and are used for the standard dynamic programming method of sequence alignment. Aligning sequences by dynamic programming using Dayhoff matrices is equivalent to finding the alignment which maximizes the probability that the two sequences evolved from an ancestral sequence, as opposed to being random sequences.

$$\prod_{i=1}^{n} f_{t_i} \left(M^d\right)_{s_i t_i} \leftarrow Liklihood$$

$$d = d_A + d_B \leftarrow We\ won't\ look\ at\ that$$

s & t are the sequences we want to compare, and therefore s_i & t_i the elements (AA) of the sequences. To make a statement if the sequences are homologous or not we compare this likelihood against the null-hypothesis that the sequences are independent by just multiplying the individuals frequencies:

$$\Rightarrow \frac{\Pr\{s, t\ are\ homologous\}}{\Pr\{s, t\ are\ independent\}} = \prod_{i=1}^{n} \frac{f_{t_i} \left(M^d\right)_{s_i t_i}}{f_{t_i} f_{s_i}} = \prod_{i=1}^{n} \frac{\left(M^d\right)_{s_i t_i}}{f_{s_i}}$$

Recall what happens by taking the log of a product:

$$log\left(\prod \frac{x_i}{y_i}\right) = \sum log\left(\frac{x_i}{y_i}\right)$$

If we use this we get:

$$log\left(\frac{\Pr\{s, t\ are\ homologous\}}{\Pr\{s, t\ are\ independent\}}\right) = \sum_{i=1}^{n} log\left(\frac{\left(M^d\right)_{s_i t_i}}{f_{s_i}}\right)$$

If this is bigger than 1 it is more likely that the sequence t is homologous to s than random.

We define now the Dayhoff matrix:

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

We see that the Dayhoff matrix is derived from the 250-PAM matrix (d=250).

Recall: Exponentials of M

How to compute M^d for $d \in \mathbb{R}$

We know that we can decompose a matrix A into an Matrix of Eigenvectors and Eigenvalues

$$A = V\Lambda V^{-1}$$

Where $V$is the Eigenvectormatrix and Λ the diagonal matrix with the Eigenvalues. Therefore

$$A^k = (V\Lambda V^{-1})^k = V\Lambda V^{-1} \cdot V\Lambda V^{-1} \cdot \ldots \cdot V\Lambda V^{-1} = V\Lambda^k V^{-1}$$

For a meromorphic function (can be written as $f(A) = \sum_i^\infty a_i A^i$) holds that

$$f(A) = \sum_i^\infty a_i\,(V\Lambda V^{-1})^i = V\sum_i^\infty a_i \begin{pmatrix} {\lambda_1}^i & & \\ & \ddots & \\ & & {\lambda_n}^i \end{pmatrix} V^{-1} = V\begin{pmatrix} f(\lambda_1) & & \\ & \ddots & \\ & & f(\lambda_n) \end{pmatrix} V^{-1}$$

We also know that $M = e^Q \Rightarrow M^d = e^{Qd}$. That means that our function is:

$$f(Q) = e^{Qd}$$

This leads to

$$M^d = e^{Qd} = f(Q) = V\begin{pmatrix} e^{\lambda_1 d} & & \\ & \ddots & \\ & & e^{\lambda_n d} \end{pmatrix} V^{-1}$$

Since $log(M) = log(e^Q) = Q$ and $M = V\begin{pmatrix} \gamma_1 & & \\ & \ddots & \\ & & \gamma_n \end{pmatrix} V^{-1}$ it must hold that

$$\gamma_i = e^{\lambda_i d} \Rightarrow \lambda_i = \frac{\ln \gamma_i}{d}$$

For a continuous step in the Markovian model we had:

$$\underline{p}(t) = \underline{p}(0)e^{Qt}$$

Q we get with the Eigenvector matrix of M and the Eigenvalues γ_i of M : $Q = V\Lambda V^{-1}$, $\lambda_i = \frac{\ln \gamma_i}{d}$

Note again:

- $\gamma_1 = 1$
- $0 \leq \gamma_i \leq 1 \Rightarrow \lambda_i < 0$, λ's are negative
- $\boldsymbol{M^d = e^{Qd} = V\begin{pmatrix} 1 & & \\ & \ddots & \\ & & e^{\lambda_n d} \end{pmatrix} V^{-1}}$

- How to derive the actual M with all these properties we learn later (Chapter 'Substitution Matrices')

How can we interpret the Dayhoff-Matrix?

- Penalty for aligning some AA (e.g. $L \leftrightarrow G = -4.4$)
- Some AA seem to enjoy remaining the same (e.g. $C \leftrightarrow C = 11.5$)
- Other AA seem to mutate quite freqyently (e.g. $E \leftrightarrow D = 2.7$)
- But most AA mutate with a little bonus or deficit (e.g. $V \leftrightarrow T = 0.0$)

Summary

The matrix **M** is the mutation matrix of the Markovian Model. How to derive it, we see later. M_{ij} tells us how the probability is that one AA changes to another AA. With the relation $M = e^{Q}$ we can model a continuous Markovian Model.

The 1-PAM matrix is defined, such that 1% of an sequence changes after one mutation. A x-PAM matrix is simply $(1 - PAM)^{x}$. So x has nothing to do with percentage, it simply a evolutionary distance measurement.

If we want now compare two sequences regarding their evolutionary origin, especially if they have the same ancestor, we do that with the likelihood method. By doing so we derive the Dayhoff-matrix. We use therefore a 250-PAM matrix which corresponds to about 17% identity and compare it to the null-hypothesis that the two sequences are independent.

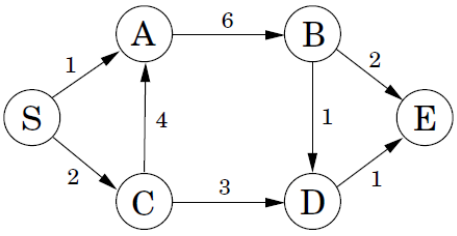
Dynamic Programming for sequence alignment

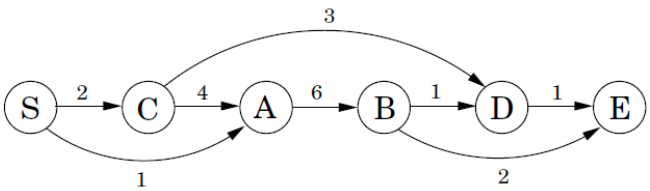
Since dynamic programming maximizes the sum of the similarity measure, dynamic programming maximizes the sum of the logarithms, i.e. maximizes the product of these quotients of probabilities. As a conclusion, dynamic programming finds the alignment which maximizes the probability of having evolved from a common ancestor (a maximum likelihood alignment) against the null hypothesis of being independent. This makes aligning sequences using Dayhoff matrices a soundly based algorithm (better founded than any other alignment algorithm).

Using Dynamic Programming for Sequence Alignment (Global Alignment). This is called the Needleman-Wunsch algorithm. It is dynamic programming applied to sequences using the scores of Dayhoff matrices.

Dynamic Programming

Let's assume the following graph and its linearization:





To find the shortest path to D we only have to compare all incoming arrows to D:

$$dist(D) = \min\{dist(B) + 1, dist(C) + 3\}$$

While we going from left-to-right we can always be sure that by the time we get to a node v, we already have all the information we need to compute dist(v). By starting with 0 at node S we can get this way the shortest path to each of the nodes.

The same principle works also for other problems. Instead of finding the min we could also search for the max, replace the + by a - or we could change the formula to something like:

$$L(j) = 1 + \max\{L(i): (i,j) \in E\}$$

which searches for the longest path in a sequence of random numbers, where the numbers on the path should increase along it.

The point about dynamic programming is:

> There is an ordering on the subproblems, and a relation that shows how to solve a subproblem given the answer to 'smaller' subproblems, that is, subproblems that appear earlier in the ordering.

Recursion

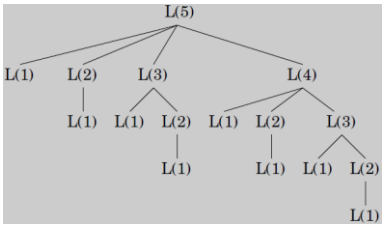
So dynamic programming is recursive? NO ITS NOT!!!

Recursion would mean that at each point we calculate all the possibilities of before again. This means

$$L(j) = 1 + \max\{L(i): (i,j) \in E\}$$

would look like

$$L(j) = 1 + \max\{L(1), \dots, L(j-1)\}$$

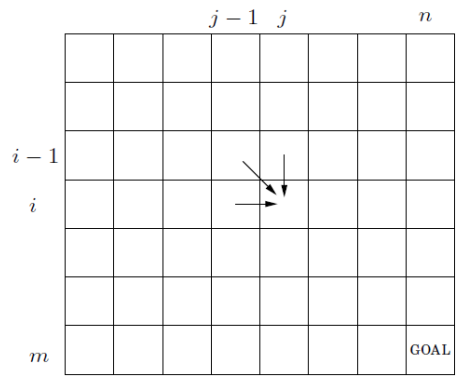


So for each node we calculate all the previous one. For L(n) this tree has exponentially many nodes which is disastrous. But why is recursion often a good thing? Recursion works good if the subproblem is substantially smaller, say half the size. This means the full recursion tree has only logarithmic depth and a polynomial number of nodes. But with dynamic programming the subproblems are only slightly smaller, e.g. L(j) relies on L(j-1). The full recursion tree generally has polynomial depth and an exponential number of nodes. However, it turns out that most of these nodes are repeats, that there are not too many distinct subproblems among them. Efficiency is therefore obtained by explicitly enumerating the distinct subproblems and solving them in the right order.

Edit distance

Our next task is to find the edit distance of two strings. This means the minimum number of insert, substitutions or deletions one string needs to be the other. This can be done by dynamic programming. But what is the subproblem to find the edit distance between x[1...m] and y[1...n]? The subproblem should go part of the way of solving the whole problem. In this case looking at the edit distance between some prefix of string one, x[1...i], and string two, y[1...j]. We call this subproblem E(i,j). Our goal is E(m,n).

(a)



(b)

		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

We have three possibilities for E(i,j):

1. x[i] is alone, which costs 1. We try to align x[1...i-1] to y[1...j]⇒add E(i-1,j)!
2. y[j] is alone, which costs 1. We try to align y[1...j-1] to x[1...i] ⇒add E(i,j-1)!
3. x[i] hits y[j] and is either the same (costs 0) or not (costs 1) plus adding E(i-1,j-1)

So in summary:

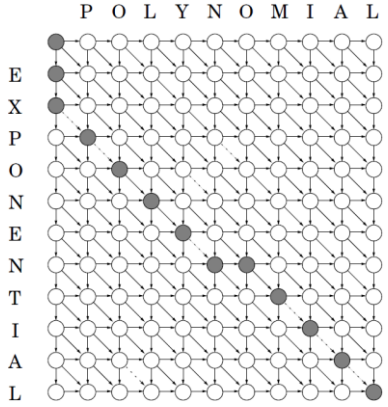
$$E(i,j) = \min\{1 + E(i-1,j), 1 + E(i,j-1), diff(i,j) + E(i-1,j-1)\}$$

where diff(i,j) is 0 when x[i]=x[j], else 1.

Whit that we are almost done. Only the smallest subproblems remain unsolved: E(0,*) and E(*,0). But what means this subproblem? It wants to compare the empty string to the prefix of length x. So the edit distance is clearly of length x. Therefore our algorithm is:

```
for i=0,..,m
      E(i,0)=0
for j=1,...,n
      E(0,j)=0
for i=1,...,m
      for j=1,...,n
            E(i,j)=min{1+E(i-1,j),1+E(i,j-1),diff(i,j)+E(i-1,j-1)}
return E(m,n)
```

The shortest path for our example has the length of 6 and the path looks like:



On this path each move down is a deletion, each move right an insertion and each diagonal move is either a match or a substitution.

Global alignment

Adapt dynamic programming for global sequence alignment.

Scoring matrix

In the introduction to dynamic programming we wanted to minimize the score for aligning strings. For gaps and mismatches we gave a penalty of 1. For matches we just counted 0. Of course we could define other weights. First of all we want to maximize the score in sequence alignment. This means a match scores a high value. When two AA hit each other with the same properties, we assign still a positive number. But when they do not match good we weight it negative. A gap will be weighted also negative. In our example this means that aligning an A with an A gives a score of 4, aligning an A with a C gives a score of -3, etc.

	A	C	D
A	4	-3	-1
C	-3	2	-4
D	-1	-4	1

Dayhoff matrix as scoring matrix

The alignment of sequences is done with dynamic programming which uses a scoring matrix to score each pair of amino acids and maximizes the sum of scores. The sums to be maximized are the sum of scores of the individual amino acids matched plus the penalties for any gaps needed. If we use Dayhoff matrices for scoring individual AAs, we find the following: let a_i and b_i be the amino acids of two sequences which are at position i of the alignment. Then, ignoring insertions/deletions, the alignment score is:

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

Under the assumption that positions evolve independently of each other, the product in the numerator represents the joint probability of the two sequences evolving from a common ancestor and the product in the denominator represents the probability of the two sequences being randomly aligned. The bottom probability does not depend on the alignment, it is constant for any two given sequences. The top part depends on the alignment.

Hence when 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.

This makes aligning sequences using Dayhoff matrices a soundly based algorithm, better founded than any other alignment algorithm. 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).

How to score insertions/deletions (indels)

How do we account for insertions/deletions when calculating the distance of alignments?

First idea

We want that opening a gap (the first indel) costs more than increasing an already open gap. That's why we assume a affine model:

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

where

- 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!!!

For DP this means that we would have to consider all rows & columns for each step, since the newest sub problem could score better with an k gap in any of the sequences 1 or 2 $\Rightarrow O(N^3)$. This is bad for computation.

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 $O(N^2)$.

Second idea

As we did with the pairing of amino acids, we want to compute the score from the ratio of the probabilities of two events:

(i) no relation, i.e. two random sequences (shown next left) and
(ii) both sequences evolved from the same ancestor (shown next right)

```
...AKPQLLT...
...AN__ILS...
```

No relation		Common ancestor		
A	A	A	$\leftarrow X_1 \rightarrow$	A
K	N	K	$\leftarrow X_2 \rightarrow$	N
P	I	P	inserted	
Q	L	Q	inserted	
L	S	L	$\leftarrow X_3 \rightarrow$	I
L		L	$\leftarrow X_4 \rightarrow$	L
T		T	$\leftarrow X_5 \rightarrow$	S

If there is no relation (and hence no common ancestor X) the event has as probability the product of all the frequencies:

$$\Pr(\overline{X}) = \left(f_A f_K f_P f_Q {f_L}^2 f_T\right) \cdot \left(f_A f_N f_I f_L f_S\right)$$

Note: the random development of ANILS has exactly the same probability as the development of NLISA. The order of the AAs does not play any role.

The probability of the sequences having X as a common ancestor is:

$$\Pr(X) = f_A M_{AA} \cdot f_K M_{NK} \cdot Pr(indel(2)) \cdot f_P \cdot f_Q \cdot f_L M_{IL} \cdot f_L M_{LL} \cdot f_T M_{ST}$$

where Pr(indel(2)) is the probability of an indel of length 2.

This formula comes from the use of the pair of amino acids formula (see above) applied to the five pairs and for the event of an indel of length 2 times the natural individual frequencies of the amino acids inserted. So the ratio to maximize is (rearranged conveniently)

$$\frac{\Pr(X)}{\Pr(\overline{X})} = \frac{f_A M_{AA}}{f_A f_A} \cdot \frac{f_K M_{NK}}{f_K f_N} \cdot \frac{Pr(indel(2)) \cdot f_P \cdot f_Q}{f_P f_Q} \cdot \frac{f_L M_{IL}}{f_L f_I} \cdot \frac{f_L M_{LL}}{f_L f_L} \cdot \frac{f_T M_{ST}}{f_T f_S}$$

and taking $10\ log_{10}$ we get:

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

We can generalize from this example that the cost of an indel of length k should be

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

This gives a precise basis to the assignment of indel costs. For example, we can collect large numbers of alignments and find the actual frequencies of indels which lets us estimate their probabilities and hence calculate their costs for dynamic programming.

For example, if we collect alignments of sequences (which we can trust as correct) and we have a total of 10 000 amino acids aligned and have found 31 indels of length 1, then the cost of an indel of length 1 should be $10\ log_{10}\left(\frac{31}{10\ 000}\right) \approx -25.09$.

Compare idea 1 and 2

With idea 1 we followed an intuitive idea which lead to the Hirschberg-Algorithm. In idea 2 we did it more accurate. Together they provide us a formula for the probability of an indel of length k:

$$Cost(indel(k)) = 10log_{10} \Pr(indel(k)) = c_0 + (k-1)c_1$$

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

This is a geometric distribution. In reality we observe a slightly other distribution: Zipf's distribution.

Probability of an indel of length k

The probability of indel's fallow Zipf's law.

Dynamic Programming algorithm

For the purpose of sequence alignment the algorithm looks the following way:

```
for i=0,..,m
      E(i,0)=0
for j=1,...,n
      E(0,j)=0
for i=1,...,m
      for j=1,...,n
            E(i,j)=min{E(i-1,j)+Del(x),
                       E(i,j-1)+Del(y),
                       match(x,y)+E(i-1,j-1)}
return E(m,n)
```

Where x is the last AA/base of sequence 1 and y the last AA/base of sequence 2. match(x,y) is a lookup function on the Dayhoff matrix.

Using this algorithm for two sequences is called the Needleman-Wunsch algorithm for global alignment. It scores poor if in the two sesquences are apart subsequences which are identical but in between these subsequences is garbage. To find such subsequences which match good we use local alignment.

Symmetry of Dynamic Programming

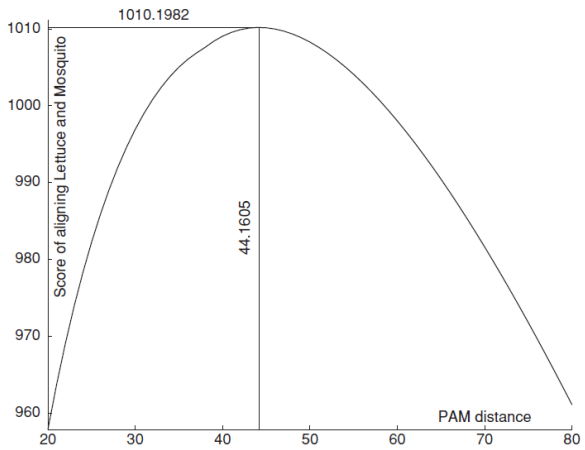
Doing DP on A^T is the same as doing DP on A from bottom-right to top-left.

Estimating distances between sequences by maximum likelihood

```
Lettuce (lactuca sativa), Mosquito (culex pipens)
lengths=165,165 simil=1010.2, PAM_dist=44.0217, identity=63.6%

VAAQNCWVKKGGAFTGEVSAEMLANLGVPWVILGHSERRALLNETNEFVGDKVAYALSQGLKVIACVGE
||||||:....|||||||!..||.:|.! ||||||||||:..|:.|.!.||!.:||::||||||||!||
VAAQNCYKVAKGAFTGEISPAMLKDLNIGWVILGHSERRAIFGESGELIADKIVHALAEGLKVIACIGE

TLEQREAGTTMEVVAAQTKAIADKISSWDNVVLAYEPVWAIGTGKVASPAQAQEVHAGLRKWFCDNVSA
||::|||..|..|. .|||||||!..|.|||:|||||||||||.|:|.|||||||.|||||.!||||
TLQEREAAKTEAVCFRQTKAIADKVKDWSNVVIAYEPVWAIGTGKTATPEQAQEVHAALRKWFTENVSA

EVSASTRIIYGGSVSGSNCKELGGQTD
!||::.|| |||||:.:||!||..:.|
DVSSAIRIQYGGSVTAANCRELAAKPD
```

The Dayhoff matrices and hence the scores (logarithms of probabilities) depend on the PAMdistance used for the mutation matrix M. Consequently we can choose the PAM distance for M which will maximize the score. As we know, maximum likelihood estimates are unbiased and best asymptotic estimates. Because it is unbiased, this estimate of the distance is additive for sufficiently long sequences.

The score for the alignment of lettuce and mosquito at different PAM distances is displayed in Figure 8.10. It can be seen that the maximum score is achieved at around 44 PAM. Using the second derivative of the logarithm of the likelihood we can estimate the variance of this PAM distance.



Local alignment

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), when these are maximized, but not with editing distances or penalty functions. 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 thewhole 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

Cost-Free End alignment (CFE)

A third variant of alignments is called cost-free end alignment (CFE). CFE is a method which can be viewed as in between global and local alignment. 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 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.)

Distance and variance matrices

When we have a set of n homologous sequences we can align each pair and compute its distance and also the variance of this distance. So we get the distance matrix:

	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

and variance matrix:

	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

Both are symmetric and very useful for computing phylogenetic trees.

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 ($10^6 - 10^8$, $human\ genome$: $3 * 10^9$)).

Algorithm: Dynamic programming

(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

Practical issues

1. A dim(s) x dim(d) matrix will be to 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.