Phylogenetic Trees – Part I (Distance-based methods)

Exercise 1: calculating sequence distances from multiple alignments

In the first two exercises, you will have (nearly) no need of your computer, but all the more for paper and pencil: we will try to obtain a **UPGMA** (**U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic **M**ean) tree from a multiple sequence alignment with the bottom-up approach.

In the previous lecture, you used different tools to produce multiple alignments of representatives of the **MFS-1** domain, based on a protein isolated from a 1000-year-old tooth plaque (dental calculus). Now, we will try to build *phylogenetic trees* to further investigate the evolutionary relationship between those different proteins. We will work with the following subset of proteins from the multiple alignment exercises (BLAST hits to the query protein MFS-1 domain, aligned with Clustal Omega):

```
Protein_Q   GDRVG RKFII WFSIL GTAPF ALWLP YAD-A DTTAI LVILI GFIIS SAFAS
Protein_A   GDRVG RKFII WFSIL GAAPF ALWLP YAD-A QTTAI LIVLI GFIIS SAFAS
Protein_B   GDKVG RKYII WFSVL GVAPF TMLLP YAS-L EWTGI LIVII GLILS SAFPS
Protein_C   GDRFG RKYVI WFSIL GTAPF ALMLP YAN-L FWTGV LIVPI GMILA SAFSA
Protein_D   GDRIG RKYVI WGSIL GVAPF TLILP YAS-L YWTGI LTVII GFILA SAFSA
```

First, we need to obtain a measure of *evolutionary distance* between these five proteins by calculating *sequence similarity* between them. For now, we will use a simple "similarity" measure: the ratio of shared vs. not-shared characters per position between the two sequences. This is also called the *Jaccard* index **J**:

$$J(A,B) = \frac{|\ A\ \cap\ B\ |}{|\ A\ \cup\ B\ |}$$

Note: $A \cap B$ is the intersection between the two sequences, i.e. the positions where the "character" between sequence A and B is equal (intersection) while $A \cup B$ the union, i.e. all the positions.

Using the Jaccard Index we can define a measure of distance, i.e. the larger the more distant:

$$d_j(A,B) = 1 - J(A,B)$$

Calculate the pairwise Jaccard distances and fill in the following distance matrix:

Distance matrix (D)

J distance	Q	A	B	C	D
Q	0.0				
A					
B					
C					
D					

Now think for a moment about the following questions:

- What is the computational *complexity* of distance matrix calculations: with increasing number of proteins, how will the run time increase? Assuming that the distance matrix for 1000 proteins takes 1s to compute, how long will it (roughly) take for 10.000, 100.000, 1.000.000, 10^9 proteins?
- As said before, the Jaccard index is quite simple, and probably provides a very inaccurate estimate of the evolutionary distance between two sequences. In which ways could the distance calculation be modified to provide a more realistic estimate of the "true" underlying evolutionary processes? Which parameters could have the strongest influence on the accuracy of any distance estimate?
- Protein sequences and nucleotide sequences (DNA/RNA) provide similar yet distinct types of information. What are the most striking differences between the two with regard to the calculation of evolutionary distances? For which kinds of problems or studies would you prefer one or the other, and why?

* Optional task:

- Can you write a Python script to compute the pairwise Jaccard distance J matrix for any number of sequences? You can use the sequences in the FASTA file `ex1_sequences.fa` to test your script.

Exercise 2: exploring the UPGMA (**U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic **M**ean) alg.

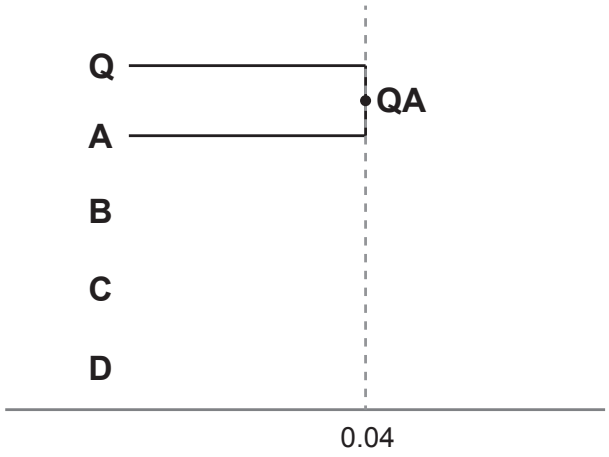
Based on the 5 x 5 distance matrix D you computed above, we will make a phylogenetic tree relating the proteins by using the **UPGMA** algorithm. We'll provide a step-by-step walkthrough here and if you like to dig deeper on the topic, you can check UPGMA on Wikipedia. A very nice and interactive explanation of the algorithm is also online (http://www.slimsuite.unsw.edu.au/teaching/upgma/).

Overview of the algorithm

We start with an unstructured, unrooted phylogenetic tree in which *n* sequences represent *n* leaves, while we do not know their appropriate connections and branch lengths. Given a pairwise distance matrix of size *n x n*, the related *n x n D matrix* of pairwise sequence relations (distances) can be computed. Based on the D matrix, the two most related leaves are joined at a tree *node*, and their respective distances from this node (the *branch lengths*) are computed. Subsequently, a new distance matrix is computed by calculating pairwise distances of all remaining elements to the newly formed node (the resulting new distance matrix is of size *(n-1) x (n-1)*). This procedure is repeated until all relations are resolved.

1. Choose first two elements (proteins) to join

Looking at the distance matrix D (the one you computed above), choose the two closest proteins (in terms of sequence distance) and create a new distance matrix D2. You will see that the closest two proteins in terms of J distance from matrix D are in fact proteins Q and A, so we will make a new "node" (cluster) named "QA" out of these two proteins and compute a new matrix D2 with distances from QA to all remaining proteins.



The branch lengths for A and Q are half the distance between A and Q (0.08 / 2 = 0.04).

2. Compute matrix D2 with updated distances from QA to remaining proteins B, C and D

In order to iterate and identify the shortest distance between nodes QA, B, C and D (4 nodes) we need to compute the new distance matrix D2. We always use the original D matrix as the basis to compute distances between individual single proteins. To compute the distance between already clustered nodes (nodes with >1 protein), we compute the mean distance between all possible protein pairs between the two nodes.

For example: if we would need to compute the distance between QAB and CD, we would take distances (Q,C) + (Q,D) + (A,C) + (A,D) + (B,C) + (B,D) and divide by 6 (mean). This would be the distance between nodes QAB and CD. This is just an example and you need to find out what proteins will be joined in the next step (which proteins will be joined next) by computing the distance matrix D2 (having nodes QA, B, C, D; so it will be of size 4 x 4).

3. Repeat

Repeat the joining of the proteins and nodes by computing a new distance matrix at each step (D2, D3, D4) and selecting the closest nodes until you reach the “root” node (QABCD). Draw the tree with all the distances.

 Optional task: write a complete Python script to compute a UPGMA tree from any number of sequences with the Jaccard measure.

Exercise 3: UPGMA computation and visualization with MAFFT web server

In this exercise you will use the MAFFT web server (https://mafft.cbrc.jp/alignment/server/) to create phylogenetic trees for the original MSF-1 proteins and visualize them.

Part 1: Conversion of the FASTA file using *Python*

In order to interpret our results more easily, we will first change the sequence identifiers to contain only the taxonomic names of each bacterial species. This information is already included in the sequence identifiers and is based on information from the NCBI taxonomy database (https://www.ncbi.nlm.nih.gov/taxonomy).

Task: Write a python script that modifies the aligned FASTA file `mfs_domain_proteins.fa` from the previous session to replace the header of each sequence with its taxonomic name. For UniProt headers, this name always follows the **OS=** prefix (field `OrganismName`) as defined in the FASTA specification (https://www.uniprot.org/help/fasta-headers).

Example:

```
>tr|A0A017H5R7|A0A017H5R7_9FUSO Acylglycerophosphoethanolamine
acyltransferase OS=Fusobacterium necrophorum subsp. funduliforme B35
OX=1226633 GN=FNF_06146 PE=4 SV=1
MLLTQRKFLPLFLTQFLGALNDNLLKMAIITFITYHLQGSLTEKGILISSVNVITILPMF
FISATAGQFADKFQRNSLVKIIKGIEILGIFFCIFFFYSGQYSLILLTLFVMSTRSAFFG
…
```

↓ [e.g. python modify_identifier.py mfs_domain_proteins.fa]

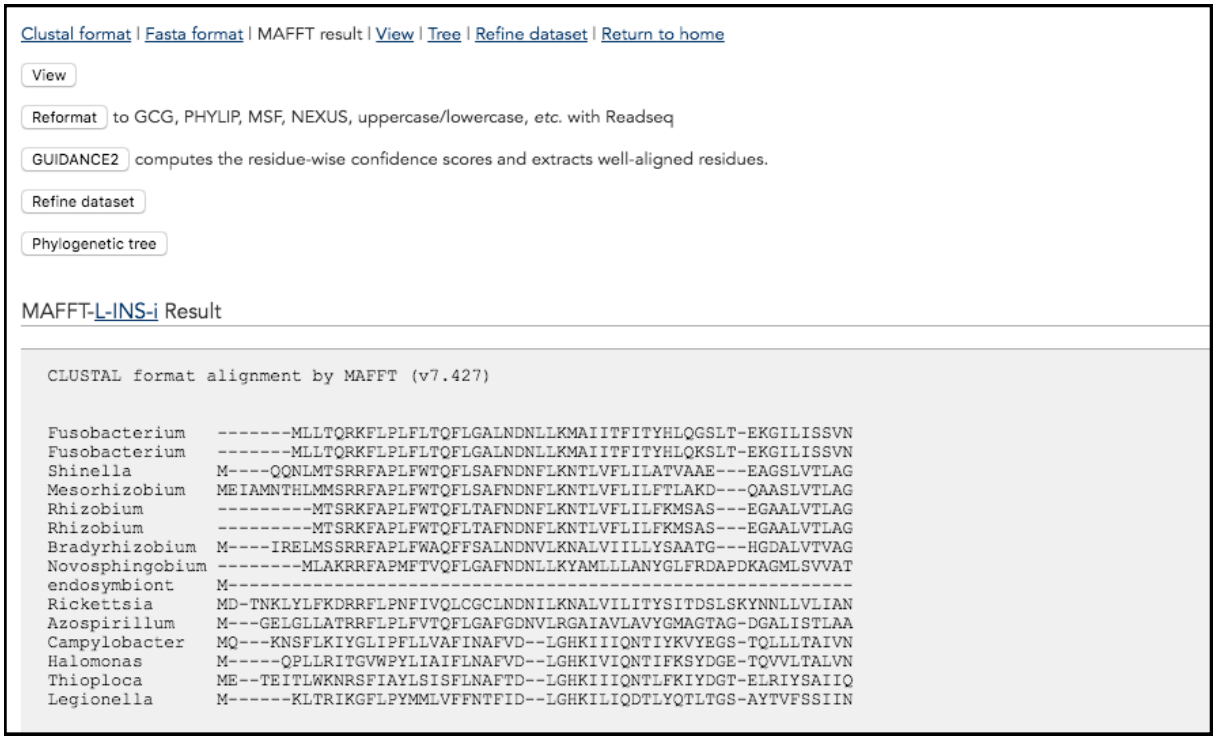
```
>Fusobacterium necrophorum subsp. funduliforme B35
MLLTQRKFLPLFLTQFLGALNDNLLKMAIITFITYHLQGSLTEKGILISSVNVITILPMF
FISATAGQFADKFQRNSLVKIIKGIEILGIFFCIFFFYSGQYSLILLTLFVMSTRSAFFG
…
```

Hint: the .split() function can split strings not only on single characters, but also on whole substrings (including "OS=" *and* "OX=" *!)*

Part 2: Creating a UPGMA tree with *MAFFT web server*
(In case the alignment with the *MAFFT web server* is not working, follow the instructions on page 8: "Running MAFFT alignment locally".)

Import the modified FASTA file into *MAFFT web server*. The platform will use MAFFT, an alternative multiple sequence alignment software tool (similar to Clustal Omega), to automatically align the input sequences for you. It offers a stunning range of alignment options, but we are content with leaving these at their default values. You can just click "Submit" and wait for your alignment results.

After a short while, a page similar to this should show up:



At the bottom of this page you can see the full multiple sequence alignment that MAFFT generated. This MSA will be used as the input for the phylogenetic tree computation in the next step. In order to proceed, now click the button “Phylogenetic tree”. *MAFFT web server* will offer you many options to fine-tune tree computations. However, in this exercise we are only interested in the “Method” parameter, which lets you select the algorithm to use. Select the option for UPGMA and click “Go!” to compute your tree.

Part 3: Tree visualization with *MAFFT web server*

The next window will offer you a number of ways to visualize and download the resulting tree. Click the “View tree on Phylo.io” button to reveal the final tree structure.

Take a bit of time to inspect the tree: How many major clades (distinct sub-trees) can you see? Which clade features, on average, the most distantly related members? What information do the taxonomic labels provide with regards to the taxonomic hierarchy?

Exercise 4: Tree building with the neighbor joining algorithm

Another, more frequently used algorithm for phylogenetic tree computation is Neighbor Joining (NJ, proposed by Saitou and Nei in 1987).

Try to repeat exercise 3 by selecting NJ as your clustering algorithm (using only conserved sites). You can then either use a new window or utilize the “Compare” tab in the Phylo.io tree viewer to compare the NJ tree to the previous UPGMA tree. Can you identify clear differences between the trees? From an evolutionary perspective, what do these differences imply? What is the most important algorithmic difference between NJ and UPGMA? (Wikipedia is your friend ;-))

Exercise 3: Running MAFFT alignment locally

In case the alignment with the *MAFFT web server* is not working, we install and run MAFFT locally. We can install MAFFT easily via Anaconda using the following commands.

1. Set up the conda environment

If you have not previously created your local conda environment, you can run:

```
. /opt/miniconda3/etc/profile.d/conda.sh
conda create –prefix ~/Documents/py39_envs python=3.9.12
```

Then, simply activate your local conda environment and install MAFFT:

```
conda activate ~/Documents/py39_envs
conda install bioconda::mafft
```

2. Using MAFFT

You can acquaint yourself with the arguments for MAFFT by typing:

```
mafft --help
```

For the purpose of this exercise, you can invoke MAFFT with the following command:

```
mafft --auto input.fa > output_aligned.fa
```

Make sure to adjust the filenames (`input.fa` and `output_aligned.fa`) to something more meaningful.

3. Creating a UPGMA tree with MAFFT web server

1. Navigate to the *Phylogeny* tab and upload or paste your previously created alignment. You can just click “Submit” and wait for your alignment results.
2. Continue on page 6.