

# Homework 3



Computational Biology course

Handed out: 28.10.2019

Due date: 11.11.2019

## Theory questions

We encourage you to answer these questions only after you have completed the programming task. Please keep your answers short, most of the questions can be answered in one or two sentences. Submit your answers in a pdf file named following the format Lastname\_Firstname.pdf.

1. What happens if you try to compute the K80 distance between two very dissimilar sequences? Take for instance the sequences *AACTCA* and *TTAGTG*. 
2. Assume you are using the UPGMA algorithm, and your initial distance matrix has multiple equal minimal entries. Does your implementation of the algorithm influence the output tree? 
3. Say we know that several sampled sequences have evolved under a JC69 model. We calculate a JC69 distance matrix for these sequences, apply the UPGMA algorithm to construct a tree, and calculate distances from the tree. However, our tree distances do not match the JC69 distances. Name two features of the sampling scheme or the true evolutionary process that might cause such a discrepancy. (2 points)
4. Can any of the potential problems you raised in your answer to question 3 be taken into account using an alternative algorithm mentioned in the lectures? If yes, which problem(s) and with which algorithm? If no, please give a short justification.

## Programming task

Submit your solution in a R file named following the format Lastname\_Firstname.R.

## Task description

In this homework you will implement the UPGMA algorithm to reconstruct an ultrametric tree from a given alignment of sequences. The objective is to be able to reconstruct the tree from an *arbitrary* given multiple sequence alignment using three different pairwise distance measures: Hamming distance, distance under the JC69 substitution model, and distance under the K80 model.

As input your code should take:

1. **sequences**: a list that holds the alignment of sequences with their names, e.g. `list(human = "AACTC", chimp = "AAGTC", orangutan = "TTAGT");`
2. **distance\_measure**: a label for the required distance measure, which can take one of the values "hamming", "JC69", "K80".

The input alignment will contain sequences that are properly aligned, are of the exact same length and the alignment will not contain any gaps.

As output the script should return the UPGMA tree constructed from the given alignment under the specified distance measure. A detailed step-by-step description and pseudocode overview of the algorithm are provided below.

Your script should follow the structure that we have laid out in the skeleton script called `CB_HW3_TreeBuilding_skeleton.R`. This skeleton splits the assignment into smaller functions that are necessary to compute the final result. Each function is listed with the input and output that it requires, as well as a short description of its task. You just have to fill in the missing code (marked with `# ???`).

Your implementation will center around a function called `upgma_one_step`, which performs one step of the UPGMA algorithm. Each time this function is run, it adds one additional (internal) node and two edges to the UPGMA tree. This is accomplished by adding a new row to a matrix named `edges` and a new element to a vector named `edge_lengths`. The `upgma_one_step` should also return a variable called `node_description`, which will contain the descriptions of nodes in the tree, their names, sizes and heights (distance to tip).

All of these structures are already initialized in the skeleton, and **must** be used in your solution. Additionally, we provide the function that defines the node description structure, as well as a function `add_new_node()` that generates new node names for a merged node and adds that node to the data structure of node descriptions. Of course, the predefined functions will not be tested and graded.

The final edge definitions, the edge lengths, and the final node description structure you compute will be transformed by the predefined `transform_to_phylo(sequences, edges, edge_lengths, node_description)` function into a phylogenetic tree object as defined by the `phylo` class in the `ape` package. You can plot the tree using the predefined `plot_tree(tree)` function.

Please do not change the structure of the code and the inputs and outputs of functions. The code will be tested and graded automatically and any structural changes will result in your code failing the tests.

To help you understand the skeleton structure, we provide a cross-reference table showing which functions in the skeleton should use which others (see Table 1).

We will also provide you with a test suite which is very similar to the one we will use to grade the homework. You can use that to verify that your code is performing as expected. Bear in mind that the suite used to grade your code will have a similar structure but will contain different parameters and may include more test conditions.

Function name	Uses
<code>build_upgma_tree</code>	<code>compute_initial_distance_matrix</code> <code>upgma_one_step</code>
<code>compute_initial_distance_matrix</code>	<code>get_hamming_distance</code> <code>get_JC69_distance</code> <code>get_TN93_distance</code>
<code>upgma_one_step</code>	<code>update_distance_matrix</code>
<code>update_distance_matrix</code>	<code>get_merge_node_distance</code>

Table 1: Function cross-reference table

## Tree reconstruction

### UPGMA pseudocode

In this section we will use the following definitions to simplify the pseudocode:

1.  $N$ : number of sequences in alignment (number of tips in the tree);
2.  $s_i$ : cluster name;
3.  $n_i$ : cluster size;
4.  $D$ : distance matrix;
5.  $d[x, y]$ : distance matrix entry for nodes  $x$  and  $y$ ;
6.  $\text{branch}(x, y)$ : branch length between nodes  $x$  and  $y$ ;
7.  $\text{distance\_to\_tip}(x)$ : distance from the node  $x$  to any of the child tips.

```

Data: Distance matrix  $D$ 
Result: Ultrametric phylogenetic tree
for  $i \leftarrow 1$  to  $N$  do
  |  $n_i \leftarrow 1$ ;
  |  $s_i \leftarrow$  label of sequence  $i$ 
end
while  $\text{size}(D) > (1, 1)$  do
  | Choose  $s_i, s_j$  such that  $\min(D) = d[s_i, s_j]$ ;
  |  $n_{i,j} \leftarrow n_i + n_j$ ;
  |  $s_{i,j} \leftarrow \{s_i, s_j\}$ ;
  |  $\text{branch}(s_{i,j}, s_i) \leftarrow d[s_i, s_j]/2 - \text{distance\_to\_tip}(s_i)$ ;
  |  $\text{branch}(s_{i,j}, s_j) \leftarrow d[s_i, s_j]/2 - \text{distance\_to\_tip}(s_j)$ ;
  | for all  $m \neq i$  and  $m \neq j$  do
  | |  $d[s_m, s_{i,j}] \leftarrow \frac{n_i d[s_i, s_m] + n_j d[s_j, s_m]}{n_i + n_j}$ ;
  | end
  | Delete node  $s_i$  from  $D$ ;
  | Delete node  $s_j$  from  $D$ ;
end

```

## Distance definitions

**Hamming distance** The Hamming distance between two sequences is computed as the total number of positions in the sequence alignment where the nucleotides are different. For example, the Hamming distance between the following two sequences is 4:

AGGTGGATAC

ACATGAATAT

**JC69 distance** Under the JC69 model the distance between two aligned sequences is computed using the formula:

$$\hat{d} = -\frac{3}{4} \log \left( 1 - \frac{4}{3}p \right)$$

where  $p$  is the proportion of sites that are different between the two sequences<sup>1</sup>. Thus, if  $H$  is the Hamming distance between the sequences and  $L$  is the (identical) length of the sequences,  $p = \frac{H}{L}$ .

**K80 distance** Under the K80 model the distance between two aligned sequences is computed using the formula:

---

<sup>1</sup>Please note that the logarithm,  $\log(\cdot)$ , is used here as a shorthand for logarithm with base  $e$ ,  $\log_e(\cdot)$ . Another common notation for the logarithm with base  $e$  is  $\ln(\cdot)$ .

$$\hat{d} = -\frac{1}{2} \log(1 - 2S - V) - \frac{1}{4} \log(1 - 2V)$$

where

$S$  = proportion of sites with transitional differences ( $T \leftrightarrow C$ ,  $A \leftrightarrow G$ );

$V$  = proportion of sites with transversional differences ( $A \leftrightarrow C$ ,  $A \leftrightarrow T$ ,  $G \leftrightarrow C$ ,  $G \leftrightarrow T$ );

## Additional material and help

### Required Packages

This homework assignment requires that the following pair of packages is installed on your R system:

**ape** : a library providing data types and methods suitable for phylogenetics, and

**Matrix** : a library containing various linear algebra functions.

To install these packages, simply enter the following command in the R command line:

```
install.packages(c("ape", "Matrix"))
```

Remember that you only need to install a package once and then you can load it for use in your code using the `library("PackageName")`. The homework skeleton loads the two aforementioned packages for you.

### Useful functions

We provide a list of R functions that you may find useful when writing your code (Table 2).

R function/operator	Description
<code>rownames(M)</code>	vector of row names of matrix <b>M</b> .
<code>colnames(M)</code>	vector of column names of matrix <b>M</b> .
<code>a %in% v</code>	Returns true if and only if the value of <b>a</b> is in the vector <b>v</b> .

Table 2: Some R functions that may be useful.

### Completed data structure example

Using the Hamming distance measure to reconstruct the tree of the 3 example sequences (`list(human = "AACTCA", chimp = "AAGCCA", orangutan = "TTAGTG")`), the completed matrix of edges (as generated by your completed solution) should look as follows:

	[,1]	[,2]
[1,]	"chimp.human"	"chimp"
[2,]	"chimp.human"	"human"
[3,]	"chimp.human.orangutan"	"chimp.human"
[4,]	"chimp.human.orangutan"	"orangutan"

In this table, the first column represents the parent node (the older node) of an edge and the second column represents the corresponding child node (the younger node) of the same edge. I.e. there are edges from node `chimp.human` to node `chimp`, `chimp.human` to `human`, `chimp.human.orangutan` to `chimp.human`, and `chimp.human.orangutan` to `orangutan`. Note that parent nodes have names that are constructed by concatenating the names of their children together, separated by a “.”. These names are created automatically by the provided function `add_new_node()`. Be aware that the exact ordering of the child names in this concatenation does not matter and may vary depending on your implementation.

The vector of edge lengths will look as follows:

```
[1] 1.0 1.0 2.0 3.0
```

Each position in the vector corresponds to the respective row in the edge definition matrix.

The corresponding node description data frame will look as follows:

	node_sizes	node_heights
human	1	0.0
chimp	1	0.0
orangutan	1	0.0
chimp.human	2	1.0
chimp.human.orangutan	3	3.0

In this data frame, each row corresponds to a node in the final tree. The rows, labelled with the node names, contain the node sizes (i.e. the number of tips the node is ancestral to) and the node heights (i.e. the distance from the node to the present time, or to any of the tips).

Figure 1 may be helpful in getting a better understanding of the relationships between the node names, and the node heights, and the tree. Note that the two representations of the tree on the left of the figure are equivalent. In both cases, the distance of the node above the lowest node on the tree (its “height”) represents the age of the node relative to the present day sequence samples.

Once these structures are complete, the completed implementation of the `build_upgma_tree()` will transform them into an APE phylogenetic tree object, which can then be plotted using the `plot_tree()` function defined in the skeleton. For this example, this will result in something similar to Figure 2.

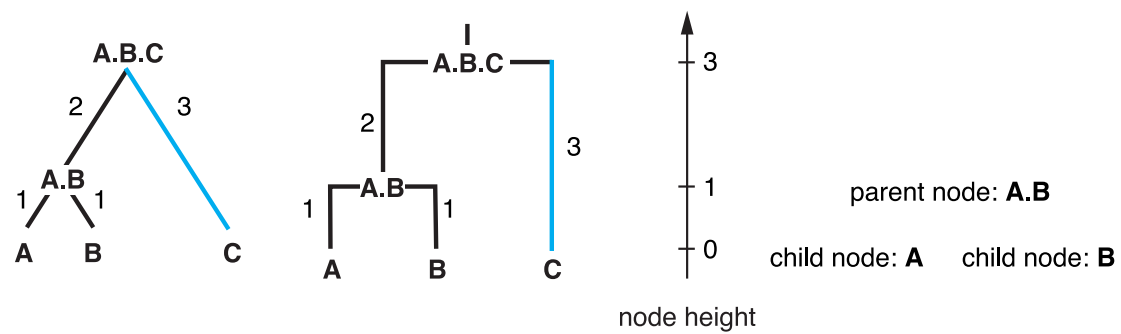


Figure 1: UPGMA nomenclature.

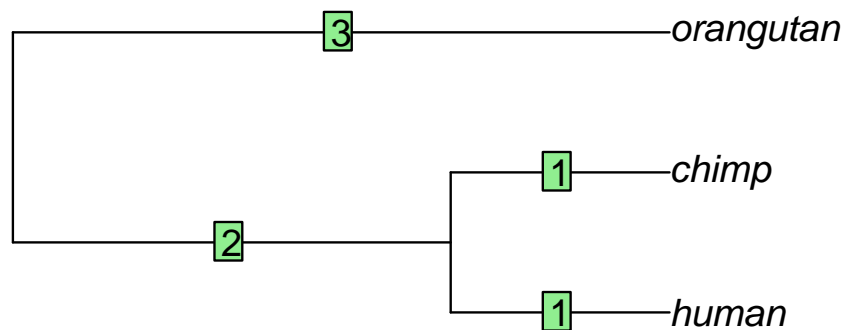


Figure 2: Example UPGMA tree built from the short example sequences included in this text using the hamming distance measure. The green labels show the length of each edge.