Article

Fernando Freire

December 2, 2018

Contents

1 Phylogenetics Evaluation **2**
1.1 Best-fit substitution model . . . 2
1.1.1 Selected models from jModelTest . . . 2
1.1.2 Selected model for Prottest . . . 3
1.2 Maximum parsimony (MP) tree. . . . 5
1.2.1 16S nucleotides . . . 5
1.2.2 RAG1 nucleotides . . . 5
1.2.3 RAG1 amino acid . . . 6
1.3 Distances (Neighbor Join) trees . . . 7
1.3.1 16S nucleotides . . . 7
1.3.2 RAG1 nucleotides . . . 8
1.3.3 RAG1 amino acid . . . 9
1.4 Maximum likelihood (ML) trees . . . 10
1.4.1 16S nucleotides . . . 10
1.4.2 RAG1 nucleotides . . . 11
1.4.3 RAG1 amino acid . . . 12
1.5 Maximum Likelihood by PhyML . . . 13
1.6 First conclusions . . . 15
1.7 Another approach: Homemade phylogenies . . . 16
1.8 Final conclusions . . . 16
1.8.1 Trees for RAG1 amino acid . . . 17
1.8.2 Trees for RAG1 Nucleotides . . . 18
1.8.3 Trees for 16S Nucleotides . . . 19
1.9 Class AlignSequences . . . 20
1.10 MSA . . . 32
1.11 MSA generic methods . . . 33
1.12 T-COFFEE methods . . . 42
1.13 Main MSA method . . . 46
1.14 T-COFFEE RAG1_AA EXEC . . . 50
1.15 CLUSTAL 16S EXEC . . . 55

1 Phylogenetics Evaluation

For each of the three matrices (16S, RAG1 nucleotides, RAG1 amino acids) present:

- Best-fit substitution model.
- Maximum parsimony (MP) tree.
- Distances (NJ) tree.
- Maximum likelihood (ML) tree.
- For every tree, show bootstrap values as well.
- Brief discussion of the results: comparison of topologies, branch lengths, support of phylogenetic relationships and resolution.
- Compare the trees of the three matrices among them, and also with the reference phylogeny.

1.1 Best-fit substitution model

1.1.1 Selected models from jModelTest

We use **jModelTest** to obtain the most appropriate model of nucleotide evolution given the sequence data of RAG1 and 16S.

These are the versions involved:

```
Phyml version = 3.0
Phyml binary = PhyML_3.0_macOS_i386
jModelTest version = 2.1
OS:       Mac OS X (10.14.1)
Arch:     x86_64
Java:     1.8.0_144 (Oracle Corporation)
```

16S: GTR+I+G The models selected under AIC and BIC criterions are equivalent:

```
Model   GTR+I+G
partition   012345
-lnL    13398.4723
K   27
freqA   0.3697  R(a)    4.4083
freqC   0.2395  R(b)    6.5698
freqG   0.1687  R(c)    5.3153
freqT   0.2222  R(d)    0.3201
ti/tv   -   R(e)    20.8173
R(f)    1.0000
p-inv   0.1950  gamma   0.9680
```

RAG1: GTR+I+G The models selected under AIC and BIC criterions are equivalent:

```
Model   GTR+I+G
partition   012345
-lnL    10202.7853
K   27
freqA   0.2859  R(a)    2.8475
```

```
freqC   0.2349  R(b)    7.3488
freqG   0.2584  R(c)    3.1158
freqT   0.2208  R(d)    1.1479
ti/tv   -   R(e)    11.5334
R(f)    1.0000
p-inv   0.3840  gamma   1.2830
```

1.1.2 Selected model for Prottest

We use **ProtTest3** to infere the most appropriate model of protein evolution given the sequence data of RAG1 amino acid.

These are the versions employed:

```
Version:  3.4.2 : 8th May 2016
OS:       Mac OS X (10.14.1)
Arch:     x86_64
Java:     1.8.0_144 (Oracle Corporation)
```

RAG1 JTT+I+G Prottest results:

```
**************************************************************************
Best model according to BIC: JTT+G
Confidence Interval: 100.0
**************************************************************************
Model          deltaBIC       BIC            BICw           -lnL
--------------------------------------------------------------------------
JTT+G          0.00           8062.58        0.76           3975.25
JTT+I+G        2.33           8064.91        0.24           3973.30
...
*********************************************
Relative importance of parameters
*********************************************
  alpha       (+G):     0.763
  p-inv       (+I):     0.000
  alpha+p-inv (+I+G):   0.237
  freqs       (+F):     No +F models

*********************************************
Model-averaged estimate of parameters
*********************************************
  alpha (+G):           0.379
  p-inv (+I):           0.520
  alpha (+I+G):         1.190
  p-inv (+I+G):         0.380


**************************************************************************
Best model according to LnL: JTT+I+G
```

```
Confidence Interval: 100.0
*************************************************************************
Model          deltaLnL       LnL            LnLw           -lnL
-------------------------------------------------------------------------
JTT+I+G        0.00           3973.30        0.71           3973.30
JTT+G          1.95           3975.25        0.27           3975.25
LG+I+G         7.29           3980.59        0.02           3980.59
...
*********************************************
Relative importance of parameters
*********************************************
  alpha       (+G):     0.276
  p-inv       (+I):     0.000
  alpha+p-inv (+I+G):   0.724
  freqs       (+F):     No +F models

*********************************************
Model-averaged estimate of parameters
*********************************************
  alpha (+G):           0.379
  p-inv (+I):           0.520
  alpha (+I+G):         1.183
  p-inv (+I+G):         0.379
```

The models are estimated under Maximum Likelihood LnL information criterion and under Bayesian Information Criterion. Both criterion selected as the best substitution models: JTT+I+G and JTT+G, exchanging the rank between them.

We launch another run, now with the graphical interface, and seek for AIC criterion (the previous run is launched in console with this command line `java -jar prottest-3.4.jar -i HRSV-A_aa.phy -all-matrices -all-distribution`).

The result is:

```
*****************************************************
              AKAIKE INFORMATION CRITERION
*****************************************************

*************************************************************************
Best model according to AIC: JTT+I+G
Confidence Interval: 100.0
*************************************************************************
Model          deltaAIC       AIC            AICw           -lnL
-------------------------------------------------------------------------
JTT+I+G        0.00           7984.61        0.59           3973.30
JTT+G          1.89           7986.50        0.23           3975.25
...
```

So, by consensus, we choose the **JTT+I+G** model.

1.2 Maximum parsimony (MP) tree.

I use the MEGA v7.0 program.

1.2.1 16S nucleotides

The bootstrap consensus tree inferred from 1000 replicates. The MP tree was obtained using the Tree-Bisection-Regrafting (TBR) algorithm with search level 1. Analysis Settings:

```
Analysis                                =  Phylogeny Reconstruction
Statistical Method                      =  Maximum Parsimony
Test of Phylogeny                       =  Bootstrap method
No. of Bootstrap Replications           =  1000
Substitutions Type                      =  Nucleotide
Gaps/Missing Data Treatment             =  Use all sites
No. of Initial Trees (random addition) =  10
MP Search level                         =  1
Max No. of Trees to Retain              =  10
```

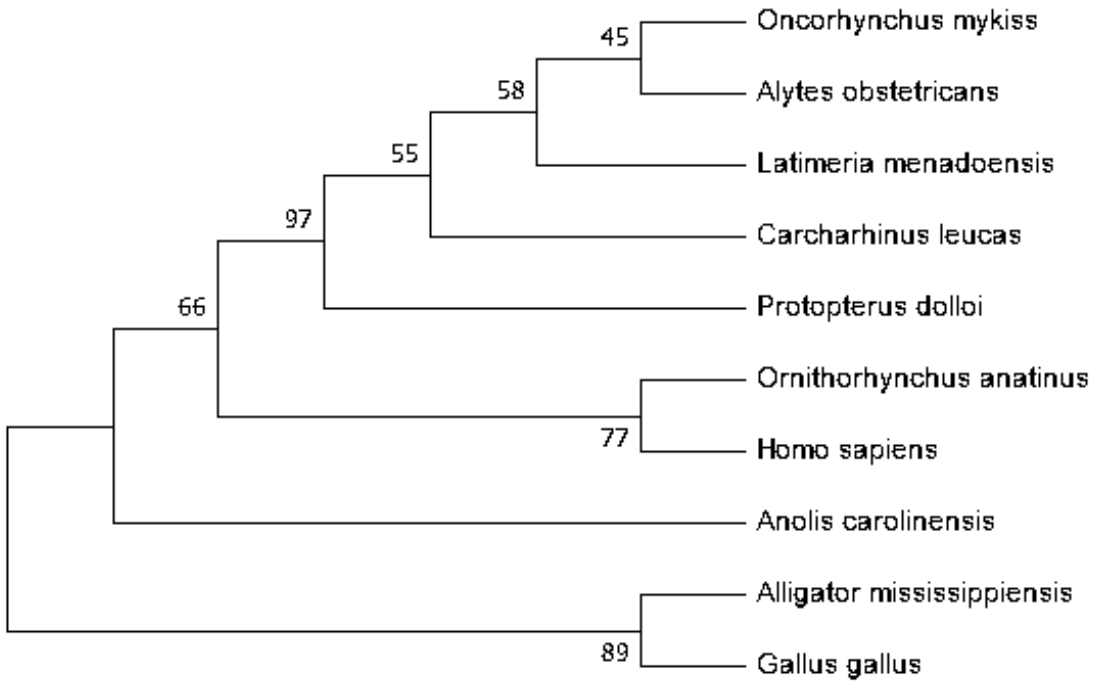


Figure 1: 16S. Maximum parsimony tree

1.2.2 RAG1 nucleotides

The bootstrap consensus tree inferred from 1000 replicates. The MP tree was obtained using the Tree-Bisection-Regrafting (TBR) algorithm with search level 1. Analysis Settings:

```
Analysis                                =  Phylogeny Reconstruction
```

```
  Statistical Method                    =  Maximum Parsimony
  Test of Phylogeny                     =  Bootstrap method
  No. of Bootstrap Replications         =  1000
  Substitutions Type                    =  Nucleotide
  Gaps/Missing Data Treatment           =  Use all sites
  No. of Initial Trees (random addition) =  10
  MP Search level                       =  1
  Max No. of Trees to Retain            =  100
Codons Included                         =  1st+2nd
```

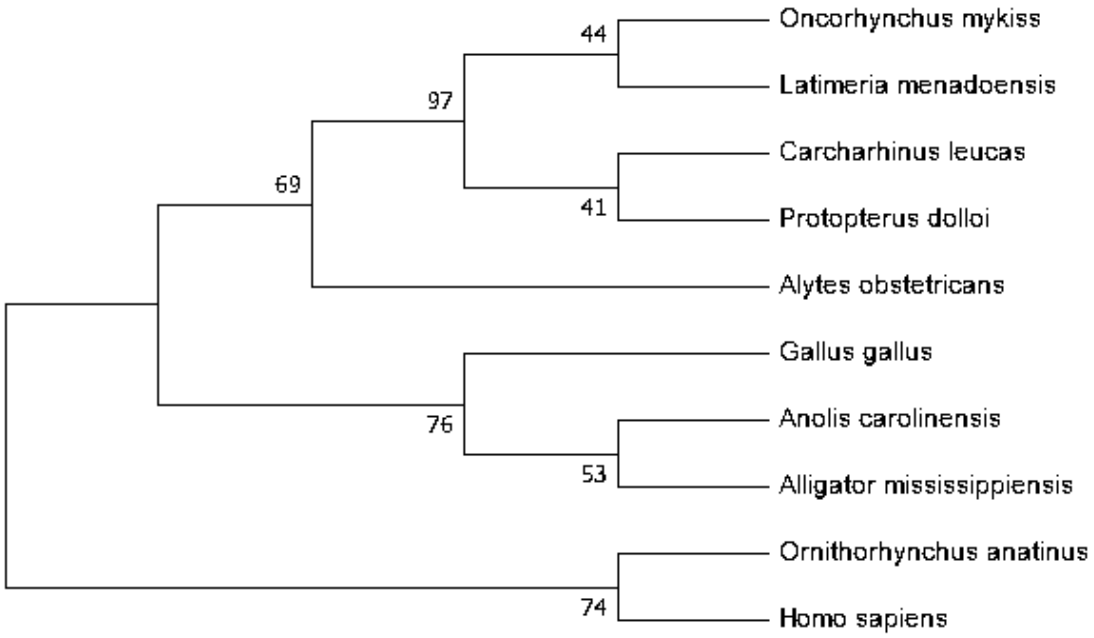


Figure 2: RAG1 nucleotides. Maximum parsimony tree

1.2.3 RAG1 amino acid

The bootstrap consensus tree inferred from 1000 replicates. The MP tree was obtained using the Tree-Bisection-Regrafting (TBR) algorithm with search level 1. Analysis Settings:

```
  Analysis                              =  Phylogeny Reconstruction
  Statistical Method                    =  Maximum Parsimony
  Test of Phylogeny                     =  Bootstrap method
  No. of Bootstrap Replications         =  1000
  Substitutions Type                    =  Amino acid
  Gaps/Missing Data Treatment           =  Use all sites
  No. of Initial Trees (random addition) =  10
  MP Search level                       =  1
  Max No. of Trees to Retain            =  100
```

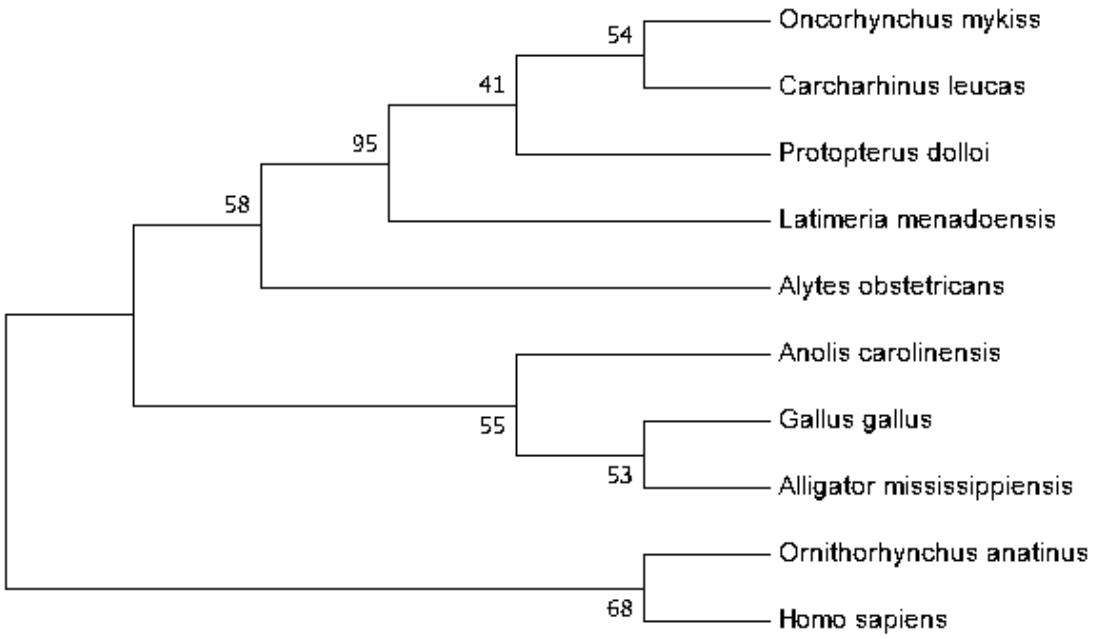


Figure 3: RAG1 AA. Maximum parsimony tree

1.3 Distances (Neighbor Join) trees

I use the MEGA v7.0 program.

1.3.1 16S nucleotides

The evolutionary distances were computed using the Kimura 2-parameter method. Analysis Settings:

```
Analysis                                =  Phylogeny Reconstruction
Scope                                   =  All Selected Taxa
Test of Phylogeny                       =  Bootstrap method
No. of Bootstrap Replications           =  1000
Substitutions Type                      =  Nucleotide
Substitutions to Include                =  d: Transitions + Transversions
Rates among Sites                       =  Gamma Distributed (G)
Gamma Parameter                         =  0.817
Pattern among Lineages                  =  Same (Homogeneous)
Gaps/Missing Data Treatment             =  Complete deletion
```

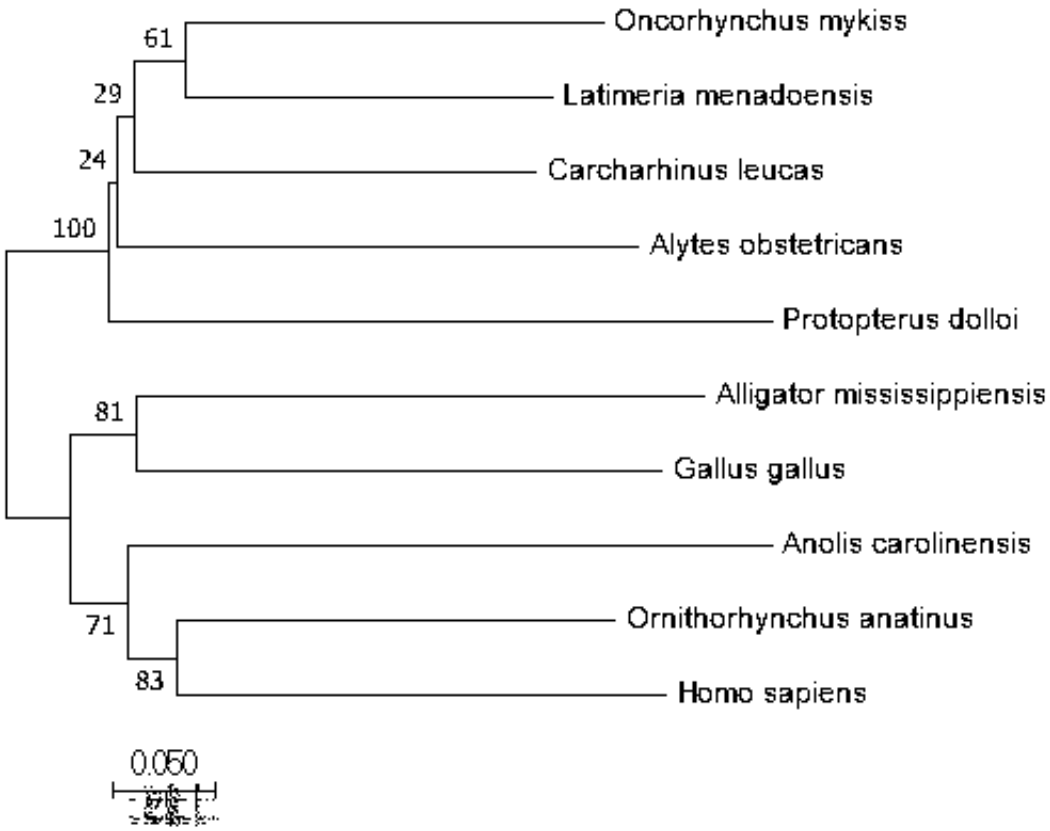


Figure 4: 16S. Neighbor join optimal tree. Sum of branch length = 2.66249603

1.3.2 RAG1 nucleotides

The evolutionary distances were computed using the Kimura 2-parameter method. Analysis Settings:

```
Analysis                              =  Phylogeny Reconstruction
Scope                                 =  All Selected Taxa
Test of Phylogeny                     =  Bootstrap method
No. of Bootstrap Replications         =  1000
Substitutions Type                    =  Nucleotide
Substitutions to Include              =  d: Transitions + Transversions
Rates among Sites                     =  Gamma Distributed (G)
Gamma Parameter                       =  0.817
Pattern among Lineages                =  Same (Homogeneous)
Gaps/Missing Data Treatment           =  Complete deletion
```

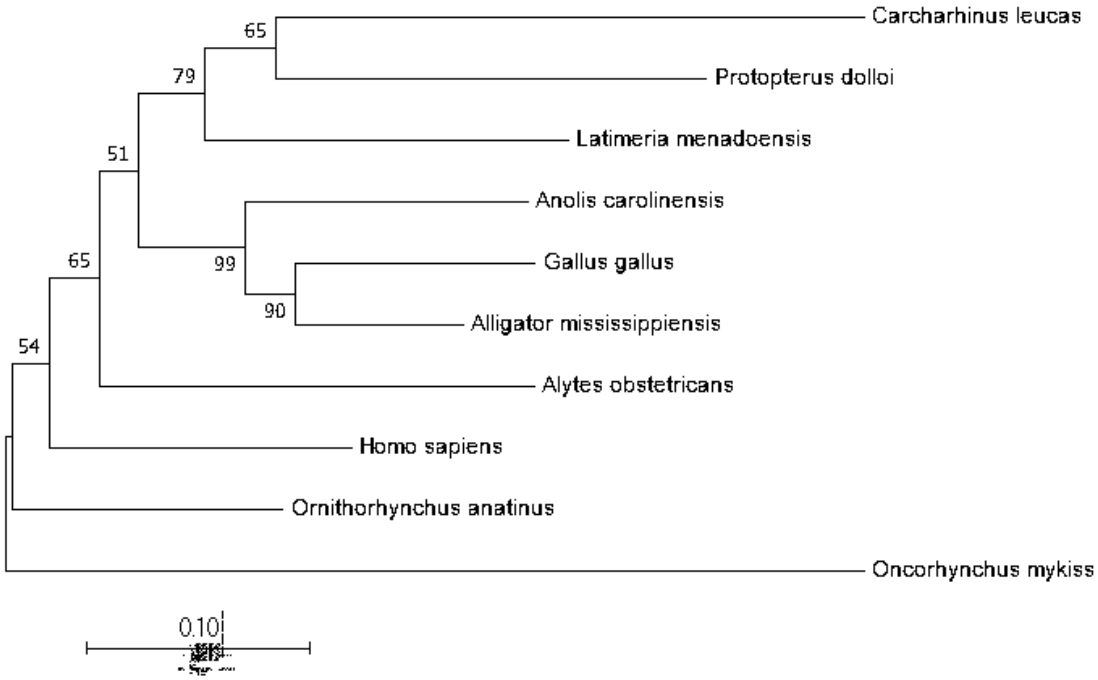


Figure 5: RAG1 nucleotides. Neighbor join optimal tree. Sum of branch length = 1.95639526

1.3.3 RAG1 amino acid

Analysis Settings:

```
Analysis                              =  Phylogeny Reconstruction
Scope                                 =  All Selected Taxa
Test of Phylogeny                     =  Bootstrap method
No. of Bootstrap Replications         =  1000
Substitutions Type                    =  Amino acid
Model/Method                          =  Poisson model
Rates among Sites                     =  Gamma Distributed (G)
Gamma Parameter                       =  0.817
Pattern among Lineages                =  Same (Homogeneous)
Gaps/Missing Data Treatment           =  Complete deletion
```

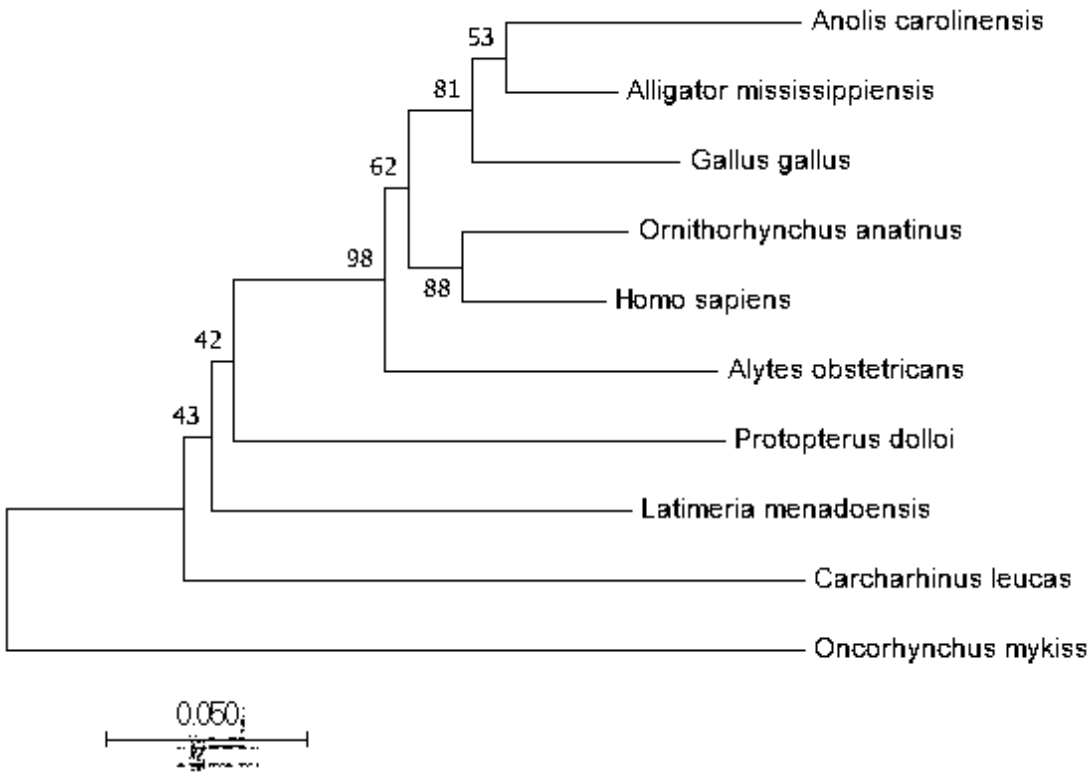


Figure 6: RAG1 AA. Neighbor join optimal tree. Sum of branch length = 1.0316

1.4 Maximum likelihood (ML) trees

I use the MEGA v7.0 program.

1.4.1 16S nucleotides

The bootstrap consensus tree inferred from 1000 replicates. The tree is computed by Maximum Likelihood method based on the Tamura-Nei. Initial tree(s) for the heuristic search were obtained automatically by applying Neighbor-Join and BioNJ algorithms to a matrix of pairwise distances estimated using the Maximum Composite Likelihood (MCL) approach, and then selecting the topology with superior log likelihood value.

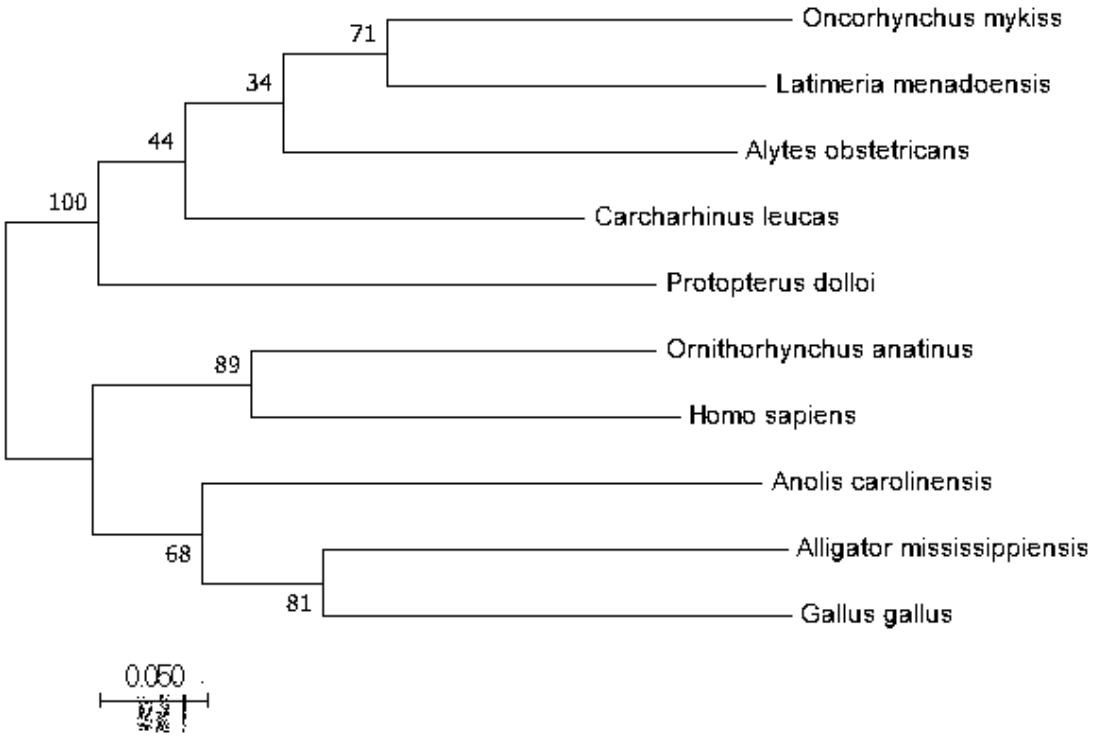


Figure 7: 16S. Maximum likelihood tree

1.4.2 RAG1 nucleotides

The bootstrap consensus tree inferred from 1000 replicates. The tree is computed by Maximum Likelihood method based on the Tamura-Nei. Initial tree(s) for the heuristic search were obtained automatically by applying Neighbor-Join and BioNJ algorithms to a matrix of pairwise distances estimated using the Maximum Composite Likelihood (MCL) approach, and then selecting the topology with superior log likelihood value.

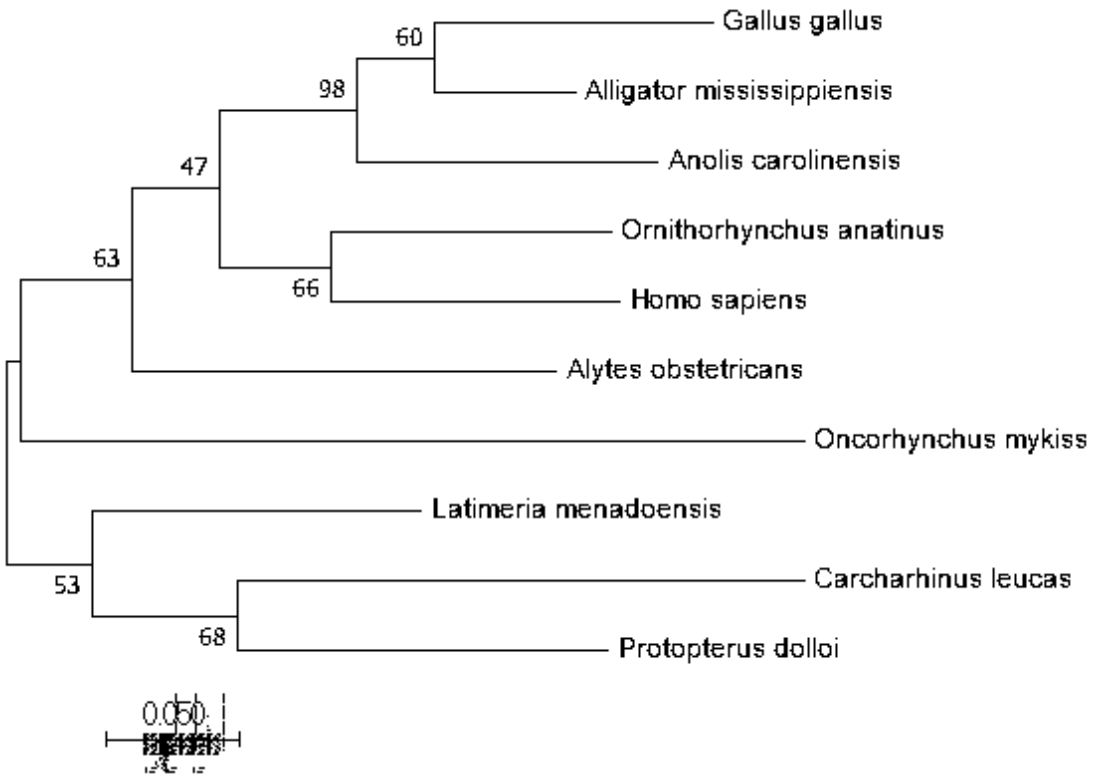


Figure 8: RAG1 nucleotides. Maximum likelihood tree

1.4.3 RAG1 amino acid

The bootstrap consensus tree inferred from 1000 replicates. The tree is computed by Maximum Likelihood method based on Equal Input model. Initial tree(s) for the heuristic search were obtained automatically by applying Neighbor-Join and BioNJ algorithms to a matrix of pairwise distances estimated using the Maximum Composite Likelihood (MCL) approach, and then selecting the topology with superior log likelihood value.

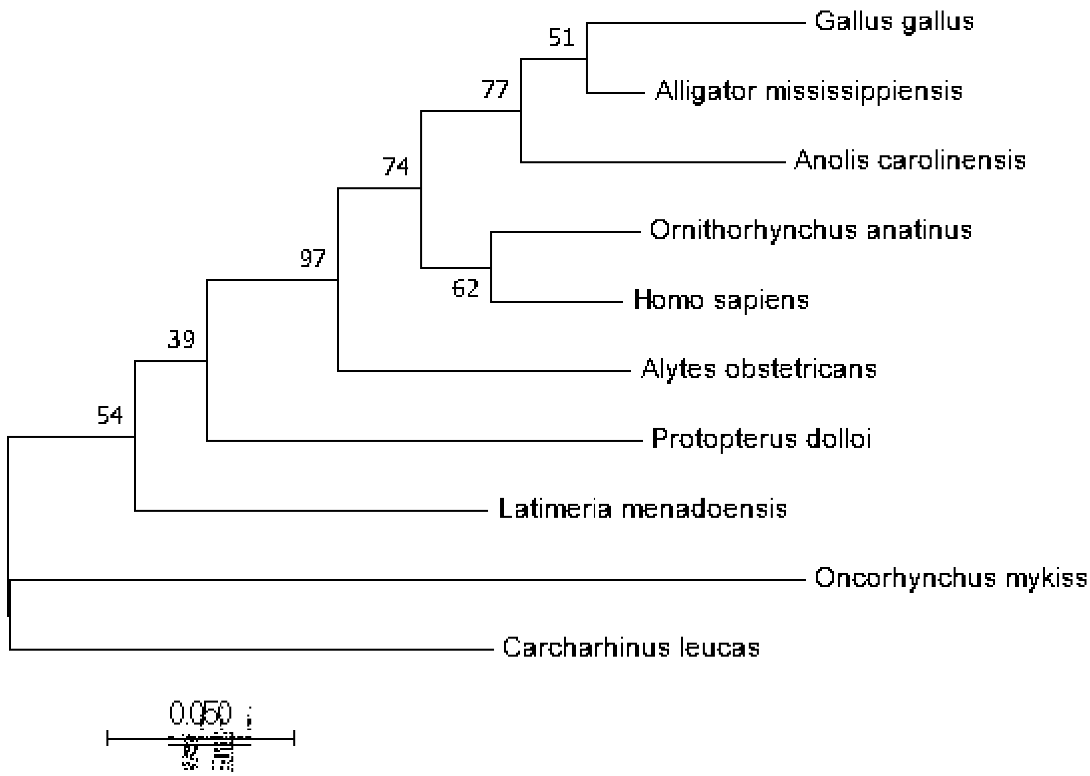


Figure 9: RAG1 AA. Maximum likelihood tree

1.5 Maximum Likelihood by PhyML

We launch another calculation of ML tree, this time with PhyML software, against the RAG1 nucleotide DATA and using the model of nucleotide substitution derived with **jModelTest** in point 1.1.

Stats:

```
 oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
                                  ---  PhyML 20120412  ---
                            http://www.atgc-montpellier.fr/phyml
                         Copyright CNRS - Universite Montpellier II
 oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo

. Sequence filename:            RAG1_nucleotides.phy
. Data set:                 #1
. Tree topology search :        Best of NNIs and SPRs
. Initial tree:             BioNJ
. Model of nucleotides substitution:    GTR
. Number of taxa:           10
. Log-likelihood:           -10221.02940
. Unconstrained likelihood:         -7159.32786
. Parsimony:                2192
. Tree size:                2.93881
. Discrete gamma model:         Yes
```

```
  - Number of categories:        4
  - Gamma shape parameter:       0.338
. Nucleotides frequencies:
  - f(A)= 0.29372
  - f(C)= 0.21009
  - f(G)= 0.25291
  - f(T)= 0.24328
. GTR relative rate parameters :
  A <-> C    5.04271
  A <-> G   11.34452
  A <-> T    3.88254
  C <-> G    2.33204
  C <-> T   19.20285
  G <-> T    1.00000

. Instantaneous rate matrix :
  [A---------C---------G---------T------]
  -0.93778   0.20387   0.55214   0.18177
   0.28503  -1.29754   0.11350   0.89900
   0.64123   0.09428  -0.78233   0.04682
   0.21945   0.77635   0.04867  -1.04447
```

We draw the guide tree with the program that we use from here, based on ETE library.

Script 1.5.1 (python)

```python
guide_tree = "(Gallus_gallus:0.12510968,(Anolis_carolinensis:0.17704815,Alligator_mississipp
↪ iensis:0.05696426)550:0.02306793,((Alytes_obstetricans:0.27201065,((Protopterus_dolloi:0
↪ .24924684,Carcharhinus_leucas:0.44036535)760:0.07012761,(Oncorhynchus_mykiss:0.73583795,
↪ Latimeria_menadoensis:0.16553616)385:0.05655032)570:0.08926547)474:0.04929311,(Ornithorh
↪ ynchus_anatinus:0.15023695,Homo_sapiens:0.14182907)572:0.07064245)955:0.06567711);"
_, _ = draw_guide_tree(guide_tree, True)
```

Output

```
    /-Gallus_gallus
   |
   |   /-Anolis_carolinensis
   |--|
   |   \-Alligator_mississippiensis
   |
 --|      /-Alytes_obstetricans
   |     |
   |   /-|      /-Protopterus_dolloi
   |  |  |   /-|
   |  |  |  |   \-Carcharhinus_leucas
   |  |   \-|
    \-|     |   /-Oncorhynchus_mykiss
      |      \-|
      |         \-Latimeria_menadoensis
```

```
|
|   /-Ornithorhynchus_anatinus
 \-|
    \-Homo_sapiens
```

1.6 First conclusions

We have obtained somehow different trees, which does not seem strange given the dispersion of algorithms and input data and the fact that the sequences are very similar. For example, we have an 83% identity between the amino acid sequence RAG1 of *Homo* and *Latimeria*, according to a global alignment *Needleman-Wunsch*, species far apart according to the consensus of the different trees.

Surely the weak differences in sequences originate much sensitivity with respect to differences of the methods used. Considering the relatively high bootstrap values and the frequency of appearance in all the trees, it seems that we can have enough confidence in this part of the tree:

```
                  /-Alligator_mississippiensis
               /-|
              |   \-Gallus_gallus
           /-|
          |  |   /-Homo_sapiens
        /-|   \-|
       |  |      \-Ornithorhynchus_anatinus
     /-|  |
    |  |   \-Anolis_carolinensis
```

In particular, the Alligator-Gallus, Homo-Ornithorhynchus and Anolis associated associations related to both groups, before the association between them, sometimes associated with one of the two before the confluence.

There is no recognized consensus pattern with respect to the rest of the species, not even with observations supported by a high bootstrap probability, what implies, if we give as true the reference tree, that the bootstrap procedure does not generate a true tree, but rather a robust tree over coherent variations of the sequence. Let's say algorithmically confident.

This part of the tree matches the reference tree. We also see in general that the NJ and ML methods tend to focus more on the different branches, than the construction methods based on UPGMA or similar, which surely underlie Maximum Parsimony. Interestingly, these trees based on simpler tree guiding algorithms are closer to the reference phylogeny:

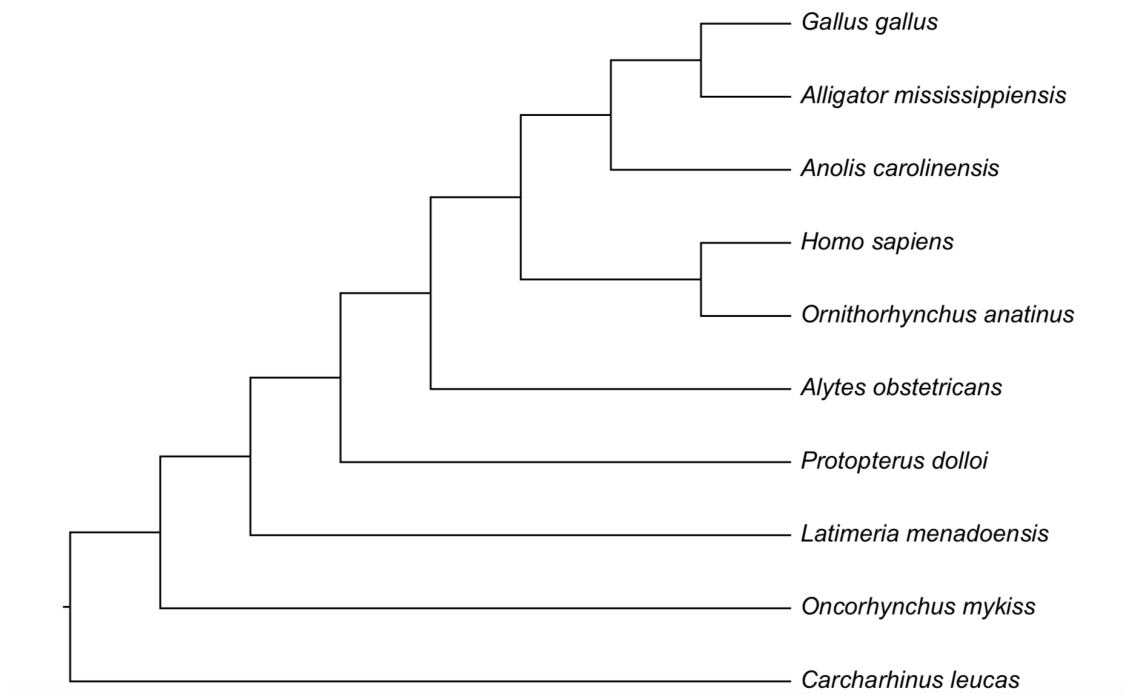


Figure 10: Reference tree

1.7 Another approach: Homemade phylogenies

To gain some major insight in the pairwise alignment, msa alignments and the construction of phylogeny trees,I have developed a few algorithms in python (not actually a fast ones, but it wasn't the purpose).

The Phylogenies were computed using CLUSTAL-W like methods and T-COFFEE inspired methods. UPGMA and NJ algorithms were developed to cluster the alignments.

All the pairwise alignments are not heuristic, based in dynamic programming (Needleman-Wunsch) and in the T-COFFEE case we use also a local alignment Smith-Waterman to help compute the weight matrix (this is not the T-COFFEE original algorithm).

The substitution matrix used in AA is a Blosum80 one and in nucleotides not matrix at all, or well score +3 for matches and score -2 for unmatches.

Not bootstrap at all, nor complex distance estimations, only based on sum of pairs, as suggested in CLUSTAL and-W T-COFFEE.

In next chapters I show some trees and the mehod used to compute them. In the bottom, as reference of all the details, the code necessary to obtain the trees and two executions (T-COFFEE and CLUSTAL-W).

Prior to that I write my conclusions.

1.8 Final conclusions

The homemade phylogenies drives me to the same conclusions than in the case of specialized and complex software.

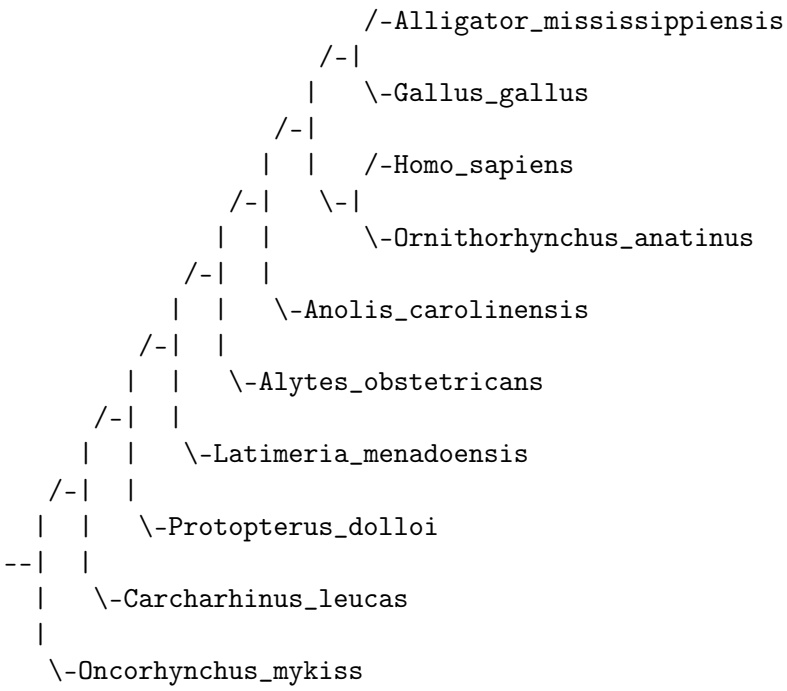
Also, the homemade phylogenies show a very good similar trees compared to the reference one, obtained by UPGMA methods, over T-COFFEE and CLUSTAL-W approaches, and RAG1 amino acid data.

Perhaps both are the trees that share the best identity as reference tree, better than all the trees obtained from the complex methods (PhyML ang Mega).

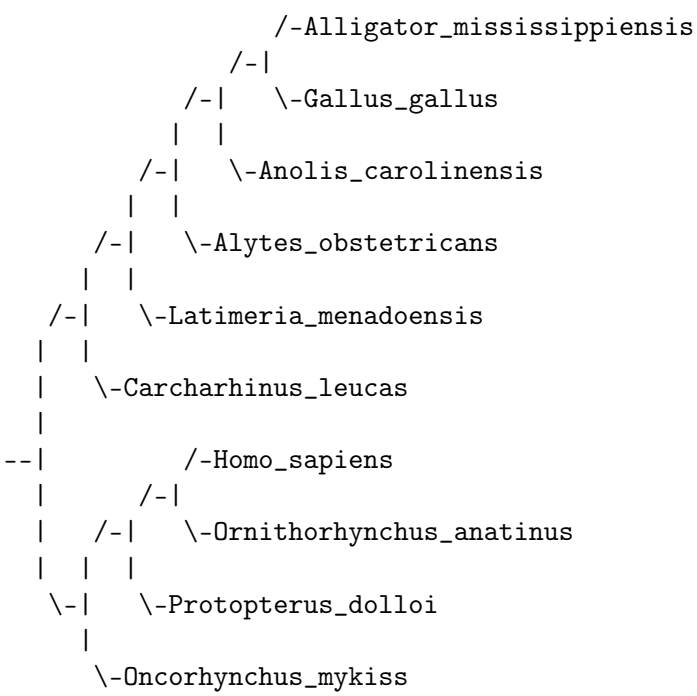
It's possible that my choose of parameters for Mega or PhyML was not the most convenient one, but also it makes me thinking on the convenience of dive so deep into the ocean of algorithms. Too many assumptions for our current evolutionary knowledge, too many parameters, too many complexity.

1.8.1 Trees for RAG1 amino acid

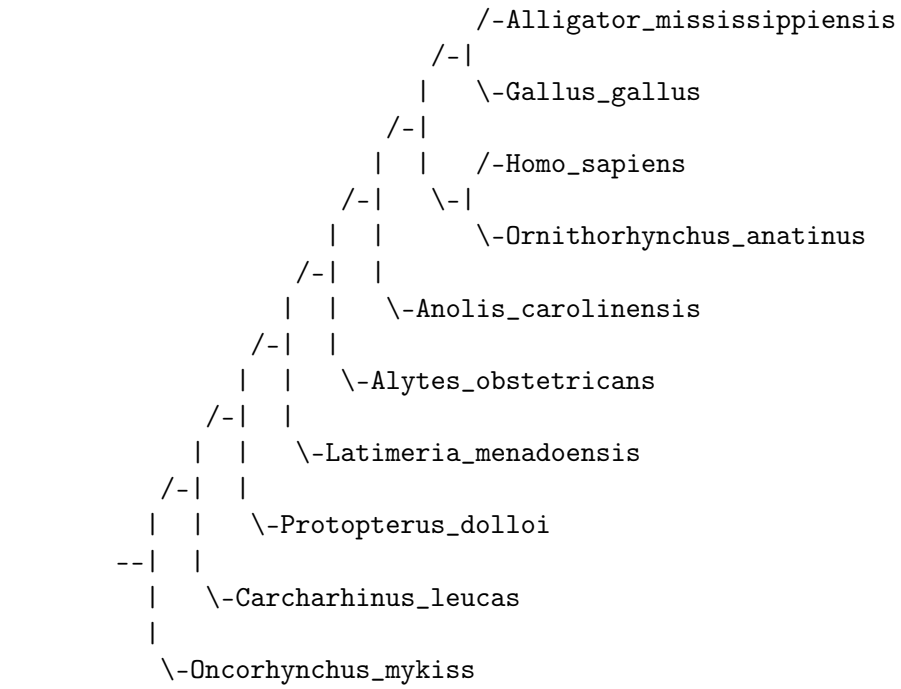
CLUSTAL-W *UPGMA*



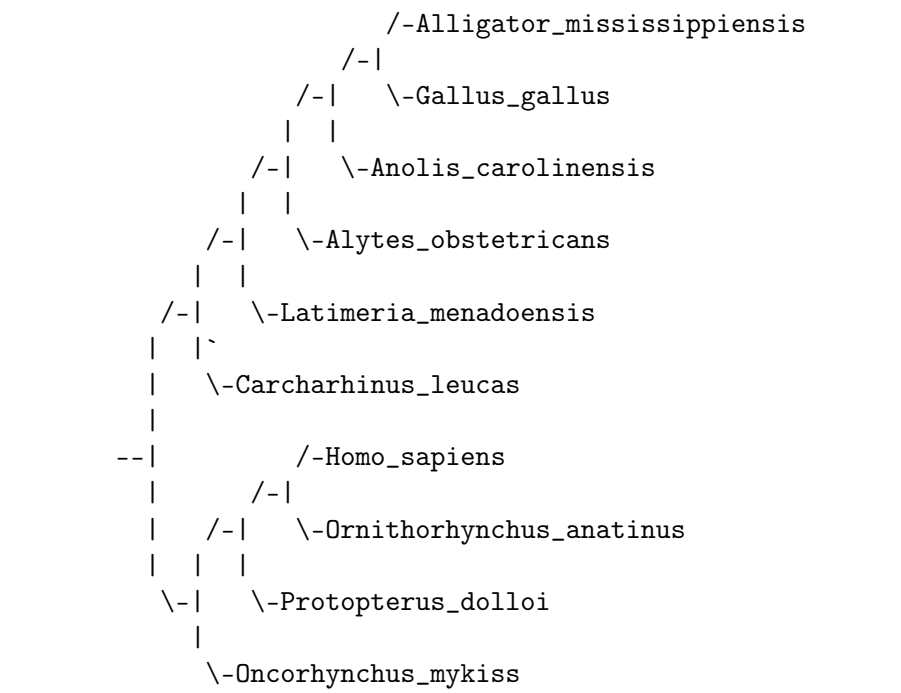
CLUSTAL-W *NJ*



T-COFFEE *UPGMA*

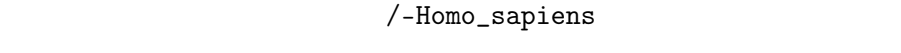


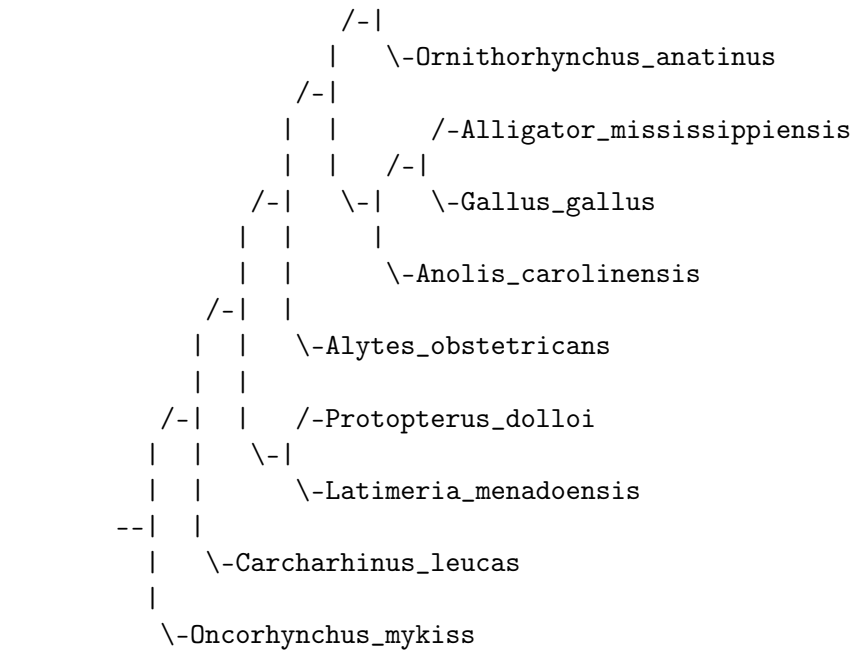
T-COFFEE *NJ*



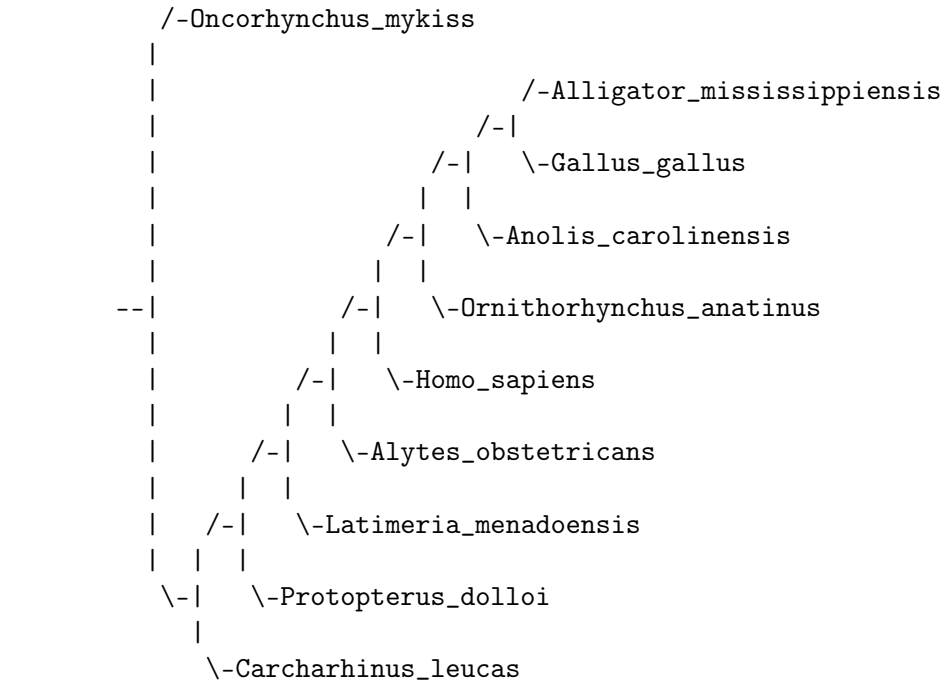
1.8.2 Trees for RAG1 Nucleotides

CLUSTAL-W *UPGMA*



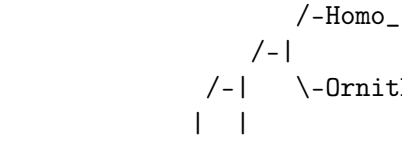


CLUSTAL-W *NJ*



1.8.3 Trees for 16S Nucleotides

CLUSTALW *UPGMA*



```
   /-|   \-Anolis_carolinensis
  |  |
  |  |   /-Gallus_gallus
  |   \-|
  |      \-Alligator_mississippiensis
--|
  |         /-Alytes_obstetricans
  |      /-|
  |     |   \-Carcharhinus_leucas
  |   /-|
  |  |  |   /-Latimeria_menadoensis
   \-|   \-|
     |      \-Oncorhynchus_mykiss
     |
      \-Protopterus_dolloi
```

1.9 Class AlignSequences

This class implements recursively three alignment algorithms: 1. Global alignment (Needleman-Wunsch inspired).

2. Local alignment (Smith-Waterman inspired).
3. Longest common substring.(the search for the longest common sequence can also be considered a type of alignment).

Script 1.9.1 (python)

```python
"""This module shows alternative recursive implementations of global sequence alignments:
    Global alignment (Needleman-Wunsch based)
    Local (Smith-Waterman based)
    Finding of the longest common substring.
Todo:
    * Return all the solutions of the alignments. Now it only returns one solution
    * Control of errors
    * Implement multi-alignments
    * Implement heuristic algorithms
"""
import re
from Bio import pairwise2
from Bio.pairwise2 import format_alignment
from Bio.SubsMat import MatrixInfo
import time
import sys

MIN = -sys.maxsize - 1
COMPAC = 100000
"""int: Constant to compact max score."""
SCORE_MATCH = 2
"""int: Default match score."""
SCORE_NO_MATCH = -3
```

```
24  """int: Default no match score."""
25  SCORE_GAP_INI = -10
26  """int: Default gap ini in affine gap penalty."""
27  SCORE_GAP_CONT = -2
28  """int: Default gap continuation in affine gap penalty"""
29  DEFAULT_SUBST_MATRIX = {('A', 'A'): 0, ('A', 'C'): 1, ('A', 'G'): 1, ('A', 'T'): 1, ('C',
    ↪  'A'): 1, ('C', 'C'): 0, ('C', 'G'): 1, ('C', 'T'): 1, ('G', 'A'): 1, ('G', 'C'): 1,
    ↪  ('G', 'G'): 0, ('G', 'T'): 1, ('T', 'A'): 1, ('T', 'C'): 1, ('T', 'G'): 1, ('T', 'T'): 0}
30  """dict: Default substitution matrix (for "ACGT" common nucleotide alphabet)"""
31
32  sys.setrecursionlimit(5000)
33
34  class AlignSequences:
35      """Recursive implementation of global, local and long substring alignments methods.
36
37      Attributes:
38          sequences (list of str): Contains the wo sequences to align. The first
39              one (index 0) is the query sequence (BLAST concept) or bottom sequence on
    ↪  alignment prints
40              or vertical sequence in the common graphical representation of score matrix.
41          len_seq0 (int): Sequence 0 length.
42          len_seq1 (int): Sequence 1 length.
43          mode (str): Computation mode:
44              'GLOBAL'            Global Alignment
45              'LOCAL'             Local Alignment
46              'LONG_SUBSTRING'    Obtain long common substring
47          score_match (int): Score of match characters.
48          score_no_match (int): Score of no match characters.
49          score_gap_ini (int): Score of gap init.
50          score_gap_cont (int): Score of gap continuation.
51          score (int): Score of last computed alignment.
52          gaps (int): Number of gaps of the last computed alignment.
53          matches (int): Number of matches of the last computed alignment.
54          unmatches (int): Number of unmatches of the last computed alignment.
55          align_seq0 (str): Sequence 0 with the gaps necessary for the alignment.
56          align_seq1 (str): Sequence 1 with the gaps necessary for the alignment.
57          matching (str): Printable line with the align relations ('|', '.', ' ') between
58              both align_seq, necessary for printing the alignment.
59          ini_time (int): Initial time of computation, for profiling purposes
60          finish_time (int): Final time of computation, for profiling purposes
61          score_store (dict of tuple int): Store of scores, for each calculated cell with
    ↪  tuple (i,j,g)
62              where i is the coordinate of the bottom sequence, j the coordinate of the top
    ↪  sequence
63              and g has the value 1 if the cell is a gap init cell and 0 if it's a gap
    ↪  continuation.
64              For a explanation of calculared cell see align method.
65          matches_store (dict of tuple int): Store of the number of matches in the calculated
    ↪  cell
66          gaps_store (dict of tuple int): Store of the number of gaps in the calculated cell
67          max_score_index (tuple of int): Cell coordinate tuple of the cell with the maximun
    ↪  score
```

```
        max_score (int): maximum computed score
        forward_arrow (dict of str): Store of the optimal displacements accomplished at a cell
            to guarantee an optimal score: 'v' vertical (down), 'h' horizontal (rigth),
            'd' diagonal.
        stacks (list of list of str): Stacks of sequences related to principal sequences in a msa
        stacks_indexes (list of str): Indexes of the sequences of stack relatives to original sequences
        stacks_refs (list of dict): References of the char in sequence os stack relatives to
        char positions on original sequences
        matrix_mode (str): If "SUBST" it's a substitution matrix, if not it's a weight matrix
            with the keys
                i = position of first sequence in stack
                j = position of second sequence on stack
                pos_i = coordinate of char on first sequence
                pos_j = coordinate of char on second sequence
            and the value is the weight to score this position
            if not match, the score is 0.
    """

    def __init__(self, sequences, mode="ALIGN", score_match=SCORE_MATCH, score_no_match=SCORE_NO_MATCH,\
                 score_gap_ini=SCORE_GAP_INI, score_gap_cont=SCORE_GAP_CONT, subst_matrix={}):
        """Init parameters of alignment"""
        self.set_sequences(sequences)
        self.set_stacks()
        self.len_seq0 = len(self.sequences[0])
        self.len_seq1 = len(self.sequences[1])
        self.init_stores()
        self.set_scores(score_match, score_no_match, score_gap_ini, score_gap_cont)
        self.set_mode(mode)
        self.score = 0
        self.matches = 0
        self.unmatches = 0
        self.gaps = 0
        self.align_seq0 = ""
        self.align_seq1 = ""
        self.matching = ""
        self.ini_time = 0
        self.finish_time = 0
        self.set_subst_matrix(subst_matrix)
        self.set_matrix_mode()

    def init_stores(self):
        """Init dictionary that store temp data of the alignment"""
        self.score_store = {}
        self.matches_store = {}
        self.gaps_store = {}
        self.max_score_index = (0, 0, 0)
        self.max_score = 0
```

```
115         self.forward_arrow = {}
116 
117     def set_sequences(self, sequences):
118         """Update the target sequences of the alignment"""
119         self.sequences = sequences
120 
121     def set_stacks(self, stack_0=[], stack_1=[],\
122                    stack_0_indexes=[], stack_1_indexes=[], stack_0_refs=[], stack_1_refs=[]):
123         """Update the stacks for msa"""
124         self.stacks = [stack_0, stack_1]
125         self.stacks_indexes = [stack_0_indexes, stack_1_indexes]
126         self.stacks_refs = [stack_0_refs, stack_1_refs]
127 
128     def set_matrix_mode(self, mode="SUBST"):
129         """Update matrix mode"""
130         self.matrix_mode = mode
131 
132     def set_subst_matrix(self, subst_matrix={}):
133         """Update the score matrix"""
134         self.subst_matrix = subst_matrix
135 
136     def set_scores(self, score_match=SCORE_MATCH, score_no_match=SCORE_NO_MATCH,\
137                    score_gap_ini=SCORE_GAP_INI, score_gap_cont=SCORE_GAP_CONT):
138         """Update the weigth scores of the alignment"""
139         self.score_match = score_match
140         self.score_no_match = score_no_match
141         self.score_gap_ini = score_gap_ini
142         self.score_gap_cont = score_gap_cont
143 
144     def set_mode(self, mode="ALIGN"):
145         """Set computation mode"""
146         self.mode = mode
147 
148     def forward_track(self, index):
149         """Calc alignments in forward direction.
150 
151             The alignment strings are calculated from init cell (0,0) in global
152             alignments or maximum score cell in local alignments.
153 
154             In local mode it's necessary to extend the alignments (local) to the total
    ↪ length of
155             the sequences to show the location of the alignment, and in order to compare with
156             BioPython outputs.
157 
158             Args:
159                 index (tuple of int): Cell coordinates of the starting cell
160 
161             Returns:
162                 string: align sequence 0 (bottom) for printing purposes
163                 string: align sequence 1 (top) for printing purposes
164                 tuple of int: Coordinates of the last cell
165         """
```

```
        ret_align_seq0, ret_align_seq1 = "", ""
        (i, j, gap_ini) = index
        ret_final_pos = (self.len_seq0, self.len_seq1)
        while i < self.len_seq0 or j < self.len_seq1:
            if self.mode == "LOCAL" and self.score_store[(i, j, gap_ini)] == 0:
                ret_final_pos = (i, j)
                break
            arrow = self.forward_arrow[(i, j, gap_ini)]
            if arrow == "d":
                ret_align_seq0 += self.sequences[0][i]
                ret_align_seq1 += self.sequences[1][j]
                i, j, gap_ini = i + 1, j + 1, 1
            elif arrow == "h":
                ret_align_seq0 += "-"
                ret_align_seq1 += self.sequences[1][j]
                i, j, gap_ini = i , j + 1, 0
            elif arrow == "v":
                ret_align_seq0 += self.sequences[0][i]
                ret_align_seq1 += "-"
                i, j, gap_ini = i + 1, j, 0
        #compute the complete align in local mode
        if self.mode == "LOCAL":
            ret_align_seq0 = self.sequences[0][0:index[0]] +\
                             ret_align_seq0 + self.sequences[0][ret_final_pos[0]:]
            ret_align_seq1 = self.sequences[1][0:index[1]] +\
                             ret_align_seq1 + self.sequences[1][ret_final_pos[1]:]
            diff_pos_ini = index[1] - index[0]
            if diff_pos_ini > 0:
                ret_align_seq0 = '-' * diff_pos_ini + ret_align_seq0
            else:
                ret_align_seq1 = '-' * -diff_pos_ini + ret_align_seq1
            diff_len = len(ret_align_seq1) - len(ret_align_seq0)
            if diff_len > 0:
                ret_align_seq0 += '-' * diff_len
            else:
                ret_align_seq1 += '-' * -diff_len
        return ret_align_seq0, ret_align_seq1, ret_final_pos

    def calc_matching(self, align_seq0, align_seq1, ini_pos=(), final_pos=()):
        """Calc matching string

            The matching string is the string line to print between the top and
            bottom alignment strings. It contains the match (|), no match (.) and
            gap ( ) indicators.

            Args:
                align_seq0 (string): Bottom sequence
                align_seq1 (string): Top sequence
                ini_pos (tuple of int): Initial cell coordinates
                final_pos (tuple of int): Final cell coordinates

            Returns:
```

```
                string: Matching string

        """
        count = 0
        ret_matching = ""
        diff_pos_ini = ini_pos[1] - ini_pos[0]
        if diff_pos_ini > 0:
            delta_pos = diff_pos_ini
        else:
            delta_pos = 0
        for n, (i, j) in enumerate(zip(align_seq0, align_seq1)):
            if self.mode == "LOCAL" and not (n >= ini_pos[0] + delta_pos and n <
            ↪  final_pos[0] + delta_pos):
                ret_matching += ' '
            else:
                if i == j: ret_matching += '|'
                elif i != j and i != '-' and j != '-': ret_matching += '.'
                else: ret_matching += ' '
            count += 1
        return ret_matching

    def store(self, key, score, matches, gaps):
        """Store info related to a computed cell
        The maximum score is computed having into account the number of matches, if there are
        most than one solution. If the score are equal, the path with more matches is
    ↪  selected.
             Args:
                key (tuple of int): Cell coordinates
                score (int): Cell score
                matches (int): Cell matches
                gaps (int): Cell gaps
        """
        self.score_store[key] = score
        super_score = score * COMPAC + 10 * matches
        if super_score > self.max_score:
            self.max_score_index = key
            self.max_score = super_score
        self.matches_store[key] = matches
        self.gaps_store[key] = gaps

    def calc_score_binary(self, seq_0, seq_1, i, j, seq_0_index=0, seq_1_index=1, pos_0=0,
    ↪  pos_1=0):
        """Compute alignment scores for two sequences
        If there are a substitution matrix (actually dictionary) defined,
        the scores are computed from the dictionary.
            Args:
                seq_0 (int): Sequence 0
                seq_1 (int): Sequence 1
                i (int): Sequence 0 char index
                j (int): Sequence 1 char index
                seq_0_index (int): Sequence 0 index on original sequences (MSA)
                seq_1_index (int): Sequence 1 index on original sequences (MSA)
```

```
                pos_0 (int): Sequence 0 index on stack 0
                pos_1 (int): Sequence 1 index on stack 0
        """
        if self.subst_matrix:
            if self.matrix_mode == "SUBST":
                #print("PAIR", i, j, seq_0[i], seq_1[j])
                subst_matrix_index = (seq_0[i], seq_1[j])
                subst_matrix_index_swap = (seq_1[j], seq_0[i])
                if subst_matrix_index in self.subst_matrix:
                    matrix_score = self.subst_matrix[subst_matrix_index]
                elif subst_matrix_index_swap in self.subst_matrix:
                    matrix_score = self.subst_matrix[subst_matrix_index_swap]
            else: #weight matrix
                if pos_0 in self.stacks_refs and i in self.stacks_refs[pos_0]:
                    i_orig = self.stacks_refs[pos_0][i]
                else:
                    i_orig = i
                if pos_1 in self.stacks_refs and i in self.stacks_refs[pos_0]:
                    j_orig = self.stacks_refs[pos_1][j]
                else:
                    j_orig = j
                if i_orig in self.subst_matrix[seq_0_index][seq_1_index] and \
                    j_orig in self.subst_matrix[seq_0_index][seq_1_index][i_orig]:
                    matrix_score = self.subst_matrix[seq_0_index][seq_1_index][i_orig][j_ori
g]
                else:
                    matrix_score = 0
        # Gaps in almost one of the sequences. This case only arises in MSA
        # There is no matrix related entry. If matrix is a weight matrix we compute
        # as zero (as defined in T-Coffee)
        if seq_0[i] == "-" or  seq_1[j] == '-':
            inc_matches = 0
            if self.subst_matrix:
                if self.matrix_mode == "SUBST":
                    inc_score = self.score_gap_cont
                else:
                    inc_score = 0
            else:
                inc_score = self.score_gap_cont
        else:
            if seq_0[i] == seq_1[j]:
                if self.subst_matrix:
                    inc_score = matrix_score
                else:
                    inc_score = self.score_match
                inc_matches = 1
            else:
                if self.subst_matrix:
                    inc_score = matrix_score
                else:
                    inc_score = self.score_no_match
                inc_matches = 0
```

```
318
319         return inc_score, inc_matches
320
321     def calc_score(self, i, j):
322         """Compute alignment scores.
323         If there are stacks associated with the sequence, we compute the score weigthing the
324         scores of the stacks (SOP: Score of Pairs). Stacks contains also the guiding
    ↪  sequences.
325             Args:
326                 i (int): Sequence 0 index
327                 j (int): Sequence 1 index
328         """
329         if self.stacks == [[],[]]:
330             return self.calc_score_binary(self.sequences[0], self.sequences[1], i, j, 0, 1,
                ↪  0, 0)
331         else:
332             computed_score = 0
333             computed_matches = 0
334             nvalues = 0
335             for pos_0, (seq_0, index_0) in enumerate(zip(self.stacks[0],
                ↪  self.stacks_indexes[0])):
336                 for pos_1, (seq_1, index_1) in enumerate(zip(self.stacks[1],
                    ↪  self.stacks_indexes[1])):
337                     score, matches = self.calc_score_binary(seq_0, seq_1, i, j, index_0,
                        ↪  index_1, pos_0, pos_1)
338                     computed_score += score
339                     computed_matches += matches
340                     nvalues += 1
341             ret_score = computed_score /  nvalues
342             ret_matches = computed_matches / nvalues
343             return ret_score, ret_matches
344
345     def align(self, i=0, j=0, ini_gap=1):
346         """Recursive align of sequences
347         For each cell, which coordinates are (i, j, ini_gap), calc the maximum score path
    ↪  from
348         three alternative displacements:
349
350         1) To (i + 1, j + 1, 1), that is, matching or no matching the seq0(i) and seq1(i)
    ↪  characters.
351         This is a diagonal displacement.
352         2) To (i, j + 1, 0), that is, seting a gap in seq0 and advance seq1. Horizontal
    ↪  displacement.
353         3) To (i + 1, j, 0), that is, seting a gap in seq1 and advance seq0. Vertical
    ↪  displacement.
354
355         The scores of these displacements are calculated adding the score ot the target cells
356         (that are computed recursively) and the matrix, default of gap scores in each case.
357
358         The score, matches, gaps and forward_arrow are stored at related dictionary entry
    ↪  based on
359         coordinates (i, j, ini_gap), all of them asociated to the maximum score of
```

```
        the three possible paths starting from the cell, avoiding recomputation
        of the cell if it's called from another recuersive path.

        Each cell has a third score coordinate, because a cell could be called from a cell with yet
        has a gap (only from horizontal or vertical prior displacement) or from a cell with has a match/no match.
        Then we need to store two scores, matches, gaps and forward_arrows related to the two possible
        cell incarnations at coordinates (i, j, 0) and (i, j, 1).

        We store matches and gaps in order to have one aditional criterion to tiebreaker
        if some of the scores are equal. We are using this aproach in local alignment computation. If two
        scores are equal we choose the solucion with the greatest number of matches.

        We store the displacement directions in forward_arrow dict to compute the alignment.
        It's posible to avoid this, using only the score information, but we have let this aproach
        as proof of concept and for clarity in the algorithm.

        In this scenario we observe that the differences between the global,
        local and long substring algorithms are minimal.

        Local algorithm:

            Starting from the global algorithm, which would be the most general,
            the local algorithm only changes two aspects:
                1. Rejection of the roads with negative values of the score, equaling these values
                to 0, that is, not letting previous alignments of poor quality affect the final result.

                2. Use as cell of beginning of the alignment the one with the highest scores.
                In our implementation we also take into account the number of matches,
                as we have already mentioned.

        Finally, but outside the algorithm of alignment itself (at forward_track and matching methods)
        it only remains to extend the alignment obtained to show its location within the chains to be aligned.

        Search algorithm of the long common substring:

            Modify the global algorithm in the following aspects:
                1. Only computes matches between characters or gaps in one or another initial sequence.

         Args:
                i (int): Sequence 0 index
                j (int): Sequence 1 index
                ini_gap (int): 1 if gap initiation, 0 if gap continuation
```

```
        """
        score_diag, score_hor, score_ver = MIN, MIN, MIN
        matches_diag, matches_hor, matches_ver = MIN, MIN, MIN
        gaps_diag, gaps_hor, gaps_ver = MIN, MIN, MIN
        #align and advance seq0 and seq1
        #in long_substring mode only matches are processed
        if i < self.len_seq0 and j < self.len_seq1 and\
        (self.mode != "LONG_SUBSTRING" or self.sequences[0][i] == self.sequences[1][j]):
            inc_score, inc_matches = self.calc_score(i, j)
            key = (i + 1, j + 1, 1)
            if key in self.score_store:
                score_diag, matches_diag, gaps_diag = \
                self.score_store[key] + inc_score, self.matches_store[key] + inc_matches,
                ↪  self.gaps_store[key]
            else:
                score, matches, gaps = self.align(i + 1, j + 1, 1)
                self.store(key, score, matches, gaps)
                score_diag, matches_diag, gaps_diag = score + inc_score, matches +
                ↪  inc_matches, gaps
        #don't align and gap in seq0 (advance seq1)
        if j < self.len_seq1:
            gap_score = self.score_gap_cont + ini_gap * self.score_gap_ini
            key = (i, j + 1, 0)
            if key in self.score_store:
                score_hor, matches_hor, gaps_hor = self.score_store[key] + gap_score,\
                self.matches_store[key], self.gaps_store[key] + 1
            else:
                score, matches, gaps = self.align(i, j + 1, 0)
                self.store(key, score, matches, gaps)
                score_hor, matches_hor, gaps_hor = score + gap_score, matches, gaps + 1
        #don't align and gap in seq1 (advance seq0)
        if i < self.len_seq0:
            gap_score = self.score_gap_cont + ini_gap * self.score_gap_ini
            key = (i + 1, j, 0)
            if key in self.score_store:
                score_ver, matches_ver, gaps_ver =\
                self.score_store[key] + gap_score, self.matches_store[key],
                ↪  self.gaps_store[key] + 1
            else:
                score, matches, gaps = self.align(i + 1, j, 0)
                self.store(key, score, matches, gaps)
                score_ver, matches_ver, gaps_ver = score + gap_score, matches, gaps + 1
        #choose the high score path
        matcher_diag, matcher_hor, matcher_ver = score_diag, score_hor, score_ver
        if i < self.len_seq0 or j < self.len_seq1:
            if self.mode == "LOCAL" and matcher_diag < 0 and matcher_hor < 0 and matcher_ver
            ↪  < 0:
                score_diag, score_hor, score_ver = 0, 0, 0
                #matcher_diag, matcher_hor, matcher_ver  = 0, 0, 0
            if matcher_diag > matcher_hor and matcher_diag > matcher_ver:
                ret_score, ret_matches, ret_gaps, ret_arrow =\
                score_diag, matches_diag, gaps_diag, "d"
```

```
450             elif matcher_hor > matcher_ver:
451                 ret_score, ret_matches, ret_gaps, ret_arrow =\
452                 score_hor, matches_hor, gaps_hor, "h"
453             else:
454                 ret_score, ret_matches, ret_gaps, ret_arrow =\
455                 score_ver, matches_ver, gaps_ver, "v"
456         else:
457             ret_score, ret_matches, ret_gaps, ret_arrow =\
458             0, 0, 0, ""
459         self.forward_arrow[(i, j, ini_gap)] = ret_arrow
460         if i == 0 and j == 0:
461             self.store((0, 0, 1), ret_score, ret_matches, ret_gaps)
462             if self.mode in ["GLOBAL", "LONG_SUBSTRING"]: self.max_score_index = (0, 0, 1)
463             else: ret_score = self.max_score // COMPAC
464             ret_matches = self.matches_store[self.max_score_index]
465             ret_gaps = self.gaps_store[self.max_score_index]
466 
467         return ret_score, ret_matches, ret_gaps
468 
469     def compute(self, mode="LOCAL", silent=False):
470         """Calc alignment
471             Args:
472                 mode (str): Type of algorithm (local, global or long substring)
473                 silent (bool): If true don't show alignment output
474         """
475         self.ini_time = time.time()
476         self.init_stores()
477         self.set_mode(mode)
478         self.score, self.matches, self.gaps = self.align()
479         self.align_seq0, self.align_seq1, final_pos = self.forward_track(self.max_score_inde
    x)
480         self.matching = self.calc_matching(self.align_seq0, self.align_seq1,
            ↪  self.max_score_index, final_pos)
481         self.unmatches = self.matching.count('.')
482         self.gaps = self.matching.count(' ')
483         self.finish_time = time.time()
484         if not silent:
485             self.view()
486 
487     def get_len_long_common_substring(self):
488         """Getter for the len of the common substring
489         That is equal to the number of matches of the alignment
490         """
491         return self.matches
492 
493     def get_long_common_substring(self):
494         """Returns the longest common substring
495         whitout alignment (positional) information
496         """
497         long_common_substring = ""
498         for (char, match_char) in zip(self.align_seq1, self.matching):
499             if match_char == '|':
```

```
                long_common_substring += char
        return long_common_substring

    def view(self):
        """Prints the alignment data"""
        #unmatches = self.matching.count('.')
        #gaps = self.matching.count(' ')
        if self.matching:
            gap_groups = self.matching.count('| ') + self.matching.count('. ') +
             ↪  self.matching[0].count(' ')
        else:
            gap_groups = 0
        print(" ")
        if self.mode == "LOCAL":
            print("### AlignSequences. Local alignment (Smith-Waterman)")
        elif self.mode == "LONG_SUBSTRING":
            print("### AlignSequences. Long substring finder")
        else:
            print("### AlignSequences. Global alignment (Needleman-Wunsch)")
        if self.subst_matrix:
            print("\tUsing score matrix with matrix mode",self.matrix_mode)
        print(self.align_seq1)
        print(self.matching)
        print(self.align_seq0)
        print("\tScore:", self.score)
        print("\tSimilarity (wo gaps):", self.matches / (self.matches + self.unmatches))
        print("\tDistance (wo gaps):", self.unmatches / (self.matches + self.unmatches))
        print("\tDistance:", self.unmatches / (self.matches + self.unmatches + self.gaps))
        print("\tInit index:", self.max_score_index)
        print("\tMatches:", self.matches, " Unmatches:", self.unmatches, " Gaps:",
         ↪  self.gaps, " Gap groups:", gap_groups)
        #simple scoring verification todo: apply to matrix
        if not self.subst_matrix:
            print("\tScore verified:", self.matches * self.score_match + self.unmatches *
             ↪  self.score_no_match \
                  + self.gaps * self.score_gap_cont + gap_groups * self.score_gap_ini)
        print("\tFinish. Execution milliseconds:", round((self.finish_time - self.ini_time)
         ↪  * 1000))
        print("\tScore Dictionary Size", len(list(self.score_store.keys())))

    def edit_distance(self, score_match=0, score_no_match=-1, score_gap_ini=0,
     ↪  score_gap_cont=-1):
        """Calculates an edit distance as requested in questions 1 and 3
        It's the same computation as a global alignment with -1 penalities applied to
        score_gap_cont and score_nomatch and 0 in score_match and score_gap_ini

            Args:
                score_match (int): Score of match characters.
                score_no_match (int): Score of no match characters.
                score_gap_ini (int): Score of gap init.
                score_gap_cont (int): Score of gap continuation.
        """
```

```
547         self.set_scores(score_match, score_no_match, score_gap_ini, score_gap_cont)
548         self.compute("GLOBAL", True)
549         return abs(self.score)
```

1.10 MSA

In these functions, what is necessary to perform a multiple alignment of sequences is developed.

The method of progressive alignment based on a guide tree is used.

The guide tree can be obtained in two alternative ways: **Unweighted Pair Group Method with Arithmetic Mean (UPGMA)** and **Neighbor Join (NJ)**, the same options present in *CLUSTAL* software.

The alignment has three known phases. These are the particularities of my implementation on each phase:

1. Perform pairwise alignments between all the sequences involved and assign them a score.

In the case of **UPGMA** we use the proportion (in percentages) between matches and matches plus no matches (without taking gaps into account). That is, we use a measure of the identity between the two sequences involved. You can also use the distance, which would be the complement to 100 of identity, but we wanted to do so to be able to compare with the information that *CLUSTAL* throws at the beginning of his output. It does not affect the result, we simply have to look for maximum identities to build the guide tree, instead of minimum distances. In the case of **NJ**, we have chosen to use distances, computed also in percentages. Also not taking into account the number of gaps in the denominators.

2. Build the guide tree. As I said, we can do it using UPGMA or NJ. The NJ method generates an unrooted tree. As we need a root, for purposes of the subsequent alignment, we have chosen to root it by clustering the two nodes that have no relation. We do not know if it is the method used by *CLUSTAL* or similar programs, but it seems to work.

3. Multiple alignment. It is, as we know, a progressive alignment following the order indicated by the guide tree. In each step we need to align two groups of sequences, of length $n >= 1$ and $m >= 1$. Each group is aligned as a whole, in the sense that the gaps entered in one of the sequences of the group must be introduced in the same positions in the rest of the sequences of their group.

To assign a score to a position, the combined score of all the residuals of that position is used. To do this we produce the Cartesian product nxm of all the characters of that position and calculate the average of scores:

$$\frac{\sum_{\substack{0 \le i < n \\ 0 \le j < m}} matrix(i,j)}{nm}$$

If in any of the positions we have a gap, we have chosen to penalize it as the sum of penalties assigned to the start of the gap plus gap continuation penalty. It is a criterion, *CLUSTAL* we know that it uses another one.

If we already have a pairwise development, as it was my case, it would be easy to extend it to address MSA?. The answer is affirmative. With slight modifications in the class **AlignSequences**, we have managed to address an MSA, in the following way: 1. Generalize the one-position scoring algorithm to take into account all the sequences of both groups, averaging the scores as indicated above.

2. Take a sequence from each group (the first) to perform a simple pairwise alignment (but with the scores calculated as indicated in 1).

3. Compare the sequences resulting from the pairwise alignment with their originals from each group, compute where the gap is introduced and introduce the gap at the same positions in the rest of the sequences of each group.

1.11 MSA generic methods

Script 1.11.1 (python)

```python
"""This methods shows alternative implementations of multiple sequence alignments, CLUSTAL,
    T-COFFEE.

TODO:
    * Many more tests. Create a test battery.

    * Achieve that the results obtained are more similar to those of CLUSTAL (if they have
    to be).
    Given the lack of detailed information it will be necessary to resort
    to the sources (in C++) of CLUSTAL.

    * Allow to configure the initial alignment and the final multialignments with different
    parameters.

    * Draw the alignments in a more standard way.

    * Draw the phylogenetic trees.

    * Include all new methods in AlignSequences or in another class.
"""
from ete3 import Tree, TreeStyle
MIN_SCORE = 0

def draw_guide_tree(tree, print_tree=True):
    """
    Draw guide tree with ETE library
    """
    t = Tree(tree + ";")
    ts = TreeStyle()
    ts.show_leaf_name = True
    ts.show_branch_length = False
    ts.show_branch_support = False
    ts.scale = 160
    ts.branch_vertical_margin = 40
    if print_tree: print(t)
    return t, ts

def readFasta(file):
    """
    Reads all sequences of a FASTA file
    Args:
        file (str): name of the imput FASTA file
    Returns:
        dict of str, str: sequences readed
    """
    ret_seqs = {}
    seq = ""
    key_found = False
    with open(file, 'r') as f:
```

```
        key = ""
        for line in f:
            line = line.replace('\n', '')
            if len(line) > 0:
                if line[0] == ">":
                    if key_found:
                        ret_seqs[key] = seq
                    key_found = True
                    key = line[1:].split(" ")[0]
                    seq = ""
                elif key_found:
                    seq += line
    if key_found:
        ret_seqs[key] = seq
    return ret_seqs

def pairwise_align(s1, s2, matrix, matrix_mode, mode, score_gap_ini=0, score_gap_cont=-8,\
                   score_match=3, score_no_match=-2):
    """
    Performs initial pairwise alignments against the class AlignSequences
    returning the %identity.

    Args:
        s1 (str): First sequence to compare.
        s2 (str): Second sequence to compare.
        matrix (dict of tuples of int): Substitution matrix, Biopython format
        matrix_mode (str): Type of matrix
            'SUBST'             Substitution matrix
            'WEIGHT'            Weight matrix
        mode (str): Computation mode:
            'GLOBAL'            Global Alignment
            'LOCAL'             Local Alignment
            'LONG_SUBSTRING'    Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)

    Returns:
        (int): % identity between sequences
    """
    align = AlignSequences([s1, s2])
    align.set_scores(score_match, score_no_match, score_gap_ini, score_gap_cont)
    align.set_subst_matrix(matrix)
    align.set_matrix_mode(matrix_mode)
    align.compute(mode.upper(), silent = True)
    return round((align.matches + 1) * 100 / (align.matches + align.unmatches + 2))

def pairwise_align_distance(s1, s2, matrix, matrix_mode, mode, score_gap_ini=0,
↪   score_gap_cont=-8):
    """
    Performs initial pairwise alignments against the class AlignSequences
```

```
    returning the distance between 0 and 100.

    Args:
        s1 (str): First sequence to compare.
        s2 (str): Second sequence to compare.
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        matrix_mode (str): Type of matrix
            'SUBST'            Substitution matrix
            'WEIGHT'           Weight matrix
        mode (str): Computation mode:
            'GLOBAL'           Global Alignment
            'LOCAL'            Local Alignment
            'LONG_SUBSTRING'   Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)

    Returns:
        (int): distance between sequences
    """
    align = AlignSequences([s1, s2])
    align.set_scores(0, 0, score_gap_ini, score_gap_cont)
    align.set_subst_matrix(matrix)
    align.set_matrix_mode(matrix_mode)
    align.compute(mode.upper(), silent = True)
    identity = round((align.matches + 1) * 100 / (align.matches + align.unmatches + 2))
    return 100 - identity

def guide_tree_UPGMA(sequences, matrix, matrix_mode, mode,\
                     score_gap_ini, score_gap_cont,\
                     score_match, score_no_match):
    """
    Performs initial pairwise alignments against the class AlignSequences
    returning the guide_tree derived from UPGMA method.

    Args:
        sequences (lit of str): Sequences to align
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        matrix_mode (str): Type of matrix
            'SUBST'            Substitution matrix
            'WEIGHT'           Weight matrix
        mode (str): Computation mode:
            'GLOBAL'           Global Alignment
            'LOCAL'            Local Alignment
            'LONG_SUBSTRING'   Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)

    Returns:
```

```
        (list of 3-tuples of int): guide three, the third position of the tuple contains the root
                                of the other two nodes.
        (dict of int, boolean = True): contains all nodes

    """
    tree = {} #initial tree
    guide_tree = [] #guided tree, pairs to align in sequence
    max_score = MIN_SCORE
    max_score_position = ()
    for i in range(0, len(sequences)):
        for j in range(0 , i):
            if (i,j) not in tree:
                score = pairwise_align(sequences[i], sequences[j], matrix, matrix_mode, mode,\
                                score_gap_ini, score_gap_cont, score_match, score_no_match)
                tree[(i,j)] = score
                if score >= max_score:
                    max_score = score
                    max_score_position = (i,j)

    print(tree)
    len_tree = len(sequences)
    guide_tree_nodes = {}
    # Generate guide tree. At every step we compute another row averaging the
    # most closer rows and removing all their row coordinates from the tree
    while len(tree) > 0:
        (imax, jmax) = max_score_position
        guide_tree.append((imax, jmax, len_tree))
        guide_tree_nodes[imax] = True
        guide_tree_nodes[jmax] = True
        guide_tree_nodes[len_tree] = False

        # Average scores from i,j rows into new row in new_row_pos
        for j in range(0, len_tree):
            if j in [imax, jmax]:
                continue
            nscores = 0.0;
            for coordinate in [(imax, j), (j, imax), (jmax, j), (j, jmax)]:
                if coordinate in tree:
                    score = tree[coordinate]
                    nscores += 1
                    if (len_tree, j) not in tree:
                        tree[(len_tree, j)] = score
                    else:
                        tree[(len_tree, j)] += score
            if nscores > 0:
                tree[(len_tree, j)] = tree[(len_tree, j)] / nscores

        # Tree cleaning and calc max score
        max_score = MIN_SCORE
        max_score_position = ()
        for i in range(0, len_tree + 1):
```

```
            for j in range(0, len_tree + 1):
                if (i,j) in tree:
                    if i == imax or i == jmax or j == imax or j == jmax:
                        del(tree[(i,j)])
                    else:
                        if tree[(i,j)] >= max_score:
                            max_score =  tree[(i,j)]
                            max_score_position = (i,j)

        len_tree += 1

    return guide_tree, guide_tree_nodes

def q(i, j, nseq, n, dmatrix):
    """
    NJ method: calculate element of intermediate Q matrix.
    """
    d = (nseq - 2) * dmatrix[(i,j)]
    for k in range(0, n):
        if (i,k) in dmatrix:
            d -= dmatrix[(i,k)]
        if (j,k) in dmatrix:
            d -= dmatrix[(j,k)]
    return d

def calc_qmatrix(nseq, n, dmatrix):
    """
    NJ method: calculate intermediate Q matrix.
    """
    qmatrix = {}
    for (i,j) in dmatrix:
        qmatrix[(i,j)] = q(i, j, nseq, n, dmatrix)
    return qmatrix

def smallest_q(qmatrix):
    """
    NJ method: returns the coordinates of the minimun score in intermediate Q matrix.
    """
    sq = ()
    min_sq = - MIN
    for key in qmatrix.keys():
        if qmatrix[key] <  min_sq:
            min_sq = qmatrix[key]
            sq = key
    return sq

def djoin(joined_pair, nseq, n, dmatrix):
    """
    NJ method: returns distances of joined nodes to the rooted node, so, it returns the
    ↪ branch lengths
    """
    (i, j) = joined_pair
```

```
    d_i_1 = dmatrix[(i,j)] / 2.0
    d_i_2 = 0
    for k in range(0, n):
        if (i,k) in dmatrix:
            d_i_2 += dmatrix[(i,k)]
        if (j,k) in dmatrix:
            d_i_2 -= dmatrix[(j,k)]
    d_i = d_i_1 - d_i_2 / (2*(nseq - 2))
    d_j = dmatrix[(i,j)] - d_i
    return d_i, d_j

def dnjoin(k, joined_pair, dmatrix):
    """
    NJ method: returns distance of sequence k to the new nodetht routes the joined_pair.
    The distance is the mean of the distances ftom k to each of nodes joined.
    """
    (i, j) = joined_pair
    d_k = 0
    if (i,k) in dmatrix:
        d_k += dmatrix[(i,k)]
    if (j,k) in dmatrix:
        d_k += dmatrix[(j,k)]
    d_k = (d_k - dmatrix[(i,j)]) / 2.0
    return d_k

def recalc_dmatrix(joined_pair, n, dmatrix):
    """
    NJ method: recalc distance matrix taking into account the joined pair
    """
    (i, j) = joined_pair
    # Recalculate distances
    for k in range(0, n):
        if (i,k) in dmatrix and (j,k) in dmatrix:
            dmatrix[(n + 1, k)] = dnjoin(k, joined_pair, dmatrix)
            dmatrix[(k, n + 1)] = dmatrix[(n + 1, k)]
    # Remove joined rows from dmatrix
    for k in range(0, n + 1):
        for l in range(0, n + 1):
            if k == i or k == j or l == i or l== j:
                if (k, l) in dmatrix:
                    del(dmatrix[(k, l)])
    return

def guide_tree_NJ(sequences, matrix, matrix_mode, mode,\
                  score_gap_ini, score_gap_cont,\
                  score_match, score_no_match):
    """
    Performs initial pairwise alignments against the class AlignSequences
    returning the guide_tree derived from NJ method.

    Args:
        sequences (lit of str): Sequences to align
```

```
304         matrix (dict of tuples of int): Substitution matrix, Biopython format.
305         matrix_mode (str): Type of matrix
306             'SUBST'             Substitution matrix
307             'WEIGHT'            Weight matrix
308         mode (str): Computation mode:
309             'GLOBAL'            Global Alignment
310             'LOCAL'             Local Alignment
311             'LONG_SUBSTRING'    Obtain long common substring
312         score_gap_ini (int): Score of gap init.
313         score_gap_cont (int): Score of gap continuation.
314         score_match (int): Score of match characters (used if no matrix informed)
315         score_no_match (int): Score of no match characters (used if no matrix informed)
316 
317     Returns:
318         (list of 3-tuples of int): guide three, the third position of the tuple contains the
    ↪ root
319                                 of the other two nodes.
320         (dict of int, boolean = True): contains all nodes
321 
322     """
323     dmatrix = {} #initial distance matrix
324     n = len(sequences)
325     for i in range(0, n):
326         for j in range(0 , i):
327             if (i,j) not in dmatrix:
328                 distance = pairwise_align_distance(sequences[i], sequences[j], matrix,
                    ↪  matrix_mode,\
329                                    mode, score_gap_ini, score_gap_cont)
330                 dmatrix[(i,j)] = distance
331                 dmatrix[(j,i)] = distance
332     nseq = n
333     new_nodes = n - 2
334     guide_tree = [] #guided tree, pairs to align in sequence
335     guide_tree_nodes = {} #guided tree rooted nodes to complete
336     for i in range(0, n):
337         guide_tree_nodes[i] = False
338     while new_nodes > 0:
339         qmatrix = calc_qmatrix(nseq, n, dmatrix)
340         (joined_i, joined_j) = smallest_q(qmatrix)
341         #print("JOIN:", (joined_i, joined_j))
342         guide_tree.append((joined_i, joined_j, n + 1))
343         guide_tree_nodes[joined_i] = True
344         guide_tree_nodes[joined_j] = True
345         guide_tree_nodes[n + 1] = False
346         recalc_dmatrix((joined_i, joined_j), n, dmatrix)
347         n += 1
348         nseq -= 1
349         new_nodes -= 1
350     # Root the tree
351     #print("DMATRIX:", dmatrix)
352     rooting_tuple = []
353     for node in guide_tree_nodes:
```

```
        if not guide_tree_nodes[node]:
            rooting_tuple.append(node)
    rooting_tuple.append(n + 1)
    guide_tree_nodes[n + 1] = True
    #print("Rooting tuple:", rooting_tuple)
    if len(rooting_tuple) == 3:
        guide_tree.append(tuple(rooting_tuple))
    assert len(rooting_tuple) == 3
    return guide_tree, guide_tree_nodes

def gapeator(a, a_gapped, b_stack, b_stack_refs):
    """
    Introduces gaps in all the sequences of b_stack taking into account the positions
    and the gaps introduced in sequence a to obtain sequence a_gapped
    Args:
        a (str): template sequence not gapped
        a_gapped (str): template sequence gapped
        b_stack (list of str): stack of b sequences ungapped
        b_stack_refs (list of dict): stack of references to original positions
    Returns:
        list of str: stack b gapped as a does
        list of dict: stack b coordinates refered to original sequence
    """
    ini_a_gapped = a_gapped
    b_gapped_stack = []
    b_references_stack = []
    len_a_gapped = len(a_gapped)
    for b, b_refs in zip(b_stack, b_stack_refs):
        b_gapped = ""
        b_gapped_references = {}
        a_gapped = ini_a_gapped
        base_ref = 0
        for k, (i, j) in enumerate(zip(a, b)):
            index = a_gapped.index(i)
            a_gapped = a_gapped[index + 1:]
            #print("a_gapped", a_gapped )
            #print(i, j, index)
            b_gapped += "-" * index + j
            if k in b_refs:
                b_gapped_references[base_ref + k + index] =  b_refs[k]
            base_ref += index
            #print("b_gapped", b_gapped )
        b_gapped += b[k+1:]
        remaining_gaps = "-" * (len_a_gapped - len(b_gapped))
        b_gapped +=  remaining_gaps
        b_gapped_stack.append(b_gapped)
        b_references_stack.append(b_gapped_references)
    return b_gapped_stack, b_references_stack

def pairwise_align_msa_step(stack_0, stack_1, sequences, matrix, matrix_mode,\
                            mode, stack_0_indexes, stack_1_indexes, stack_0_refs,
                             ↪  stack_1_refs,\
```

```
                            score_match, score_no_match, score_gap_ini, score_gap_cont):
    """
    Performs msa alignment of sequence stack 0 and 1.

    Args:
        stack_0 (list of str): First stack of sequences to align.
        stack_1 (list of str): Secong stack of sequences to align.
        sequences (list of str): Sequences to align.
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        matrix_mode (str): Type of matrix
            'SUBST'             Substitution matrix
            'WEIGHT'            Weight matrix
        mode (str): Computation mode:
            'GLOBAL'            Global Alignment
            'LOCAL'             Local Alignment
            'LONG_SUBSTRING'    Obtain long common substring
        stack_0_indexes(list of int) : Indexes of initial sequences related to stack
    ↪  sequences 0
        stack_1_indexes(list of int) : Indexes of initial sequences related to stack
    ↪  sequences 1
        stack_0_refs(list of dict) : stack_0 references to original sequences
        stack_1_refs(list of dict) : stack_1 references to original sequences
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.

    Returns:
        list of str: stack_0 gapped (with the gaps necessary for the alignment)
        list of str: stack_0 gapped (with the gaps necessary for the alignment)
        list of dict: stack_0 references to original sequences
        list of dict: stack_1 references to original sequences
    """
    align = AlignSequences([stack_0[0], stack_1[0]])
    align.set_scores(score_match, score_no_match, score_gap_ini, score_gap_cont)
    align.set_subst_matrix(matrix)
    align.set_matrix_mode(matrix_mode)
    align.set_stacks(stack_0, stack_1, stack_0_indexes, stack_1_indexes, stack_0_refs,
    ↪  stack_1_refs)
    align.compute(mode.upper(), silent = True)
    # align_seq0 align_seq1 are the seq0 and seq1 alignments
    # we need to deduce the rest of alignments.
    # what we do is perform the same gap insertions, if any, as the first sequence of the
    ↪  stacks
    # the gap insertions where performed taken into account the initial sequence
    # to compute the references to inital sequence in order to employ a weight matrix if
    ↪  informed
    stack_0_gapped, stack_0_references = gapeator(stack_0[0], align.align_seq0, stack_0,
    ↪  stack_0_refs)
    stack_1_gapped, stack_1_references = gapeator(stack_1[0], align.align_seq1, stack_1,
    ↪  stack_1_refs)
    return stack_0_gapped, stack_1_gapped, stack_0_references, stack_1_references
```

```python

def get_name(index, sequence_names):
    """
    Obtain sequence name from index
    """
    name = ""
    if index < len(sequence_names):
        name = sequence_names[index]
    else:
        name = str(index)
    return name

def to_newick(tree, sequence_names):
    """
    Obtain guide tree in newick format
    """
    # Change format to intermediate roots
    roots = {}
    newick_tree = ""
    for branch in tree:
        (i, j, k) = branch
        name_i = get_name(i, sequence_names)
        name_j = get_name(j, sequence_names)
        name_k = get_name(k, sequence_names)
        if name_i in roots:
            new_root_i = roots[name_i]
        else:
            new_root_i = name_i
        if name_j in roots:
            new_root_j = roots[name_j]
        else:
            new_root_j = name_j
        roots[name_k] = [new_root_i, new_root_j]

    for root in roots.values():
        s_root = str(root)
        if len(s_root) > len(newick_tree):
            newick_tree = s_root.replace("[","(").replace("]",")").replace("'","")

    return newick_tree
```

1.12 T-COFFEE methods

Script 1.12.1 (python)

```python
# T-COFFEE specific methods
def pairwise_align_coffee(s1, s2, matrix, mode, score_gap_ini=0, score_gap_cont=-8,\
                  score_match=3, score_no_match=-2):
    """
    Performs initial pairwise alignments against the class AlignSequences
```

```
    returning the %identity and the alignments to construct the primary library

    Args:
        s1 (str): First sequence to compare.
        s2 (str): Second sequence to compare.
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        mode (str): Computation mode:
            'GLOBAL'           Global Alignment
            'LOCAL'            Local Alignment
            'LONG_SUBSTRING'   Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)

    Returns:
         int: % identity between sequences
         str: sequence 1 aligned
         str: sequence 2 aligned
    """
    align = AlignSequences([s1, s2])
    align.set_scores(score_match, score_no_match, score_gap_ini, score_gap_cont)
    align.set_subst_matrix(matrix)
    align.compute(mode.upper(), silent = True)
    return round((align.matches + 1) * 100 / (align.matches + align.unmatches + 2)), \
            align.align_seq0, align.align_seq1

def get_pos(k, seq_i, align_i):
    """
    Obtain position of a character in the original sequence given the
    position in the alignment(k), the original sequence (seq_i)
    and the align_i (gapped) sequence
    """
    char = align_i[k]
    count_char = align_i[0:k+1].count(char)
    index = -1;
    for _ in range(0, count_char):
        index = seq_i.find(char, index + 1)
    return index

def update_weight_at_pos(weight_library, i, j, pos_i, pos_j, identity):
    """
    Update weight at pos i , j, pos_i, pos_j
    """
    if i not in weight_library:
        weight_library[i] = {}
    w_i = weight_library[i]
    if j not in w_i:
        w_i[j] = {}
    w_i_j = w_i[j]
    if pos_i not in w_i_j:
        w_i_j[pos_i] = {}
```

```
    w_i_j_pi = w_i_j[pos_i]
    if pos_j not in w_i_j_pi:
        w_i_j_pi[pos_j] = identity
    else:
        w_i_j_pi[pos_j] += identity

def update_weight_library(weight_library, i, j, identity,\
                  seq_i, seq_j, align_i, align_j):
    """
    Update weights library from alignments and %identity
    """
    for k, (c_i, c_j) in enumerate(zip(align_i, align_j)):
        if c_i != "-" and c_j != "-":
            pos_i = get_pos(k, seq_i, align_i)
            pos_j = get_pos(k, seq_j, align_j)
            #print("Position:", pos_i, pos_j)
            update_weight_at_pos(weight_library, i, j, pos_i, pos_j, identity)
            update_weight_at_pos(weight_library, j, i, pos_j, pos_i, identity)

def compute_library(sequences, matrix={}, weight_library={}, mode="GLOBAL",\
                  score_gap_ini=0, score_gap_cont=-8,\
                  score_match=3, score_no_match=-2):
    """
    Compute initial library of identities based on scores of PA
     Args:
        sequences (list of str): Sequences to compare.
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        mode (str): Computation mode:
            'GLOBAL'           Global Alignment
            'LOCAL'            Local Alignment
            'LONG_SUBSTRING'   Long substring alignment
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)

    Returns:
        list of str: primary library of alignments
    """
    primary_library = {}
    for i in range(0, len(sequences)):
        for j in range(0 , i):
            if (i,j) not in primary_library:
                identity, align_i, align_j = pairwise_align_coffee(sequences[i],
                 ↪  sequences[j],\
                                matrix, mode, score_gap_ini, score_gap_cont,\
                                score_match, score_no_match)
                update_weight_library(weight_library, i, j, identity,\
                               sequences[i], sequences[j], align_i, align_j)
                primary_library[(i,j)] = (align_i, align_j, identity)
    #print(weight_library)
    return primary_library
```

```
109
110 def extend_weigths(weight_library, i, k, j):
111     """
112     Extend weigths for pair of sequences (i,j) at pos (pos_i_posj)
113     taken into account the routes using k as
114     intermediate, by means of the alignments (i, k) and (k, j).
115     """
116     for pos_i, pos_i_j in weight_library[i][j].items():
117         for pos_j in pos_i_j.keys():
118             if pos_j in weight_library[j][k].keys():
119                 for pos_k in weight_library[j][k][pos_j].keys():
120                     if pos_k in weight_library[k][i].keys():
121                         for pos_i_new in weight_library[k][i][pos_k].keys():
122                             if pos_i_new == pos_i:
123                                 #print("Extension", pos_i, pos_j, pos_k, weight_library[i][k
                                    ↪  ][pos_i][pos_k],weight_library[j][k][pos_j][pos_k])
124                                 m = min(\
125                                     weight_library[i][k][pos_i][pos_k],\
126                                     weight_library[j][k][pos_j][pos_k])
127                                 #print("++", m, weight_library[i][j][pos_i][pos_j])
128                                 weight_library[i][j][pos_i][pos_j] +=  m
129
130 def extend_library_weigths(sequences, weight_library):
131     """
132     Extend library for all triplets of sequences
133     Taken into account the simetry i -> k -> j
134     """
135     len_sequences = len(sequences)
136     for i in range(0, len_sequences):
137         for k in range(0, len_sequences):
138              if k != i:
139                 for j in range(0, len_sequences ):
140                     if j != k and j != i:
141                         #print("Triplet:", i, k, j)
142                         extend_weigths(weight_library, i, k, j)
143
144
145 def compute_libraries(sequences, matrix,\
146                       score_gap_ini, score_gap_cont, score_match, score_no_match):
147     """
148     Compute initial library of identities based on scores of pairwise alignments
149      Args:
150         sequences (list of str): Sequences to compare.
151         matrix (dict of tuples of int): Substitution matrix, Biopython format.
152         score_gap_ini (int): Score of gap init.
153         score_gap_cont (int): Score of gap continuation.
154         score_match (int): Score of match characters (used if no matrix informed)
155         score_no_match (int): Score of no match characters (used if no matrix informed)
156
157     Returns:
158         dict : primary library of alignments
159         dict : weight matrix
```

```
    """
    weight_library = {}
    primary_library = compute_library(sequences, matrix,\
                weight_library, "GLOBAL", score_gap_ini, score_gap_cont,\
                score_match, score_no_match)

    _ = compute_library(sequences, matrix,\
                weight_library, "LOCAL", score_gap_ini, score_gap_cont,\
                score_match, score_no_match)

#     _ = compute_library(sequences, matrix,\
#                 weight_library, "LONG_SUBSTRING", score_gap_ini, score_gap_cont,\
#                 score_match, score_no_match)

    extend_library_weigths(sequences, weight_library)

    return primary_library, weight_library
```

1.13 Main MSA method

Script 1.13.1 (python)

```
# Generic MSA method
def do_msa_from_fasta(file, main_alg="CLUSTAL", method="NJ", matrix={}, matrix_mode="SUBST",\
                      mode="GLOBAL", score_gap_ini =-10, score_gap_cont=-5, score_match=3,\
                      score_no_match=-2, verbose=False):
    """
    Performs MSA alignments from fasta file
    Args:
        file (str): Name of the FASTA file.
        main_alg (str): Main algorithm:
            "CLUSTAL"       Clustal like
            "T-COFFEE"      T-COFFEE like
        method (str): NJ neighbor join / UPGMA
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        matrix_mode (str): Type of matrix
            'SUBST'            Substitution matrix
            'WEIGHT'           Weight matrix
        mode (str): Computation mode:
            'GLOBAL'           Global Alignment
            'LOCAL'            Local Alignment
            'LONG_SUBSTRING'   Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)
        verbose (bool): If True prints verbose info

    Returns:
```

```
        list of 3-tuples of int: guide three, the third position of the tuple contains the root
                                of the other two nodes.
        list of str: alignments
        list of str: sequence_names
        list of int: sequence indexes relating strings in alignment to original sequences
    """
    seq_fasta = readFasta(file)
    sequences = list(seq_fasta.values())
    sequence_names = list(seq_fasta.keys())
    return do_msa(sequences, sequence_names,\
                  main_alg, method, matrix, matrix_mode,\
                  mode, score_gap_ini, score_gap_cont, score_match,\
                  score_no_match, verbose)

def do_msa(sequences, sequence_names, main_alg="CLUSTAL", method="NJ", matrix={}, matrix_mode="SUBST",\
                      mode="GLOBAL", score_gap_ini=-10, score_gap_cont=-5, score_match=3,\
                      score_no_match=-2, verbose=False):
    """
    Performs MSA alignments from sequences
    Args:
        sequences (list of str): Sequences
        sequence_names (list of str): Names of sequences
        main_alg (str): Main algorithm:
            "CLUSTAL"       Clustal like
            "T-COFFEE"      T-COFFEE like
        method (str): NJ neighbor join / UPGMA
        matrix (dict of tuples of int): Substitution matrix, Biopython format.
        matrix_mode (str): Type of matrix
            'SUBST'             Substitution matrix
            'WEIGHT'            Weight matrix
        mode (str): Computation mode:
            'GLOBAL'            Global Alignment
            'LOCAL'             Local Alignment
            'LONG_SUBSTRING'    Obtain long common substring
        score_gap_ini (int): Score of gap init.
        score_gap_cont (int): Score of gap continuation.
        score_match (int): Score of match characters (used if no matrix informed)
        score_no_match (int): Score of no match characters (used if no matrix informed)
        verbose (bool): If True prints verbose info

    Returns:
        list of 3-tuples of int: guide three, the third position of the tuple contains the root
                                of the other two nodes.
        list of str: alignments
        list of str: sequence_names
        list of int: sequence indexes relating strings in alignment to original sequences
    """
    if main_alg == "T-COFFEE":
        primary_library, weight_library = compute_libraries(sequences, matrix,\
```

```
                        score_gap_ini, score_gap_cont, score_match, score_no_match)
        #print("Primary library", primary_library)
        #print("Weight library", weight_library)
        # Matrix mode and other MSA parameters
        matrix = weight_library
        matrix_mode = "WEIGHT"
#         score_gap_ini = 0
#         score_gap_cont = 0
        # From here only we need is to compute a MSA with weight matrix as reference.
    else:
        matrix_mode = "SUBST"
    if method == "NJ":
        guide_tree, guide_tree_nodes =\
            guide_tree_NJ(sequences, matrix, matrix_mode,\
            mode, score_gap_ini, score_gap_cont,\
            score_match, score_no_match)
    else: #UPGMA
        guide_tree, guide_tree_nodes =\
            guide_tree_UPGMA(sequences, matrix, matrix_mode,\
            mode, score_gap_ini, score_gap_cont,\
            score_match, score_no_match)
    sequences_store = {}
    sequences_store_indexes = {}
    sequences_store_refs = {}
    print("Guide Tree",guide_tree, sequence_names)
    #return guide_tree, "", sequence_names
    # Create MSA
    for i in guide_tree_nodes.keys():
        if i < len(sequences):
            sequences_store[i] = [sequences[i]]
            sequences_store_indexes[i] = [i]
            sequences_store_refs[i] = []
            autorefs = {}
            for k in range(0, len(sequences[i])):
                autorefs[k] = k
            sequences_store_refs[i].append(autorefs)

    if verbose: print(sequences_store)
    for (i ,j ,k) in guide_tree:
        stack_i = sequences_store[i]
        stack_j = sequences_store[j]
        stack_i_indexes = sequences_store_indexes[i]
        stack_j_indexes = sequences_store_indexes[j]
        stack_i_references = sequences_store_refs[i]
        stack_j_references = sequences_store_refs[j]
        if verbose: print("Stack i", i, stack_i)
        if verbose: print("Stack j", j, stack_j)
        if verbose: print("Stack i_indexes", i, stack_i_indexes)
        if verbose: print("Stack j_indexes", j, stack_j_indexes)
        stack_0, stack_1, stack_0_references, stack_1_references =\
            pairwise_align_msa_step(stack_i, stack_j, sequences, matrix, matrix_mode,\
                                    mode, stack_i_indexes, stack_j_indexes,\
```

```
                                    stack_i_references, stack_j_references,\
                                    score_match, score_no_match, score_gap_ini,
                                    ↪  score_gap_cont)
        sequences_store[k] = []
        sequences_store_indexes[k] = []
        sequences_store_refs[k] = []
        if verbose: print("==================")
        for s in stack_0:
            if verbose: print(s)
            sequences_store[k].append(s)
        for s in stack_1:
            if verbose: print(s)
            sequences_store[k].append(s)
        for seq_index in stack_i_indexes:
            sequences_store_indexes[k].append(seq_index)
        for seq_index in stack_j_indexes:
            sequences_store_indexes[k].append(seq_index)
        for seq_refs in stack_0_references:
            sequences_store_refs[k].append(seq_refs)
        for seq_refs in stack_1_references:
            sequences_store_refs[k].append(seq_refs)
        if verbose: print("==================")
        if verbose: print("Sequences store indexes",sequences_store_indexes[k])
        if verbose: print("Sequences store references",sequences_store_refs[k])
        if verbose: print("New stack:", k, sequences_store[k])
    alignment = sequences_store[k]
    newick_tree = to_newick(guide_tree, sequence_names)
    return newick_tree, alignment, sequence_names, sequences_store_indexes[k]

def score(alignment, matrix, score_gap_ini=0, score_gap_cont=0):
    """
    Score based on sum of pair scores (SOP) taking into account substitution matrix
    Derived from the objetive score of MUSCLE refinement stage
    """
    msa_score = 0
    for k in range(0, len(alignment[0])): #columns of msa
        score_column_k = 0
        nvalues = 0
        for i in range (0, len(alignment)):
            for j in range (i + 1, len(alignment)):
                if alignment[i][k] == "-" and alignment[j][k] != "-":
                    score_column_k += score_gap_cont
                    if k == 0 or alignment[i][k-1] != "-":
                            score_column_k += score_gap_ini
                if alignment[j][k] == "-" and alignment[i][k] != "-":
                    score_column_k += score_gap_cont
                    if k == 0 or alignment[j][k-1] != "-":
                            score_column_k += score_gap_ini
                elif (alignment[i][k], alignment[j][k]) in matrix:
                    score_column_k += matrix[(alignment[i][k], alignment[j][k])]
                    nvalues += 1
                elif (alignment[j][k], alignment[i][k]) in matrix:
```

```
                    score_column_k += matrix[(alignment[j][k], alignment[i][k])]
                    nvalues += 1
        if nvalues > 0:
            #score += score_column_k / nvalues
            msa_score += score_column_k
    return msa_score

def score_from_fasta(file, matrix, score_gap_ini=0, score_gap_cont=0):
    seq_fasta = readFasta(file)
    sequences = list(seq_fasta.values())
    return score(sequences, matrix, score_gap_ini, score_gap_cont)
```

1.14 T-COFFEE RAG1_AA EXEC

Script 1.14.1 (python)

```
matrix = MatrixInfo.blosum80
file = "files/RAG1_AA.fas"
quality_score_gap_ini = -10
quality_score_gap_cont = -5

guide_tree_upgma, align, sequence_names, indexes = do_msa_from_fasta(file,\
                main_alg = "T-COFFEE", method = "UPGMA", \
                matrix = matrix, matrix_mode = "SUBST",\
                mode = "GLOBAL", score_gap_ini = -10,\
                score_gap_cont = -5, score_match = 3, score_no_match = -2, verbose = False)

print("# Guide Tree:", guide_tree_upgma)
t_upgma, ts_upgma = draw_guide_tree(guide_tree_upgma)
print("# Alignment:")
for i, s in enumerate(align):
    print(">" + sequence_names[indexes[i]])
    print(s)
print()
print("Score", score(align, MatrixInfo.blosum62, quality_score_gap_ini,
 ↪  quality_score_gap_cont))
print()

guide_tree_nj, align, sequence_names, indexes = do_msa_from_fasta(file,\
                main_alg = "T-COFFEE", method = "NJ", \
                matrix = matrix, matrix_mode = "SUBST",\
                mode = "GLOBAL", score_gap_ini = -10,\
                score_gap_cont = -5, score_match = 3, score_no_match = -2, verbose = False)
print("# Guide Tree:", guide_tree_nj)
t_nj, ts_nj = draw_guide_tree(guide_tree_nj)
print("# Alignment:")
for i, s in enumerate(align):
    print(">" + sequence_names[indexes[i]])
    print(s)
print()
```

```
34 print("Score", score(align, matrix, quality_score_gap_ini, quality_score_gap_cont))
```

Output

```
{(1, 0): 72, (2, 0): 75, (2, 1): 79, (3, 0): 74, (3, 1): 77, (3, 2): 81, (4, 0): 73, (4, 1):
 ↪  79, (4, 2): 82, (4, 3): 80, (5, 0): 73, (5, 1): 79, (5, 2): 81, (5, 3): 79, (5, 4): 85,
 ↪  (6, 0): 73, (6, 1): 78, (6, 2): 82, (6, 3): 81, (6, 4): 88, (6, 5): 87, (7, 0): 73, (7,
 ↪  1): 79, (7, 2): 83, (7, 3): 82, (7, 4): 87, (7, 5): 91, (7, 6): 93, (8, 0): 75, (8, 1):
 ↪  79, (8, 2): 83, (8, 3): 82, (8, 4): 87, (8, 5): 87, (8, 6): 89, (8, 7): 91, (9, 0): 74,
 ↪  (9, 1): 80, (9, 2): 83, (9, 3): 84, (9, 4): 88, (9, 5): 87, (9, 6): 89, (9, 7): 91, (9,
 ↪  8): 93}
Guide Tree [(9, 8, 10), (7, 6, 11), (11, 10, 12), (12, 5, 13), (13, 4, 14), (14, 2, 15), (15,
 ↪  3, 16), (16, 1, 17), (17, 0, 18)] ['Oncorhynchus_mykiss', 'Carcharhinus_leucas',
 ↪  'Latimeria_menadoensis', 'Protopterus_dolloi', 'Alytes_obstetricans',
 ↪  'Anolis_carolinensis', 'Gallus_gallus', 'Alligator_mississippiensis',
 ↪  'Ornithorhynchus_anatinus', 'Homo_sapiens']
# Guide Tree: (((((((Alligator_mississippiensis, Gallus_gallus), (Homo_sapiens,
 ↪  Ornithorhynchus_anatinus)), Anolis_carolinensis), Alytes_obstetricans),
 ↪  Latimeria_menadoensis), Protopterus_dolloi), Carcharhinus_leucas), Oncorhynchus_mykiss)

                      /-Alligator_mississippiensis
                   /-|
                  |   \-Gallus_gallus
                /-|
               |  |   /-Homo_sapiens
             /-|   \-|
            |  |      \-Ornithorhynchus_anatinus
          /-|  |
         |  |   \-Anolis_carolinensis
       /-|  |
      |  |   \-Alytes_obstetricans
    /-|  |
   |  |   \-Latimeria_menadoensis
 /-|  |
|  |   \-Protopterus_dolloi
--|  |
  |   \-Carcharhinus_leucas
  |
   \-Oncorhynchus_mykiss
# Alignment:
>Alligator_mississippiensis
RTVKAVTGRQIFQPLHALRTAEKALLPGYHSFEWKPPLKNVSANTEVGIIDGLSGLPHTVDDYPIDTIAKRFRYDAALVSALMDMEEDILEGM
 ↪  KAHDLDDYLNG-PFTVVVKESCDGMGDVSEKHGCGPAVPEKAVRFSFTVMTIAI--THGNTNVRIFEEVKPNSELCCKPLCLMLADESD
 ↪  HETLTAILSPLIAERETMKNSVLLLEMGGILRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLVLHSITRSHTEN
 ↪  LERYEVWRSNPYHESVDELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQFEIGEVYKNPDASKEERKRWQSALDKHLRKKMN
 ↪  LKPIMRMNGNFARKLMTKETVEAVCELIKCEERHEALKELMDLYLKMKPVWRSSCPAKECPELLCQYSFNSQRFAELLSTKFKYRYEGK
 ↪  ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
>Gallus_gallus
```

```
RTVKAVTGRQIFQPLHALRTAEKALLPGYHPFEWKPPLKNVSTNTEVGIIDGLSGLPLSIDDYPIDTIAKRFRYDTALVSALKDMEEEILEGM
↪ KAKNLDDYLNG-PFTVVVKECCDGMGDVSEKHGSGPAVPEKAVRFSFTVMNIAI--DHENERIRIFEEVKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKNSELLLEIGGILRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLVFHSITRSHAEN
↪ LERYEIWRSNPYHESVDELRDRVKGVSAKPFIETVPSIDALHCDIGNATEFYRIFQMEIGEVYKNPDATKEERKRWQLTLDKHLRKKMK
↪ LKPMMRMSGNFARKLMSKETVEAVCELIKCEERHEALKELMDLYLKMKPVWRSSCPAKECPELLCQYSYNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Homo_sapiens
RTVKAITGRQIFQPLHALRNAEKVLLPGYHHFEWQPPLKNVSSSTDVGIIDGLSGLSSSVDDYPVDTIAKRFRYDSALVSALMDMEEDILEGM
↪ RSQDLDDYLNG-PFTVVVKESCDGMGDVSEKHGSGPVVPEKAVRFSFTIMKITI--AHSSQNVKVFEEAKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKSSELMLELGGILRTFKFIFRGTGYDEKLVREVEGLEASGSVYICTLCDATRLEASQNLVFHSITRSHAEN
↪ LERYEVWRSNPYHESVEELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQLEIGEVYKNPNASKEERKRWQATLDKHLRKKMN
↪ LKPIMRMNGNFARKLMTKETVDAVCELIPSEERHEALRELMDLYLKMKPVWRSSCPAKECPESLCQYSFNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
>Ornithorhynchus_anatinus
RTVKAITGRQIFQPLHSLRTAEKVLLPGYHPFEWDPPLKNVSANTEVGIMDGLSGLPVSVDDYPVDTIAKRFRYDAALVSALMDMEEDILEGM
↪ KSQDLDDYLSG-PFTVVIKESCDGMGDVSEKHGSGPAVPEKAVRFSFTVMNITL--AYEQENVKIFEEAKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKDSELKLEMGGILRSFRFIFRGTGYDEKLVREVEGLEASGSVYICTLCDATRLEASQNLVLHSITRSHAEN
↪ LERYEVWRSNPFHESVEELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQLEIGEAYKNPNASKEERKRWQATLDKHLRKKMK
↪ LKPIMRMNGNFARKLMTKETVEAVCELVHCEERHEALRELMDLYLKMKPVWRSSCPAKECPESLCQYSFNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
>Anolis_carolinensis
RTVKAVTGRQIFQPLHALRTAEKALLPGYHQFEWKPPLKNVSSNTEVGIIDGLSGIQHLVDDYPVDTIAKRFRYDAALASALMDMEEDILEGL
↪ KRQDLDDYFKG-PFTVVIKESCDGMGDVSEKHGCGPAVPEKAVRFSFTLMTISV--THGNASIRVFEECKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLVAEREAMKDSVLILDMAGIPRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLILHSITRSHAEN
↪ LERYELWRTNPYHETVDELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQFEIGEVYKNTDASKEERRRWQSTLDKHLRKKMN
↪ LKPMTRMNGNFARKLMTKETVEAVCELIKSEERHEALRELMDLYLKMKPVWRSSCPTKECPELVCQYSFNSQRFAELLATKFRYRYAGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Alytes_obstetricans
RTVKATTGRQIFQPLHALRNAEKALLPGYHPFEWKPPLKNVSTCTDTGIIDGLSGLNRSIDEYPVEAISKRFRYDTALVSALKDMEEDILEGL
↪ RSHDMDDYLNG-PFTVVIKESCDGMGDVSEKHGSGPAVPEKAVRFSFTVMYISV--PNNNECVRIFDETKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKTSELMLEMGGILRNFKFIFRGTGYDEKLVREVEGLEASGSVYICTLCDSTRLEASQNLVFHSITRCHTEN
↪ LQRYETWRANPHHESVDELRDRVKGVSAKPFIETLPSIDALHCDIGNAAEFYRLFQLEIGEVYKNPNATKEERKRWQSTLDKHLRKKMN
↪ LKPIMRMNGNFARKLMSKETVEAVCELVHSEERQEILRELMDLYLKMKPVWRSSCPAKECPELLYQYSFHSQRFAELLSTKFKYRYAGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Latimeria_menadoensis
RTVKATTGKQIFQPLHSLRNAEKALLPGFHPFEWQPPLKNVSSTTEVGIIDGMSGMTQFVDEYPLDTISKRFRYDAALVSALKDLEEELLKGL
↪ IEEDLEDYLSG-PFTVIIKESCDGMGDVSEKHGSGPAVPEKAVRYSFTIMTISV-ANSHNENVTIFEEGKPNSELCCKPLCLMLADESD
↪ HETLTAILGPVIAEREAMKNSELFLEMGGILRSFKFIFRGTGYDEKLIRDVEGLEASGSSYICTLCDSTRSEASQNFILHSITRSHKEN
↪ LERYEIWRSNPYQEPVEELRDRVKGVSAKPFIETLPSIDALHCDIGNATEFYKIFQDEIGEIYKNPNPSREEKKRWHSVLDKHLRKNMN
↪ LKPVMRMNGNYARKLMTKETVNAVCELIPSEERQEALKELVDLYLKMKPVWRSTCPAKECPELLCQYSFHSQRFAELLSTMYRYRYEGK
↪ ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKYYELEDVL
>Protopterus_dolloi
RTVKAATGRQIFQPLHALRSAEKALLPGYHPFEWQPPLVGVSSSTDVGIINGLSGLTSSVDEYPVEALAKRFRYDAALVSALKDIEENILEGM
↪ KQNGLDEYLSG-PFTVVIKESCDGMGDVSEKHGSGPPVPEKAVRFSFTIMSISV-AMSDSENVQIFEEFKPNSELSCKPLCLMIADESD
↪ HETLTVILGPVIAEREAMKTSELMLELGGILRTFKFFFRGTGYDEKLVREVEGLEASGSHYICTLCDATRQEASRNLVLHSITRSHAEN
↪ LERYEVWRSNPYNESVDELRDRVKGVSAKPFIETRPCIDALQCDIGNATEFYKIFQDEVGEVYKRPNPSKEDRKRWHMTLDKHLRKKLS
↪ LKPVMRMNGNFARKLITKEAVDAVCELIPSEERRAAIRDLVHLYMLMKPVWRSTYPAKECPELLCQYSFNSQRFAELLSTKFQYRYEAK
↪ ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYELEDVL
>Carcharhinus_leucas
```

```
KTVKAITGKQIFQPLHALRNAEKTLLPGYYSFEWQPPLANISTNTRVGIIDGLSGWVQCVDDYPMETISRRLSYDVALASAVKEMEDDILEGL
↪  RSQNVDEFVSG-PFTVVIKESCDGMGDVSEKHGCGPTVPEKAVRYSFTIMSISV-MNENNEKVKVFEEMKPNSELCCRPLCLMLADESD
↪  RETLTAILGPVIAERQSMKTSDLIVEIGDLYRSFQFIFRGTGYDEKLVREVEGLEASGSIYICTLCDSTRSEASKNMVLHSITRNHAEN
↪  LERYEIWRSNPYHETADELRDRVKGVSAKPFIETQPSIDALHCDIGNATEFYRIFQDEIGEVYKNSNSSKEERKRWQSMLDKHLRKKMN
↪  LKPIMRMNGNFARKLMTKETVEAVCELIPSEERREILRELMHLYLLMKPVWRSTFPTTECPDLLCQYSFNSQRFAELLHTEFSHRYEGK
↪  ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKSYELEDIL
>Oncorhynchus_mykiss
RTVKATSGRQIFQPLHTLRTAEKELLPGYHPFEWQPALKSVSTSCHVGIIDGLSGWIASVDDSPADTVTRRFRYDVALVSALKDLEEDIMEGL
↪  RERGLEDSACTSGFSVMIKESCDGMGDVSEKHGGGPPVPEKPVRFSFTIMSVSIQAEGEDEAITIFREPKPNSEMSCKPLSLMFVDESD
↪  HETLTGVLGPVVAERNAMKHSRLILSVGGLSRSFRFHFRGTGYDEKMVREMEGLEASGSTYICTLCDSTRAEASQNMTLHSVTRSHDEN
↪  LERYELWRTNPHSESAEELRDRVKGVSAKPFMETQPTLDALHCDIGNATEFYKIFQDEIGEVYHKANPSREQRRSWRAALDKQLRKKMK
↪  LKPVMRMNGNYARKLMTREAVEAVCELVCSEERQEALRELMGLYIQMKPVWRSTCPAKECPDELCRYSFNSQRFAELLSTVFKYRYDGK
↪  ITNYLHKTLAHVPEIVERDGSIGAWASEGNESGNKLFRRFRKMNARQSKTFELEDVL

Score 99825

Guide Tree [(7, 6, 11), (11, 5, 12), (9, 8, 13), (12, 4, 14), (13, 3, 15), (14, 2, 16), (16,
↪  1, 17), (15, 0, 18), (17, 18, 19)] ['Oncorhynchus_mykiss', 'Carcharhinus_leucas',
↪  'Latimeria_menadoensis', 'Protopterus_dolloi', 'Alytes_obstetricans',
↪  'Anolis_carolinensis', 'Gallus_gallus', 'Alligator_mississippiensis',
↪  'Ornithorhynchus_anatinus', 'Homo_sapiens']
# Guide Tree: ((((((Alligator_mississippiensis, Gallus_gallus), Anolis_carolinensis),
↪  Alytes_obstetricans), Latimeria_menadoensis), Carcharhinus_leucas), (((Homo_sapiens,
↪  Ornithorhynchus_anatinus), Protopterus_dolloi), Oncorhynchus_mykiss))

                  /-Alligator_mississippiensis
                /-|
              /-|   \-Gallus_gallus
             |  |
           /-|   \-Anolis_carolinensis
          |  |
        /-|   \-Alytes_obstetricans
       |  |
     /-|   \-Latimeria_menadoensis
    |  |
    |   \-Carcharhinus_leucas
    |
--|          /-Homo_sapiens
    |       /-|
    |    /-|   \-Ornithorhynchus_anatinus
    |   |  |
     \-|   \-Protopterus_dolloi
       |
        \-Oncorhynchus_mykiss
# Alignment:
>Alligator_mississippiensis
RTVKAVTGRQIFQPLHALRTAEKALLPGYHSFEWKPPLKNVSANTEVGIIDGLSGLPHTVDDYPIDTIAKRFRYDAALVSALMDMEEDILEGM
↪  KAHDLDDYLNG-PFTVVVKESCDGMGDVSEKHGCGPAVPEKAVRFSFTVMTIAI--THGNTNVRIFEEVKPNSELCCKPLCLMLADESD
↪  HETLTAILSPLIAERETMKNSVLLLEMGGILRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLVLHSITRSHTEN
↪  LERYEVWRSNPYHESVDELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQFEIGEVYKNPDASKEERKRWQSALDKHLRKKMN
↪  LKPIMRMNGNFARKLMTKETVEAVCELIKCEERHEALKELMDLYLKMKPVWRSSCPAKECPELLCQYSFNSQRFAELLSTKFKYRYEGK
↪  ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
```

```
>Gallus_gallus
RTVKAVTGRQIFQPLHALRTAEKALLPGYHPFEWKPPLKNVSTNTEVGIIDGLSGLPLSIDDYPIDTIAKRFRYDTALVSALKDMEEEILEGM
↪ KAKNLDDYLNG-PFTVVVKECCDGMGDVSEKHGSGPAVPEKAVRFSFTVMNIAI--DHENERIRIFEEVKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKNSELLLEIGGILRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLVFHSITRSHAEN
↪ LERYEIWRSNPYHESVDELRDRVKGVSAKPFIETVPSIDALHCDIGNATEFYRIFQMEIGEVYKNPDATKEERKRWQLTLDKHLRKKMK
↪ LKPMMRMSGNFARKLMSKETVEAVCELIKCEERHEALKELMDLYLKMKPVWRSSCPAKECPELLCQYSYNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Anolis_carolinensis
RTVKAVTGRQIFQPLHALRTAEKALLPGYHQFEWKPPLKNVSSNTEVGIIDGLSGIQHLVDDYPVDTIAKRFRYDAALASALMDMEEDILEGL
↪ KRQDLDDYFKG-PFTVVIKESCDGMGDVSEKHGCGPAVPEKAVRFSFTLMTISV--THGNASIRVFEECKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLVAEREAMKDSVLILDMAGIPRTFKFIFRGTGYDEKLVREVEGLEASGSTYICTLCDATRLEASQNLILHSITRSHAEN
↪ LERYELWRTNPYHETVDELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQFEIGEVYKNTDASKEERRRWQSTLDKHLRKKMN
↪ LKPMTRMNGNFARKLMTKETVEAVCELIKSEERHEALRELMDLYLKMKPVWRSSCPTKECPELVCQYSFNSQRFAELLATKFRYRYAGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Alytes_obstetricans
RTVKATTGRQIFQPLHALRNAEKALLPGYHPFEWKPPLKNVSTCTDTGIIDGLSGLNRSIDEYPVEAISKRFRYDTALVSALKDMEEDILEGL
↪ RSHDMDDYLNG-PFTVVIKESCDGMGDVSEKHGSGPAVPEKAVRFSFTVMYISV--PNNNECVRIFDETKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKTSELMLEMGGILRNFKFIFRGTGYDEKLVREVEGLEASGSVYICTLCDSTRLEASQNLVFHSITRCHTEN
↪ LQRYETWRANPHHESVDELRDRVKGVSAKPFIETLPSIDALHCDIGNAAEFYRLFQLEIGEVYKNPNATKEERKRWQSTLDKHLRKKMN
↪ LKPIMRMNGNFARKLMSKETVEAVCELVHSEERQEILRELMDLYLKMKPVWRSSCPAKECPELLYQYSFHSQRFAELLSTKFKYRYAGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKFYEMEDVL
>Latimeria_menadoensis
RTVKATTGKQIFQPLHSLRNAEKALLPGFHPFEWQPPLKNVSSTTEVGIIDGMSGMTQFVDEYPLDTISKRFRYDAALVSALKDLEEELLKGL
↪ IEEDLEDYLSG-PFTVIIKESCDGMGDVSEKHGSGPAVPEKAVRYSFTIMTISV-ANSHNENVTIFEEGKPNSELCCKPLCLMLADESD
↪ HETLTAILGPVIAEREAMKNSELFLEMGGILRSFKFIFRGTGYDEKLIRDVEGLEASGSSYICTLCDSTRSEASQNFILHSITRSHKEN
↪ LERYEIWRSNPYQEPVEELRDRVKGVSAKPFIETLPSIDALHCDIGNATEFYKIFQDEIGEIYKNPNPSREEKKRWHSVLDKHLRKNMN
↪ LKPVMRMNGNYARKLMTKETVNAVCELIPSEERQEALKELVDLYLKMKPVWRSTCPAKECPELLCQYSFHSQRFAELLSTMYRYRYEGK
↪ ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKYYELEDVL
>Carcharhinus_leucas
KTVKAITGKQIFQPLHALRNAEKTLLPGYYSFEWQPPLANISTNTRVGIIDGLSGWVQCVDDYPMETISRRLSYDVALASAVKEMEDDILEGL
↪ RSQNVDEFVSG-PFTVVIKESCDGMGDVSEKHGCGPTVPEKAVRYSFTIMSISV-MNENNEKVKVFEEMKPNSELCCRPLCLMLADESD
↪ RETLTAILGPVIAERQSMKTSDLIVEIGDLYRSFQFIFRGTGYDEKLVREVEGLEASGSIYICTLCDSTRSEASKNMVLHSITRNHAEN
↪ LERYEIWRSNPYHETADELRDRVKGVSAKPFIETQPSIDALHCDIGNATEFYRIFQDEIGEVYKNSNSSKEERKRWQSMLDKHLRKKMN
↪ LKPIMRMNGNFARKLMTKETVEAVCELIPSEERREILRELMHLYLLMKPVWRSTFPTTECPDLLCQYSFNSQRFAELLHTEFSHRYEGK
↪ ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKSYELEDIL
>Homo_sapiens
RTVKAITGRQIFQPLHALRNAEKVLLPGYHHFEWQPPLKNVSSSTDVGIIDGLSGLSSSVDDYPVDTIAKRFRYDSALVSALMDMEEDILEGM
↪ RSQDLDDYLNG-PFTVVVKESCDGMGDVSEKHGSGPVVPEKAVRFSFTIMKITI--AHSSQNVKVFEEAKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKSSELMLELGGILRTFKFIFRGTGYDEKLVREVEGLEASGSVYICTLCDATRLEASQNLVFHSITRSHAEN
↪ LERYEVWRSNPYHESVEELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQLEIGEVYKNPNASKEERKRWQATLDKHLRKKMN
↪ LKPIMRMNGNFARKLMTKETVDAVCELIPSEERHEALRELMDLYLKMKPVWRSSCPAKECPESLCQYSFNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
>Ornithorhynchus_anatinus
RTVKAITGRQIFQPLHSLRTAEKVLLPGYHPFEWDPPLKNVSANTEVGIMDGLSGLPVSVDDYPVDTIAKRFRYDAALVSALMDMEEDILEGM
↪ KSQDLDDYLSG-PFTVVIKESCDGMGDVSEKHGSGPAVPEKAVRFSFTVMNITL--AYEQENVKIFEEAKPNSELCCKPLCLMLADESD
↪ HETLTAILSPLIAEREAMKDSELKLEMGGILRSFRFIFRGTGYDEKLVREVEGLEASGSVYICTLCDATRLEASQNLVLHSITRSHAEN
↪ LERYEVWRSNPFHESVEELRDRVKGVSAKPFIETVPSIDALHCDIGNAAEFYKIFQLEIGEAYKNPNASKEERKRWQATLDKHLRKKMK
↪ LKPIMRMNGNFARKLMTKETVEAVCELVHCEERHEALRELMDLYLKMKPVWRSSCPAKECPESLCQYSFNSQRFAELLSTKFKYRYEGK
↪ ITNYFHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYEMEDVL
>Protopterus_dolloi
```

```
RTVKAATGRQIFQPLHALRSAEKALLPGYHPFEWQPPLVGVSSSTDVGIINGLSGLTSSVDEYPVEALAKRFRYDAALVSALKDIEENILEGM
↪  KQNGLDEYLSG-PFTVVIKESCDGMGDVSEKHGSGPPVPEKAVRFSFTIMSISV-AMSDSENVQIFEEFKPNSELSCKPLCLMIADESD
↪  HETLTVILGPVIAEREAMKTSELMLELGGILRTFKFFFRGTGYDEKLVREVEGLEASGSHYICTLCDATRQEASRNLVLHSITRSHAEN
↪  LERYEVWRSNPYNESVDELRDRVKGVSAKPFIETRPCIDALQCDIGNATEFYKIFQDEVGEVYKRPNPSKEDRKRWHMTLDKHLRKKLS
↪  LKPVMRMNGNFARKLITKEAVDAVCELIPSEERRAAIRDLVHLYMLMKPVWRSTYPAKECPELLCQYSFNSQRFAELLSTKFQYRYEAK
↪  ITNYLHKTLAHVPEIIERDGSIGAWASEGNESGNKLFRRFRKMNARQSKCYELEDVL
>Oncorhynchus_mykiss
RTVKATSGRQIFQPLHTLRTAEKELLPGYHPFEWQPALKSVSTSCHVGIIDGLSGWIASVDDSPADTVTRRFRYDVALVSALKDLEEDIMEGL
↪  RERGLEDSACTSGFSVMIKESCDGMGDVSEKHGGGPPVPEKPVRFSFTIMSVSIQAEGEDEAITIFREPKPNSEMSCKPLSLMFVDESD
↪  HETLTGVLGPVVAERNAMKHSRLILSVGGLSRSFRFHFRGTGYDEKMVREMEGLEASGSTYICTLCDSTRAEASQNMTLHSVTRSHDEN
↪  LERYELWRTNPHSESAEELRDRVKGVSAKPFMETQPTLDALHCDIGNATEFYKIFQDEIGEVYHKANPSREQRRSWRAALDKQLRKKMK
↪  LKPVMRMNGNYARKLMTREAVEAVCELVCSEERQEALRELMGLYIQMKPVWRSTCPAKECPDELCRYSFNSQRFAELLSTVFKYRYDGK
↪  ITNYLHKTLAHVPEIVERDGSIGAWASEGNESGNKLFRRFRKMNARQSKTFELEDVL

Score 107036
```

1.15 CLUSTAL 16S EXEC

Script 1.15.1 (python)

```python
matrix = {}
file = "files/16S.fas"
quality_score_gap_ini = -10
quality_score_gap_cont = -5
guide_tree_upgma, align, sequence_names, indexes = do_msa_from_fasta(file,\
                main_alg = "CLUSTAL", method = "UPGMA", \
                matrix = matrix, matrix_mode = "SUBST",\
                mode = "GLOBAL", score_gap_ini = -10,\
                score_gap_cont = -5, score_match = 3, score_no_match = -2, verbose = False)
print("# Guide Tree:", guide_tree_upgma)
t_upgma, ts_upgma = draw_guide_tree(guide_tree_upgma)
print("# Alignment:")
for i, s in enumerate(align):
    print(">" + sequence_names[indexes[i]])
    print(s)
print()
#print("Score", score(align, MatrixInfo.blosum62, quality_score_gap_ini,
↪  quality_score_gap_cont))

# print()
# guide_tree_nj, align, sequence_names, indexes = do_msa_from_fasta(file,\
#                 main_alg = "CLUSTAL", method = "NJ", \
#                 matrix = matrix, matrix_mode = "SUBST",\
#                 mode = "GLOBAL", score_gap_ini = -10,\
#                 score_gap_cont = -5, score_match = 3, score_no_match = -2, verbose = False)
# print("# Guide Tree:", guide_tree_nj)
# t_nj, ts_nj = draw_guide_tree(guide_tree_nj)
# print("# Alignment:")
# for i, s in enumerate(align):
#     print(">" + sequence_names[indexes[i]])
```

```
#     print(s)
# print()
# print("Score", score(align, matrix, quality_score_gap_ini, quality_score_gap_cont))
```

Output

```
{(1, 0): 70, (2, 0): 71, (2, 1): 72, (3, 0): 69, (3, 1): 68, (3, 2): 69, (4, 0): 71, (4, 1):
↪  71, (4, 2): 70, (4, 3): 69, (5, 0): 68, (5, 1): 64, (5, 2): 67, (5, 3): 64, (5, 4): 65,
↪  (6, 0): 66, (6, 1): 65, (6, 2): 67, (6, 3): 65, (6, 4): 67, (6, 5): 67, (7, 0): 67, (7,
↪  1): 66, (7, 2): 68, (7, 3): 65, (7, 4): 66, (7, 5): 67, (7, 6): 69, (8, 0): 69, (8, 1):
↪  65, (8, 2): 67, (8, 3): 67, (8, 4): 67, (8, 5): 70, (8, 6): 67, (8, 7): 67, (9, 0): 66,
↪  (9, 1): 65, (9, 2): 68, (9, 3): 64, (9, 4): 67, (9, 5): 67, (9, 6): 67, (9, 7): 66, (9,
↪  8): 71}
Guide Tree [(2, 1, 10), (9, 8, 11), (4, 0, 12), (12, 10, 13), (7, 6, 14), (13, 3, 15), (11,
↪  5, 16), (16, 14, 17), (17, 15, 18)] ['Carcharhinus_leucas', 'Oncorhynchus_mykiss',
↪  'Latimeria_menadoensis', 'Protopterus_dolloi', 'Alytes_obstetricans',
↪  'Anolis_carolinensis', 'Alligator_mississippiensis', 'Gallus_gallus',
↪  'Ornithorhynchus_anatinus', 'Homo_sapiens']
# Guide Tree: ((((Homo_sapiens, Ornithorhynchus_anatinus), Anolis_carolinensis),
↪  (Gallus_gallus, Alligator_mississippiensis)), (((Alytes_obstetricans,
↪  Carcharhinus_leucas), (Latimeria_menadoensis, Oncorhynchus_mykiss)), Protopterus_dolloi))

            /-Homo_sapiens
          /-|
       /-|   \-Ornithorhynchus_anatinus
      |  |
    /-|   \-Anolis_carolinensis
   |  |
   |  |   /-Gallus_gallus
   |   \-|
   |      \-Alligator_mississippiensis
---|
   |         /-Alytes_obstetricans
   |       /-|
   |      |   \-Carcharhinus_leucas
   |    /-|
   |   |  |   /-Latimeria_menadoensis
    \-|   \-|
      |      \-Oncorhynchus_mykiss
      |
       \-Protopterus_dolloi
# Alignment:
>Homo_sapiens
```

```
GCT--------------AAACCTAGCCCCAAACCCACTCCACCTTA-----------CTACCAGACAACCTTA----------GCCAAACCA
↪  TTTAC----CCAAATAAAGTATAGGCGATAGAAATTG-AAACCTGGCGCAATAGA----TATAGTACCGCAAGGGAAAGA-TGAAAAAT
↪  TA-----------TAACCAAGCATAATATAGCAAGGACTAACCCCTATACCTTCTGCATAATGAATTAACTAGAAATAACTTTGCAAGG
↪  AG-AGCCAAAGCTAAGACCCCCGAAA-CCAGACGAGCTACCTAAGAACAGCTAA---AAGAGCACACCCGTCTATGTAGCAAAATAGTG
↪  GGAAGATTTATAGGTAGAGGCGACAAACCTACCGAGCCTGGTGATAGCTGGTTGTCCAAGATA-GAATCTTAGTTCAACTTTAAATTTG
↪  CCC---ACAGAACCCTCTA-AATCCCCT-----------TGTAAATTTAACTGTTAGTCCAAAGAGGAACAGCTCTTTGGACACTAGG
↪  AAAAAACCTTGTAGAG-AGAGTAAAAAATTTA--------------ACACCCATAGTAGGCCTAAAAGCAGCCACCAATTAAGAAA-G
↪  CGTTCAAGCTCAAC--ACCCACTACCTAAAAAATCCCAAACATA-TAACTGAACTCCTCAC-ACCCAATTGGACCAATCTATCACCCTA
↪  TAGAAGAACTAATGTTAGTATAAGT-AACATGAAAAC---------ATTCTCCTCCGCATAAGCCTGCGTCAGATTAAAACACTGAACT
↪  GACAATTAACAGCCCAATATCTAC-------------------AATCAACCAACAAGTCATTATTACC-CTCACTGTC-AACCCAACA
↪  CAGGCATGCTCATAAGGAAAGGTTAAAAAAAGTAAAAGGAACTCGGCAAA--TCTTACCCCGCCTGTTTACCAAAAACATCACCTCTAG
↪  CA----TCACCAGTATTAGAGGCACCGCCTGCCCAGTGACAC---ATGTTTAACGGCCGCGGTAC-CCTAACCGTGCAAAGGTAGCATA
↪  ATCACTTGTTCCTTAAATAGGGACCTGTATGAATGGCTCCACGAGGGTTCAGCTGTCTCTTACTTTTAACCAGTGAAATTGACCTGCCC
↪  GTGAAGAGGCGGGCATAACACAGCAAGACGAGAAGACCCTATGGAGCTTTAATTTATTAATGCAAACAG---------------TACCT
↪  AACAAACCCACAGGTCCTAAACTACCAAACCTGCATTAAAAATTTCGGTTGGGGCGACCTCGGAGCAGAACCCAACCTCCGA--GCAGT
↪  ACA----------------TG-CTAAGACTTCACCAGTCAAAGCGAACTACTATACT----CAATTGATCCAA-----TAACTTGACC
↪  AACGGAACAAGTTACCCTAGGGATAACAGCGCAATCCTATTCTAGAGTCCATATCAACAATAGGGTTTACGACCTCGATGTTGGATCAG
↪  GACATCCCGATGGTGCAGCCGCTATTAAAGGTTCGTTTGTTCAACGATTAA-AGTCCTACGTGATCTGAGTTCAGACCGGAGTAATCCA
↪  GGTCGGTTTCTATCTACNTTCAAA-TTCCTCCCTGTAC-GAAAGGACAAGAGAAATAAGGCCTACTTCACAAAGCGCCTTCC------C
↪  CCGTAAATGATATCATCTCAACTTAGTATTATACCCACACCCACC--------CAAGAACAGGGTTT---
>Ornithorhynchus_anatinus
GCC--------------AACTCTAGCCCAAATTAATCTTAATACACATATAAA-----------CTCTAATAA----------ACTAAAACA
↪  TTTTAACGCTAACCCCTAGTATAGGAGATAGAAAAGG--AAATTGGAGCCTTAGA----CATAGTACTGTGAAGGAAAAA-TGAAAGAT
↪  TAG----------CTTAAAGCACAAAAAAGCCAGGATTTAACCCTGTACCTTTTGCATAATGGTTTAGCTAGAAA-ACTTATATACAA
↪  AG-AATTTTAATATAAAATCCCGAAA-CTAGATGAGCTACTATCGAGCAATTTAT--TAGAATCAACCCGTCTATGTTGCAAAATAGTG
↪  GGATGACTTTATAGTAGAGGTGAAAAATCAACCGGACCTAGTGATAGCTGGTTAGCCAAAAAACGAATTTAAGTTCAACAATAAGTTTA
↪  TTT---CCTGATATACTAT-ATT-------------AACCATAAACTTATAAGTTATTCATAAGAGGGTCAGCCCTTATGA-ATAAGG
↪  AAACAACCTCTAATAG-AGGGAAATTTATTCT-------------TATTTACATAGTAGGCCTAAAAGCAGCCATCAATTAAAAAA-G
↪  CGTTAAAGCTTAAA--CTTCATTCTTTCTTAATCCCTTAACTT--CTTAACTACCCCTAAA-ATAATATTGGCTTAATCTATGTCCCCA
↪  TAGAAGAAATAATGCTAAAATAAGTAAACCAGAATT---------TATTCTCC-ATGCACTAGCTTAAATTAGAACGGAGC-ATCCACT
↪  AATAATTAACAGCTAAATAATCATAA-----------------ATATTAACTAG-AAACATTATTTTA-TAAACTGTT-AACCCGACA
↪  CAGGAGTGCATCTAAGGAAAGATTAAAAGGAGTAAAAGGAACTCGGCAAA-CTAGGATTTCGCCTGTTTACCAAAAACATCGCCTCTAG
↪  CA----TAACAAGTATTAGAGGTCCTGCCTGCCCAGTGATT----CTATTAAACGGCCGCGGTAT-CCTGACCGTGCAAAGGTAGCATA
↪  ATCATTTGTCTCCTAATTAGAGACTAGTATGAATGGCTAGACGAAAATCCAACTGTCTCTTACTCCCAATCAGTGAAATTGCCCTCCCC
↪  GTGCAGAGACGGGGATAACACCATAAGACGAGAAGACCCTGTGGAGCTTTAATCATAGATTTATTCTTT-------TTCATACAGACCC
↪  AAAAGCTCAACACCTTAAGAATT--------TTTTCAACTGATTTTGGTTGGGGTGACCTCGGAGTACAACAAAACCTCTGA--ATGCA
↪  TAG--------------------TAATAACTTACAAGTTTAAACGCAAAACTGCCAG----TAATAGACCCAA-----ATTATTGATC
↪  AAAGGACCAAGTTACCCCAGGGATAACAGCGCAATCCTATTCAAGAGTTCATATCGACAATAGGGTTTACGACCTCGATGTTGGATCAG
↪  GACATCCAAATGGTGCAGCCGCTATTAATGGTTCGTTTGTTCAACGATTAA-AGTCCTACGTGATCTGAGTTCAGACCGGAGTAATCCA
↪  GGTCGGTTTCTATCTATGGGTAA--TTTCTCCTAGTACGGAAAGGACCAGAGAAATCAGGCCAATCTTAAAAAGAAGCCTCC---AACT
↪  TAACAAATGATTTCATCTCAATTTGCCACGTTACCCTTACTAATC--------CTAGACCAGGAATTC-T
>Anolis_carolinensis
```

```
GCAAAACAAC----------AACCTAATTAAACAAACAAA----ATAGCACACACAACAA----------AACAAACCA
↪ TTTGA-----CAAAAAAAGTAGAGGCGATCGAACCTT-AACCGTAGAGTTTTATT----GATAGTACCGCAAGGGAAAAA-TGAAATAA
↪ TAGTGAAA------ACAAAAGCAAAACATAGCAAAGACTTCCCCTTGTACCTCTTGCATCACGATCTAGCAAGCATAAAACAAGCAAGA
↪ AGAAAACACAGCCTGTTTCCCCGAAC-TCAAGTGAGCTATTTTAAAGCAACTA----AAGAGTTAACCCGTCCCTGTAGCAAAAGGGTG
↪ GGAAGACTTTAAAATAGAGGTGAAAAGCCAACCGAACTTGCTGATAGCTGGTTACCAGATAGACGAATTTTAGTTCAACTTAAAACTTT
↪ ATTAACCCGCCCTTAGTAA-AA----------ACTTAGTTTTTAAGATACTCAATGAGGGGACAGCCTTATTGA-GCCAGA
↪ ATACAGCCTGAACTAG-AGAGAAACCTCAACA----------AAAACAACAGTAGGCCTTAAAACAGCCATCTAATATAATA-A
↪ CGTCGCAGTCTTCA----------TAATCAAAATACCAACCGCA-CTTTAAAACTCCTATA-CTACCACCTGGTAATTCAATAAATTTA
↪ TAGAAGATACTATGCTAGAACTAGT-AACAAGAAAT----------TTTCTCTCACGCACAACCATAAATCAGACATAGAAAAACTACT
↪ GAAAATTAACAAAACAATAAAAAC----------AACTTACATAG-------TATATTA-CTAGCTGTT-AACCCAACA
↪ CAGGTGTGCATAAA-AGAAAGATTAAAAGTTAAAAAAGGAACTCGGCAACAAACAAATTTCAACTGTTTACCAAAAACATAGCCTTTAG
↪ CA----AACCAAGTATTAAAGGTAACGCCTGCCCAGTGA------AACTTAAACGGCCGCGGTAT-TCTAACCGTGCAAAGGTAGCGTA
↪ ATCACTTGTCTCCTAAATAGGGACCTGTATGAACGGCTAAATGAATATTTAACTGTCTCTTTTAACTAATCAGTGAAACTGATCTTCCA
↪ GTACAAAAGCTGGGATAATAATATAAGACGAGAAGACCCTGTGGAGCTTTAAATTTC----------
↪ AACCAACATATAATTATCACAGTTATTT---ATGCTTAACATTTTAAGTTGGGGCGACTTCGGATAAAAACTAAACATCCGA--GCAAA
↪ TAA--------GGCACTGCCTA-AAAAAGGCCTACAAGCCAAAGCTA----------ATATTGACCCAG----TAATACTGATT
↪ AACGAACCAAGTTACCCCAGGGATAACAGCGCCATCTTCTTTAAGAGTTCATATCGACAAGAAGGTTTACGACCTCGATGTTGGATCAG
↪ GACACCCAAATGGTGCAGCCGCTATTAAAGGTTCGTTTGTTCAACGATTAACAGTCCTACGTGATCTGAGTTCAGACCGGAGCAATCCA
↪ GGTCGGTTTCTATCTATACAGTTG-CTTTCTTTAGTAC-GAAAGGACCAAGAAAGCAGGACCAATATAACCTACGTCCTTTA------C
↪ AATAGATTAAAAAAAACTAAAAT--TTAACAAACAACAAATTTTCCCC-----TAGACATAGGGGTTA--
>Gallus_gallus
CTTG----------CCCCCCCTCTAGCCCGACAAACTCGTACCCTTAACATAAAAAACTTACCTCCCCCTCTTA----------ACCAAAACA
↪ TTATA---AATTGTCCCAGTATAGGCGATAGAAAAGACTACCCCGGCGCAATAGAGGCTAACTGTACCGCAAGGGAAAGA-TGAAATAG
↪ CAATGAAA-----ACCATAAGCAAAAAACAGCAAAGACCAACCCTTGTACCTTTTGCATCATGATTTAGCAAGAAC-AACCAAGCAAAG
↪ TG-AGCTAAAGTTTGCCTTCCCGAAACCCAAGCGAGCTACTTGCGAGCAGCTAAAATTTGAGCGAACCCGTCTCTGTTGCAAAAGAGCG
↪ GGATGACTTGCCAGTAGAGGTGAAAAGCCTACCGAGCTGGGTGATAGCTGGTTACCTGTCAAACGAATCTAAGTTCCCCCTTAACCCAC
↪ CCC---CTAAAGACACCCACCTT-TGTCAACCTTGAGAACGTTGGGGTTAAGAGCAATTCGATGGGGGTACAGCTCCATCGA-AAAAGA
↪ ACACAACCTCCTCCAG-CGGATAATAATCACCCC----------TCCCCGCACTGTGGGCCTTCAAGCAGCCACCAACAAAAGAGTG
↪ CGTCAAAGCTCCCT--CATTAAAAAATCTAAAACC----------CTATTTGACTCCCTCA-ACCAAAGCAGGTTAACCTATG--ACAA
↪ TAGAAGAATCAATGCTAAAATGAGT-AATCTGGAACCTA-------TCCTCCTACGGCGTAAACTTACATTA----------ATACATT
↪ ATTAACAGAACTCAACTTATACCCC------C----------ACACTAACAAGCAATACGTATTCCT-CAATCTGTT-AAGCCAACC
↪ CAGGAGCGCCCAC--AGGATGATTAAAACCTACAGAAGGAACTCGGCAAA-CCAAAGACCCGACTGTTTCCCAAAAACATAGCCTTCAG
↪ CTA--ACAACAAGTATTGAAGGTGATGCCTGCCCAGTGACCCCCAAAGTTCAACGGCCGCGGTAT-CCTAACCGTGCGAAGGTAGCGCA
↪ ATCAATTGTCCCGTAAATTGAGACTTGTATGAATGGCTAAACGAGGTCTTAACTGTCTCCTGTAGGTAATCTATGAAATTAGTATTCCC
↪ GTGCAAAAACGAGAATGTGAACATAAGACGAGAAGACCCTGTGGAACTTTAAAATCACGACCACCTTACAA----CCTTACACAGCCCC
↪ ACTGGGTCCACCCACACATAAACCCC-----TGGTCGACATTTTTCGGTTGGGGCGACCTTGGAGAAAAAAAAATCCTCCAA--ACCCA
↪ CAG------ACCACAACTCTTC-ACTAAGACCAACTCCTCAAAGTACCAACAGT--------AACCAGACCCAA----TATAATTGAGC
↪ AATGGACCAAGCTACCCCAGGGATAACAGCGCAATCTCCTCCAAGAGCCCATATCGACAAGGAGGTTTACGACCTCGATGTTGGATCAG
↪ GACAACCTAATGGTGCAACCGCTATTAAGGGTTCGTTTGTTCAACGATTAACAGTCCTACGTGATCTGAGTTCAGACCGGAGCAATCCA
↪ GGTCGGTTTCTATCTATGGACA---CTCCTCCTAGTAC-GAAAGGACCGGAGAAGTGGGGTCAATACCACTGAGCACACCCCAACCTTC
↪ TAAGCAATGAATACAACTCAACTGCCAAGAACCCCTCCCCCACACCCGAACTCCTAGAAAAGGATCCA--
>Alligator_mississippiensis
```

```
AC--------------CCATATCTAGCCCTACCTCCTTTCAACATGC-----------TTAATACAAACCTCAA----------ACCAAAACA
↪  TTTATC--TAACACTTTAGTATTGGCGAAAGAAAATA-TAATCAGGCGCAACAGA----TAAAGTACCGCAAGGGAAAAAATGAAATAA
↪  AAATGAAA------ACATAAGTAAAACACAGCAAAGACCAACCCTTCTACCTTTCGCATCATGGTTTAGCAAACCA-CAACTGGCAAAA
↪  AG-AATTTAAGTCACACAACCCGAAA-CTAGGTGAGCTACTAACTAGCAGCTAAA-TTTGAGCTAACCCCCCTCTGTGGCAAAAGAGCG
↪  GGAAGACTACTTAGTAGAAGTGAAAAGCCTACCGAACCTAGTGATAGCTGGCTGCTTGGAAAATGAATCTTAGTTCCACTGTAA-ATTC
↪  TCCTAACTACCACAACGAAGGAT--------------AAGGAAGAATTTTCAAGCTATTTAATAGAGGTACAGCTCTATTAA-TGCAGG
↪  ACTCAACCTCCACTTA-AGGGTAATCCATCCCAT------------TCCTCGGACTGTGGGCCTTAAAGCAGCCACCA--CAAAAAA-G
↪  CGTCAAAGCTTAA-----------CTCCAAAAAATACCAACAA--CAAATCGAACCCTACA-TCATAACCAAGCCATCCTACAACCC--
↪  TAGAAAAGATTATGCTAAAATGAGT-AATAAGAAAACGAC------TTTCTCCTTTGCGTCAGCCTACATCCTACATGACA-CCCTATA
↪  GATTATTAACAGCCGCTTCCTACA--------CCACCCCACAA-AAAACAAAAAG-AAGACCT--------CACCTGTTGAACCCAACA
↪  CAGGAACGCAACA--GGAAAGGCTAAACCTTGCAAAAGGAACTCGGCAAA-CAAAGATTCCGACTGTTTACCAAAAACACAGCCCCTAG
↪  CC----CCCTCAGTATTAGGGGTGATGCCTGCCCAATGATTTC--CAATTGAATGGCCGCGGTATATACAACCGTGCGAAGGTAGCGTA
↪  ATCACTTGTTCTCCAAATAAGGACCAGTATGAACGGCTAAACGAGAATCTAACTGTCTCCTGCAAGCAACCAATGAAATTGATCTCCCT
↪  GTGCAAAAGCAGGAATGACCCCACCAGACGAGAAGACCCTGTGAAACTTTAACCGACTAAGTCACACAC------------TAGGAACA
↪  ACACAACCCACAACCACCTTAAACCCA----TGACTTAGCGCTTTTGGTTGGGGTGACCCTAAAACAAAAAAAAACTTTTAA--GACAA
↪  TCATAACAAAATTAGACTATTA-ACTAAGACCCACACCTCAAAGTACTTAACTGT-------AATTAGATCCGA----CAATGTCGATC
↪  CACGAACTAAGCTACTCCAGGGATAACAGCGCAATCCCCTTCAAGAGCCCCTATCGACAAGGGGGTTTACGACCTCGATGTTGGATCAG
↪  GACACCCCATTGGTGTAACCGCTATTAATGGTTCGTTTGTTCAACGATTAA-AGTCCTACGTGATCTGAGTTCAGACCGGAGTAATCCA
↪  GGTCGGTTTCTATCTATGACTATGTCTCTTCCTAGTAC-GAAAGGACCGGGGAAACAAGGCCCATGCCACCAAAGTACGCCTTACCTAT
↪  AATTTAATGCACGTAACTAAATTAATAATTAGGATAAACACTACTTCT-----CAAAACAAGAGAAC---
>Alytes_obstetricans
GCT-------------GTAAACCTAGCCC-------------TCAAAAAACAA-----TAACCCTCAAATCCCA----------ACTAAATCA
↪  TTTAC-----ACTCTTTAGTATAGGTGATAAAAAAAG--AAATAAGCGCAACAGA----CAGAGTACCGTAAGGGAAAGA-TGAAATAG
↪  CTATGAAATAAA-CACCAAAGCACTAAAAAGCAGAGATACATCCTCGTACCTTTTGCATCATGGTCTAGCAAGAAA-CCTTAAGCAAAA
↪  CG-CATTAAAGTTTAACCCCCCGAAA-CTAGACGAGCTACTCCGAGGCAGCTAT---AAGAGCAAACCCATCTCTGTGGCAAAAGAGTG
↪  GGAAGACCCCCGAGTAGAGGTGACAAGCCTACCGAGCCTAGTGATAGCTGGTTGCTTACGAAGAGAATATTAGTTCTGCCTCATAACTG
↪  TTTAAAGCACTATAAAGCAAAC----------------AACAA-ACTTGAGAGTTAGTCAAAAGGGGTACAGCCCATTTGA-CTCAGG
↪  AAACAACCTAACACCC-GGGGTAAAGATTATAATAAACAAGG---ATTTTGACCCAGTGGGCCTAAAAGCAGCCACCTGTTAAGAAA-G
↪  CGTCATAGCTCTAT--CAGCACCCGACCCAACAATCCCAATAAC-CACTCTGATCCCTCCT-A--GTAGTAAGCCCTCCTATTCACATA
↪  TAGAATTGTTTATGCTAGAACTAGT-AACTAGAAT----------AATTCTCCTAATGCAAGTGCACATCAGCCCGAAAC-ACTCACT
↪  GATAACTAACGACTACTGAACCCA----------CTGTAGTA-ACACCCCCCAGAAAAACCTACACTA-ACCATCGTT-AACCTAACA
↪  CAAGAGCATCCCA--GAAAAGATTTAAAGACGTGGAAGGAACTCGGCAAA-CACGAACTTCGCCTGTTTACCAAAAACATCGCCTCTTG
↪  TC--------AACTATAAGAGGTCCCGCCTGCCCAGTGACTAC--ATGTTCAACGGCCGCGGTAT-TATGACCGTGCAAAGGTAGCGTA
↪  ATCACTTGTCTTTTAAATAAAGTCTAGTATGAAAGGCACCACGAAAGTTCAACTGTCTCCCACATCTAATCAGTGAAACTGATCCCCCC
↪  GTGCAGAAACGGGGATACTCTCATAAGACGAGAAGACCCCATGGAGCTTCAAACACGCAAGCAGCCGCTTTAAACCTCATCTAACCACA
↪  AGCAAGCGGACA------------------TGACCAGCTGTTTTCGGTTGGGGCGACCACGGAGAACAAAAAATCCTCCGA--GACGA
↪  AAG----GGCCAACCCCCCTTA-CCCAAGAATCACCACTCTAAGGAATTGAATATCTAAC--ATCATGATCCAA------TATTTGATC
↪  AACGAACCAAGTTACCCTGGGGATAACAGCGCAATCCATTTTTAGAGCCCCTATCGACAAATGGGTTTACGACCTCGATGTTGGATCAG
↪  GGTACCCCAGTGGTGCAGCCGCTACTAAAGGTTCGTTTGTTCAACGATTAA-AACCCTACGTGATCTGAGTTCAGACCGGAGCAATCCA
↪  GGTCAGTTTCTATCTATGCAATG--CTCTCTCTAGTAC-GAAAGGACCGAAAGAACCTGGCCAATGCTAATGCAAGCCCTAA---CTTA
↪  GCCTTAGTGAAATCAGCTCAACTAAGACAAAAATTGCAACTAAGC--------CCGAGATAAGGGCTT--
>Carcharhinus_leucas
```

```
ACCTTAAAGCTAGCCTATATAT-TAATTCAACTAGACCTTATAAATCCAATC------TATATTACAGTTTTTA----------ATCAAAACA
↪  TTCTC----AACCTTTTAGTATGGGTGACAGAACAAT-AACCCCAGCGCGATAGC----TTATGTACCGCAAGGGAAAGC-TGAAAAAG
↪  AAATGAAATAAATCATTAAAGTACTAAAAAGCAGAGATGATACCTCGTACCTTTTGCATCATGATTTAGCTAGAAA-AACTAGGCAAAA
↪  AG-ACCTTAAGTCTATCTCCCCGAAA-CTAAACGAGCTACTCCGAAGCAGCATTA--TAGGGCTAACCCGTCTCTGTGGCAAAAGAGTG
↪  GGAAGACTTCCGAGTAGCGGTGAAAAGCCTACCGAGTTTAGTGATAGCTGGTTACCCAAGAAAAGAACTTTAGTTCTGCATTA-ATTTT
↪  TCACTATCTAGACAAGACTTACTCGTCAAA---GAATTCCATAAGAATTAATAGTTATTTAGAAGAGGTACAACCCTTCTAA-ATCAAG
↪  ACACAACTTTTTAAGG-TGGGAAATGATCATAATTATTAAGG---TTACCATCCTAGTGGGCCTAAAAGCAGCCACCTGTTAAGTAA-G
↪  CGTCGCAGCTCTAATCTAATAATTAAACCTTTAATTCAGATATT-CATTCACAACCCCCTTCATTCTATTGGGTTATTTTATATTTATA
↪  TAAAAGAACTTATGCTAAAATGAGT-AATAAAGAGAATAA------ATCTCTCCCGACACAAGTGTATGTCAGAAAGAATTAAATCACT
↪  GATAATTAAACGACCCCAAACTGAG---GTCATTATATCACAATCAATCAACTAGAAAACCCTATTTTC-CTACTCGTT-ACCCCTACA
↪  CAGGAGTGTCATCA-GGAAAGATTAAAAGAAAATAAAGGAACTCGGCAAA-CATAAACTCCGCCTGTTTACCAAAAACATCGCCTCTTG
↪  AT-------AAACTATAAGAGGTCCCGCCTGCCCTGTGATA----ATGTTCAACGGCCGCGGTAT-TTTGACCGTGCAAAGGTAGCGTA
↪  ATCACTTGTCTTTTAAATGAAGACCTGTATGAAAGGCATCACGAGAGTTTAACTGTCTCTATTTTCTAATCAATGAAATTGATCTATTC
↪  GTGCAGAAGCGAATATAATAACATTAGACGAGAAGACCCTATGGAGCTTTAAACACTTAAATTAATTATGTAACCATCCACTTCTCAGG
↪  ATATAAACAAAATATACAATACTTCT-----AATTTAACTGTTTTTGGTTGGGGTGACCAAGGGGAAAAACAAATCCCCCTT--ATCGA
↪  TTG--AGTACTAAGTACTTAAA-AATTAGAATGACAATTCTAATTAATAAAATATTTATCGAAAAATGACCCAGG---ATTTCCTGATC
↪  AATGAACCAAGTTACCCTAGGGATAACAGCGCAATCCTTTCTCAGAGTCCCTATCGAAGAAAGGGTTTACGACCTCGATGTTGGATCAG
↪  GACATCCTAATGGTGCAACCGCTATTAAGGGTTCGTTTGTTCAACGATTAATAGTCCTACGTGATCTGAGTTCAGACCGGAGAAATCCA
↪  GGTCAGTTTCTATCTATGAATACA-CTTTTCCTAGTAC-GAAAGGACCGGAAAAGTAGGGCCAATGCCACTAGTACGCCCTA---TTTT
↪  CATCTATTGAATAAAACTAAAATAGATAAGAAAAGATCACCTAGTGCC-----CAAAAAAAGGGTT----
>Latimeria_menadoensis
GCT------------ATATAGCTAGGCCAACAAAACCACAACCACTATACCA-----TCAACCAAAAACCCAAACAAGCAAAACCTAAACCA
↪  TTTA------CCATCCAAGTATAGGCGATAGAAAAGA--CACCAGGCACAATAAT----GAAAGTACCGCAAGGGAACGC-TGAAAAAG
↪  AAATGAAACAACTCGTAAAAGCAACAAAAACCAAAGACTAACCCTTGTACCTTTTGCATTATGATCTAGTTAGACCCCACCGGGCAAAA
↪  TG-AATTTAAGTCCAACCCCCCGAAA-CTAAGTGAGCTACTTCGAAACAGCCTAT--AAGGGCAAACCCTTCTCTGTAGCAAAAGAGTG
↪  GGAAGATTTCCAAGTAGAGGCGATAAACCTAACGAACTCAGTGATAGCTGGTTATTCGAGAAATGATTTTTAGTTCAGCCCCTCAGCCT
↪  CCA-AACCCTAACAACACAAAACCAATG------------TGAACCAGAGAGAGTTATTCAAAGGGGGTACAGCCCCTTTGA-AAAAGG
↪  ATACAACCTTCTAAGACCGGATAAAGACCATATTAACTCAAGGATCAGCTGCCTAAGTGGGCCTAAAAGCAGCCACC---TAAAAAA-G
↪  CGTTAAAGCTTAAG--CAGCACTCAACCAACCTATAATGATAAATTAACCTCATTCCACCA-CCACTATCGAATTATTCTATA--TATA
↪  TAGAAGAATAAATGCTAGAATTAGT-AACAAGAAGGCCAATAAACCTTCTCTAACTGCATAAGTGTACATCAGATTAGATT-AACCACT
↪  GATAATTAACGACTTCAAAGAGAA-----------TACTATGA-CATAAAACAAGAAAAACACACACACCCACATCGTT-AATCCAACA
↪  CAGGAATGCAACCA-CGAAAGATTAAAAGAAAGAAAAGGAACTCGGCAAACCATAAGCCCCGCCTGTTTACCAAAAACATCGCCTCCTG
↪  CCAACAACAGAAGTATTGGAGGTCCCGCCTGCCCAGTGACA----AGATTTAACGGCCGCGGTAT-CCTGACCGTGCAAAGGTAGCGAA
↪  ATCACTTGTCTTTTAAATGAAGACCTGTATGAATGGCACCACGAGGGCTTAACTGTCTCCTCTTTCCAATCAGTAAAATTGATCTGTCC
↪  GTGCAGAAGCGGACTTAACCACATTAGACGAGAAGACCCTGTGGAGCTTCAGACACAAAGCCAACTACAAATAA-ACACTTAGATAACA
↪  AGTTGATAAACCAGTAGCCTAACACT-----GGCCCTATTGTCTTTGGTTGGGGCGACCACGGAGAAAAAGTAATCCTCCAA--GCCGA
↪  TTG--GTACCACCTGTACTAAA-ACAAAGGGTGACACCCCAAAGTAATAAAAATTTTATC-GGACATGACCCAGGAACTGAACCTGATC
↪  AACGAACCAAGTTACCCCAGGGATAACAGCGCAATCCTTTCCAAGAGTCCGAATCGACGAAAGGGTTTACGACCTCGATGTTGGATCAG
↪  GACACCCCAATGGTGAAGCCGCTATTAAAGGTTTGTTTGTTCAACAATTAA-AGTCCTACGTGATCTGAGTTCAGACCGGAGAGATCCA
↪  GGTCAGTTTCTATCTATGACGTTA-ATTCTCCCAGTAC-GAAAGGACCGGAGAACTTGAGCCAATACCACAAGCACGCTCAT---CTTC
↪  AACCTGATGAAAACAACTAAAACAGATAAAGAAGAACGTAACCACCCC-----CGTAAACAACGGGTAAT
>Oncorhynchus_mykiss
```

```
GCTGACTAGCTAGCCAACATATTTGGTCCAACACCACAACATACATACCCCAA-----TAAAACTTAGAATTAAGTC-------AACAAACCA
↪  TTTTT-----CCACCTTAGTAGGGGCGACCGAAAAGGAGATAATTGAGCAACAGA----AAAAGTACCGCAAGGGGAAGC-TGAAAGAG
↪  AATTGAAATAACCCATTTAAGCCTAGAGAAGCAGAGATTAAATCTCGTACCTTTTGCATCATGATTTAGCCAGCAC-ACCTGAGCAAAG
↪  AG-AACTTTAGTTTAGGCCCCCGAAA-CTAGACGAGCTACTCCGGGACAGCCTATTGTAGGGCCAACCCGTCTCTGTGGCAAAAGAGTG
↪  GGACGAGCCCCGAGTAGAGGTGATAAACCTATCGAGCCTAGTTATAGCTGGTTGCTTAGGAAATGAATAGAAGTTCAGCCCCCCGGCTT
↪  TCT-TAGGACCTTAAGGTAAAACTAATATT---GTCCCAAAGAAACCAGGAGAGTTAGTCAAAGGAGGTACAGCTCCTTTGA-ACAAGG
↪  ACACAACCTTAACAGG-CGGCTAAGGATCATAGTTCCAAGGT---AACCTGTTACAGTGGGCCTAAGAGCAGCCACCTGCACAGAAA-G
↪  CGTTAAAGCTCAGA--CAG-ATACAAACCTCTTATCCTGATAAG-AAATCCCACCCCCCTA-ACCGTACTAAGCCGTTCCATGCCCCCA
↪  TGGAAGAGATTATGCTAGAATGAGT-AATAAGAGAGT-----ACAACTCTCTCCCAGCACATGTGTAAGTCGGACCGGACC-CCCCACC
↪  GACAAATAACGAACCCAAACCAAGAGGGAACTGTAGGCCAGAA-CAAACACCAAGAAAAACCTACACCAACAAATCGTT-ACCCCCACA
↪  CAGGAGTGCCCCAAGGGAAAGACCCAAAGGAAGAGAAGGAACTCGGCAAA-CACAAGCCTCGCCTGTTTACCAAAAACATCGCCTCTTG
↪  CA----AATCAAAACATAGAGGT-CCGCCTGCCCTGTGACTAT--GGGTTTAACGGCCGCGGTAT-TTTGACCGTGCGAAGGTAGCGCA
↪  ATCACTTGTCTTTTAAATGAAGACCTGTATGAATGGCATCACGAGGGCTTAGCTGTCTCCTCTTCCAAGTCAATGAAATTGATCTGCCC
↪  GTGCAGAAGCGGACATAAGCACATAAGACGAGAAGACCCTATGGAGCTTTAGACACCAGGCAGATCACGTCAAGCAACCTT---GAATT
↪  AACAAGTAAAAACGCAGTAGACCCCT-----AGCCCATATGTCTTTGGTTGGGGCGACCGCGGGGGGAAAATTAAGCCCCCACTGTGGA
↪  CTG-GGGGGCACTGCCCCACCA-GCCGAGAGCTACAGCTCTAAGCACCAGAATATCTGACCAAATATGATCCGG----CGAACGCATTC
↪  AACGGACCGAGTTACCCTAGGGATAACAGCGCAATCCTCTCCCAGAGTCCCTATCGACGAGGGGGTTTACGACCTCGATGTTGGATCAG
↪  GACATCCTAATGGTGCAGCCGCTATTAAGGGTTCGTTTGTTCAACGATTAA-AGTCCTACGTGATCTGAGTTCAGACCGGAGTAATCCA
↪  GGTCAGTTTCTATCTATGAAGTGA-TGTTTCCTAGTAC-GAAAGGACCGGAAAGAAGGGGCCCATGCTTGAGGCACGCCCCA---CCCC
↪  CACCTGATGAAGGCAACTAAAACAGACAAGGGGGCACACCAAGATTGC-----CTAAAAGAACGGCGC--
>Protopterus_dolloi
ACTAAATGG--------------TAAGCCTGTCATTTTTTAACATGTAAATTTT-ATATAACACACACTCTGTA----------AGTAAACCA
↪  TTTA------TACTTCTAGTATTGGAGAAAGAAAGAA--ATTCAAGCGCAT--------AAAAGTACCGCAAGGGAAAAC-TGAAAAAC
↪  TAGTGAAA------AATTAAGTTTTAAAAAGCAAAGACTAACACTTGTACCTTTTGCATCATGGTCTAGCTAGTCA-TGAAGGGCAGAA
↪  AG-AATTTTAGTCC-CACCCCCGAAA-CTAGGCGAGCTACTCCGAGACAGCCAA---ACGGGCCAACTCGTCCATGTGGCAAAATGGTG
↪  AGAAGAGCTCCGAGTAGCGGTGAAAAGCCAAACGAGCCTAGAGATAGCTGGTTGCGCGAGAAACGAATCTTAGTTCTACCCTAAATTTT
↪  TCACGGGCTACAGCCTTAAACCCCGTA------------ATTAAATTTAGAG-GCTACTCAAAGGGGGGACAGCCCCTTTGT-GAAAGG
↪  ACACAACCTCAACAAA-CGGATAATGATTAATCTACA---AG---GTAAAATTTAAGTGGGCCTAAAAGCAGCCACCAACAAAGAAA-G
↪  CGT-ATAGCTCCCC--CATAACTTTATCCATCAATTCAATTAAA-CAATCATAATCCATCTTAC-GTATCGAGCTGACTTATATCATTA
↪  TAAGTGCAAAAATGCTAGAATGAGT-AACAAGAAAAT-----ACAATTTTCTCCTCACATAGGTGTTAGTCAGAACAGATT-ACCTACT
↪  GACAATTACTGATAATGAACACAA-------TGTATTAACCTG-TCATAAGCAAGAAAACCCTACACTGATATATCATA-AATTCTACA
↪  CTGAAGTGTGCTT--GGAAAAATTAAGGGGGGGAGAAGGAATTCGGCAA--CTATGGCCTCGCCTGTTTACCAAAAACATCGCCTCCTG
↪  CC--------AAATATAGGAGGTACTGCCTGCCCTGTGACTC---TGTTTAACTGGCCGCGGTAT-TTTGACCGTGCGAAGGTAGCGTA
↪  ATCACTTGTCTCTTAATTAGGGACCTGTATGAATGGCAACACGAGGGTCCAACTGTCTCCTCCCCCAGATTAGTGAAATTGATCTATCC
↪  GTTCAAAAGCGGATATTTTTTCATAAGACGAGAAGACCCTGTGGAGCTTAAAGTTCTAATATTAAACAT--------------------
↪  -GAGCGTAAATATATACTTTAAGTTTTG-TAATATTAAATACTTTCGGTTGGGGCGACCACGGAGTATAAAAAACCCTCCGC-------
↪  ----------AATACAATTCTATTTTAAGAAGGACACTTCTCA-AACTAGAATATCTAGC-ACAATTGACCCAG----TCTAACTGAGC
↪  AATGAACCAAGTTACCCCAGGGATAACAGCGCAATCCCCTTTAAGAGTCCCCATCGACGAGGGGGTTTACGACCTCGATGTTGGATCAG
↪  GGTATCCTGGTGGTGCAGCCGCTACCAAGGGTTTGTTTGTTCAACAATTAA-TATCCTACGTGATCTGAGTTCAGACCGGAGCAATCCA
↪  GGTCAGTTTCTATCTATGACTTC--TTTTTTCTAGTAC-GAAAGGACTGAAAAAGGGGGGCCTATAT-AAAAATATGCCCCA---CCCA
↪  CTACTACTGAATTTATATAAAGTAGCCAAGTGGGAAACCCCCCACACG-----GGAGAAAACCACACT--
```