n01_alignment

May 7, 2025

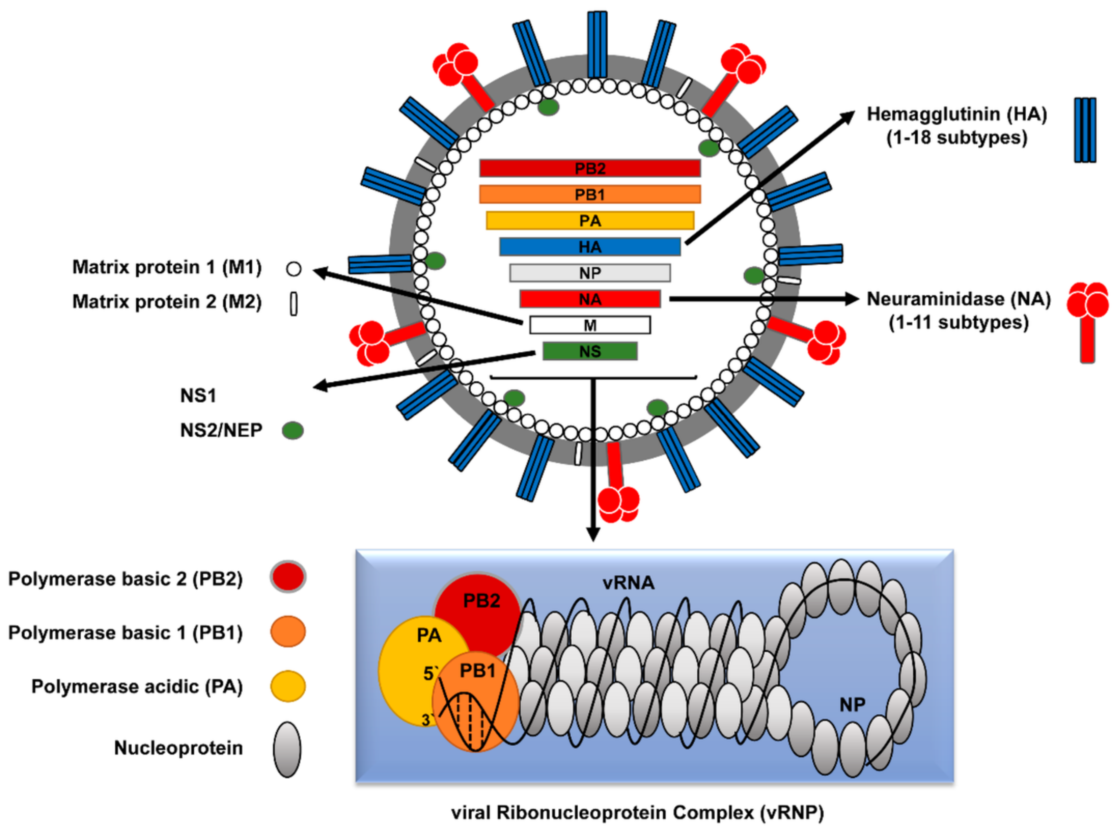
1 A look at influenza

1.1 the plan

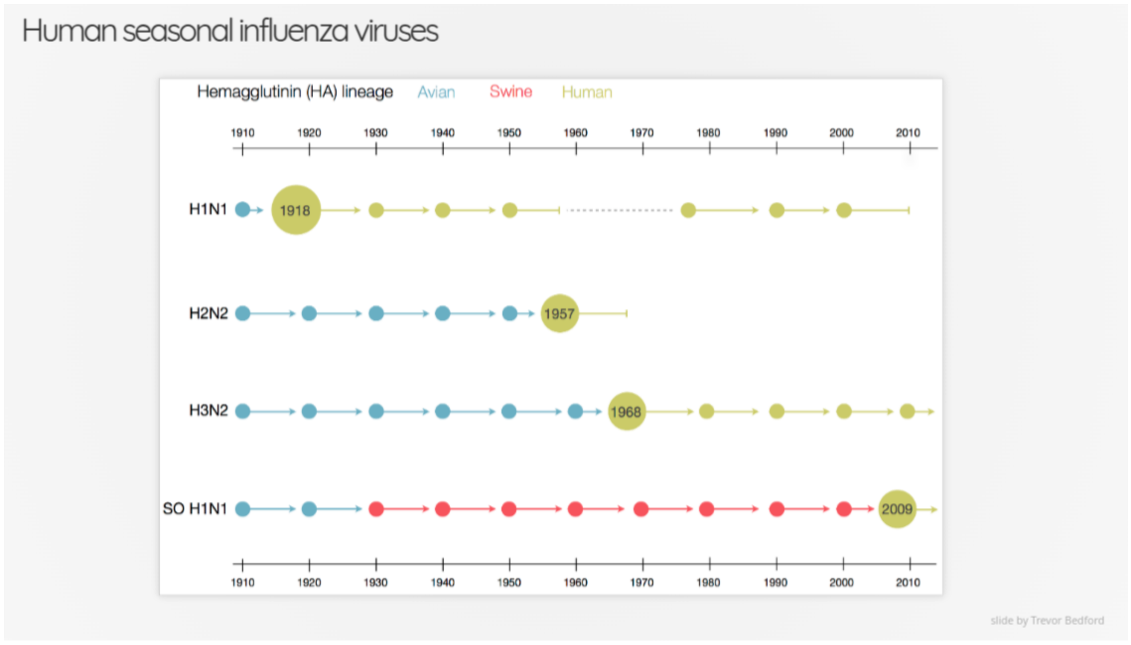
- A look at the pathogen: influenza A
  - a pathogen evolving rapidly and under strong selection pressure
- Download data and metadata
  - quick inspection and quality control
- Building a multiple sequence alignment
  - inspecting and trimming the alignment.
  - looking for signatures of selection
- Building a phylogenetic tree
  - building a tree from the alignment
  - manipulating trees with Biopython
  - augmenting the tree with metadata
  - simple parsimonious ancestral state reconstruction: Fitch's algorithm
  - rooted vs unrooted trees: the problem of picking a root

2 Influenza A virus

- A segmented, negative-sense RNA virus
- 8 segments of RNA, ~13.5 kb in total



- classified into subtypes based on two surface proteins:
  - hemagglutinin (HA), involved in binding to host cells
  - neuraminidase (NA), involved in the release of new virions
  - both are targets for the immune system
- Rapidly evolving virus (~4×10^{-3} substitutions per site per year on fast-evolving genes)
  - high mutation rate
  - frequent reassortment
- Infects a wide range of hosts
  - humans, birds, pigs...
- Both seasonal circulation and pandemic outbreaks



(from this presentation).

We'll focus on the H3N2 subtype: - first emerging in 1968 (Hong Kong flu) via reassortment with avian strain.

To get a sense of the data, here is a phylogenetic tree for the HA gene of H3N2 influenza A viruses for the last 12 years of evolution: nextstrain H3N2 tree - ladder-like structure - fast substitution rate - let's look at the annotations on the segment: HA1 and HA2 encode the receptor binding domain and the fusion peptide, respectively. Cleaved post-translationally by host proteases.

3 Downloading the dataset

Head to the NCBI virus database and search for "influenza A" virus

- select *segment 4*: hemagglutinin (HA)
- under "genotype" select subtype : H3N2
- should be ~1700 bp long: you can impose a minimum length filter of 1700 bp
- you can also filter by host and restrict to human samples
- optionally, you can also require that the sequence does not contain any ambiguous bases.

Here is a link to the query.

This will still give you too many sequences for a quick analysis (~50-60k) so we will need to sub-sample the dataset

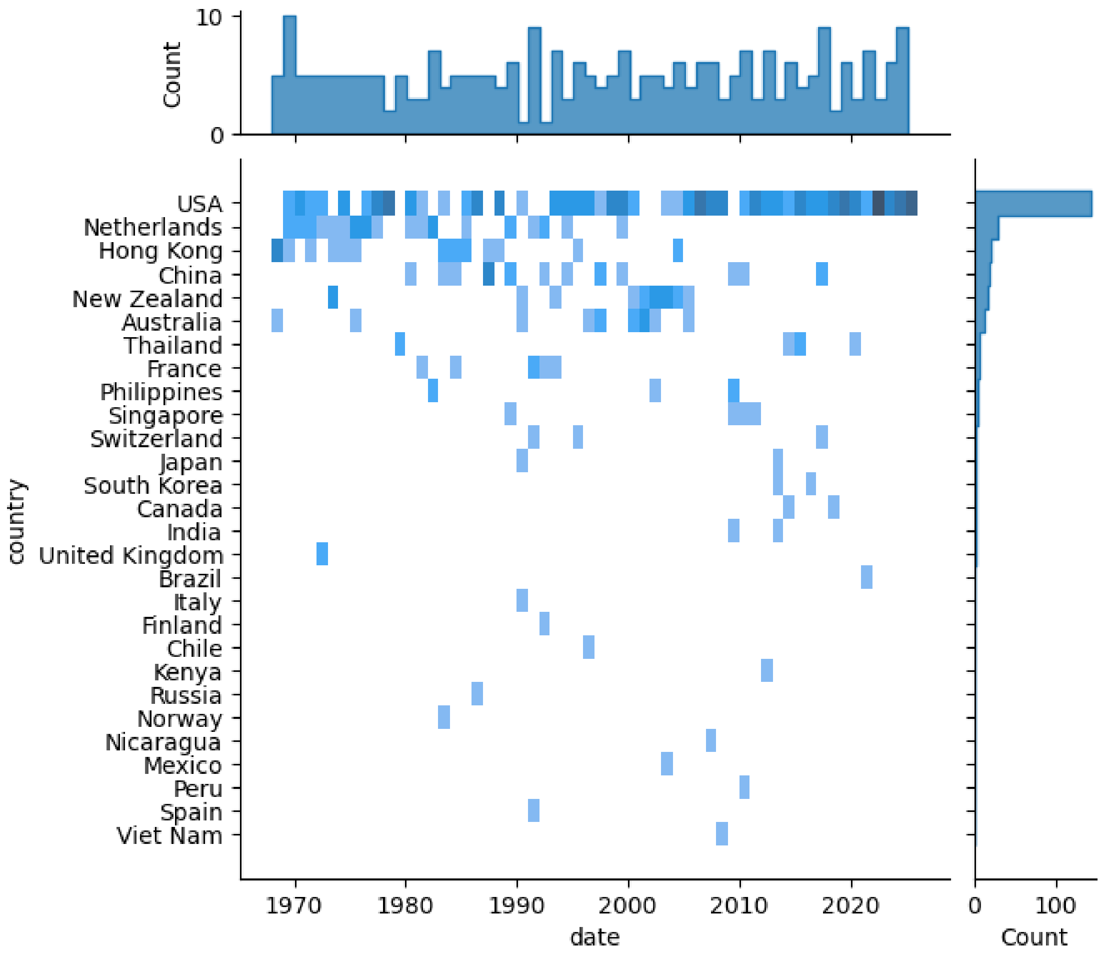
- select "download" and choose "nucleotide" (fasta) format
- select "download a randomized subset"
  - stratify by "collection year", and select 5 samples per category
  - other than `Accession`, `GenBank title` and `Collection Date`, also add `Country` to the fasta definition line. This will add metadata that we will analyze later.

This should give you a file with ~300 sequences.

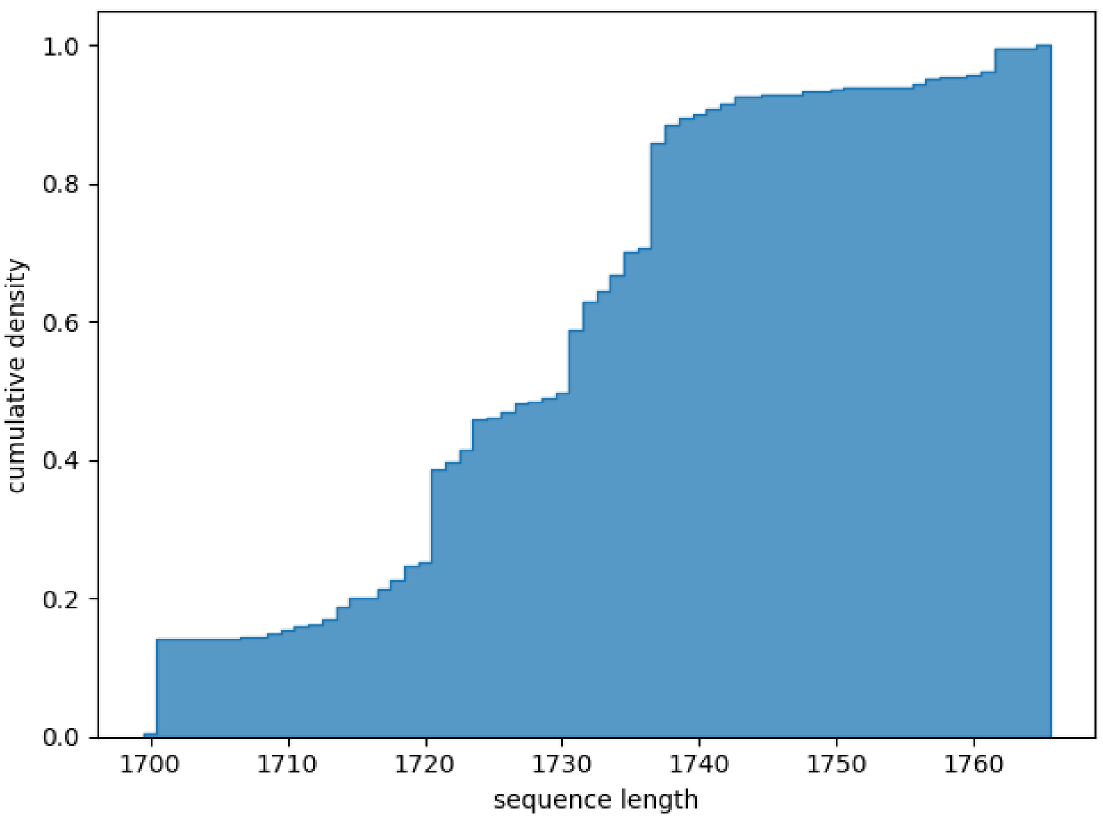
4 Preliminary quality control: a look at the fasta file

- how many sequences do you have?
- how long are they on average?
- how does the description line look like?
- can you parse the metadata from the description line? What is the distribution of sampling dates? And of countries?
- Are there sequences with many ambiguous bases `N`?

```
                                                   description       date  \
accession
CY009356.1  Influenza A virus (A/England/72(H3N2)) segment… 1972-01-01
OP844051.1  Influenza A virus (A/Oregon/01/2022(H3N2)) seg… 2022-01-03
CY012728.1  Influenza A virus (A/New York/767/1993(H3N2)) … 1993-04-05
CY033553.1  Influenza A virus (A/Hong Kong/1-11-MA21-3/196… 1968-01-01
MK744280.1  Influenza A virus (A/Hawaii/03/2019(H3N2)) seg… 2019-01-12

                   country   len  ambiguous  collection_year
accession
CY009356.1  United Kingdom  1723          0             1972
OP844051.1             USA  1737          0             2022
CY012728.1             USA  1724          0             1993
CY033553.1       Hong Kong  1719          0             1968
MK744280.1             USA  1737          0             2019
```



10
0
Count
USA
Netherlands
Hong Kong
China
New Zealand
Australia
Thailand
France
Philippines
Singapore
Switzerland
Japan
South Korea
Canada
India
United Kingdom
Brazil
Italy
Finland
Chile
Kenya
Russia
Norway
Nicaragua
Mexico
Peru
Spain
Viet Nam
country
1970
1980
1990
2000
2010
2020
date
0
100
Count



```
Number of sequences with ambiguous bases:

 ambiguous
 0    266
 1     15
 2      1
 3      2
 Name: count, dtype: int64
```

5 Building a multiple sequence alignment

After preliminary quality control, we can align all sequences. We will not align all sequences to a reference, but rather align all sequences to each other in a multiple sequence alignment.

We will use MAFFT for this. This is a fast and accurate multiple sequence alignment program. It scales well to relatively large datasets, and uses different algorithms, including Fast Fourier Transform (FFT) to speed up alignment.

It can be installed via conda:

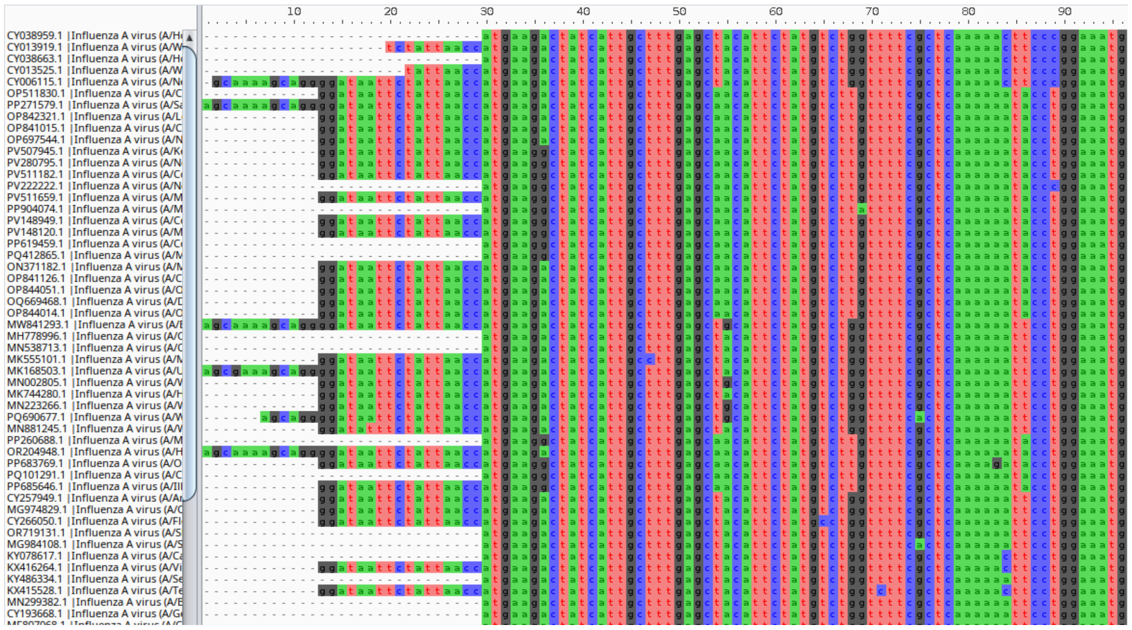
```
conda install bioconda::mafft
```

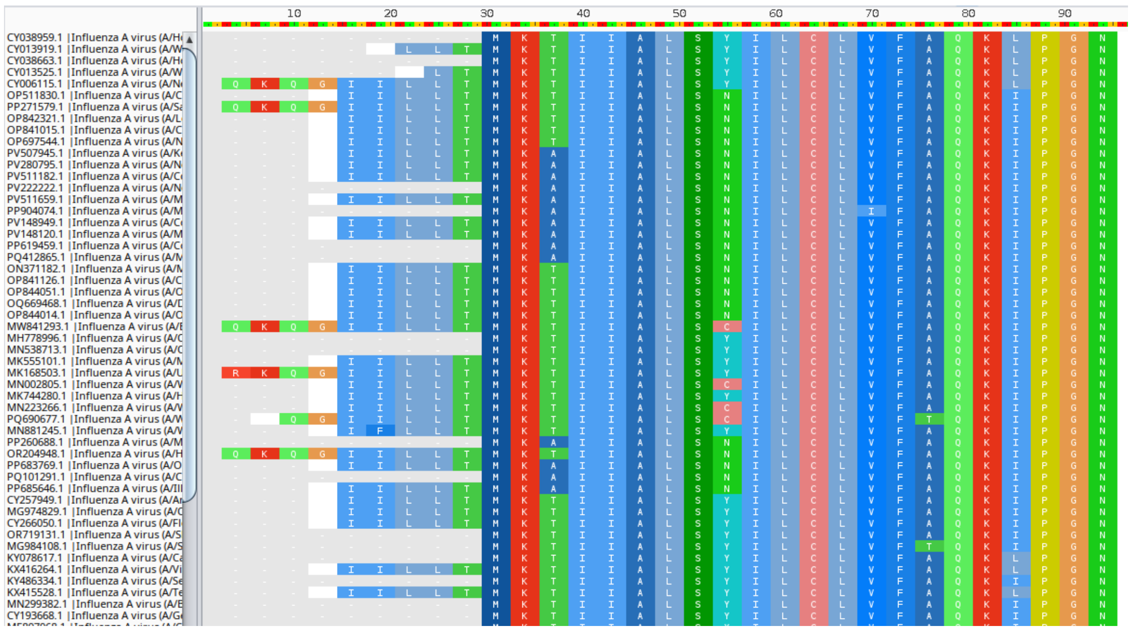
And can be run from the command line:

```
mafft --auto h3n2.fa > h3n2.aln.fa
```

5.1 Inspecting the alignment

- you can visualize the alignment with your tool of choice. If you don't have one, you can use AliView for example.
- can you find the start-end of the coding region?
- are there any gaps in the alignment?
- are there sequences that do not align well and should be removed?
- polish the alignment if necessary (e.g. remove low-quality sequences) and trim the alignment to the coding region.
- is the diversity different in different parts of the alignment? What about amino-acid diversity?


10
20
30
40
50
60
70
80
90
CY038959.1 |Influenza A virus (A/H
CY013919.1 |Influenza A virus (A/W
CY038663.1 |Influenza A virus (A/H
CY013525.1 |Influenza A virus (A/W
CY006115.1 |Influenza A virus (A/N
OP511830.1 |Influenza A virus (A/C
PP271579.1 |Influenza A virus (A/Sa
OP842321.1 |Influenza A virus (A/L
OP841015.1 |Influenza A virus (A/C
OP697544.1 |Influenza A virus (A/N
PV507945.1 |Influenza A virus (A/K
PV280795.1 |Influenza A virus (A/N
PV511182.1 |Influenza A virus (A/C
PV222222.1 |Influenza A virus (A/N
PV511659.1 |Influenza A virus (A/M
PP904074.1 |Influenza A virus (A/M
PV148949.1 |Influenza A virus (A/C
PV148120.1 |Influenza A virus (A/M
PP619459.1 |Influenza A virus (A/C
PQ412865.1 |Influenza A virus (A/M
ON371182.1 |Influenza A virus (A/M
OP841126.1 |Influenza A virus (A/C
OP844051.1 |Influenza A virus (A/O
OQ669468.1 |Influenza A virus (A/D
OP844014.1 |Influenza A virus (A/O
MW841293.1 |Influenza A virus (A/E
MH778996.1 |Influenza A virus (A/C
MN538713.1 |Influenza A virus (A/C
MK555101.1 |Influenza A virus (A/M
MK168503.1 |Influenza A virus (A/U
MN002805.1 |Influenza A virus (A/W
MK744280.1 |Influenza A virus (A/H
MN223266.1 |Influenza A virus (A/W
PQ690677.1 |Influenza A virus (A/W
MN881245.1 |Influenza A virus (A/W
PP260688.1 |Influenza A virus (A/M
OR204948.1 |Influenza A virus (A/H
PP683769.1 |Influenza A virus (A/O
PQ101291.1 |Influenza A virus (A/C
PP685646.1 |Influenza A virus (A/Il
CY257949.1 |Influenza A virus (A/Ar
MG974829.1 |Influenza A virus (A/C
CY266050.1 |Influenza A virus (A/Fl
OR719131.1 |Influenza A virus (A/S
MG984108.1 |Influenza A virus (A/S
KY078617.1 |Influenza A virus (A/Ca
KX416264.1 |Influenza A virus (A/Vi
KY486334.1 |Influenza A virus (A/Se
KX415528.1 |Influenza A virus (A/Te
MN299382.1 |Influenza A virus (A/E
CY193668.1 |Influenza A virus (A/G



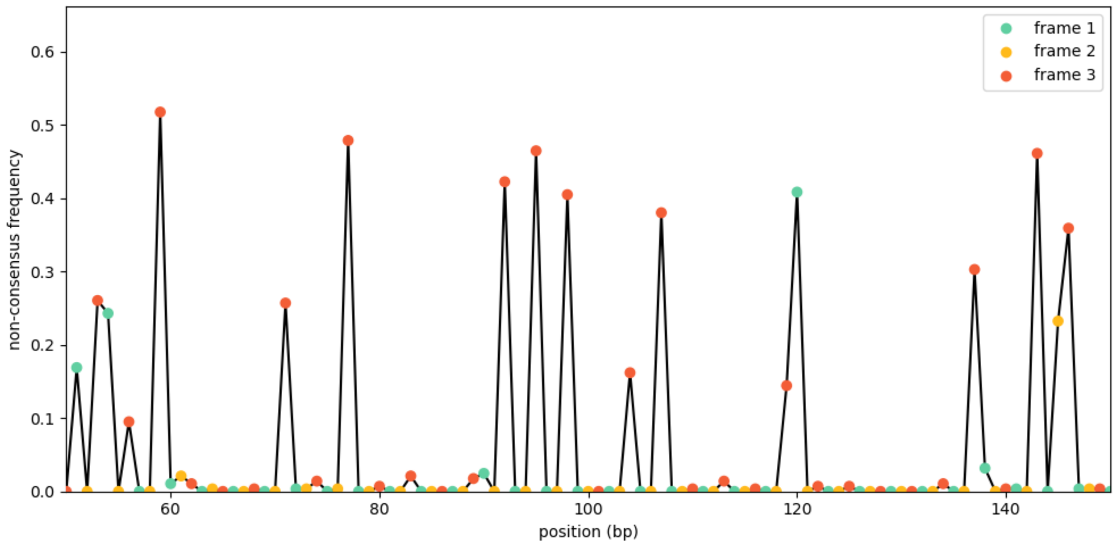
```
Alignment with 284 rows and 1768 columns
------------------------------atgaagactatcatt…--- CY038959.1
--------------------tctattaaccatgaagactatcatt…--- CY013919.1
------------------------------atgaagactatcatt…--- CY038663.1
----------------------tattaaccatgaagactatcatt…--- CY013525.1
-gcaaaagcaggggataattctattaaccatgaagactatcatt…--- CY006115.1
------------ggataattctattaaccatgaagactatcatt…--- OP511830.1
agcaaaagcaggggataattctattaaccatgaagactatcatt…--- PP271579.1
------------ggataattctattaaccatgaagactatcatt…--- OP842321.1
------------ggataattctattaaccatgaagactatcatt…--- OP841015.1
------------ggataattctattaaccatgaagactatcatt…--- OP697544.1
------------ggataattctattaaccatgaaggctatcatt…--- PV507945.1
------------ggataattctattaaccatgaaggctatcatt…--- PV280795.1
------------ggataattctattaaccatgaaggctatcatt…--- PV511182.1
------------------------------atgaaggctatcatt…--- PV222222.1
------------ggataattctattaaccatgaaggctatcatt…--- PV511659.1
------------------------------atgaaggctatcatt…--- PP904074.1
------------ggataattctattaaccatgaaggctatcatt…--- PV148949.1
------------ggataattctattaaccatgaaggctatcatt…--- PV148120.1
…
----------------------tattaaccatgaagactatcatt…--- CY006707.1

Alignment with 284 rows and 1698 columns
atgaagactatcattgctttgagctacattctatgtctggtttt…att CY038959.1
atgaagactatcattgctttgagctacattctatgtctggtttt…att CY013919.1
atgaagactatcattgctttgagctacattctatgtctggtttt…att CY038663.1
atgaagactatcattgctttgagctacattctatgtctggtttt…att CY013525.1
atgaagactatcattgctttgagctacattctatgtctggtttt…att CY006115.1
atgaagactatcattgctttgagcaacattctatgtcttgtttt…att OP511830.1
atgaagactatcattgctttgagcaacattctatgtcttgtttt…att PP271579.1
```

```
atgaagactatcattgctttgagcaacattctatgtcttgtttt…att OP842321.1
atgaagactatcattgctttgagcaacattctatgtcttgtttt…att OP841015.1
atgaagactatcattgctttgagcaacattctatgtcttgtttt…att OP697544.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV507945.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV280795.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV511182.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV222222.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV511659.1
atgaaggctatcattgctttgagcaacattctatgtcttatttt…att PP904074.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV148949.1
atgaaggctatcattgctttgagcaacattctatgtcttgtttt…att PV148120.1
…
atgaagactatcattgctttgagctacattttctgtcaggtttt…att CY006707.1
```

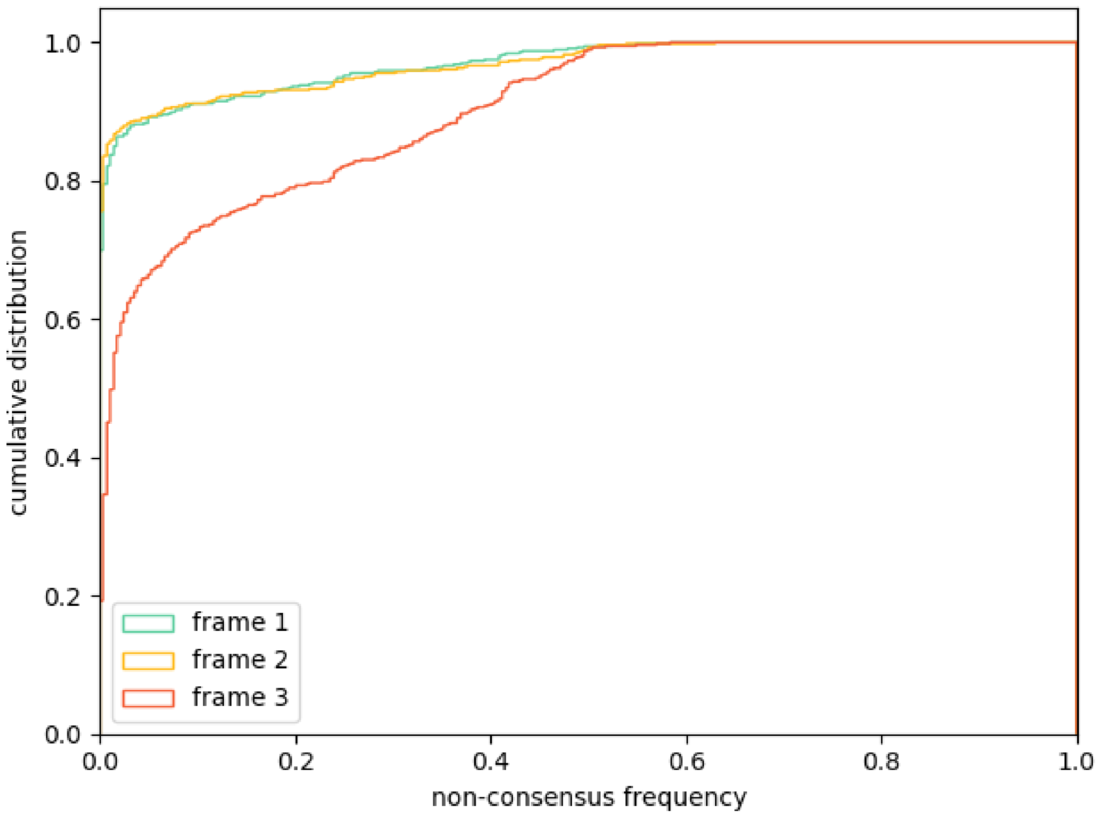
```
Alignment with 284 rows and 566 columns
MKTIIALSYILCLVFAQKLPGNDNSTATLCLGHHAVPNGTIVKT…ICI CY038959.1
MKTIIALSYILCLVFAQKLPGNDNSTATLCLGHHAVPNGTIVKT…ICI CY013919.1
MKTIIALSYILCLVFAQKLPGNDNSTATLCLGHHAVPNGTIVKT…ICI CY038663.1
MKTIIALSYILCLVFAQKLPGNDNSTATLCLGHHAVPNGTIVKT…ICI CY013525.1
MKTIIALSYILCLVFAQKLPGNDNSTATLCLGHHAVPNGTIVKT…ICI CY006115.1
MKTIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTVVKT…ICI OP511830.1
MKTIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PP271579.1
MKTIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI OP842321.1
MKTIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI OP841015.1
MKTIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI OP697544.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV507945.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV280795.1
MKAIIALSNILCLVFAQKIPGNDKSTATLCLGHHAVPNGTIVKT…ICI PV511182.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV222222.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV511659.1
MKAIIALSNILCLIFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PP904074.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV148949.1
MKAIIALSNILCLVFAQKIPGNDNSTATLCLGHHAVPNGTIVKT…ICI PV148120.1
…
MKTIIALSYIFCQVFAQNLPGNDNSTATLCLGHHAVPNGTLVKT…ICI CY006707.1
```

5.2 Diversity along the sequence

- look at how the diversity varies along the sequence. You can measure it in terms of non-consensus frequency or in terms of entropy.
- can you spot any patterns? What do you think is going on?



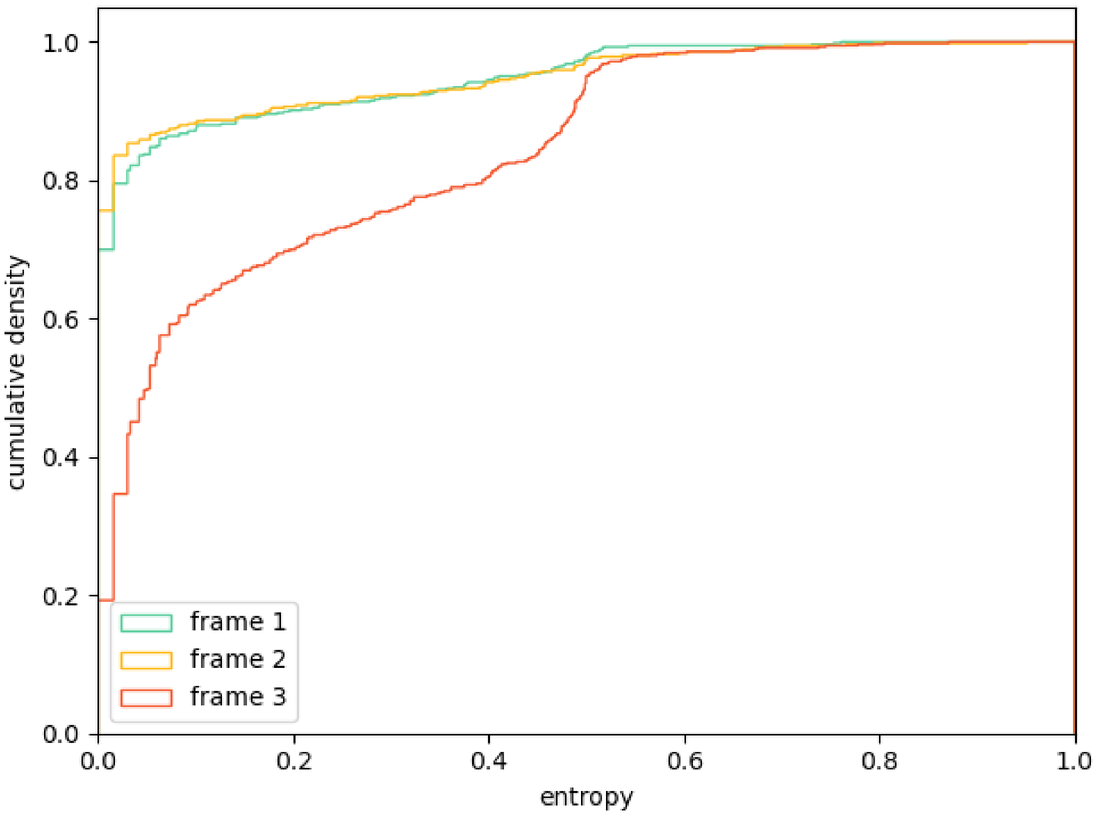
We can indeed observe a pattern: there seems to be a pattern in the high-diversity sites. These are mostly sites that are located at the third position of a codon. These sites are often more free to mutate, as mutations in these sites are more likely to result in **synonymous mutations**.



The same can be measured in terms of entropy of the distribution of nucleotides at each position. This is a measure of diversity that is independent of the consensus sequence. The entropy at position l is defined as:

$$H_l = -\sum_{n\in\{A,C,G,T\}} f_l(n)\log f_l(n)$$

where $f_l(n)$ is the frequency of nucleotide n at position l in the alignment. This quantity is zero if the position is invariant, and reaches its maximum when all nucleotides are present at equal frequencies. We use here logarithm in base 4, such that in this case the maximum possible entropy is 1.



5.2.1 Signatures of selection

More diversity in third-codon position is indicative of imbalance between the ratio of synonymous and non-synonymous mutations, which itself is a signature of selection. There are more rigorous measures of such imbalance, such as the dN/dS ratio, which is the ratio of the rate of non-synonymous mutations to the rate of synonymous mutations. Calculating this ratio is a bit more involved, as it requires a phylogenetic tree and ancestral state reconstruction.

But even from this simple measure, we can see that (unsurprisingly) the virus is under strong selection pressure. - the virus mutates rapidly, and most mutations are lost due to purifying selection - at the same time, the virus is under pressure to escape the immune system. This is especially true for the HA protein.

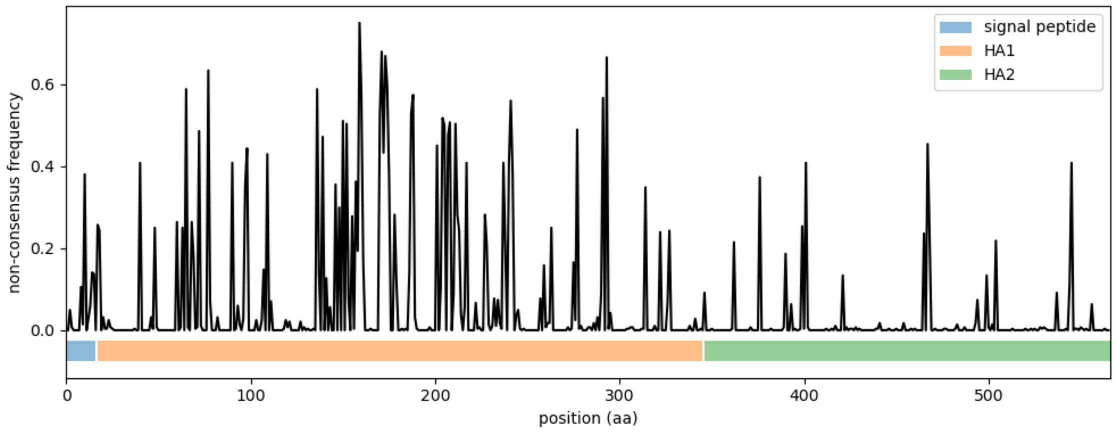
It is important to keep this in mind. Systems under selection often can violate some of the assumptions of evolutionary models, e.g. heterogeneity of rates of evolution…

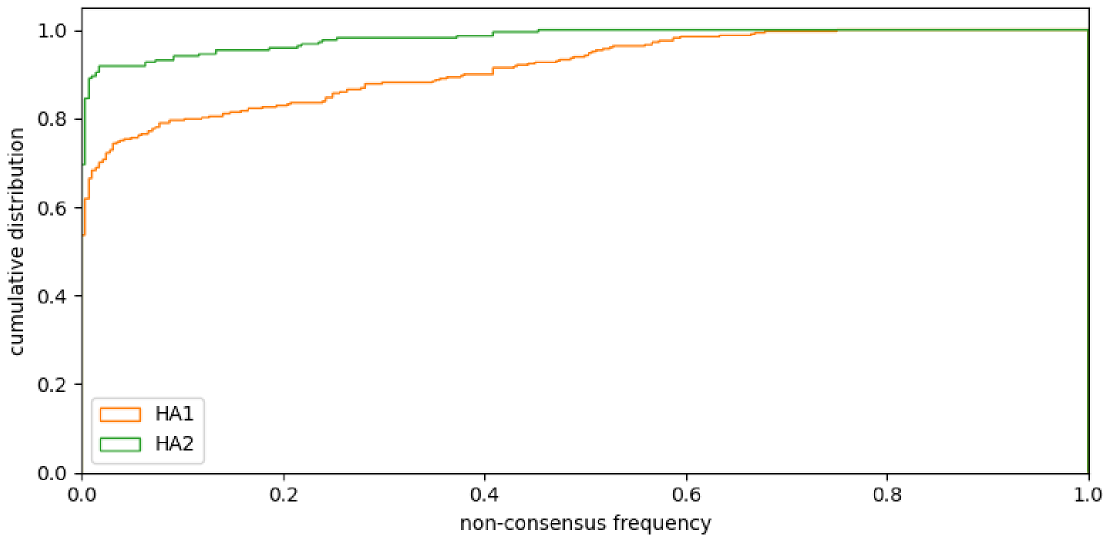
5.2.2 Diversity along the sequence

The diversity is also not uniform along the sequence. The HA gene is divided into two subunits that are cleaved post-translationally:

- HA1: head of the protein. Contains the receptor binding domain. More exposed to the immune system (primary target of antibodies).
- HA2: stem region of the protein. Contains the fusion peptide. Less exposed to the immune system.

These two domains are subject to different levels of selection pressure and consequent antigenic drift. This is reflected in the amino-acid level diversity of the two domains along the sequence.





n02_tree

May 7, 2025

1 Building a phylogenetic tree

In this second part of the tutorial, we will start from the multiple sequence alignment we previously generated, containing nucleotide sequences for the coding region of the HA influenza gene from H3N2 subtype viruses. We will use this alignment to reconstruct a phylogenetic tree that approximates the evolutionary history of these viruses.

We'll then discuss some common ways to visualize, manipulate and explore the tree.

1.1 Building a tree with IQtree

To investigate evolutionary relationships between sequences in the alignment, we have to reconstruct the phylogenetic tree that approximates the evolutionary history of the viruses whose genomes we are analyzing. There are several commonly used tools for this, including

- IQtree
- RAxML
- FastTree

We will use IQtree here, which can again be installed with `conda` via `conda install -c bioconda iqtree`.

We can call IQtree (the program has looooots of options) via

```
iqtree -s <input_alignment> -m GTR
```

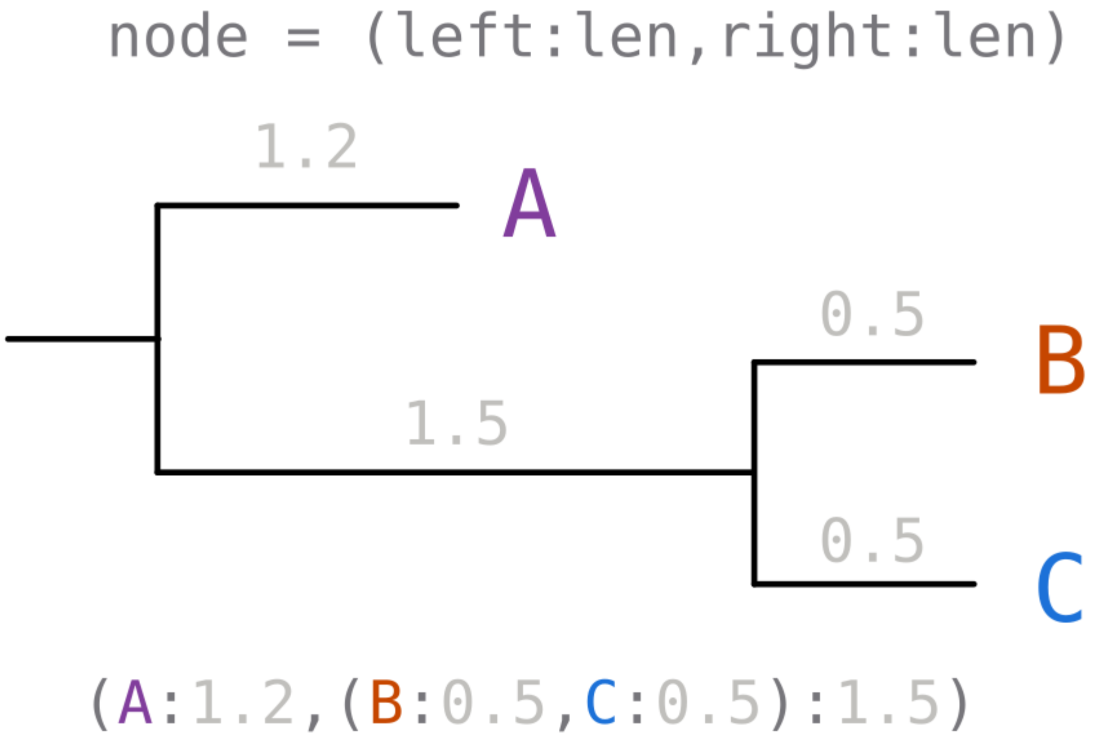
The flag `-m GTR` tells IQtree to use a generalized time-reversible substitution model.

After running the command (it should run for 3-4 minutes), you'll find a number of files in your directory.

```
<input_alignment>.fasta.bionj
<input_alignment>.fasta.ckp.gz
<input_alignment>.fasta.iqtree
<input_alignment>.fasta.log
<input_alignment>.fasta.mldist
<input_alignment>.fasta.treefile
```

Some of these are log files, or checkpoints to restart the analysis. You can check out the `.iqtree` file to get information on the inferred parameters for the substitution model (transition rates, etc.).

The file we care most about is the file ending in `.treefile`. It contains the reconstructed tree in `newick` format.

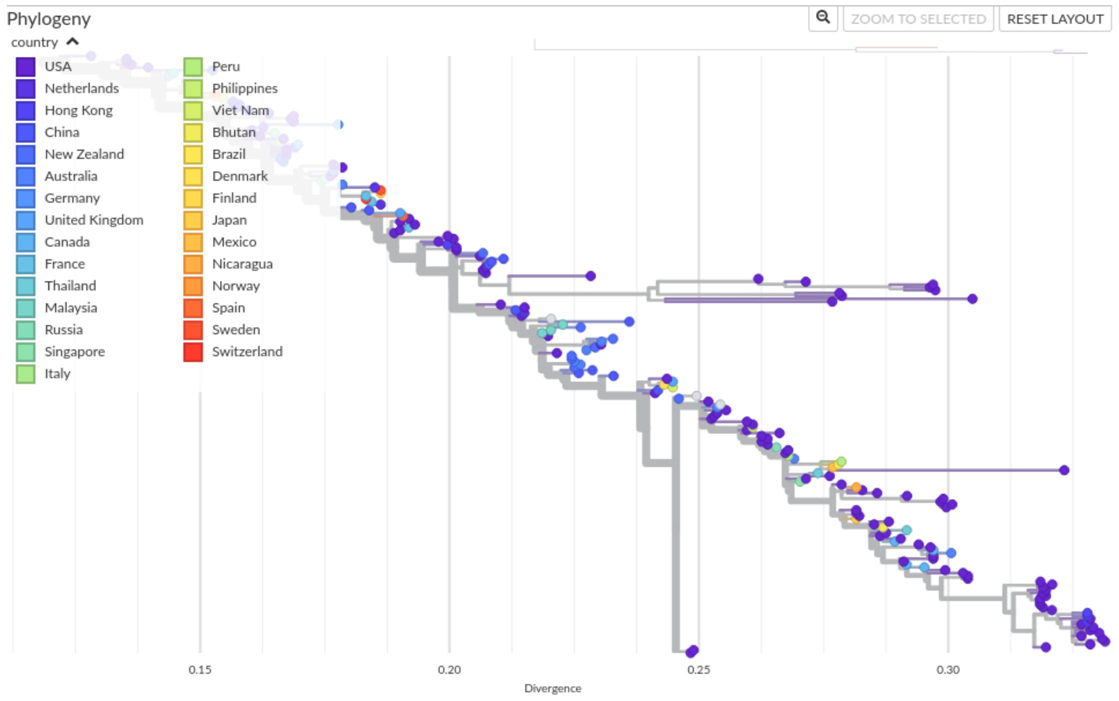


1.2 visualizing the tree

There are many excellent tools for visualizing phylogenetic tree. A very convenient one is Auspice, which is a web-based tool that allows you to visualize and explore phylogenetic trees interactively.

After changing the file extension of the `.treefile` to `.nwk`, upload the file on Auspice and explore the tree.

Phylogenetic analysis often require integrating the phylogenetic tree with other information such as phenotype, geographic location, collection date… For the data at hand, you can also upload the metadata `csv` file generated previously, which lets you also color the tree by metadata.



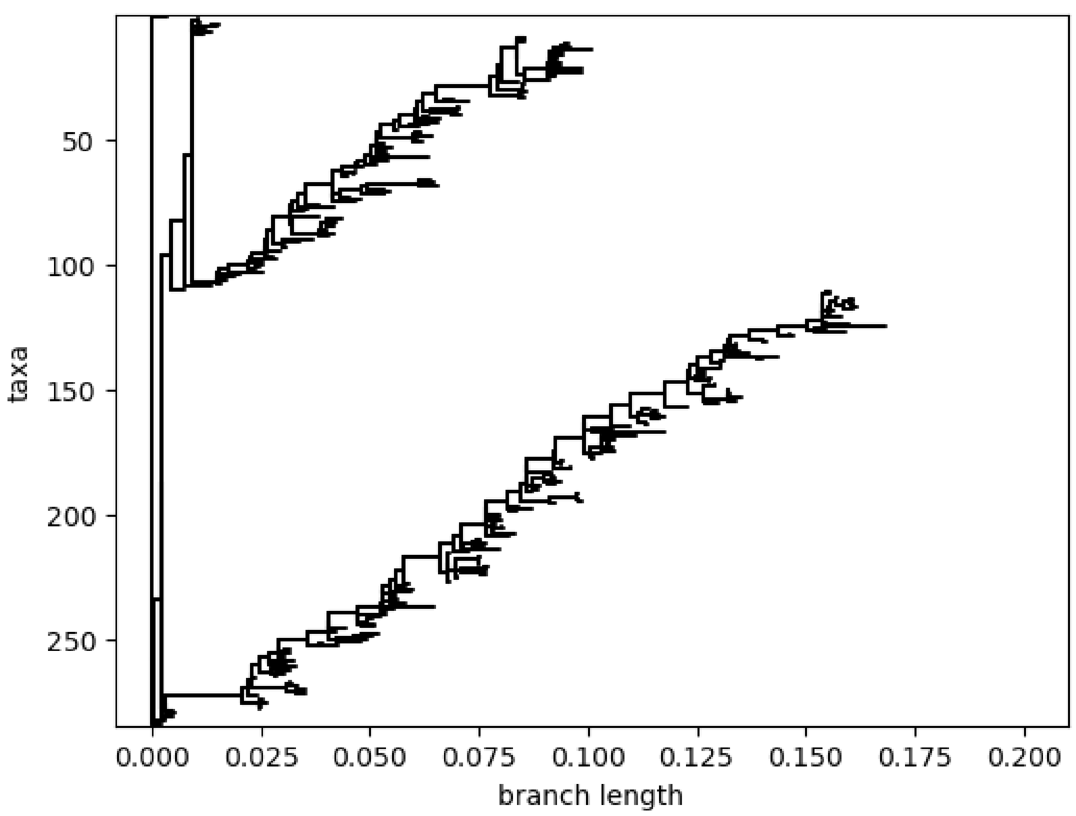
2 Manipulating trees with Biopython

Once obtained your tree, you might want to perform different operations on it, such as calculating node properties, assigning a root…

Biopython provides the `Phylo` library with a number of very useful basic function that allow you to work with phylogenetic trees.

We'll discuss a few common operations here: - loading / drawing trees - ladderizing trees - rooting trees (better discussed in the next tutorial) - preorder/postorder traversal

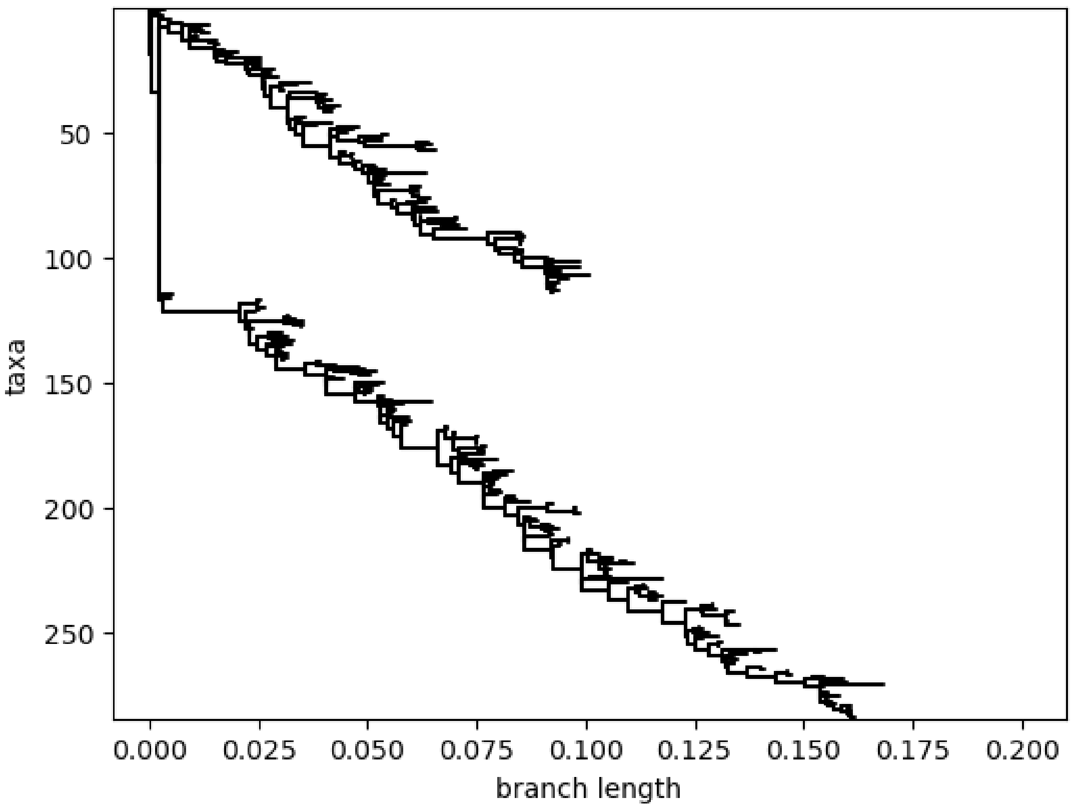
Let's load an visualize the tree. You can use the `read` and `draw` function for this.



2.0.1 Ladderization

Note that whenver we have an internal node, there is no unique way to order the children of that node. Any flip between the order of the children will result in the same tree topology.

A conventional way to resolve this ambiguity is to "ladderize" the tree. This is a common operation in phylogenetics, and it is often used to make trees easier to read and interpret. "Ladderizing" consists in sorting the children of each node such that the child with the smallest number of terminal descendants is placed first (or last). Biopython conveniently implements the `ladderize` function for this.



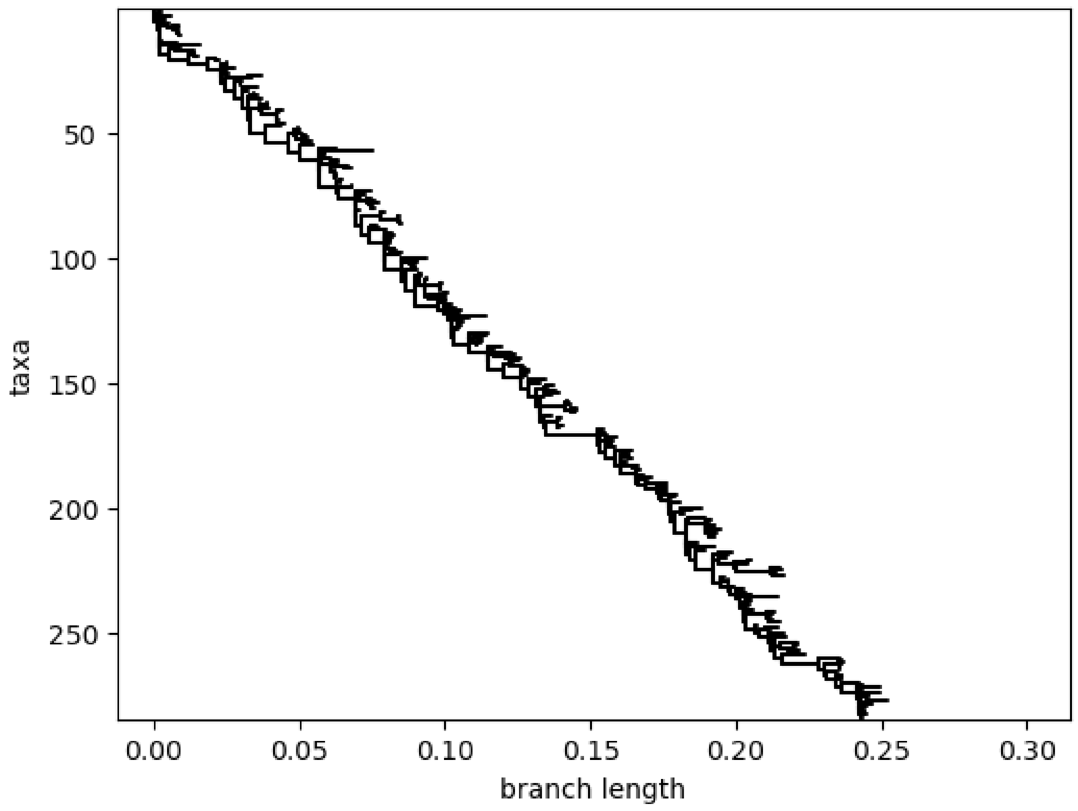
2.1 Rooting the tree

Another important decision when it comes to analyzing phylogenetic trees is the placement of the root. This decides the direction of time along the branches.

There are several ways to root a tree. We will discuss them in more detail in the next section. For now, we will root the tree using the oldest sequence in the dataset as an outgroup.

```
                                                          description       date  \
accession
CY034012.1  Influenza A virus (A/Hong Kong/1-6-MA21-2/1968… 1968-01-01
CY033553.1  Influenza A virus (A/Hong Kong/1-11-MA21-3/196… 1968-01-01
CY033017.1  Influenza A virus (A/Hong Kong/1-4/1968(H3N2))… 1968-01-01
CY033988.1  Influenza A virus (A/Hong Kong/1-1-MA-12D/1968… 1968-01-01
CY146809.1  Influenza A virus (A/Victoria/JY2/1968(H3N2)) … 1968-01-01

              country   len  ambiguous  collection_year
accession
CY034012.1  Hong Kong  1715          0             1968
CY033553.1  Hong Kong  1719          0             1968
CY033017.1  Hong Kong  1720          0             1968
```

```
CY033988.1  Hong Kong  1719          0             1968
CY146809.1  Australia  1743          0             1968
```



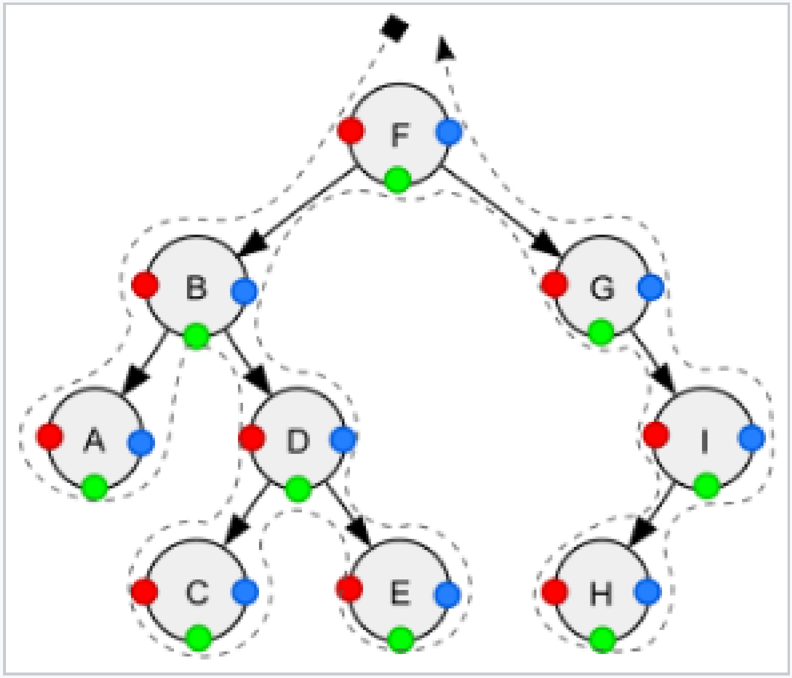
```
1
```

2.2 Tree traversal

Unlike lists, trees are not linear data structures, but rather hierarchical data structures. Many of the algorithms that operate on trees require you to visit all nodes, but there is no single order in which to do this, and the order often matters.

There are several common ways to traverse a tree:

- **pre-order**: visit the parents before its children, starting at the root. This is often useful if the information needed at a node can be calculated from the information of its parent. (E.g. distance from the root).
- **post-order**: visit children before its parent Useful if the information needed at a node can be calculated from the information of its children. (E.g. number of leaves below a node).
- level or **breadth-first** traversal: root -> children of the root -> grand-children of the root -> …

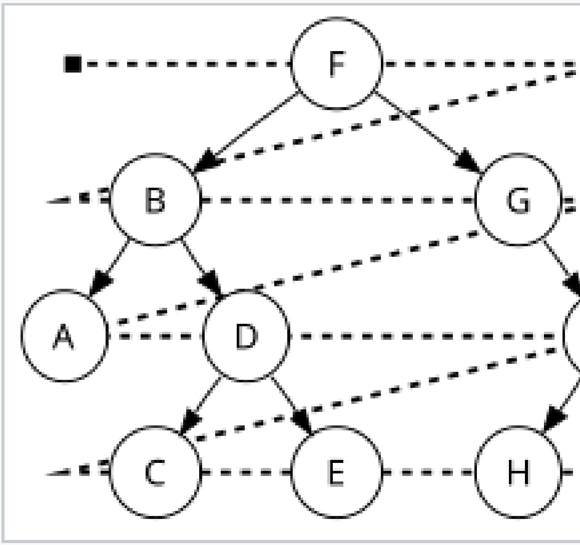


Depth-first traversal (dotted path) of a binary tree:

Pre-order (node visited at position red ●*):*
F, B, A, D, C, E, G, I, H;

In-order (node visited at position green ●*):*
A, B, C, D, E, F, G, H, I;

Post-order (node visited at position blue ●*):*
A, C, E, D, B, H, I, G, F.

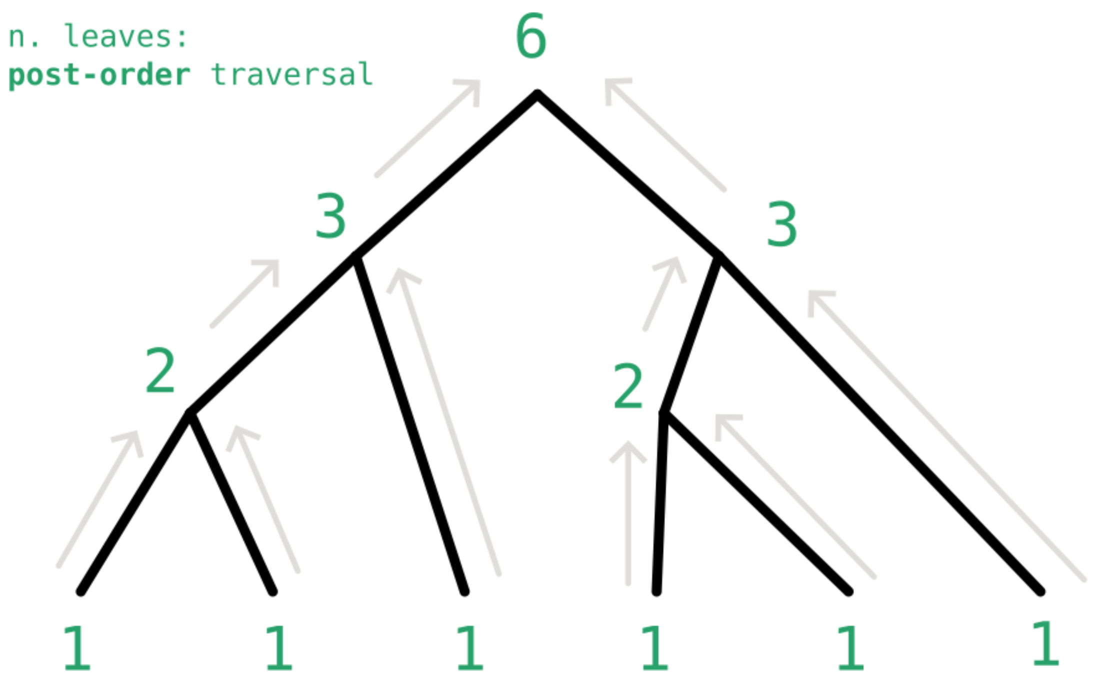


Level-order: F, B, G, A, D, I, C, E,

Pre-order and post-order traversal are at the heart of many efficient recursive algorithms.

2.2.1 Simple example: calculate the number of leaves below each node.

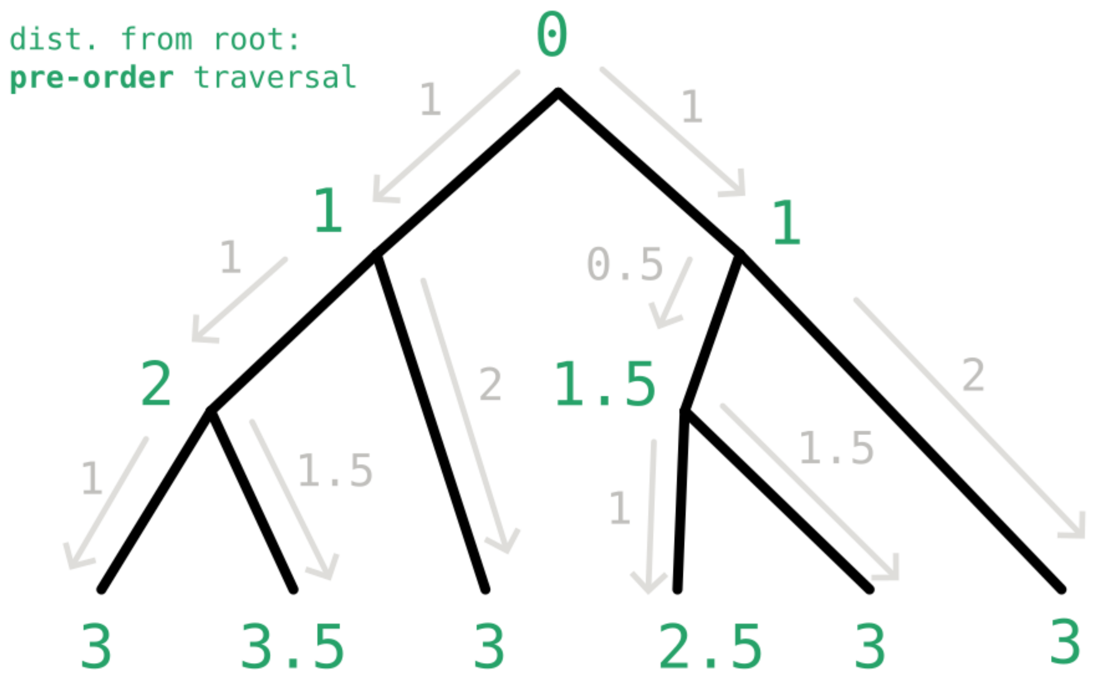
If we want to calculate the number of leaves below each internal node, we can use a post-order traversal. We can assign to each terminal node `n_leaves=1`, and for each internal node, we can sum the number of leaves of its children.



```
total n. leaves: 284
```

2.2.2 Simple example: calculate the distance to the root.

If we want to calculate the distance of each internal node to the root, we can use a pre-order traversal. We can assign `distance=0` to the root, and for each other node we can set its `distance` equal to the distance of its parent plus the node branch length.



```
few root-to-tip distances:
node: CY034012.1, distance: 0.000000
node: CY033988.1, distance: 0.002354
node: CY112257.1, distance: 0.005885
node: CY019899.1, distance: 0.003527
node: CY021837.1, distance: 0.004115
```

3 Ancestral reconstruction

A very important problem in phylogenetics is to infer properties of "unobserved" internal nodes of the tree. For example, we might want to infer the nucleotide sequence of an ancestor of the sequences we are analyzing. Knowing the sequences of ancestors is extremely useful, as it allows for example to study the frequency and type of mutations observed. This is called "ancestral sequence reconstruction".

Other than sequence, one can also infer other properties of internal nodes, such as for example the geographic location. As a next example we will use the operations we discussed above to implement a simple algorithm for ancestral reconstruction.

3.1 A simple example: Fitch parsimony

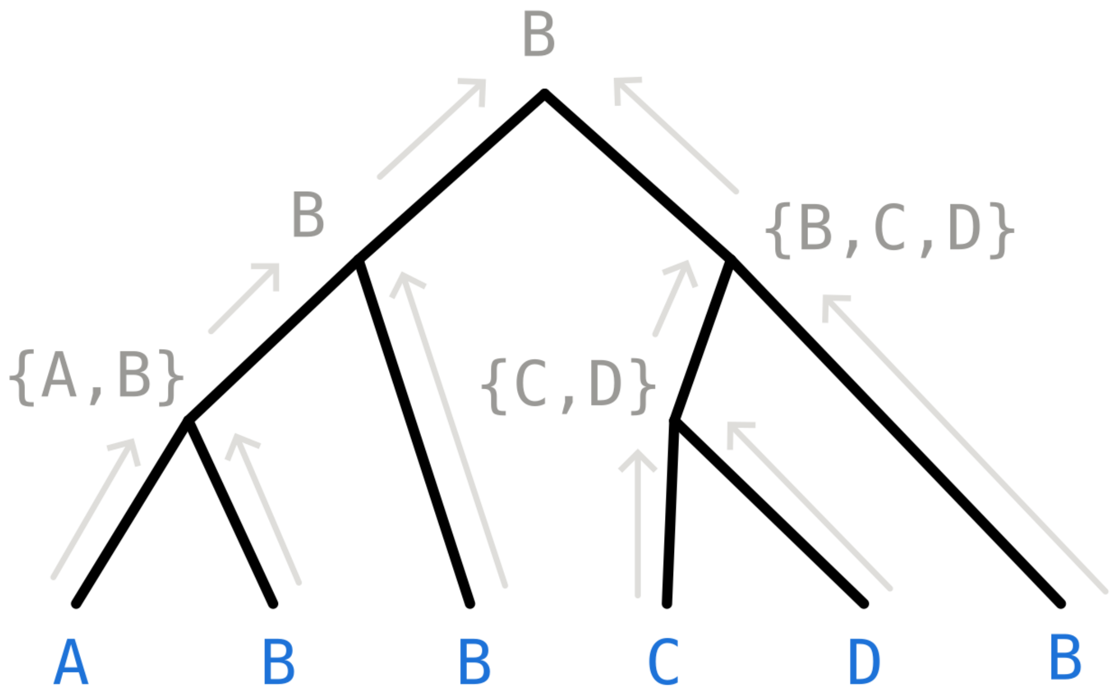
Fitch parsimony is a simple algorithm for inferring the state of internal nodes of a tree, given the state of all terminal nodes. It is a dynamic programming algorithm that is guaranteed to find the most parsimonious solution, i.e. the one that requires the least number of changes of state to explain the pattern observed on the leaves. Note that this solution is not guaranteed to be unique, and there might be multiple equally parsimonious solutions. In this case the algorithm will return one of them.

3.1.1 Algorithm description

The algorithm works with two three traversals.

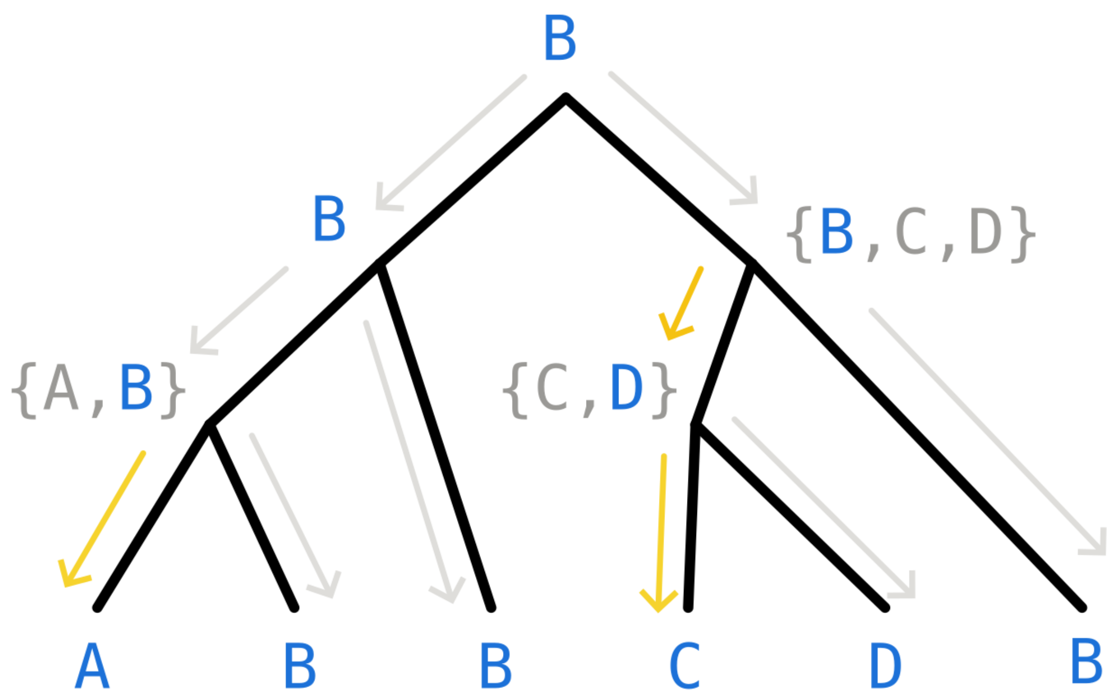
In the first traversal nodes are visited post-order, starting from the leaves. In this traversal we assign to each node **a set of possible states**, given the set of possible states of its children.

- the state set of each terminal node is known
- the state of internal node is derived from the states of its children.
  - The state set of the node is the **intersection** of the state sets of its children, if this intesection is not empty.
  - If the intersection is empty, the state set of the node is the **union** of the state sets of its children. In this case we know that at least one change of state is required to explain the pattern observed in the leaves.



The second traversal is a pre-order traversal, where we **assign state to each node**.

- We assign to the root one of the states in its state set. If more than one state is possible, we can pick one at random.
- For each other node:
  - if the state set contains the state of its parent, we assign the same state to the node.
  - if the state set does not contain the state of its parent, we assign to the node one of the states in its state set, chosen at random. This corresponds to a change of state (yellow arrows).

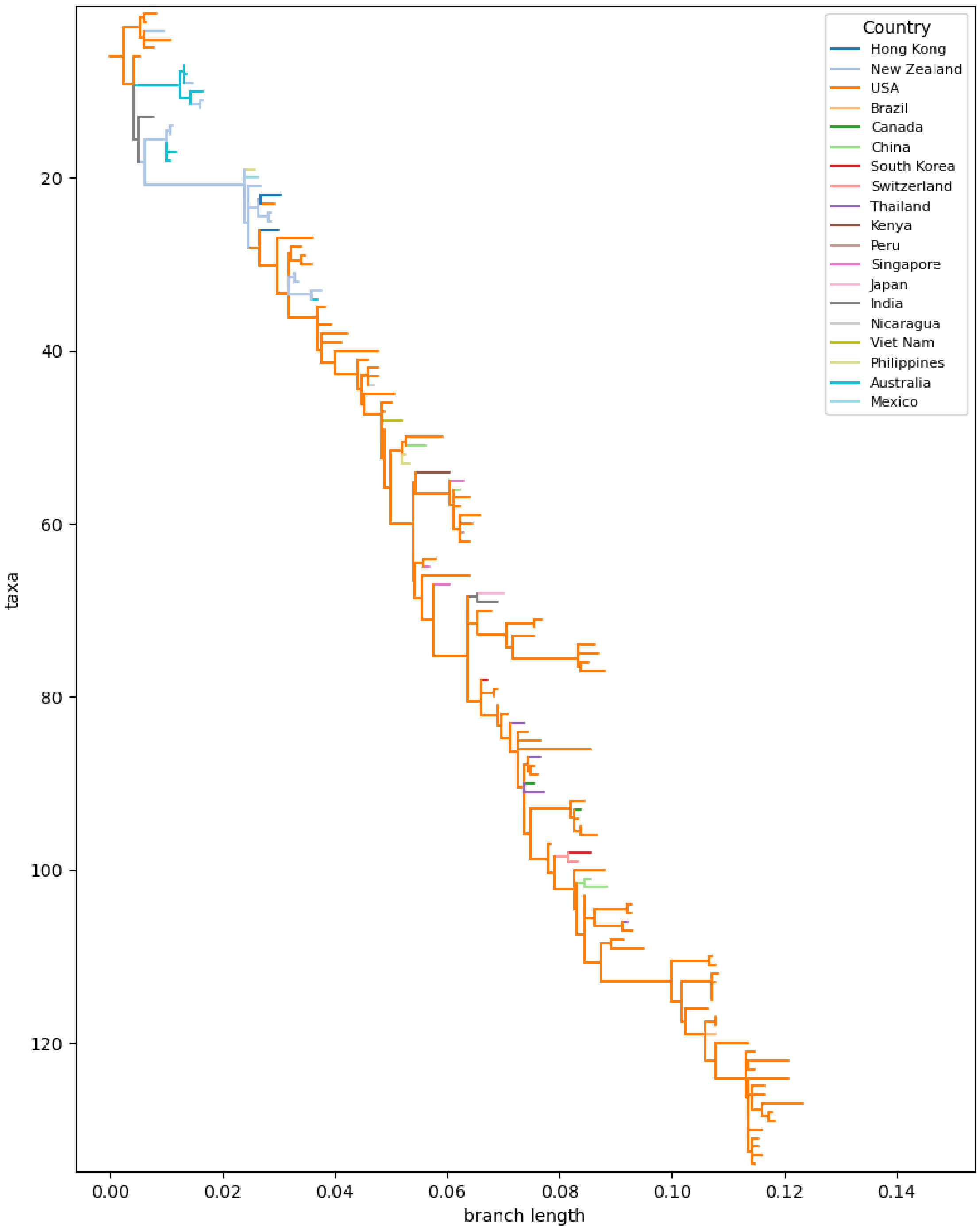


The final solution is guaranteed to have the minimal number of changes of state possible (yellow arrows).

3.1.2 Infer geographic location of internal nodes

Let's use the functions we discussed above to implement the Fitch algorithm and infer the geographic location of the internal nodes of the tree we built previously.

```
Preorder traversal terminated
Possible root states: {'Hong Kong'}
Root state set to: Hong Kong
Preorder traversal terminated

total number of transitions: 104
out of a total of  567 nodes
most common transitions:

 transitions
 USA -> Netherlands          9
 USA -> China                9
 Netherlands -> USA          8
 USA -> Hong Kong            7
 Netherlands -> France       4
 USA -> Thailand             4
 Hong Kong -> USA            3
 Australia -> New Zealand    3
 USA -> Australia            3
 USA -> New Zealand          3
 Name: count, dtype: int64
```



3.1.3 Some caveats

This algorithm is very simple and has a number of limitations:

- this approach completely ignores branch lengths. For example, a change of state on a short branch should be less likely than a change of state on a long branch.
- the algorithm also ignores the fact that rates of transitions are not uniforms (e.g. countries have different sizes, and different sampling biases… Nucleotides are not equally likely to mutate into each other…).
- often there are many possible parsimonious solutions, and the algorithm only returns one.

This algorithm is appropriate only when changes of state are rare and unambiguous. When we need a more precise solution, it is advisable to use more sophisticated methods with a proper probabilistic model that takes these variables into account.

4 Consistency check: alignment distance vs tree distance

When reconstructing a phylogenetic tree, the algorithm tries to maximize the likelihood of the tree given the alignment. This means that the distance between two sequences in the alignment should be closely related to the distance between the two sequences in the tree.

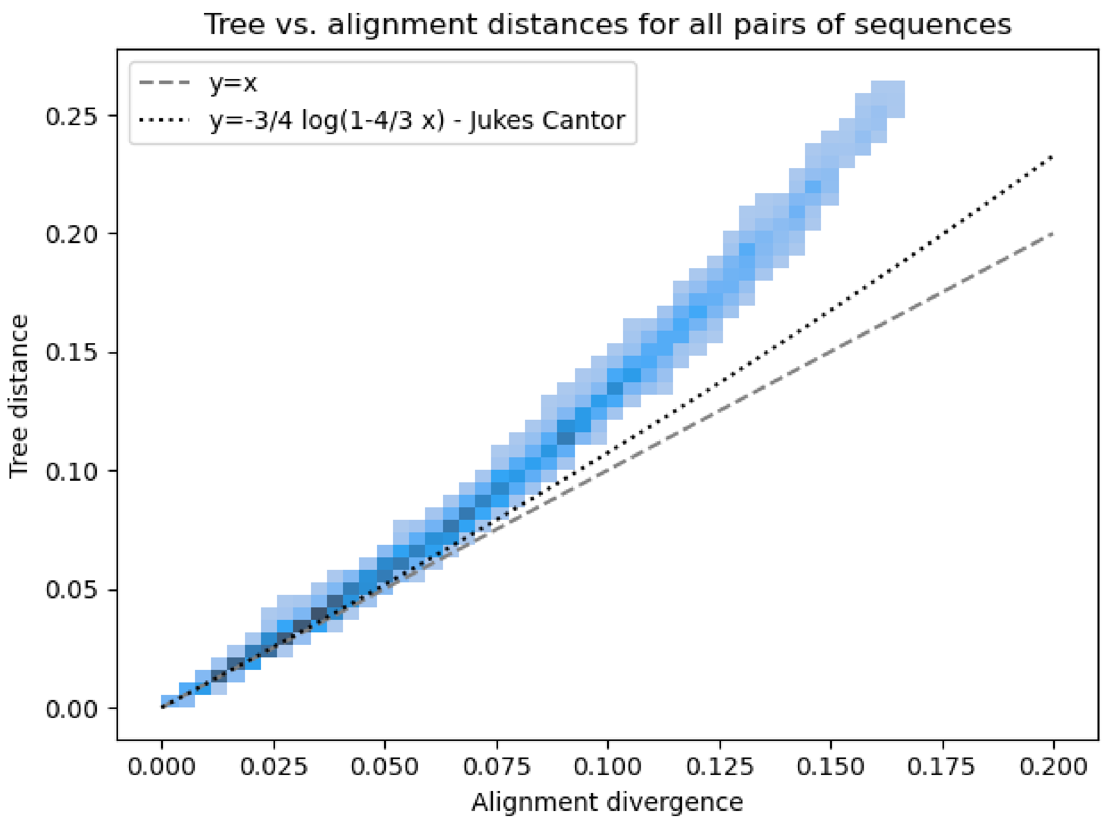
We can check that this is the case by calculating these two distances: - sequence divergence can be calculated from the hamming distance in alignment - distances of leaves on the tree can be calculated using the `distance` method of the tree object.

Look at the joint distribution of the two distances. What do you observe? Why is their relationship not linear?

```
                       aln_divergence
id1        id2
CY013919.1 CY038959.1        0.012956
CY038663.1 CY038959.1        0.010012
CY013525.1 CY038959.1        0.013545
CY006115.1 CY038959.1        0.014723
CY038959.1 OP511830.1        0.072438
…                                 …
CY006691.1 CY020325.1        0.001767
           CY006707.1        0.002945
CY006699.1 CY020325.1        0.004711
           CY006707.1        0.004711
CY006707.1 CY020325.1        0.003534

[40186 rows x 1 columns]
```

```
                       tree_distance
id1        id2
CY033988.1 CY034012.1       0.002354
CY034012.1 CY112257.1       0.005885
CY019899.1 CY034012.1       0.003527
```

```
CY021837.1 CY034012.1       0.004115
CY034012.1 CY113045.1       0.005291
…                               …
PP619459.1 PP904074.1       0.002954
PP685646.1 PP904074.1       0.001772
PP619459.1 PQ412865.1       0.002955
PP685646.1 PQ412865.1       0.001772
PP619459.1 PP685646.1       0.002361

[40186 rows x 1 columns]
```



n03_rooting_trees

May 7, 2025

1 The direction of time: rooting a tree

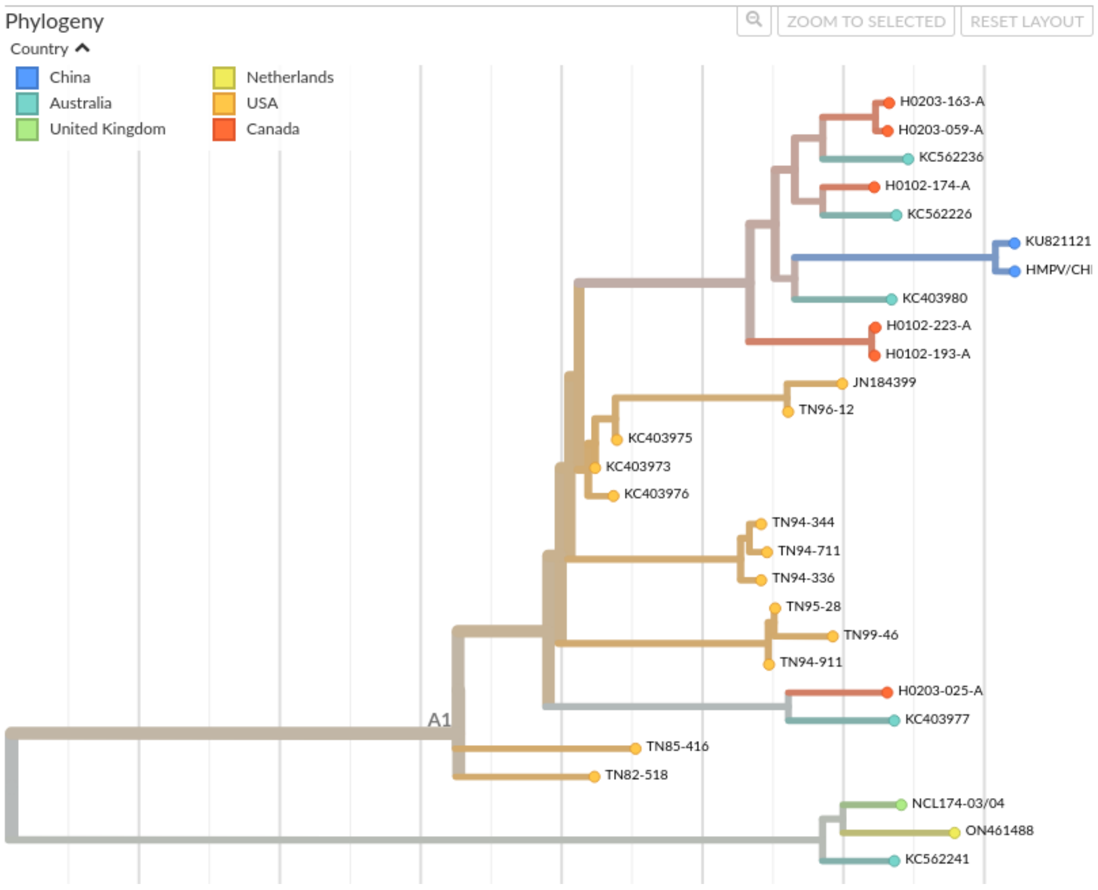
In the last part of this tutorial we'll discuss the problem of rooting a tree.

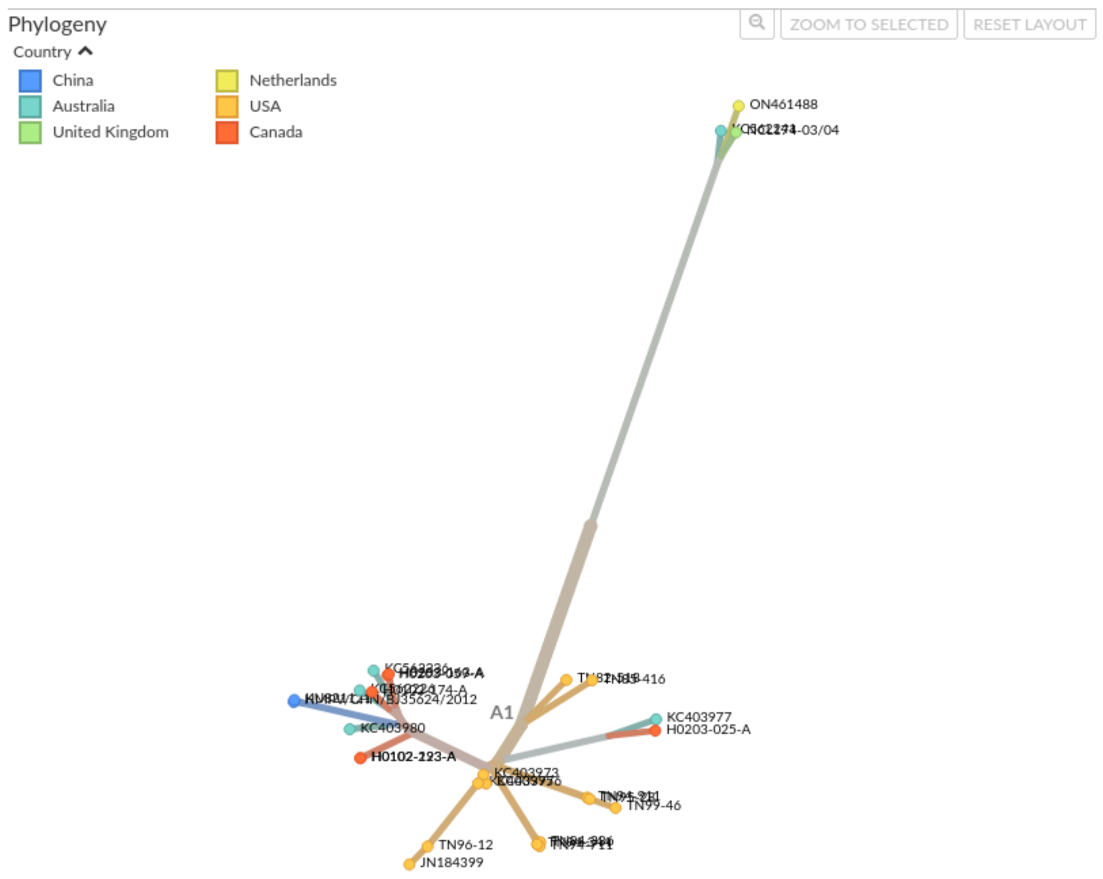
By picking a root, one defines a direction of time along branches. We'll discuss few commonly used methods to root a tree: - midpoint rooting - outgroup rooting - rooting using information on collection times and reconstructing timetrees

1.1 Rooted vs unrooted trees

Trees inferred by the canonical tree-building programs are typically "rooted", meaning they have singled out a particular node in the tree to act as the "parent" of all other nodes. However, as you saw in previous lectures, the models of evolution used to infer the tree are typically "time reversible", meaning they don't differentiate between past and future and the likelihood of the tree does not depent on the choice of root. And **while tree-building programs return a rooted tree, these trees are typically rooted arbitrarily**. This is an important point to keep in mind.

The two trees below are a rooted and unrooted version of the same tree.


Phylogeny
ZOOM TO SELECTED
RESET LAYOUT
Country
China
Australia
United Kingdom
Netherlands
USA
Canada
H0203-163-A
H0203-059-A
KC562236
H0102-174-A
KC562226
KU821121
HMPV/CHI
KC403980
H0102-223-A
H0102-193-A
JN184399
TN96-12
KC403975
KC403973
KC403976
TN94-344
TN94-711
TN94-336
TN95-28
TN99-46
TN94-911
H0203-025-A
KC403977
A1
TN85-416
TN82-518
NCL174-03/04
ON461488
KC562241

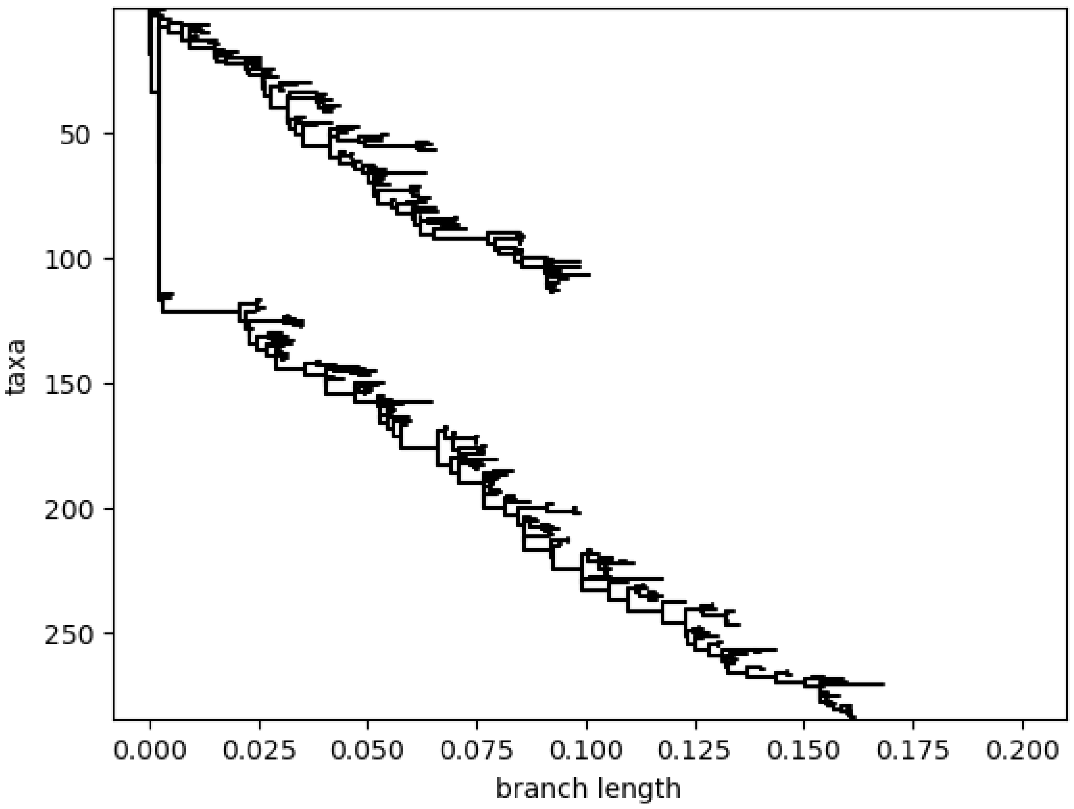


How does one pick a root?

1.2 Preliminary steps: loading our data

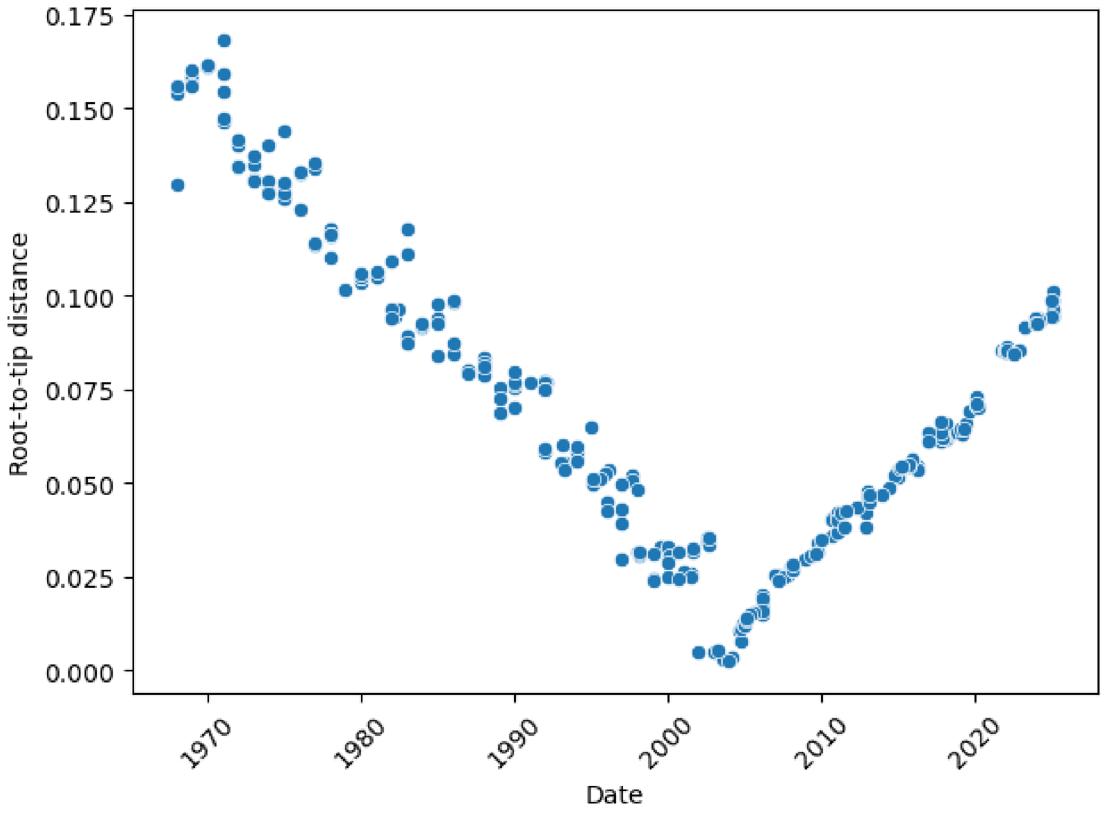
We will experiment with rooting the three that we build in the previous tutorial part. We can start by loading and visualizing the tree.

Our tree has two relatively long arms, and the roots sits in the middle.



In our case we have informations that we could use to verify how good our rooting is: we are considering a fast-evolving virus, and we have samples spread over several decades, with know collection times. This is a very special case, but one in which we can verify the choice of root. A good rooting is one in which samples close to the root are more ancient, and samples further away from the root are more recent.

To verify how the current rooting is, we can load the metadata information, containing the collection date of each sample, and scatter-plot the date against the distance of each sample from the root.



With the current rooting, the most recent sample sit at intermediate distance from the root, while samples closer to the root are older. This is a sign of a bad rooting point.

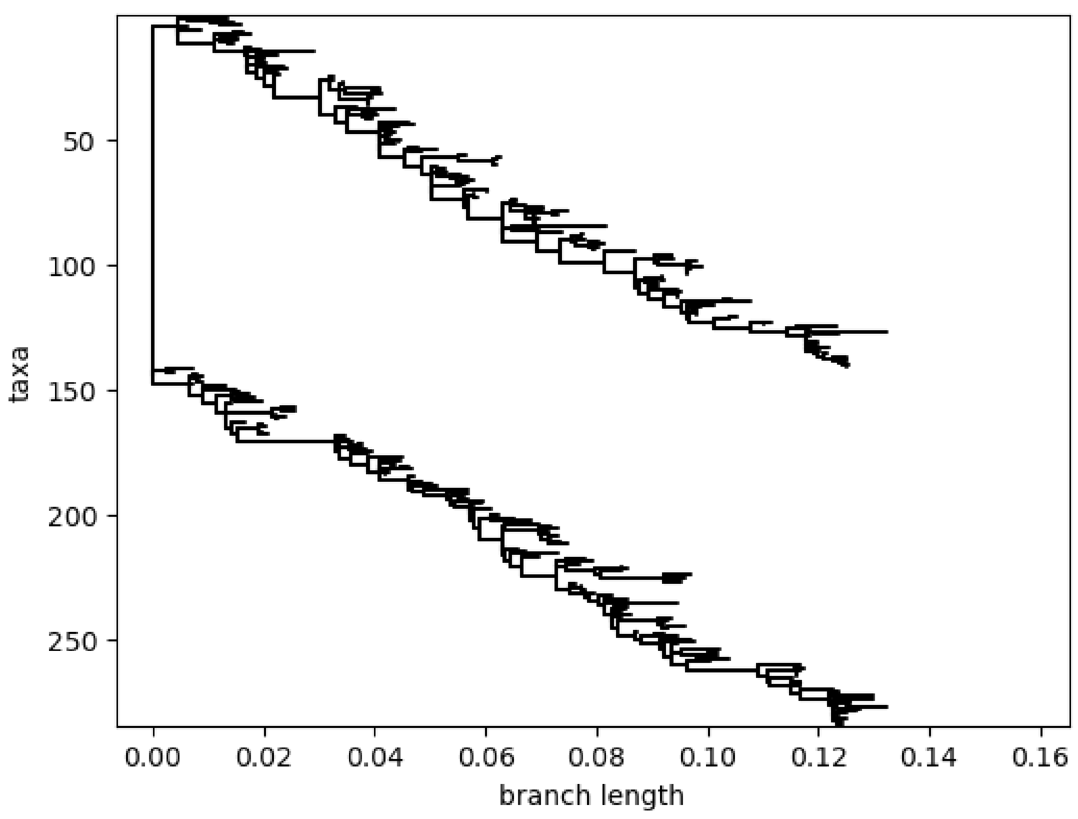
2 Common ways of picking a root

Below we discuss few common ways of picking a root.

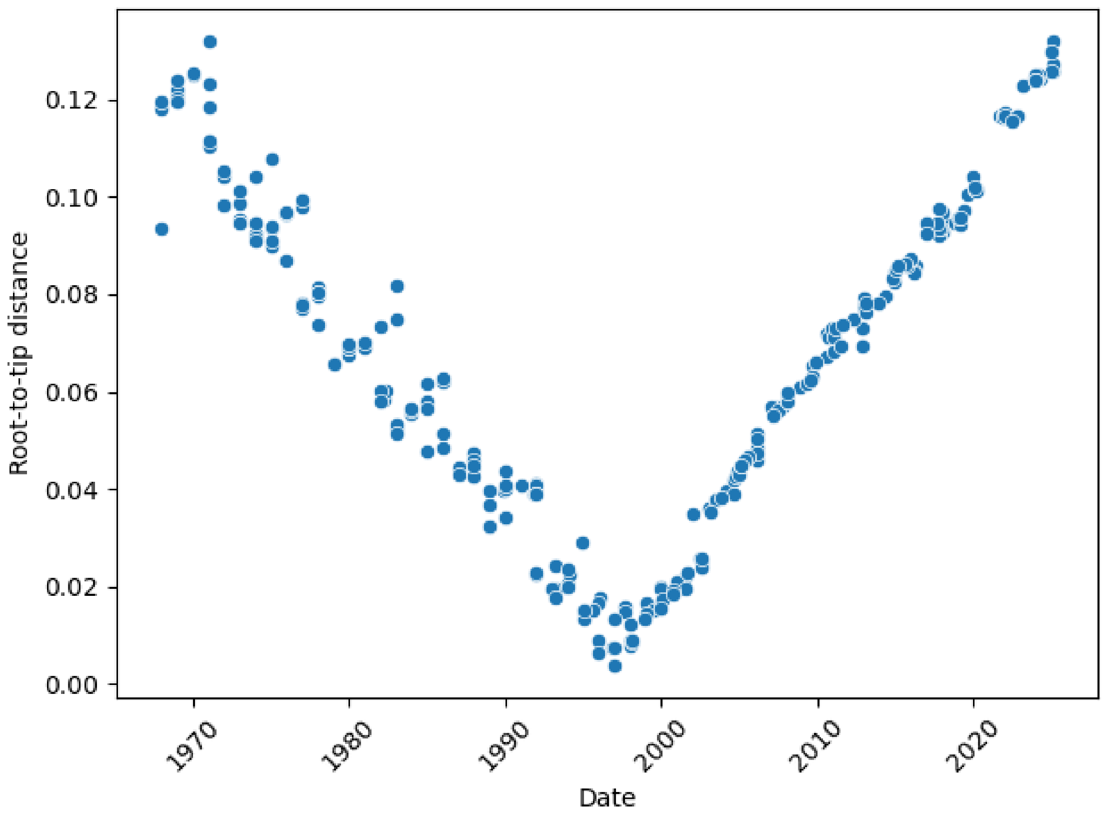
2.1 Midpoint rooting

Midpoint rooting picks the root to be half-way between the two tips that are furthest away of each other. This choice of root is often natural when the samples have evolved from a common ancestor at similar rates and with similar divergence times. In this case you expect all tips to have a similar distance from the tree root, and the midpoint between the two most diverged tips is an approximation of that. There are also variations of mid-point rooting minimize the variance if root-to-tip distance. One of the advantages of midpoint rooting is that it does not require any additional information, such as outgroup or collection times.

In our case this method is not expected to work well, since we are violating one core assumptions: samples are not equally diverged from the root, since we sample the evolution of the H3N2 flu A virus from the pandemic onset (close to the root) to the present day (far from the root). In our case this method erroneously places the root in the middle of the tree.



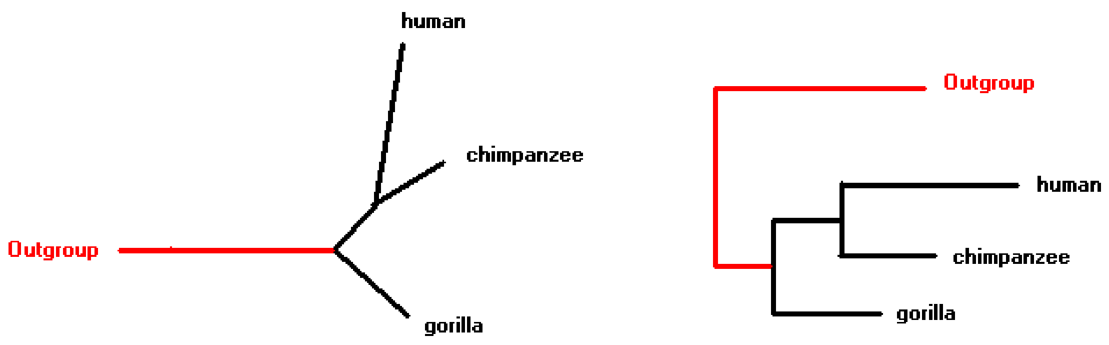
taxa
50
100
150
200
250
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
branch length



2.2 Outgroup rooting

Another common way of rooting a tree is by using an outgroup, i.e. a sequence that is clearly further diverged from the dataset of interest (the ingroup) than any other sequence in the dataset. For example, if you were investigating how different mice species are related to each other, you could use the genetic sequence of a rat as an outgroup. All sequences from the ingroup should form one clade and the point where the outgroup joins the ingroup is a good estimate of the common ancestor of the ingroup.

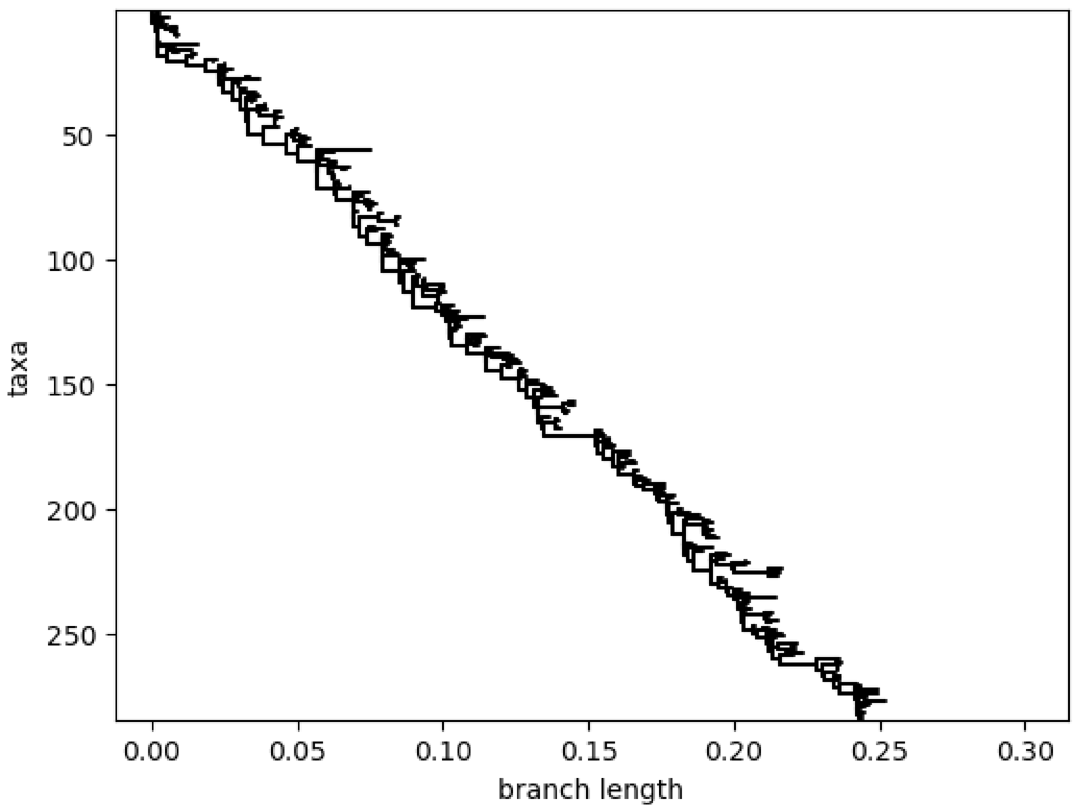
See this illustration from U Manchester

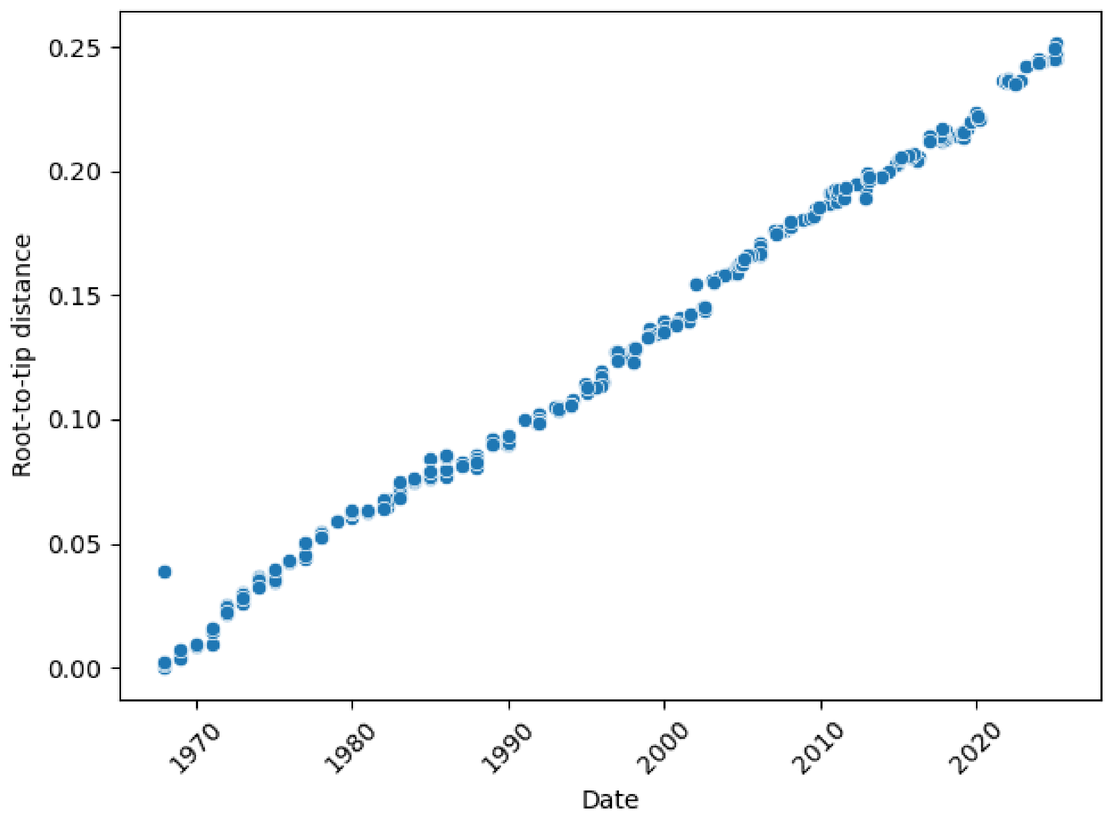


In our case for simplicity we will not add an extra sequence as an outgroup, but as an approximation we will use the oldest sample in our dataset as an outgroup. This will be the sample that should be closer to the root.

Indeed, we see that with this choice of root, sample times are well correlated with the distance from the root.

```
                                                   description       date  \
accession
CY034012.1  Influenza A virus (A/Hong Kong/1-6-MA21-2/1968… 1968-01-01
CY033553.1  Influenza A virus (A/Hong Kong/1-11-MA21-3/196… 1968-01-01
CY033017.1  Influenza A virus (A/Hong Kong/1-4/1968(H3N2))… 1968-01-01
CY033988.1  Influenza A virus (A/Hong Kong/1-1-MA-12D/1968… 1968-01-01
CY146809.1  Influenza A virus (A/Victoria/JY2/1968(H3N2)) … 1968-01-01

              country   len  ambiguous  collection_year
accession
CY034012.1  Hong Kong  1715          0             1968
CY033553.1  Hong Kong  1719          0             1968
CY033017.1  Hong Kong  1720          0             1968
CY033988.1  Hong Kong  1719          0             1968
CY146809.1  Australia  1743          0             1968
```





2.3 Rapidly evolving viruses and timetrees

As discussed above, our example comes from a very special case: a rapidly evolving virus.

Rapidly-evolving viruses adapt rapidly enough that they accumulate significant numbers of mutations over a few years, and we can follow their evolution in real time. Example include SARS-CoV-2, Influenza viruses, and other RNA viruses. They typically accumulate between 10 and 40 mutations per genome per year.

As mentioned above, in these cases information on sampling time can be used to inform the choice of the root. We expect the distance of the individual sequences from the root of the tree to increase with their sampling date: later sequences should have more differences compared to the ancestor. We can use this expectation to more accurately estimate the root. Specifically, we expect that the distance d_i of genome i from the ancestor should grow linearly with the time $t_i - T_{MRCA}$ since the common ancestor

$$d_i = \mu(t_i - T_{MRCA}) + \epsilon$$

where ϵ captures the stochasticity of the process (more precisely, we expect the number of mutations d_i to be Poisson distributed with mean $\mu(t_i - T_{MRCA})$.

This expectation ca be used for two purposes.

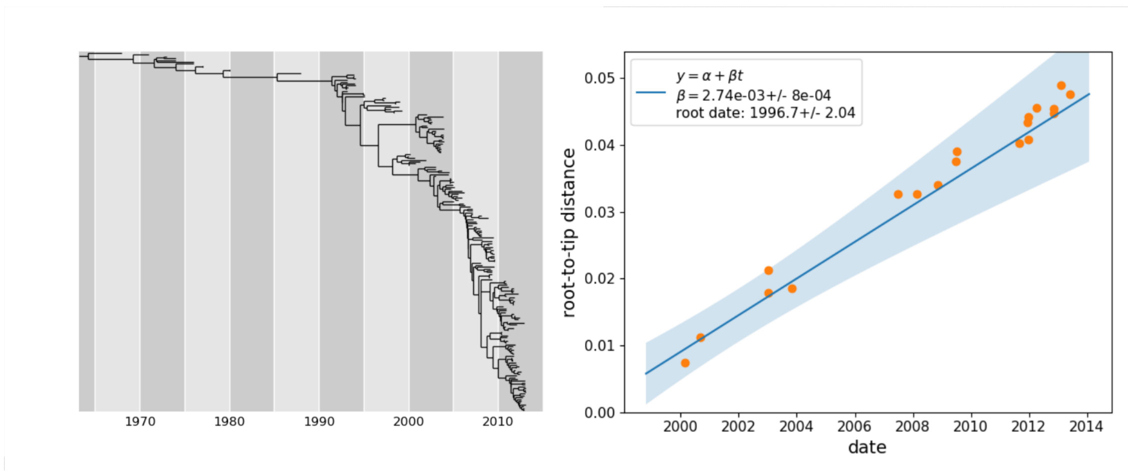
2.3.1 Optimization of the root position

The position of the root can be optimized efficiently without exhaustively searching for the minimum of

$$\sum_i (d_i - \mu(t_i - T_{MRCA}))^2$$

for all possible choices of roots. The best-fitting slope μ serves as an estimate of the evolutionary rate, i.e. the number of mutations that a genome accumulates per unit time.

Such a search for the root is implemented in several packages, including TreeTime. This algorithm will place the root such that the root-to-tip vs time regression is straight:

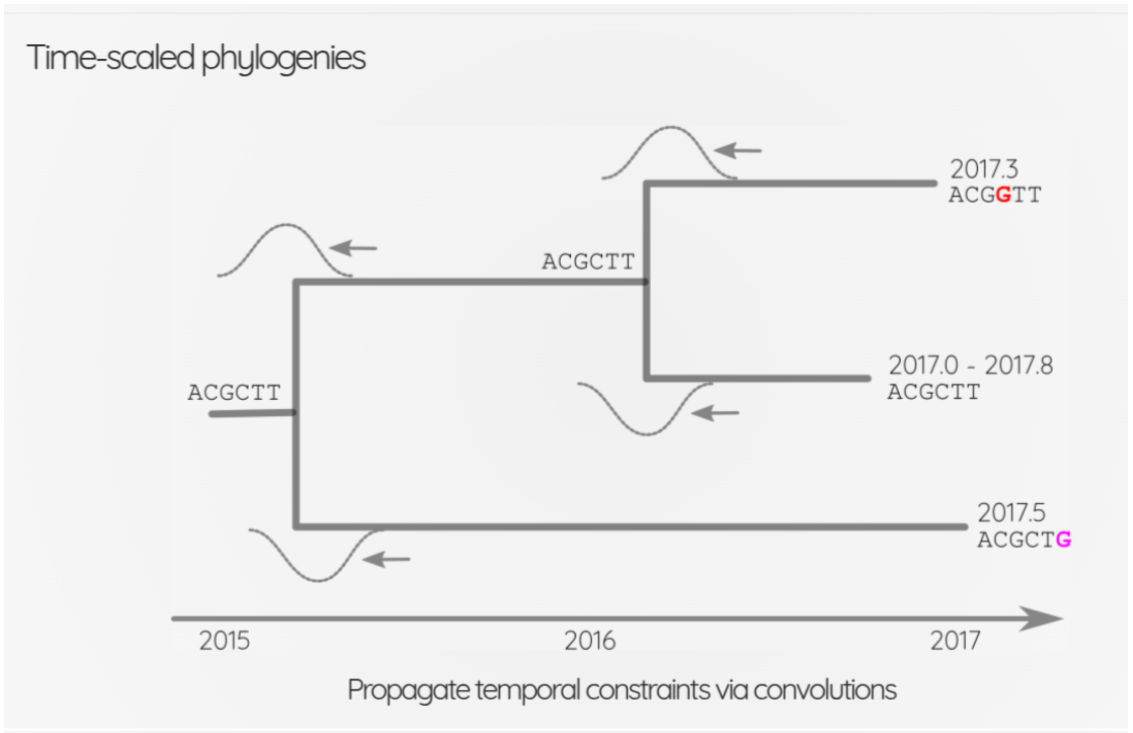


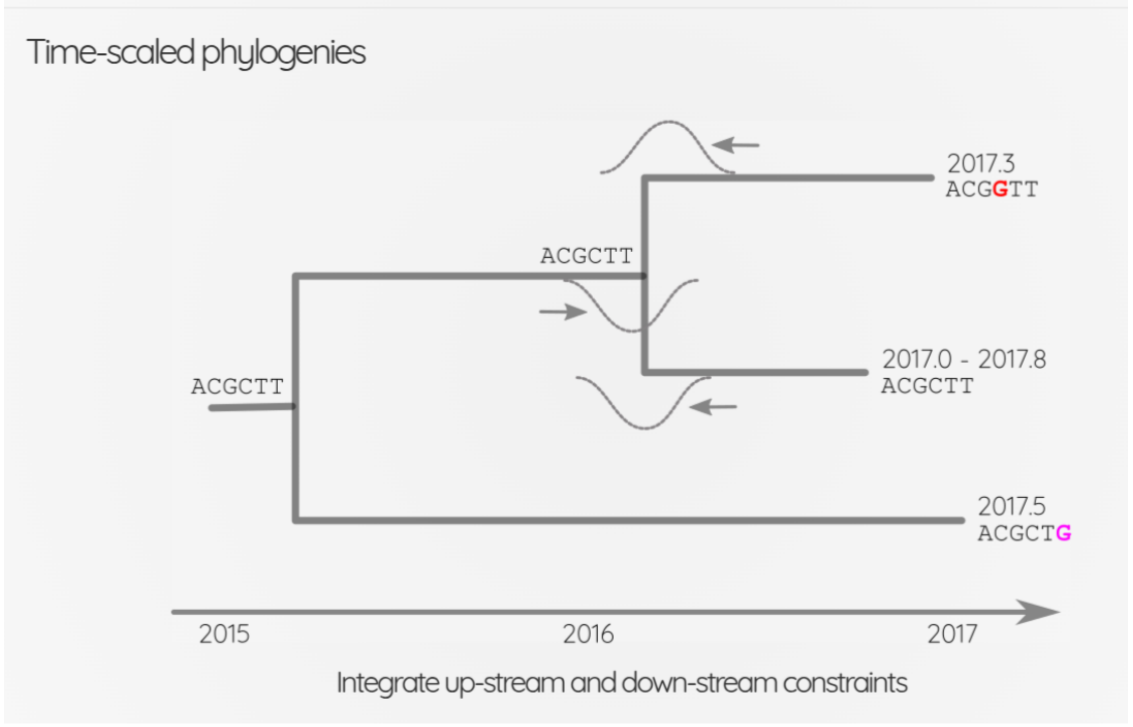
The slope of the regression line is an estimate of the evolutionary rate, that is the rate at which mutations accumulate in the sequences.

2.3.2 Time trees and time-scaled phylogenies

The same expectation can be used to estimate the time of internal nodes in the tree, and build a time-scaled phylogeny.

You can find more details on how this can be done in the treetime paper, but the general idea is that you can combine time constraints that come from the sampling times of sequences with the expectation of evolutionary times that come from their genetic distance. Similarly to what we saw for the Fitch parsimony algorithm, you can propagate this information back through the tree towards the root, to find the most likely time for each internal node, then assign the most likely time to the root. Then the information can be propagated towards the leaves, and progressively pick the most likely time for each branch and internal node.



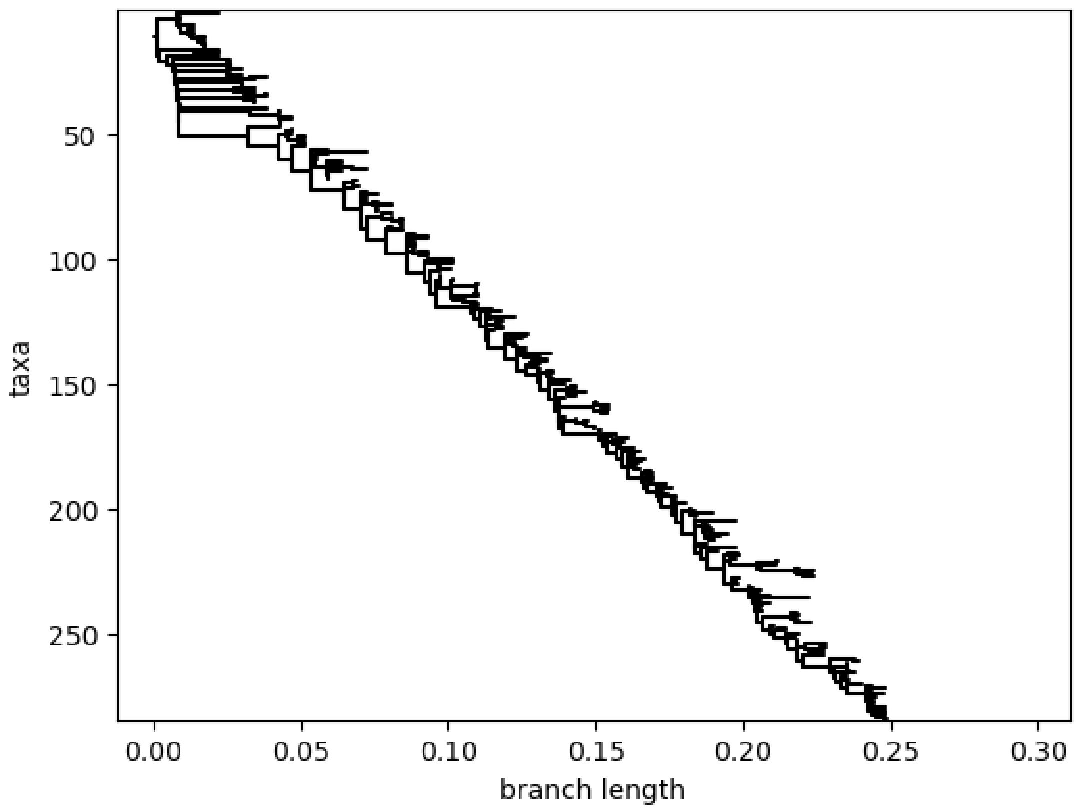


From this presentation

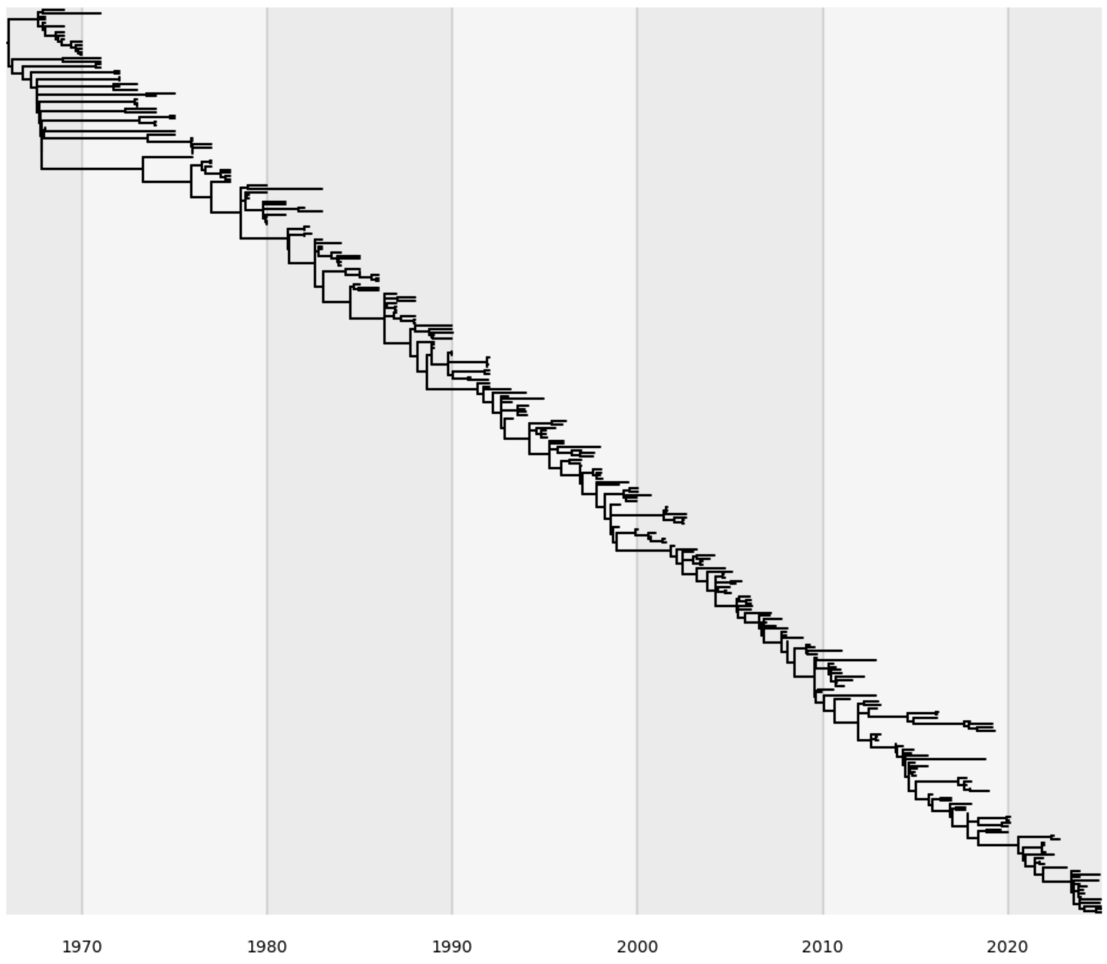
2.3.3 Application to our data

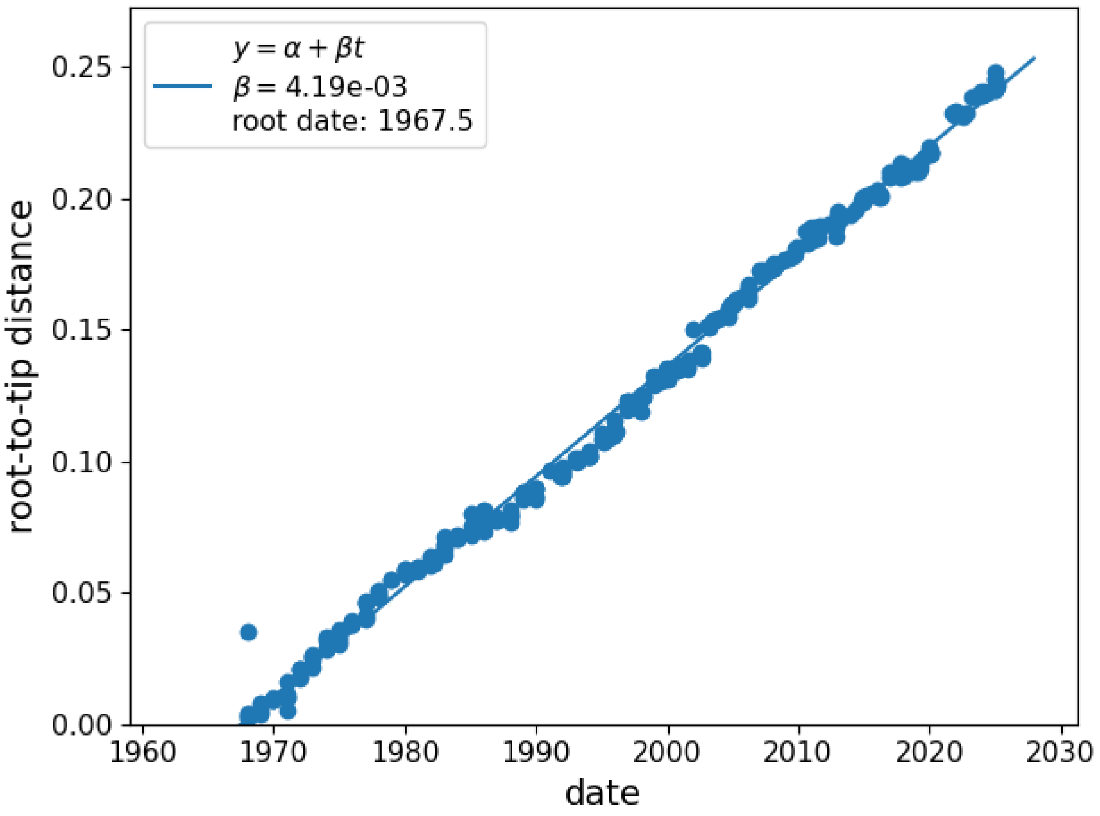
We can apply this approach to our data, and use *TreeTime* to reconstruct a time-scaled phylogeny, and estimate the root position that best fits the sampling times.

```
Attempting to parse dates…
        Using column 'accession' as name. This needs match the taxon names in
the tree!!
        Using column 'date' as date.


1.09    TreeTime.reroot: with method or node: least-squares

1.09    TreeTime.reroot: rerooting will ignore covariance and shared ancestry.

1.41    ###TreeTime.run: INITIAL ROUND

4.60    TreeTime.reroot: with method or node: least-squares

4.60    TreeTime.reroot: rerooting will ignore covariance and shared ancestry.

4.76    ###TreeTime.run: rerunning timetree after rerooting

8.11    ###TreeTime.run: ITERATION 1 out of 2 iterations

8.11    DEPRECATION WARNING. TreeTime.resolve_polytomies: You are resolving
        polytomies using the old 'greedy' mode. This is not well suited for
large
        polytomies. Stochastic resolution will become the default in future
        versions. To switch now, rerun with the flag `--stochastic-resolve`. To
        keep using the greedy method in the future, run with `--greedy-resolve`

11.93   ###TreeTime.run: ITERATION 2 out of 2 iterations
```



```
clock rate: 0.00419
```


1970
1980
1990
2000
2010
2020



$y = \alpha + \beta t$
$\beta = 4.19\text{e-}03$
root date: 1967.5
root-to-tip distance
0.00
0.05
0.10
0.15
0.20
0.25
date
1960
1970
1980
1990
2000
2010
2020
2030