**Course: Advanced Bioinformatics**

**Module title: Phylogenetic Tree IO**

**Module no. : 204**

**Phylogenetics with Bio.Phylo**

The Bio.Phylo module was introduced in Biopython 1.54. Following the lead of SeqIO and AlignIO, it aims to provide a common way to work with phylogenetic trees independently of the source data format, as well as a consistent API for I/O operations.

**Demo: What’s in a Tree?**

Create a simple Newick file named simple.dnd using your favorite text editor, or use [simple.dnd](http://biopython.org/SRC/biopython/Doc/examples/simple.dnd) provided with the Biopython source code:

(((A,B),(C,D)),(E,F,G));

This tree has no branch lengths, only a topology and labeled terminals. (If you have a real tree file available, you can follow this demo using that instead.)

Launch the Python interpreter of your choice:

% ipython -pylab

For interactive work, launching the IPython interpreter with the -pylab flag enables **matplotlib** integration, so graphics will pop up automatically. We’ll use that during this demo.

Now, within Python, read the tree file, giving the file name and the name of the format.

>>> from Bio import Phylo

>>> tree = Phylo.read("simple.dnd", "newick")

Printing the tree object as a string gives us a look at the entire object hierarchy.

>>> print(tree)

Tree(rooted=False, weight=1.0)

Clade()

Clade()

Clade()

Clade(name='A')

Clade(name='B')

Clade()

Clade(name='C')

Clade(name='D')

Clade()

Clade(name='E')

Clade(name='F')

Clade(name='G')

The Tree object contains global information about the tree, such as whether it’s rooted or unrooted. It has one root clade, and under that, it’s nested lists of clades all the way down to the tips.

The function draw\_ascii creates a simple ASCII-art (plain text) dendrogram. This is a convenient visualization for interactive exploration, in case better graphical tools aren’t available.

>>> from Bio import Phylo

>>> tree = Phylo.read("simple.dnd", "newick")

>>> Phylo.draw\_ascii(tree)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ A

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ B

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ C

| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

\_| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ D

|

| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ E

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ F

|

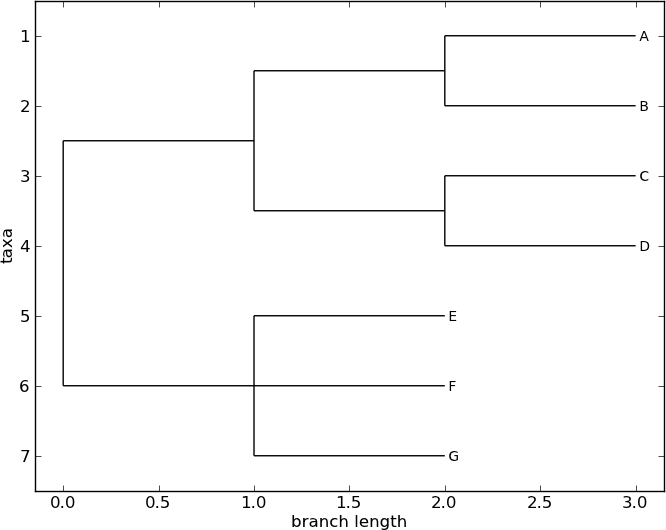
|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ G

<BLANKLINE>

If you have **matplotlib** or **pylab** installed, you can create a graphic using the draw function:

>>> tree.rooted = True

>>> Phylo.draw(tree)



**Coloring branches within a tree**

The functions draw and draw\_graphviz support the display of different colors and branch widths in a tree. As of Biopython 1.59, the color and width attributes are available on the basic Clade object and there’s nothing extra required to use them. Both attributes refer to the branch leading the given clade, and apply recursively, so all descendent branches will also inherit the assigned width and color values during display.

In earlier versions of Biopython, these were special features of PhyloXML trees, and using the attributes required first converting the tree to a subclass of the basic tree object called Phylogeny, from the Bio.Phylo.PhyloXML module.

In Biopython 1.55 and later, this is a convenient tree method:

>>> tree = tree.as\_phyloxml()

In Biopython 1.54, you can accomplish the same thing with one extra import:

>>> from Bio.Phylo.PhyloXML import Phylogeny

>>> tree = Phylogeny.from\_tree(tree)

Note that the file formats Newick and Nexus don’t support branch colors or widths, so if you use these attributes in Bio.Phylo, you will only be able to save the values in PhyloXML format. (You can still save a tree as Newick or Nexus, but the color and width values will be skipped in the output file.)

Now we can begin assigning colors. First, we’ll color the root clade gray. We can do that by assigning the 24-bit color value as an RGB triple, an HTML-style hex string, or the name of one of the predefined colors.

>>> tree.root.color = (128, 128, 128)

Or:

>>> tree.root.color = "#808080"

Or:

>>> tree.root.color = "gray"

Colors for a clade are treated as cascading down through the entire clade, so when we colorize the root here, it turns the whole tree gray. We can override that by assigning a different color lower down on the tree.

Let’s target the most recent common ancestor (MRCA) of the nodes named “E” and “F”. The common\_ancestor method returns a reference to that clade in the original tree, so when we color that clade “salmon”, the color will show up in the original tree.

>>> mrca = tree.common\_ancestor({"name": "E"}, {"name": "F"})

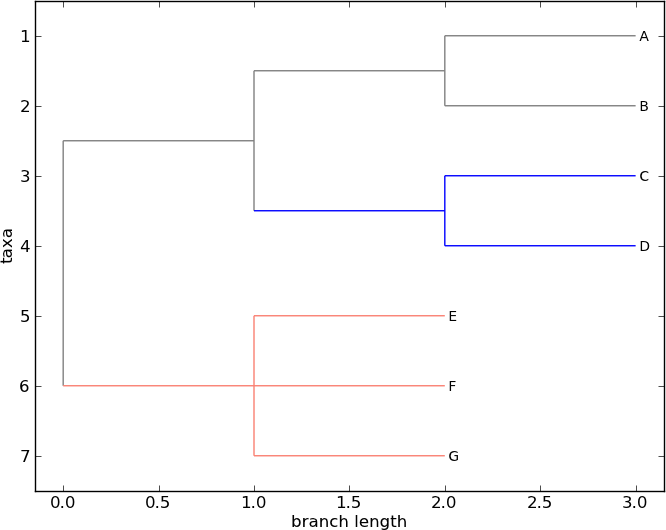
>>> mrca.color = "salmon"

If we happened to know exactly where a certain clade is in the tree, in terms of nested list entries, we can jump directly to that position in the tree by indexing it. Here, the index [0,1] refers to the second child of the first child of the root.

>>> tree.clade[0, 1].color = "blue"

Finally, show our work:

>>> Phylo.draw(tree)



Note that a clade’s color includes the branch leading to that clade, as well as its descendents. The common ancestor of E and F turns out to be just under the root, and with this coloring we can see exactly where the root of the tree is.

My, we’ve accomplished a lot! Let’s take a break here and save our work. Call the write function with a file name or handle — here we use standard output, to see what would be written — and the format phyloxml. PhyloXML saves the colors we assigned, so you can open this phyloXML file in another tree viewer like Archaeopteryx, and the colors will show up there, too.

>>> import sys

>>> Phylo.write(tree, sys.stdout, "phyloxml")

<phy:phyloxml xmlns:phy="http://www.phyloxml.org">

<phy:phylogeny rooted="true">

<phy:clade>

<phy:branch\_length>1.0</phy:branch\_length>

<phy:color>

<phy:red>128</phy:red>

<phy:green>128</phy:green>

<phy:blue>128</phy:blue>

</phy:color>

<phy:clade>

<phy:branch\_length>1.0</phy:branch\_length>

<phy:clade>

<phy:branch\_length>1.0</phy:branch\_length>

<phy:clade>

<phy:name>A</phy:name>

...

The rest of this chapter covers the core functionality of Bio.Phylo in greater detail. For more examples of using Bio.Phylo, see the cookbook page on Biopython.org:

<http://biopython.org/wiki/Phylo_cookbook>

**12.2  I/O functions**

Like SeqIO and AlignIO, Phylo handles file input and output through four functions: parse, read, write and convert, all of which support the tree file formats Newick, NEXUS, phyloXML and NeXML, as well as the Comparative Data Analysis Ontology (CDAO).

The read function parses a single tree in the given file and returns it. Careful; it will raise an error if the file contains more than one tree, or no trees.

>>> from Bio import Phylo

>>> tree = Phylo.read("Tests/Nexus/int\_node\_labels.nwk", "newick")

>>> print(tree)

(Example files are available in the Tests/Nexus/ and Tests/PhyloXML/ directories of the Biopython distribution.)

To handle multiple (or an unknown number of) trees, use the parse function iterates through each of the trees in the given file:

>>> trees = Phylo.parse("../../Tests/PhyloXML/phyloxml\_examples.xml", "phyloxml")

>>> for tree in trees:

... print(tree)

Write a tree or iterable of trees back to file with the write function:

>>> trees = list(Phylo.parse("../../Tests/PhyloXML/phyloxml\_examples.xml", "phyloxml"))

>>> tree1 = trees[0]

>>> others = trees[1:]

>>> Phylo.write(tree1, "tree1.nwk", "newick")

1

>>> Phylo.write(others, "other\_trees.nwk", "newick")

12

Convert files between any of the supported formats with the convert function:

>>> Phylo.convert("tree1.nwk", "newick", "tree1.xml", "nexml")

1

>>> Phylo.convert("other\_trees.xml", "phyloxml", "other\_trees.nex", "nexus")

12

To use strings as input or output instead of actual files, use StringIO as you would with SeqIO and AlignIO:

>>> from Bio import Phylo

>>> from StringIO import StringIO

>>> handle = StringIO("(((A,B),(C,D)),(E,F,G));")

>>> tree = Phylo.read(handle, "newick")

**View and export trees**

The simplest way to get an overview of a Tree object is to print it:

>>> from Bio import Phylo

>>> tree = Phylo.read("PhyloXML/example.xml", "phyloxml")

>>> print(tree)

Phylogeny(description='phyloXML allows to use either a "branch\_length" attribute...', name='example from Prof. Joe Felsenstein's book "Inferring Phyl...', rooted=True)

Clade()

Clade(branch\_length=0.06)

Clade(branch\_length=0.102, name='A')

Clade(branch\_length=0.23, name='B')

Clade(branch\_length=0.4, name='C')

This is essentially an outline of the object hierarchy Biopython uses to represent a tree. But more likely, you’d want to see a drawing of the tree. There are three functions to do this.

As we saw in the demo, draw\_ascii prints an ascii-art drawing of the tree (a rooted phylogram) to standard output, or an open file handle if given. Not all of the available information about the tree is shown, but it provides a way to quickly view the tree without relying on any external dependencies.

>>> tree = Phylo.read("example.xml", "phyloxml")

>>> Phylo.draw\_ascii(tree)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ A

\_\_\_\_\_\_\_\_\_\_|

\_| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ B

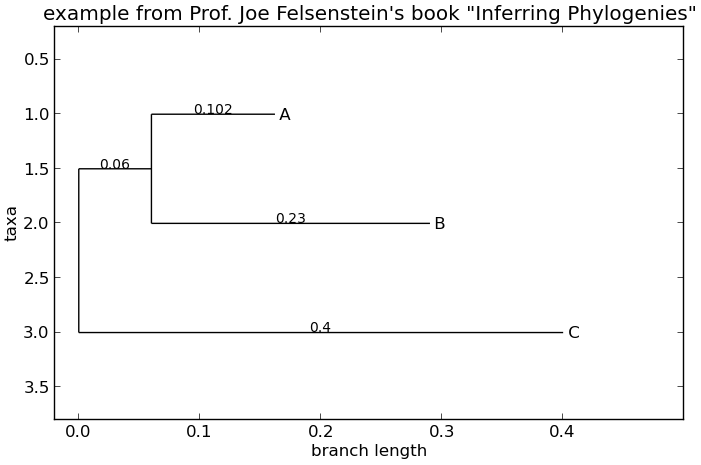
|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ C

The draw function draws a more attractive image using the matplotlib library. See the API documentation for details on the arguments it accepts to customize the output.

>>> tree = Phylo.read("example.xml", "phyloxml")

>>> Phylo.draw(tree, branch\_labels=lambda c: c.branch\_length)



draw\_graphviz draws an unrooted cladogram, but requires that you have Graphviz, PyDot or PyGraphviz, NetworkX, and matplotlib (or pylab) installed. Using the same example as above, and the dot program included with Graphviz, let’s draw a rooted tree:

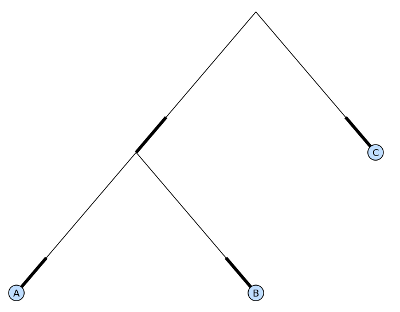
>>> tree = Phylo.read("example.xml", "phyloxml")

>>> Phylo.draw\_graphviz(tree, prog='dot')

>>> import pylab

>>> pylab.show() # Displays the tree in an interactive viewer

>>> pylab.savefig('phylo-dot.png') # Creates a PNG file of the same graphic



(Tip: If you execute IPython with the -pylab option, calling draw\_graphviz causes the matplotlib viewer to launch automatically without manually calling show().)

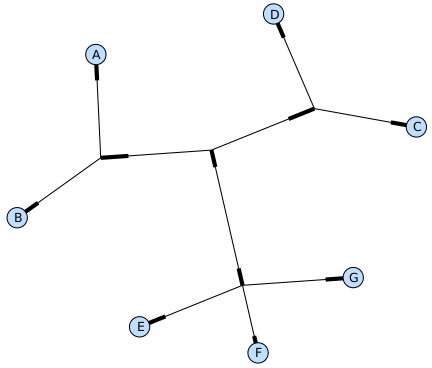
This exports the tree object to a NetworkX graph, uses Graphviz to lay out the nodes, and displays it using matplotlib. There are a number of keyword arguments that can modify the resulting diagram, including most of those accepted by the NetworkX functions networkx.draw andnetworkx.draw\_graphviz.

The display is also affected by the rooted attribute of the given tree object. Rooted trees are shown with a “head” on each branch indicating direction:

>>> tree = Phylo.read("simple.dnd", "newick")

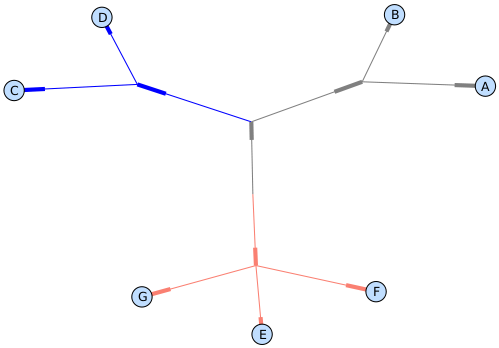
>>> tree.rooted = True

>>> Phylo.draw\_graphviz(tree)



The “prog” argument specifies the Graphviz engine used for layout. The default, twopi, behaves well for any size tree, reliably avoiding crossed branches. The neato program may draw more attractive moderately-sized trees, but sometimes will cross branches. The dot program may be useful with small trees, but tends to do surprising things with the layout of larger trees.

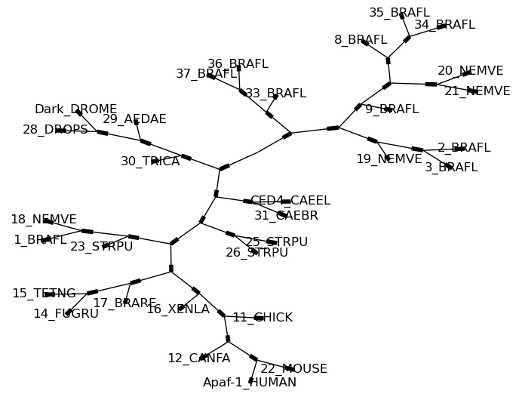
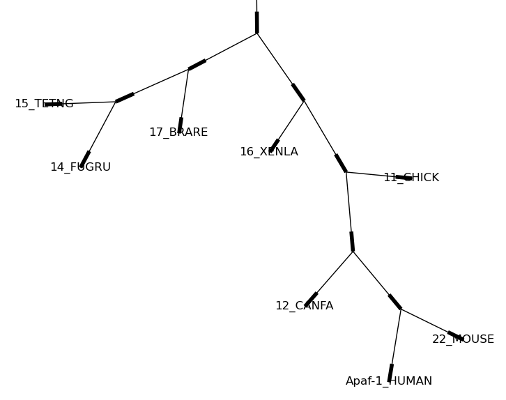
>>> Phylo.draw\_graphviz(tree, prog="neato")



This viewing mode is particularly handy for exploring larger trees, because the matplotlib viewer can zoom in on a selected region, thinning out a cluttered graphic.

>>> tree = Phylo.read("apaf.xml", "phyloxml")

>>> Phylo.draw\_graphviz(tree, prog="neato", node\_size=0)

Note that branch lengths are not displayed accurately, because Graphviz ignores them when creating the node layouts. The branch lengths are retained when exporting a tree as a NetworkX graph object (to\_networkx), however.

See the Phylo page on the Biopython wiki (<http://biopython.org/wiki/Phylo>) for descriptions and examples of the more advanced functionality in draw\_ascii, draw\_graphviz and to\_networkx.