PieTree drawing phylogenetic trees and ancestral state reconstructions

# PieTree

***Release 0.4***

**Emma E. Goldberg**

November 09, 2011

## Contents

# 1 Preliminaries

## 1.1 Introduction

### Purpose

*PieTree* is a program for drawing pretty pictures of phylogenetic trees. Its main purpose is displaying discrete character states, including ancestral state reconstructions. Its name comes from using a pie chart to show the probability of a node being in the various possible states.

*PieTree* doesn't do any kind of analysis—it just produces images. Carrying out ancestral state reconstruction in another program and assembling the results for use with *PieTree* is illustrated in *Creating a Tree & States Data File*.

### History

The first incarnation of *PieTree* was written in C by Walter Brisken in 2007. In 2008, I rewrote it all in Python (as an excuse to practice Python and learn about the Cairo graphics library) and gave it a more useable interface. Features have been gradually accumulating since then.

### Feedback

Please send bug reports, feature requests, and any other comments to me at eeg@uic.edu.

### License

*PieTree* is released under the GNU General Public License.

## 1.2 Installation

### Dependencies

*PieTree* requires:

- The Python programming language, version 2.5.x, 2.6.x, or 2.7.x
- The Cairo graphics library, version 1.4.x or greater

- The pycairo Python module for cairo
- The Python module argparse
- Some other Python modules that seem to be standard on all installations.

When the dependencies are satisfied, this should work without errors:

```
$ python
Python 2.6.5
>>> import cairo
>>> import argparse
```

#### Linux

On Ubuntu:

```
$ sudo apt-get install python-cairo
$ sudo easy_install argparse
```

On Gentoo or Fedora, the package is called `pycairo` instead.

#### Mac OS X

These instructions are courtesy of Lesley Lancaster, for Leopard 10.5.7 in Aug 2009 (and corrected by a MacPorts manager).

- Install Xcode developer tools for Mac
- Install MacPorts
- Install `py25-cairo` using MacPorts:

  ```
  $ sudo port install py25-cairo
  ```

#### Windows

It shouldn't be hard, but I haven't tried.

### PieTree itself

The latest version of *PieTree* is available from http://www.uic.edu/~eeg/code.html.

The program to run is `PieTree/src/PieTree.py`. How to add this to your path is a matter of personal taste. Here are examples of two possibilities:

```
$ ln -s /home/emma/PieTree/src/PieTree.py /home/emma/bin/PieTree
# or
$ alias PieTree="/home/emma/PieTree/src/PieTree.py"
```

The rest of this documentation assumes that running simply `PieTree` calls the program.

## 2 Usage

When running *PieTree*, you provide it with

- an input data file, containing the Newick-formatted tree and a list of the tip and node states
- optionally (but likely), some options for formatting the image output

If no input is provided, a usage message will be printed summarizing the available options:

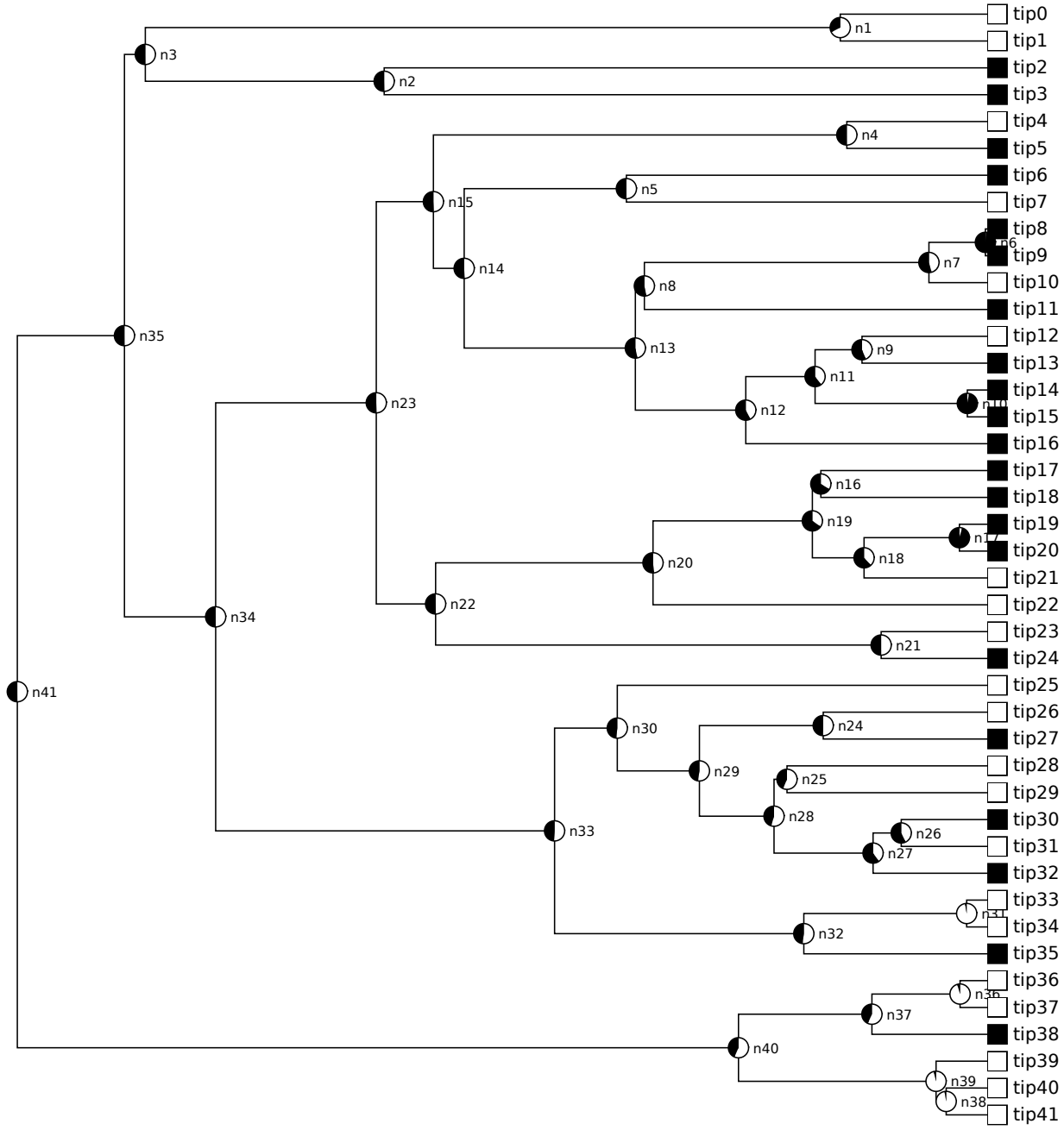
```
$ PieTree
usage: PieTree.py [-h] [--version] [--treefile TREEFILE] [--optfile OPTFILE]
                  [--outformat {pdf,ps,svg,png}] [--shape {rect,radial}]
                  ... and a bunch more options
```

### 2.1 Quick start

A sample tree file with a binary character, called `tree2.ttn`, is included. To use it (adjust paths as necessary):

```
$ PieTree --treefile PieTree/examples/tree2.ttn
created pietree.pdf
```

The result, in the file `pietree.pdf`, looks like this:




## 2.2 Input tree file

The tree data file contains the tree structure itself and the character states of its tips and nodes.

- Tree structure: A Newick string. If node states will be used, nodes as well as tips should be labeled in this string. This line must come before the character state lines.
- Tip states: One line for each tip, giving its name and the number corresponding to its character state.
- Node states: One line for each node, giving its name and the probability of it being in each possible

state.

- Comments: Lines beginning with # will be ignored.

This is best illustrated with a small example:

```
# the Newick tree string, with tips and nodes labeled
((tipA:1, tipB:1)node1:2, tipC:3)node2;
# tip states, either 0 or 1 for a binary trait
tipA   1
tipB   1
tipC   0
# node states, with one value for each of the two possible states
node1  0.2  0.8
node2  0.5  0.5
```

The tree string must be specified first. The order of the lines for the tip and node states doesn't matter. The tip and node names must not contain spaces, and they must match those in the tree string. The names and states must be separated by whitespace (of any kind).

I use the extension `.ttn` for files formatting like this, which stands for "trees, tips, nodes". Such files can easily be constructed by hand or by script. I've included an example of how to perform ancestral state reconstructions and generate a `.ttn` file, using the R package diversitree:

- *Creating a Tree & States Data File*

## 2.3 Specifying options

Plotting options may be specified on either the command line or in a separate file. Elsewhere is a *List of All Options*. The example here will use the options `treefile` (the name of the input tree and character state file), `shape` (for either a rectangular or circular tree image), and `colorX` (the color of state X, where X is 0, 1, 2, etc.).

### Command line

On the command line, the name of an option is preceded with `--`, and its value comes immediately after it, separated by a space or by =. Extending the example above:

```
$ PieTree --treefile PieTree/examples/tree2.ttn --shape=radial
```

The color of each state is specified as an RGB triple, where each component is given a value between 0 and 1. On the command line, this string must be surrounded by quotes. For more details, see *Note on RGB colors*. For example, to make state 0 be red:

```
$ PieTree --treefile PieTree/examples/tree2.ttn --color0 "(1, 0, 0)"
```

### Option file

When specifying a lot of options, it can be easier to put them all into a separate file. In such a file, the first line must be `[pietree]`, and then each option is on its own line, with the option name separated from its

value by either = or :. For example:

```
[pietree]
treefile = PieTree/examples/tree2.ttn
shape = radial
color0 = (1, 0, 0)
color1 = (0, 0.75, 0.75)
```

Suppose that file is called `opt.pie`. Then to use it:

```
$ PieTree --optfile opts.pie
```

When the same argument is provided on both the command line and in the options file, the command line value takes precedence. For example, with the options file above, this produces a rectangular tree:

```
$ PieTree --optfile opts.pie --shape rect
```

### More options

There are many more options for tweaking the look of a tree:

- *List of All Options*
- *Gallery of Examples*

## 3 Creating a Tree & States Data File

*PieTree* does not perform ancestral state reconstructions itself. So if node states are to be estimated and displayed, that analysis must be done elsewhere.

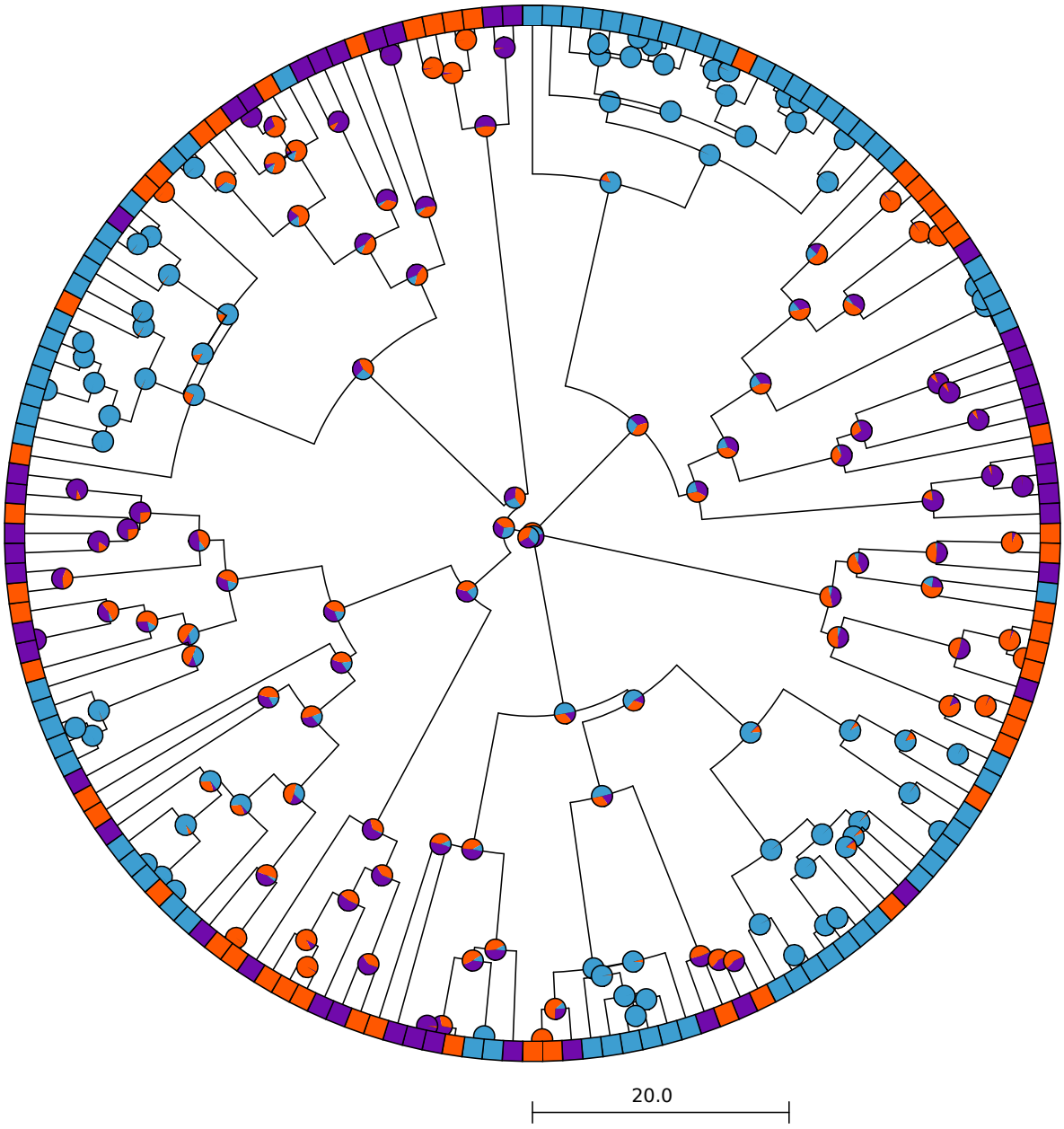
Here is an example of such an analysis, using the R package diversitree. The results are written to a file formatted for use by *PieTree*. (That file format is described here: *Input tree file*)

```
library(diversitree)
set.seed(2)

### Simulate a tree with three character states.  (See the diversitree tutorial.)

# lambda = speciation, mu = extinction, q = character state transition
# lambda1 lambda2 lambda3 mu1 mu2 mu3 q12 q13 q21 q23 q31 q32
pars <- c(0.1, 0.15, 0.2, 0.03, 0.045, 0.06, 0.05, 0, 0.05, 0.05, 0, 0.05)

phy <- tree.musse(pars, max.t = 50, x0 = 1)
states <- phy$tip.state

### Alternatively, you could read in your real data with ape commands.
### If your tree lacks node labels, use something like:
###    phy$node.label <- paste("node", seq(Nnode(phy)), sep="")

### Estimate MuSSE rates.  Note: this is an illustration, not a thorough analysis!

lik <- make.musse(phy, states, k = 3)
```

```
p <- starting.point.musse(phy, 3)
rates <- coef(find.mle(lik, p))

### Perform ancestral state reconstruction.

anc <- asr.marginal(lik, rates)
colnames(anc) <- phy$node.label

### Write the results to a .ttn file.  Note that the state numbers must start from 0.
### tree string, tip states, node state estimates

outfile <- "tree3.ttn"
write.tree(phy, file = outfile)
write.table(states-1, file = outfile, col.names = F, quote = F, append = T)
write.table(t(anc), file = outfile, col.names = F, quote = F, append = T)
```

Take a look at the results:

```
$ PieTree --treefile tree3.ttn --shape radial --color0 "(0.44,0.04,0.67)" \
  --color1 "(1,0.34,0)" --color2 "(0.24,0.62,0.82)" --tipnamesize 0 --nodenamesize 0 \
  --scalebar 20
```




It's of course possible to produce a similar image directly in R. I don't think it looks quite as nice, however, and I find it much more frustrating to tweak things like spacing there.

## 4 List of All Options

Recall that option names must be preceded with `--` when used on the command line. Alternatively, they can be placed in a file whose first line is `[pietree]`. See *Specifying options*.

### 4.1 Input and output logistics

**treefile** Name of file containing tree and tip states. See *Input tree file*.

**optfile** Name of file containing formatting options. See *Specifying options*.

**outfile** Name for the output file. If it doesn't have a suffix (like `.pdf`), an appropriate one will be appended. [default is `pietree`]

**outformat** File format of the output image. If an `outfile` with a suffix is also specified, `outformat` takes precedence and an appropriate suffix will be appended.

= `pdf` Adobe's format [the default]

= `eps` (encapsulated) postscript

= `svg` scalable vector graphics (works with Inkscape and Illustrator)

= `png` portable network graphics (lossless bitmap)

**width** Width of the canvas. The image is scaled horizontally to match this width.

= any positive number [default is 800]

**height** Height of the canvas. Doesn't affect scaling of the image.

= any positive number

[default is determined by tree size for `shape = rect` or is `width` for `shape = radial`]

**xmargin** Margin size on the left and right of the image. For `shape = radial`, these are based on the longest tip name.

= any positive number [default is 20]

**ymargin** Margin size on the top and bottom of the image.

= any positive number [default is 10]

### 4.2 Graphical elements

**shape** Shape in which the tree is drawn.

= `rect` rectangular, with all the tips on the right [the default]

= `radial` radial, with the tips along the outside of a circle

**scalebar** Draw a bar showing the time scale of the plot.

= `no` no scale bar is drawn [the default]

= `yes` a scale bar of some auto-calculated length is drawn

= `X` a scale bar of length `X` is drawn (replace `X` with a number, obviously)

**colorX** Color representing state `X`. Replace `X` with `0`, `1`, etc.

= `"(R, G, B)"` see *Note on RGB colors*

[defaults: `color0 = "(1, 1, 1)"` (white), `color1 = "(0, 0, 0)"` (black), no defaults for additional states]

**`pieradius`** Radius of the pie chart showing the node reconstruction.

= any positive number [default is 7]

**`boxsize`** Height of the box showing the tip state.

= any positive number [default is 1.9 * `pieradius`]

**`rimthick`** Thickness of the lines around the pies and boxes.

= any positive number [default is 2]

**`linethick`** Thickness of the branch lines.

= any positive number [default is 1]

**`linecolor`** Color of the branches of the tree and other lines.

= `"(R, G, B)"` see *Note on RGB colors* [default is black]

**`backcolor`** Color of the image background.

= `"(R, G, B)"` see *Note on RGB colors* [default is transparent for `outformat = png`, white otherwise]

**`tipspacing`** Space between tip markers.

= any positive number [default is 1.5 * `boxsize`]

### 4.3 Text formatting

**`tipnamesize`** Font size of the tip labels.

= any positive number [default is `boxsize`]

**`nodenamesize`** Font size of the internal node labels.

= any positive number [default is 0.75 * `tipnamesize`]

**`italic`** Whether to italicize the text

= `yes` italics

= `no` normal upright [the default]

**`serif`** Whether to use a serif font.

= `yes` serif font

= `no` sans-serif font [the default]

**`underscorespace`** Whether to replace _ with a space in tip and node names.

= `yes` do replace underscores

= `no` don't replace underscores [the default]

**textcolor** Color of tip and node labels

= "(R, G, B)" see *Note on RGB colors* [default is black]

**tipnamestatecolor** Whether to color the tip text label according to the tip state.

= yes take tip label color from tip state

= no use textcolor for tip label [the default]

### 4.4 Note on RGB colors

R, G, and B are the values for the red, green, and blue components of the color. Each takes a value between 0 and 1. These three numbers must be separated with commas and the whole thing surrounded with parentheses. On the command line, put quote marks outside the parentheses. For example, color0 = (0.24, 0.62, 0.82) in an option file, and --color0 = "(0.24, 0.62, 0.82)" on the command line.

## 5 Gallery of Examples

The files needed for each example here are included in PieTree/examples/.

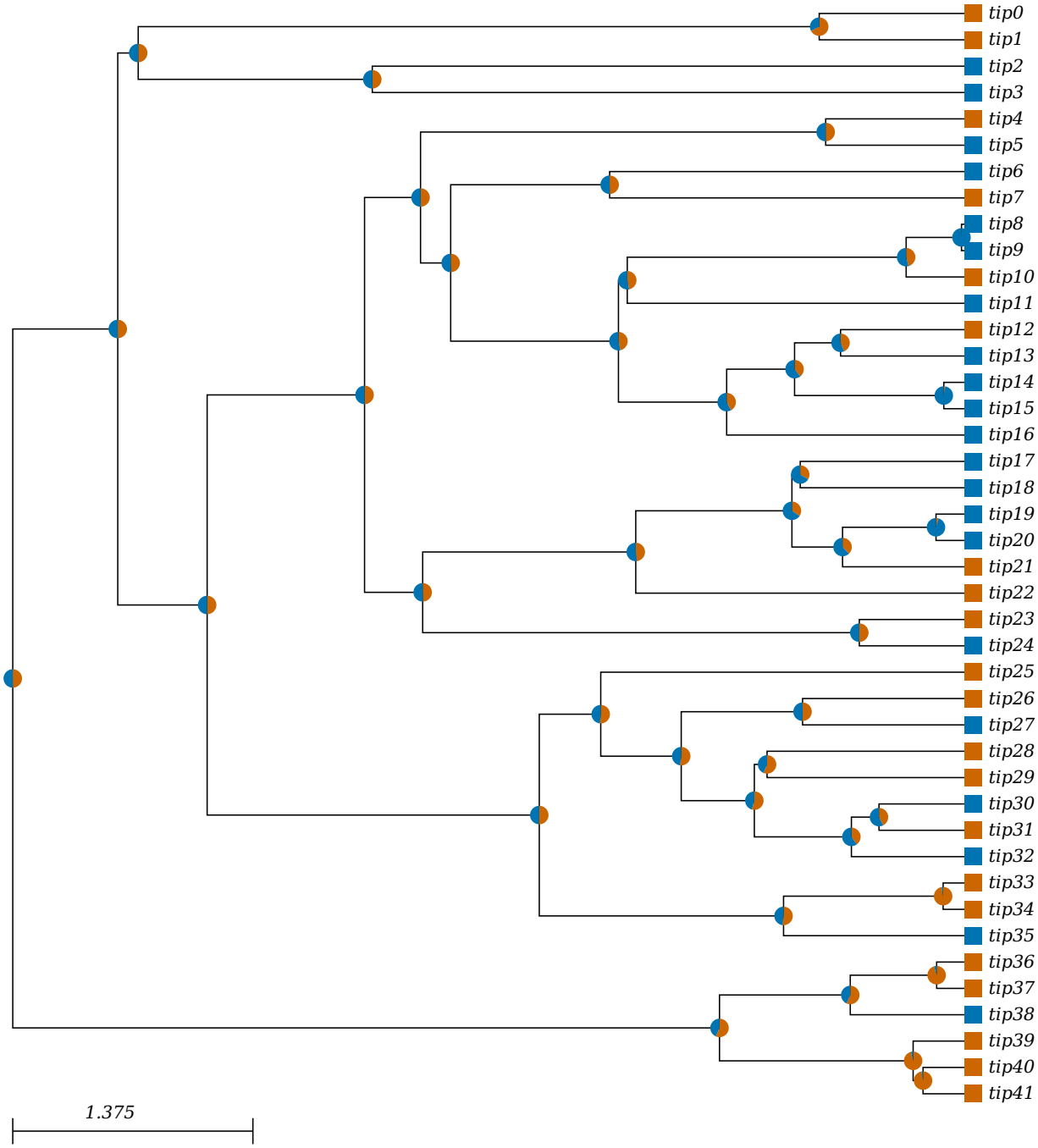
Each example below is generated with a command like this:

```
$ PieTree --treefile tree2.ttn --optfile opt1.pie
```

(or substitute the other treefile or a different optfile).

### 5.1 Example 1

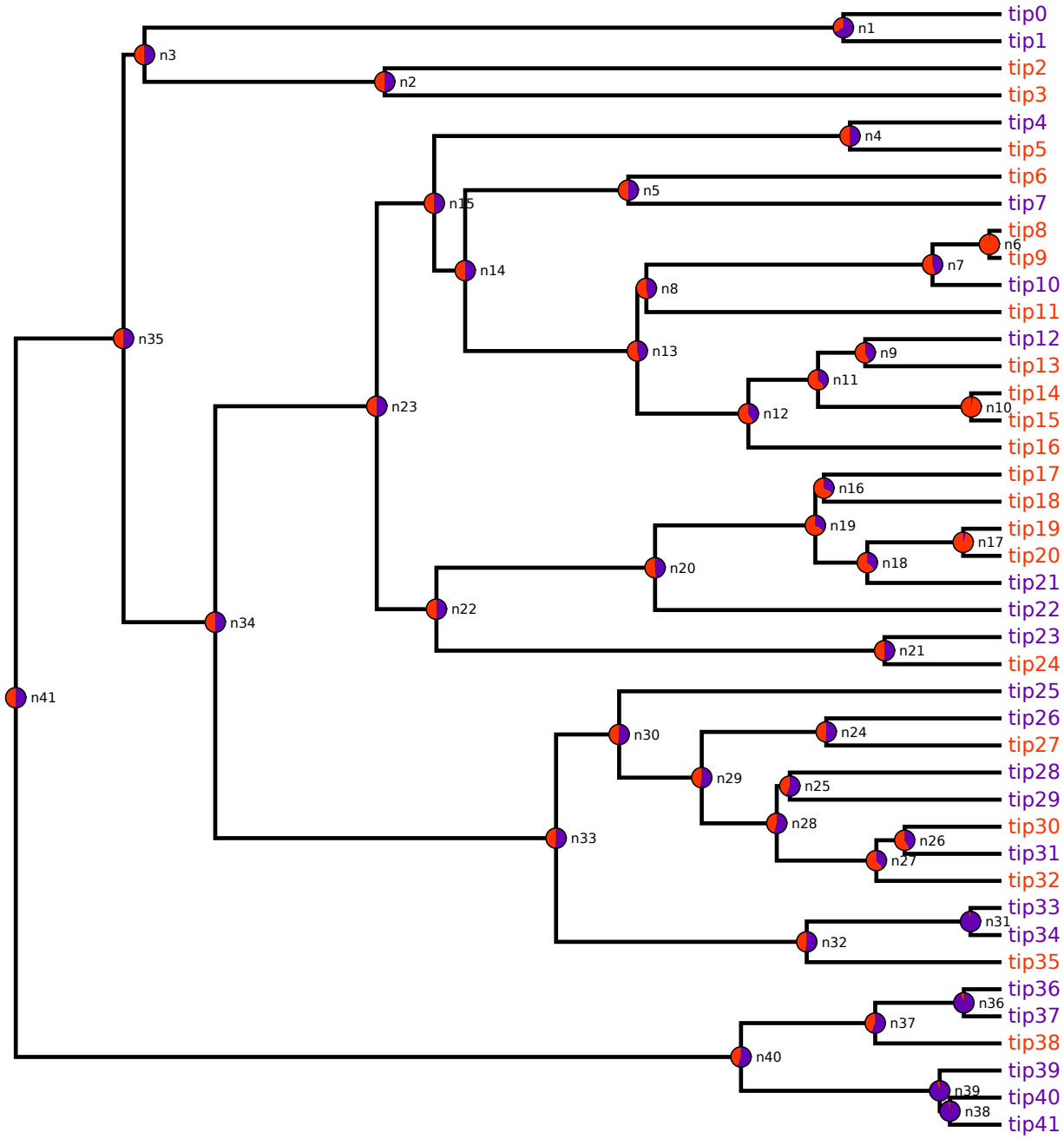
```
[pietree]
outfile = example1
pieradius = 7
nodenamesize = 0
italic = yes
serif = yes
color1 = (0, 0.45, 0.70)
color0 = (0.8, 0.4, 0)
rimthick = 0
scalebar = yes
```




## 5.2 Example 2

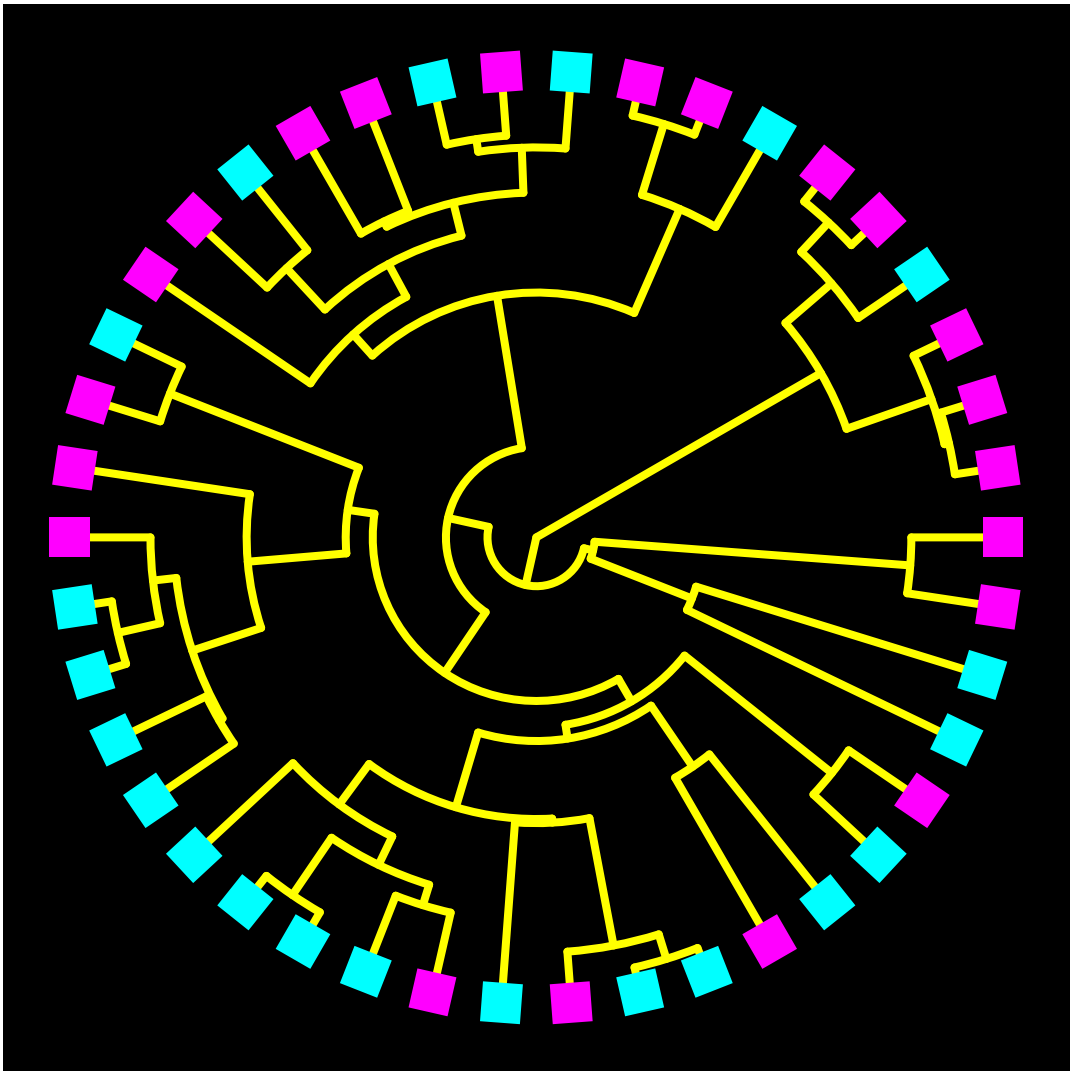
```
[pietree]
outfile = example2
shape = rect
boxsize = 0
tipspacing = 20
```

```
tipnamesize = 15
nodenamesize = 10
color1 = (1, 0.2, 0)
color0 = (0.4, 0, 0.7)
tipnamestatecolor = yes
linethick = 3
```



### 5.3 Example 3

```
[pietree]
outfile = example3
shape = radial
backcolor = (0, 0, 0)
linecolor = (1, 1, 0)
color0 = (1, 0, 1)
color1 = (0, 1, 1)
rimthick = 0
linethick = 3
pieradius = 0
boxsize = 15
nodenamesize = 0
tipnamesize = 0
width = 400
```



### 5.4 Example 4

```
[pietree]
outfile = example4
shape = radial
tipnamesize = 0
nodenamesize = 0
color0 = (0.44, 0.04, 0.67)
color1 = (1, 0.34, 0)
color2 = (0.24, 0.62, 0.82)
scalebar = 20
```