# **PieTree** Manual
version 0.3.1

Emma Goldberg
`eeg@umd.edu`

August 21, 2009

## Contents

## Background

### ***Purpose***

`PieTree` is a program for drawing pretty pictures of phylogenetic trees, particularly for the purpose of displaying ancestral state reconstructions of discrete characters. It doesn't do any kind of analysis—it just produces images.

### ***History***

The first incarnation of `PieTree` (v0.1) was written in C by Walter Brisken in December, 2007. In April, 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 (v0.2). I added radial plotting capabilities a year later (v0.3). Beta-testers would be very welcome!

## Installation

Sorry these instructions are so bad now. I really don't know what will be required in various cases. If you get it working, please tell me what you did so I can pass the info along.

### ***Dependencies***

To use `PieTree`, you will need the Cairo graphics library (`cairographics.org`; version 1.4.x), the Python programming language (`python.org`; version 2.5.x), the Python package for Cairo, and the Python module `configparse` (`gustaebel.de/lars/configparse/`).

Good places to start are `cairographics.org/download` and `python.org/download`. I have included a copy of `configparse.py` with my `PieTree` files.

#### *Linux*

On Ubuntu, it should be sufficient to run

```
sudo apt-get install python-cairo
```

which will pull in `libcairo2` as a dependency. You might also need `libcairo2-dev`.

On Gentoo, try

```
emerge pycairo
```

On Fedora, try

```
sudo yum install pycairo
```

#### *Mac OS X*

These instructions are courtesy of Lesley Lancaster, for Leopard 10.5.7 in Aug 2009.

1. Install developer tools for Mac: `http://developer.apple.com/technology/Xcode.html`
2. Install MacPorts: `http://www.macports.org/`
3. Install Darwin Ports: `http://darwinports.com/`
4. See `http://py25-cairo.darwinports.com/`. In a terminal window, type: `sudo port install py25-cairo`
5. Follow additional instructions that appear in terminal window to make Python 2.5 the default version.

#### *Windows*

I'm sure it's possible, but I haven't tried.

### *PieTree itself*

Once you've gotten to the point where you can start Python and execute `import cairo` and `import optparse` (I think that's a good test. . . ), download `PieTree` from `www.biology.ucsd.edu/∼goldberg/code/code.html`. In the `src/` directory are a few `.py` files, including the main executable `PieTree.py`.

Remember where you put this whole directory—you will either need to add it to your path or include the full path when you execute `PieTree`. In this document, I'll write it as `/path/to/PieTree.py`. I found it useful to create the file shown below to add `PieTree` to my Python path:

File contents: `/usr/lib/python2.5/site-packages/PieTree.pth`

```
/home/emma/src/PieTree-0.3.1/src/
```

## Usage

### *Quick start*

`PieTree-0.3.1/examples/` contains a sample tree file called `example.ttn`. If you execute

```
/path/to/PieTree.py example.ttn
```

you should see a note saying

```
created pietree.pdf
```

### *Input tree file*

You will need to create a plain text file containing your tree and character states. It's a very simple format: first a Newick string for the tree, including tip and node labels and branch lengths, and then a list of the tip/node labels and their corresponding character state values, one per line.

A minimalist example is

File contents: `minimal.ttn`

```
# this is a comment
((tipA:1, tipB:1)node1:2, tipC:3)node2;
tipA 1
tipB 0
tipC 1
node1 0.5
node2 0.8
```

and look at the included `example.ttn` for a larger example.

I give my tree files the suffix `.ttn`, which stands for "trees, tips, nodes," but you can call yours whatever you want.

The character states must be 0 and 1 (only binary characters for now). For a node with an uncertain state, give the proportion of the weight given to state 1; i.e. `node2` above was reconstructed as 20% state 0 and 80% state 1.

Note that all tips and nodes must be labeled in the tree string, and those labels must match the ones given in the list of character states. If there's a label mismatch or omission, you should get a warning message and/or have a missing "pie" in the output figure. The order of the lines in the character state list is not important. Blank lines and lines beginning with `#` are ignored.

### ***Image options***

There are lots of options for tweaking the image that's produced. To see a help message with a summary of them all, just type

```
/path/to/PieTree.py
```

#### *Specifying options*

As an example, say you want to increase the size of the "pies" and add some color.

One way to do this is with command line options:

```
/path/to/PieTree.py --pieradius=9 --color1='(0, 0.5, 0.7)' example.ttn
```

Note that you need quote marks (either single or double) around the color value because it contains special characters (punctuation, since it's an RGB triplet). I used = signs in the example above, but you can replace them with spaces instead, as I show below. If you do use = signs, there can't be any spaces around them.

Another method is to create a file containing your desired options, e.g.

File contents: `opts.pie`

```
pieradius = 9
color1 = '(0, 0.5, 0.7)'
outfile = prettytree.pdf
```

and then to specify that file on the command line:

```
/path/to/PieTree.py --opt opts.pie example.ttn
```

In option files, the = sign is required but spaces around it are fine. Color strings still need to be surrounded by quotes. Filenames with hyphens also need quotes around them, e.g. `outfile = "pretty-tree.pdf"`.

If an option is given in both the input file and on the command line, the command line value takes effect.

```
/path/to/PieTree.py --opt opts.pie --color1 '(0.4, 0.8, 0)' example.ttn
```

*Note that the final argument must always be your input tree file.*

All the other options, detailed below, can be specified in the same manner.

#### *All the options*

Here are all the possible options, their allowed values, and their default values. Again, to get a quick (and possibly more up-to-date) summary, type:

```
/path/to/PieTree.py
```

`opt` A file containing your desired option values. Use one line per value, each in the form option = value.

`outfile` A name for the output file. If it doesn't have a suffix (like `.pdf`), an appropriate one will be appended. [default is `pietree`] `outformat` The file format of the output image. If you also specify an `outfile` with a suffix, `outformat` takes precedence and an appropriate suffix will be appended.

- `= pdf` Adobe's format
- `= ps` (encapsulated) postscript
- `= svg` scalable vector graphics
- `= png` portable network graphics (lossless bitmap)

`shape` The 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 (note that `pdf` and `svg` output looks best for this)

`pieradius` The radius of the pie chart showing the node reconstruction.

- `=` any number $\geq 0$ [default is 7]

`boxsize` The height of the box showing the tip state.

- `=` any number $\geq 0$ (default is $1.9 *$ `pieradius`)

`tipspacing` The distance between tip box centers.

- `=` any number $\geq 0$ (default is $1.5 *$ `boxsize`)

`tipnamesize` The font size of the tip labels.

- `=` any number $\geq 0$ (default is `boxsize`)

`nodenamesize` The font size of the internal node labels.

- `=` any number $\geq 0$ (default is $0.75 *$ `tipnamesize`)

`italic` Whether to italicize the text.

- `= yes` italics
- `= no` normal upright (default)

`serif` Whether to use a serif font.

- `= yes` serif font
- `= no` sans-serif font (default)

`color0` The color for state 0.

- `= (R, G, B)` `R`, `G`, and `B` are the values for the red, green, and blue components of the color (each between 0 and 1). These three numbers must be separated with commas and the whole thing surrounded with parentheses and quote marks. [default is white = '(1, 1, 1)']

`color1` The color for state 1.

- `= (R, G, B)` [default is black = '(0, 0, 0)']

`textcolor` The color of tip and node labels.

- `= (R, G, B)` (default is black)

`tipnamestatecolor` Whether to color the tip text according to the tip state.

- `= yes` base tip name color on state
- `= no` use `textcolor` for tip names (default)

`linecolor` The color of the branches of the tree and other lines.

- `= (R, G, B)` (default is black)

`backcolor` The color of the background.

- `= (R, G, B)` (default is transparent for PNG, white otherwise)

`rimthick` The thickness of the lines around the pies and boxes.

- `=` any number $\geq 0$ (default is 2)

`linethick` The thickness of the branch lines.

- `=` any number $\geq 0$ (default is 1)

`width` The width of the canvas; determines the horizontal scaling of the image.

- `=` any number $\geq 0$ (default is 800)

`height` The height of the canvas; doesn't affect the scaling of the image.

- `=` any number $\geq 0$ (default is determined by the tree size for `shape = rect` or is `width` for `shape = radial`)

`xmargin` The margin size on the left and right of the image.

| | |
|---|---|
| `=` | any number $\geq 0$ (default is 20) |
| `ymargin` | The margin size on the top and bottom of the image. |
| `=` | any number $\geq 0$ (default is 10) |

For `shape = radial`, the left and right margins are based on the longest tip name and the top and bottom margins are best controlled with `height`.

## Procedural stuff

### *Bugs and help*

I've done my best to make this code correct and robust, but bugs happen. If you encounter a bug, either an unexplainable crash or a result that seems incorrect, please let me know. You can send just a good description of the problem with enough information for me to reproduce it, or a patch if you have one.

I hope this code will be useful for other people, so if you have read the instructions here but are still having problems getting things working, just let me know and I will try to help. I would also be happy to hear if you have usability suggestions (e.g. making the documentation clearer or error-checking during use of the program), ideas for additional features that would be useful, or if you are making substantial changes to take the code in a new direction. Improvements to the Installation section would be very much appreciated. And of course I would be especially glad to hear if the program just worked for your purposes!

### *License*

This software is issued under the Gnu Public License (`http://www.gnu.org/licenses/gpl.html`). The full text of the license is included with the code, but the gist of it is (1) you can do whatever you want with this code and modify it however you like, and (2) if you redistribute this code or your modified version, it must also be under the GPL.