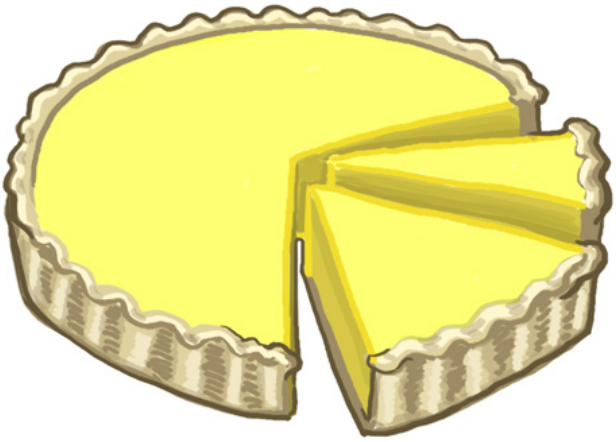
PartitionFinder v1.1.0
and
PartitionFinderProtein v1.1.0

Manual

Rob Lanfear, August 2011
Last updated May 2013

Icon © Ainsley Seago. Thanks Ainsley!



Questions, suggestions, problems, bugs? Search or post on the discussion group at:
http://groups.google.com/group/partitionfinder

Step-by-step tutorial: http://www.robertlanfear.com/partitionfinder/tutorial/

News: http://www.robertlanfear.com/partitionfinder/news/

FAQs: http://www.robertlanfear.com/partitionfinder/faq/

Citations

If you use PartitionFinder or PartitionFinderProtein at all:
Lanfear R, Calcott B, Ho SYW, Guindon S (2012). PartitionFinder: combined selection of partitioning schemes and substitution models for phylogenetic analyses. *Molecular Biology and Evolution* 29 (6): 1695-1701. http://dx.doi.org/10.1093/molbev/mss020

If you use the RAxML version, or the ‘rcluster’ or ‘hcluster’ search options:
Lanfear, Calcott, Kainer, Mayer, and Statmatakis In prep. Selecting optimal partitioning schemes for phylogenomic datasets: a comparison of clustering methods.

Disclaimer

What PartitionFinder and PartitionFinderProtein are for

PartitionFinder and PartitionFinderProtein are programs for selecting best-fit partitioning schemes and models of molecular evolution for nucleotide and amino acid alignments, respectively. The user provides an alignment with some pre-defined data blocks (e.g. 9 data blocks defining the 1st, 2nd and 3rd codon positions of 3 protein-coding genes, see Figure 1). The programs then find the best partitioning scheme for this dataset, at the same time as selecting best-fit substitution models for each subset of sites. Here are a few things you can do with the programs:

1. Find the best-fit partitioning scheme for a given nucleotide or amino acid dataset
2. Compare any number of user-defined partitioning schemes
3. Find best-fit models of molecular evolution for each subset in any partitioned dataset (much like you might do with ModelTest or ProtTest).

The idea is that finding best-fit partitioning schemes and models of molecular evolution will improve any downstream analyses of your data, like estimating phylogenetic trees or molecular dates. All of those kinds of analyses assume that your model of evolution is correct, and PartitionFinder helps make the model as good as it can be.

PartitionFinder and PartitionFinderProtein come in a single download from www.robertlanfear.com/partitionfinder, and are designed to take the hard work out of comparing partitioning schemes, and to help find a scheme that maximises the fit of the data to the model, without including more parameters than are necessary. Both programs implement three information-theoretic measures for comparing models of molecular evolution and partitioning schemes: the Akaike Information Criterion (AIC), the corrected Akaike Information Criterion (AICc), and the Bayesian Information Criterion (BIC). At the end of a run, you are given output files that tell you the best partitioning scheme, along with the best-fit model of molecular evolution for each subset (sometimes called a ‘partition’, but that term is a bit misleading) in that scheme. So you can then move straight on to your phylogenetic analyses.

Operating systems (Mac, Windows and Linux work)

Mac OSX and Windows are supported. For Linux, details are provided in the FAQs on the website: http://www.robertlanfear.com/partitionfinder/faq/

QuickStart – simple use cases

For a small multilocus dataset (e.g. ~10 loci) use a greedy search with PhyML:

1. Define data blocks by gene and codon position
2. In the .cfg file, set the following options:

```
branchlengths = linked;
models = all;
model_selection = bic;
search=greedy;
```

3. Run PartitionFinder from the commandline as follows:

```
python “<PartitionFinder.py>” “<InputFoldername>”
```

For a larger dataset (e.g. ~100 loci) use a greedy search with RAxML:

1. Define data blocks by gene and codon position
2. Set the .cfg file options as above.
3. Run PartitionFinder using RAxML, as follows:

```
python “<PartitionFinder.py>” “<InputFoldername>” --raxml
```

For a really big dataset (e.g. ~1000 loci) use clustering algorithms with RAxML:

1. Define data blocks by gene and codon position
2. Set the .cfg file options as above, except:

```
search=rcluster;
```

3. Run PartitionFinder using RAxML, as follows:

```
python “<PartitionFinder.py>” “<InputFoldername>” --raxml
```

This implements the relaxed clustering algorithm described in Lanfear et al 2013 (in prep: Lanfear, Calcott, Kainer, Mayer, and Statmatakis “Selecting optimal partitioning schemes for phylogenomic datasets: a comparison of clustering methods.”). The default is to check the top 10% of schemes, based on those that are expected to give the biggest improvements. If that’s still too slow on your huge dataset, reduce that to 1% or 0.1% of schemes like this:

```
python “<PartitionFinder.py>” “<InputFoldername>” –raxml --rcluster-percent 0.1
```

4. If the relaxed clustering algorithm is still too slow, use the strict hierarchical clustering algorithm by setting this option in the .cfg file:

```
search=hcluster;
```

This implements the strict clustering algorithm described in Lanfear et al 2013 (in prep: Lanfear, Calcott, Kainer, Mayer, and Statmatakis “Selecting optimal partitioning schemes for phylogenomic datasets: a comparison of clustering methods.”).

Overview

Partitioning involves splitting sites in your alignment into sets that have evolved under similar models. For example, if you have a 3 gene dataset you might suspect that each of the three genes has been evolving differently – perhaps they come from different chromosomes, or have experienced different evolutionary constraints. Furthermore, you might think that each codon position within each gene has been evolving differently – different codon positions tend to evolve at different rates, and experience different substitutional processes thanks to the triplet structure of the genetic code. Because of this, you might split your datfa into 9 sets of sites for this alignment – one for each codon position in each gene. But is this too many different sets? Perhaps it would be better to join together the 1st and 2nd codon sites of each gene, so defining 6 sets of sites. Or perhaps it would be better to forget the divisions between genes, and define only 2 sets of sites – 1st and 2nd codon sites versus 3rd codon sites. The trouble is that if you start with 9 possible sets of sites, there are a lot of different possible partitioning schemes you might consider, 21147 in fact. This creates a problem – how do we find the best scheme from that many schemes?

PartitionFinder solves this problem by quickly and efficiently comparing all of these schemes. All you need to do is define your 9 possible sets of sites (i.e. the largest number of sets of sites you think is sensible to define) as data blocks, and PartitionFinder will do the rest. At the end of a run you are told which partitioining scheme is the best, and also which model of molecular evolution you should use for each subset of sites in that scheme (i.e., you don’t have to use ModelTest or ProtTest or similar programs on your partitioned dataset, PartitionFinder does all of this model selection for you at the same time as finding a partitioning scheme). You can then go straight on to performing your phylogenetic analysis, without any additional model-testing or comparisons of partitioning schemes.

If you don’t want to compare all possible schemes (which can be almost impossible for large datasets), you can define exactly the schemes you do want to compare (see `search = user`, below), or use a heuristic search algorithm to find a good scheme (see `search = greedy, search = rcluster, search = hcluster` below). You can also tell the PartitionFinder programs exactly which models of molecular evolution to consider (see `models`, below). And you can define how they compare partitioning schemes and models (see `model_selection`, below). PartitionFinder uses a number of methods to speed up partitioning scheme comparison and model selection, such as running on multiple processors when they’re available.

Running PartitionFinder on Macs

Installing Python on Macs (most Macs already have it)

If you have mac OSX Lion (i.e. OSX 10.7) or later, you already have Python 2.7 installed, so ignore the rest of this section. If you don't have Lion, you need to make sure you have Python 2.7 or later installed (but avoid installing Python 3.0 or above). Installing Python is really easy, if you already know what version of OSX you have, just go to this link and click the appropriate installer: http://www.python.org/getit/ .

If you don't know what version of OSX you have, click the apple symbol at the top left of your screen and then click 'About This Mac'. A window will come up, and under the picture of the apple is your version number (something like this: 10.6.6).

If you have version 10.6 or above, use this link to get Python 2.7:
http://www.python.org/ftp/python/2.7.2/python-2.7.2-macosx10.6.dmg

If you have anything before 10.6 (i.e. 10.5 or lower), use this link:
http://www.python.org/ftp/python/2.7.2/python-2.7.2-macosx10.3.dmg

Installing PartitionFinder on Macs

1. Download the latest version of PartitionFinder from www.robertlanfear.com/partitionfinder, PartitionFinderProtein is included with this download
2. Double-click the .zip file, and it will automatically unzip. You will get a folder called something like 'PartitionFinder1.0.0' (depending on the version you have.
3. Move it to wherever you want to store the PartitionFinder program

Running PartitionFinder and PartitionFinderProtein on Macs

These instructions describe how to run the 'example/nucleotide' analysis using PartitionFinder. To run PartitionFinderProtein, just follow these instructions but replace 'PartitionFinder' with 'PartitionFinderProtein' in step 2, and 'example/nucleotide' with 'example/aminoacid'.

1. Open Terminal (on most Macs, this is found in Applications/Utilities)
2. In the Terminal, you need to tell the computer where to find PartititionFinder, and where to find your input files. The easiest way to do this is as follows:
    a. Type "python" followed by a space
    b. Drag and drop the "PartitionFinder.py" file (which is in the PartitionFinder folder you just unzipped) onto the command prompt. The path to 'PartitionFinder.py' will be added automatically.
    c. Type another space
    d. Drag and drop the blue 'example/nucleotide' folder (in the PartitionFinder folder) onto the command prompt
3. Hit Enter/Return to run PartitionFinder

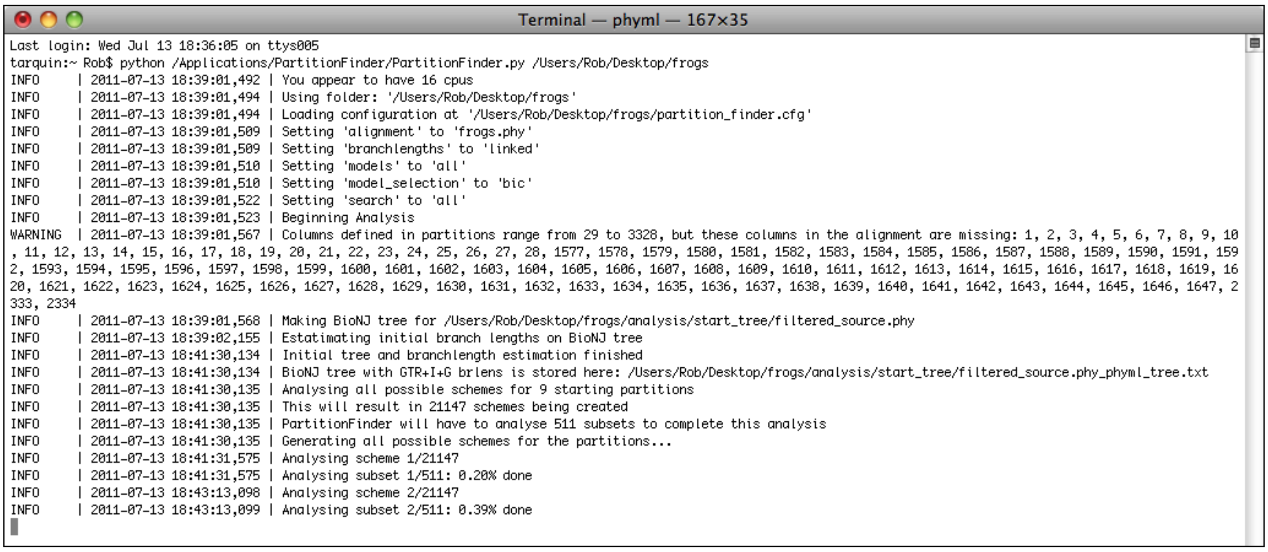
That's it!

More generally, you run PartitionFinder by typing a command line that looks like this:

```
python "<PartitionFinder.py>" "<InputFoldername>"
```

Where `<PartitionFinder.py>` is the full path to the PartitionFinder.py (or PartitionFinderProtein.py) file, and `<InputFoldername>` is the full path to your input folder, which should contain an alignment and a .cfg file. Note that the input folder can be anywhere on your computer, it doesn't have to be in the PartitionFinder folder like the example file.

Once PartitionFinder is running, it will keep you updated about its progress. If it hits a problem, it will (hopefully) provide you with a useful error message that will help you correct that problem. Hopefully, you won't have too many problems and your terminal screen will look something like that shown below.



```
Last login: Wed Jul 13 18:36:05 on ttys005
tarquin:~ Rob$ python /Applications/PartitionFinder/PartitionFinder.py /Users/Rob/Desktop/frogs
INFO     | 2011-07-13 18:39:01,492 | You appear to have 16 cpus
INFO     | 2011-07-13 18:39:01,494 | Using folder: '/Users/Rob/Desktop/frogs'
INFO     | 2011-07-13 18:39:01,494 | Loading configuration at '/Users/Rob/Desktop/frogs/partition_finder.cfg'
INFO     | 2011-07-13 18:39:01,509 | Setting 'alignment' to 'frogs.phy'
INFO     | 2011-07-13 18:39:01,509 | Setting 'branchlengths' to 'linked'
INFO     | 2011-07-13 18:39:01,510 | Setting 'models' to 'all'
INFO     | 2011-07-13 18:39:01,510 | Setting 'model_selection' to 'bic'
INFO     | 2011-07-13 18:39:01,522 | Setting 'search' to 'all'
INFO     | 2011-07-13 18:39:01,523 | Beginning Analysis
WARNING  | 2011-07-13 18:39:01,567 | Columns defined in partitions range from 29 to 3328, but these columns in the alignment are missing: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 159
2, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 16
20, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 2
333, 2334
INFO     | 2011-07-13 18:39:01,568 | Making BioNJ tree for /Users/Rob/Desktop/frogs/analysis/start_tree/filtered_source.phy
INFO     | 2011-07-13 18:39:02,155 | Estatimating initial branch lengths on BioNJ tree
INFO     | 2011-07-13 18:41:30,134 | Initial tree and branchlength estimation finished
INFO     | 2011-07-13 18:41:30,134 | BioNJ tree with GTR+I+G brlens is stored here: /Users/Rob/Desktop/frogs/analysis/start_tree/filtered_source.phy_phyml_tree.txt
INFO     | 2011-07-13 18:41:30,135 | Analysing all possible schemes for 9 starting partitions
INFO     | 2011-07-13 18:41:30,135 | This will result in 21147 schemes being created
INFO     | 2011-07-13 18:41:30,135 | PartitionFinder will have to analyse 511 subsets to complete this analysis
INFO     | 2011-07-13 18:41:30,135 | Generating all possible schemes for the partitions...
INFO     | 2011-07-13 18:41:31,575 | Analysing scheme 1/21147
INFO     | 2011-07-13 18:41:31,575 | Analysing subset 1/511: 0.20% done
INFO     | 2011-07-13 18:43:13,098 | Analysing scheme 2/21147
INFO     | 2011-07-13 18:43:13,099 | Analysing subset 2/511: 0.39% done
```

Running PartitionFinder on Windows

Installing Python on Windows

The first thing you'll need to do is install Python. This is simple and takes just a couple of minutes. Download Python from here: http://www.python.org/getit/. Make sure you download version 2.7. The instructions that follow assume you have installed python in its default folder, which is c:\Python27.

Once python is installed you'll need to update your "PATH", so that your computer can find it. To do this, follow these steps:

On Windows 7

1. Select "Computer" from the Start menu
2. Choose "System Properties" from the menu
3. Click "Advanced system settings" (it's on the left) then click the "Advanced" tab
4. Click on "Environment Variables", under "System Variables", find "**Path"**, and click on it.
5. Click "Edit…", and add this text to the end of the **Path** in the box "Variable value". Note that there should be no spaces anywhere at all:
   ;C:\Python27
   then click "OK" and you're done.

Windows XP

1. Click the "Start" menu, then "Control Panel" -> "System" -> "Advanced"
2. Click on "Environment Variables", under "System Variables", find "**Path**", and click on it.
3. Click "Edit…", and add this text to the end of the **Path**. Note that there should be no spaces anywhere at all:
   ;C:\Python27
   then click "OK" and you're done.

Windows Vista

1. Right click "My Computer" icon
2. Choose "Properties" from the menu
3. Click "Advanced" tab (it might also be called "Advanced system settings")
4. Click "Edit…", and add this text to the end of the **Path**. Note that there should be no spaces anywhere at all:
   ;C:\Python27
   then click "OK" and you're done.

Installing PartitionFinder on Windows

1. Download the latest version of PartitionFinder from www.robertlanfear.com/partitionfinder, PartitionFinderProtein is included with this download
2. Unzip the .zip file by right-clicking on the .zip file and choosing 'Extract All'. Inside the unzipped folder, find the folder called something like 'PartitionFinder1.0.0' (the numbers will depend on the version you have).
3. Move this folder to wherever you want to store the PartitionFinder program.

Running PartitionFinder and PartitionFinderProtein on Windows

These instructions describe how to run the ‘example/nucleotide’ analysis using PartitionFinder. To run PartitionFinderProtein, just follow these instructions but replace ‘PartitionFinder’ with ‘PartitionFinderProtein’ in step 2, and ‘example/nucleotide’ with ‘example/aminoacid’.

1. Open a command prompt. To do this, click on the Start Menu, then navigate to the command prompt like this: “All Programs” -> “Accessories” -> “Command Prompt”. On **Windows 7** you can just type “cmd” into the search box area, and you’ll see it.
2. In the command prompt, you need to tell the computer where to find PartititionFinder, and where to find your input files. The easiest way to do this is as follows:
    a. Type “python“ followed by a space
    b. Drag and drop the “PartitionFinder.py” file (which is in the PartitionFinder folder you just unzipped) onto the command prompt. The path to ‘PartitionFinder.py’ will be added automatically.
    c. Type another space
    d. Drag and drop the blue ‘example/nucleotide’ (in the PartitionFinder folder) onto the command prompt
3. Hit Enter/Return to run PartitionFinder

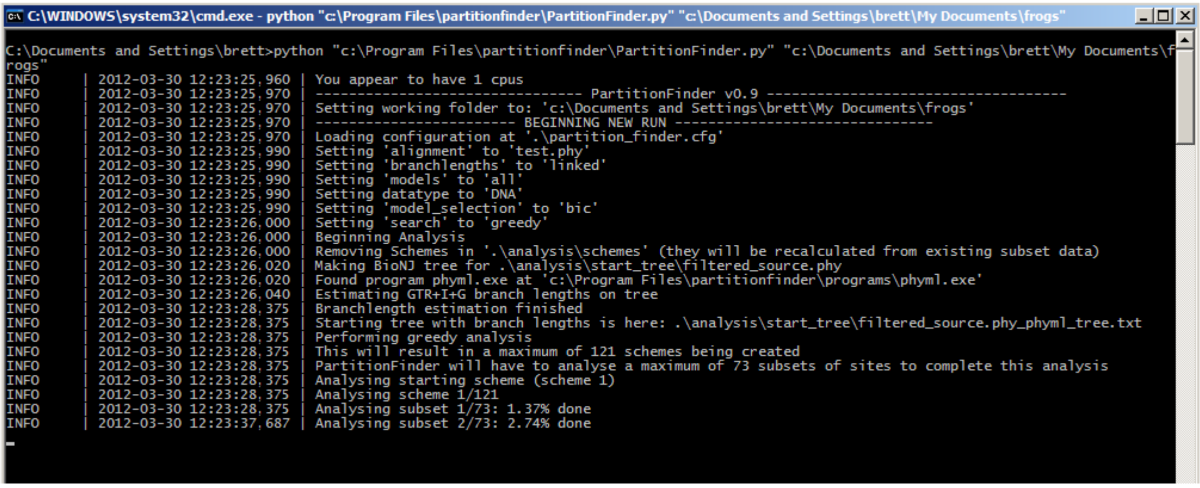
That’s it!

More generally, you run PartitionFinder by typing a command line that looks like this:

```
python “<PartitionFinder.py>” “<InputFoldername>”
```

Where `<PartitionFinder.py>` is the full path to the PartitionFinder.py (or PartitionFinderProtein.py) file, and `<InputFoldername>` is the full path to your input folder, which should contain an alignment and a .cfg file. Note that the input folder can be anywhere on your computer, it doesn’t have to be in the PartitionFinder folder like the example file.

Once PartitionFinder is running, it will keep you updated about its progress. If it hits a problem, it will (hopefully) provide you with a useful error message that will help you correct that problem. Hopefully, you won’t have too many problems and your terminal screen will look something like that shown below.


C:\WINDOWS\system32\cmd.exe - python "c:\Program Files\partitionfinder\PartitionFinder.py" "c:\Documents and Settings\brett\My Documents\frogs"

```
C:\Documents and Settings\brett>python "c:\Program Files\partitionfinder\PartitionFinder.py" "c:\Documents and Settings\brett\My Documents\f
rogs"
INFO     | 2012-03-30 12:23:25,960 | You appear to have 1 cpus
INFO     | 2012-03-30 12:23:25,970 | -------------------------------- PartitionFinder v0.9 ------------------------------------
INFO     | 2012-03-30 12:23:25,970 | Setting working folder to: 'c:\Documents and Settings\brett\My Documents\frogs'
INFO     | 2012-03-30 12:23:25,970 | ------------------------ BEGINNING NEW RUN -------------------------------
INFO     | 2012-03-30 12:23:25,970 | Loading configuration at '.\partition_finder.cfg'
INFO     | 2012-03-30 12:23:25,990 | Setting 'alignment' to 'test.phy'
INFO     | 2012-03-30 12:23:25,990 | Setting 'branchlengths' to 'linked'
INFO     | 2012-03-30 12:23:25,990 | Setting 'models' to 'all'
INFO     | 2012-03-30 12:23:25,990 | Setting datatype to 'DNA'
INFO     | 2012-03-30 12:23:25,990 | Setting 'model_selection' to 'bic'
INFO     | 2012-03-30 12:23:26,000 | Setting 'search' to 'greedy'
INFO     | 2012-03-30 12:23:26,000 | Beginning Analysis
INFO     | 2012-03-30 12:23:26,000 | Removing Schemes in '.\analysis\schemes' (they will be recalculated from existing subset data)
INFO     | 2012-03-30 12:23:26,020 | Making BioNJ tree for .\analysis\start_tree\filtered_source.phy
INFO     | 2012-03-30 12:23:26,020 | Found program phyml.exe at 'c:\Program Files\partitionfinder\programs\phyml.exe'
INFO     | 2012-03-30 12:23:26,040 | Estimating GTR+I+G branch lengths on tree
INFO     | 2012-03-30 12:23:28,375 | Branchlength estimation finished
INFO     | 2012-03-30 12:23:28,375 | Starting tree with branch lengths is here: .\analysis\start_tree\filtered_source.phy_phyml_tree.txt
INFO     | 2012-03-30 12:23:28,375 | Performing greedy analysis
INFO     | 2012-03-30 12:23:28,375 | This will result in a maximum of 121 schemes being created
INFO     | 2012-03-30 12:23:28,375 | PartitionFinder will have to analyse a maximum of 73 subsets of sites to complete this analysis
INFO     | 2012-03-30 12:23:28,375 | Analysing starting scheme (scheme 1)
INFO     | 2012-03-30 12:23:28,375 | Analysing scheme 1/121
INFO     | 2012-03-30 12:23:28,375 | Analysing subset 1/73: 1.37% done
INFO     | 2012-03-30 12:23:37,687 | Analysing subset 2/73: 2.74% done
_
```

Using PartitionFinder with RAxML (the --raxml option).

Why bother?

By default, PartitionFinder uses PhyML to estimate likelihoods. As of version 1.1.0 PartitionFinder also includes the option to use RAxML. To do this, add '--raxml' to the end of your command line, e.g.:

```
python “~/Applications/PartitionFinder.py” “~/Desktop/frogs” --raxml
```

There are three main considerations when choosing whether to use the RAxML option:

- RAxML is the only program that will work with very large datasets (e.g. 1000s of genes, 1000s of taxa, or both).
- RAxML includes fewer models of molecular evolution than PhyML.
- Using the RAxML option allows you to use the very fast ‘clustering’ algorithms.

In general, if you need results fast or if you are working with very large datasets, then you should use RAxML.

Advanced tips

1. Download and compile RAxML for your computer

We supply a windows and a mac version of RAxML with PartitionFinder, but for various reasons these won’t be optimised for your computer. In fact, they might not work at all on your computer. This is because RAxML is written so that it compiles differently for different computers. PartitionFinder may run faster if you download and compile the latest version of RAxML by following the instructions here: https://github.com/stamatak/standard-RAxML

Once you have compiled RAxML, rename the executable ‘raxml’ or ‘raxml.exe’ if you’re on windows, and put it in the ‘programs’ folder of PartitionFinder (replace the existing file).

2. Use the clustering algorithms

When you’re using the --raxml option, you can use the clustering algorithms ‘rcluster’ and ‘hcluster’ (see below). These algorithm use default settings to find a good partitioning scheme, but you can alter these settings to try and optimise partitionfinder for your own dataset using the --weights and --rcluster-percent options (see below). As a rule of thumb, it’s far better to run a single ‘rcluster’ search with a higher --rcluster-percent than to spend time trying a lot of different weighting schemes with the ‘—weights’ option (we have a paper in prep on this: Lanfear, Calcott, Kainer, Mayer, and Statmatakis “Selecting optimal partitioning schemes for phylogenomic datasets: a comparison of clustering methods.”).

Input Files

PartitionFinder and PartitionFinderProtein both need two input files, a Phylip alignment and a configuration file. The best way to get a feel for how this works is to have a look in the examples we've provided in the 'example' folder. There is also an online tutorial at www.robertlanfear.com/partitionfinder/tutorial. You can copy and paste these folders onto your desktop (or anywhere) and try running them by following the instructions above. Playing around with the options in the .cfg files give you a good idea of what's possible.

In the rest of this section, we describe in detail exactly what the two input files should look like, and what they do.

Alignment File in phylip format

The phylip format: Your alignment needs to be in Phylip format. We use the same version of Phylip format that PhyML uses, which is described in detail here http://www.atgc-montpellier.fr/phyml/usersguide.php?type=phylip. In brief, this format should contain a line at the top with the number of sequences, followed by the number of sites in the alignment. After that, there should be one sequence on each line, where a sequence contains a name, followed by some whitespace (either spaces or tabs) and the sequence. Names can be up to 100 characters long. There should be nothing else on the line other than the name and the sequence – watch out if you use MacClade, which adds some extra things to the end of each line.

Converting other formats to phylip: If you have an alignment in some other format and want to convert it into phylip format, the best (free!) tool to use is Geneious. Other alignment editors tend to cut the names short in phylip files (the original definition had a 10 character limit on names), but Geneious doesn't. If you don't have Geneious, it's free and you can download it from http://www.geneious.com/. Once you have Geneious, follow these steps to convert your alignment file to phylip:

1. Open up your alignment file in Geneious, and highlight it
2. Go the 'File' menu and click 'Export', then 'Selected documents...'
3. Scroll down the list of options and choose 'Phylip (*.phy)', and click 'OK'.
4. Now a box of options will come up, choose 'Export full length'.
5. Save the phylip alignment file in the same folder as your .cfg file for PartitionFinder.

One more thing: Often, you'll have sites in your alignment that you don't intend to use in your final analysis, or perhaps you have an alignment of mixed data types like DNA, protein, and morphological data. In PartitionFinder and PartitionFinder protein, this is OK. You don't need to make separate alignments of each datatype. You can just ignore the sites you're not interested in by setting the '[data_blocks]' option appropriately, more instructions below.

Configuration File

PartitionFinder programs get all of their information on the analysis you want to do from a configuration file. This file should always be called "partition_finder.cfg", regardless of whether you're using PartitionFInder or PartitionFinderProtein. The best thing to do is to base your own .cfg on the example file provided in the "example" folder. An exhaustive list of everything in that file follows. **Note that all lines in the .cfg file except comments and lines with square brackets have to end with semi-colons.**

In the configuration file, white spaces, blank lines and lines beginning with a "#" (comments) don't matter. You can add or remove these as you wish. All the other lines do matter, and they must all stay in the file in the order they are in below. There is one exception – the user_tree_topology option (see below).

The basic configuration file looks like this:

```
# ALIGNMENT FILE #
alignment = test.phy;

# BRANCHLENGTHS #
branchlengths = linked;

# MODELS OF EVOLUTION #
models = all;
model_selection = bic;

# DATA BLOCKS #
[data_blocks]
Gene1_pos1 = 1-789\3;
Gene1_pos2 = 2-789\3;
Gene1_pos3 = 3-789\3;

# SCHEMES #
[schemes]
search = user;

# user schemes
allsame   = (Gene1_pos1,  Gene1_pos2,  Gene1_pos3);
1_2_3     = (Gene1_pos1) (Gene1_pos2) (Gene1_pos3);
12_3      = (Gene1_pos1,  Gene1_pos2) (Gene1_pos3);
```

The options in the file are described below. Where an option has a limited set of possible commands, they are listed on the same line as the option, separated by vertical bars like this "|". Most of these options don't differ between PartitionFinder and PartitionFinderProtein, the only one that does is the 'models' option. The options have slightly different meanings depending on whether you're using the default version of PartitionFinder (which is based on PhyML) or the RAxML version (which, perhaps obviously, is based on RAxML). Options that differ in the RAxML version are explained at the end of this section.

alignment

The name of your sequence alignment. This file should be in the same folder as the .cfg file.

`branchlengths: linked | unlinked`

This sets how to branch lengths of will be estimated. How you set this will depend to some extent on which program you intend to use for you final phylogenetic analysis. Almost all phylogeny programs support linked branchlengths, but only some support unlinked branchlenghts (e.g. MrBayes, BEAST, and RaxML).

`branchlengths = linked;` only one underlying set of branch lengths is estimated. Each subset has its own scaling parameter (i.e. its own subset-specific rate). This allows subsets to evolve at different rates, but doesn't change the length of any one branch relative to any other. The total number of branch length parameters here is quite small. If there are N species in your dataset, then there are 2N-3 branch lengths in your tree, and each subset after the first one adds an extra scaling parameter. For instance, if you had a scheme with 10 subsets and a dataset with 50 species, you would have 106 branch length parameters.

`branchlengths = unlinked;` each subset has its own independent set of branch lengths. In this case, branch lengths are estimated independently for each subset, so each subset has it's own set of 2N-3 branch length parameters. With this setting, the number of branch length parameters can be quite large (2NS – 3S). So, a scheme with 10 subsets and a dataset with 50 species would have 970 branch length parameters. (

`models (PartitionFinder): all | raxml | mrbayes | beast | <list>`
`(PartitionFinderProtein): all_protein | <list>`

Sets which models of molecular evolution to consider during model selection. PartitionFinder analyses nucleotide sequences, so uses only nucleotide models of evolution, like the GTR and HKY models. PartitionFindeProtein analyses amino acid sequences, so uses only amino acid models, like the WAG and LG models.

PartitionFinder and PartitionFinderProtein perform model selection on each subset in much the same way as other programs like jModelTest, ProtTest, MrModelTest, or ModelGenerator. Your results therefore tell you not only the best partitioning scheme, but also which model of molecular evolution is most appropriate for each subset in that scheme. This means that you don't need to do any further model selection after PartitionFinder is done.

`models = all;` in PartitionFinder, compare 56 models of nucleotide evolution for each subset. These 56 models comprise the 12 most commonly used models of molecular evolution (JC, K80, TrNef, K81, TVMef, TIMef, SYM, F81, HKY, TrN, K81uf, TVM, TIM, and GTR), each of which comes in four flavours: on its own, with invariant sites (+I), with gamma distributed rates across sites (+G), or with both gamma distributed rates and invariant sites (+I+G).

`models = all_protein;` in PartitionFinderProtein, compare 112 models of amino acid evolution for each subset. These 112 models comprise the 14 most commonly used models of protein evolution (LG, WAG, mtREV, Dayhoff, DCMut, JTT, VT, Blosum62, CpREV, RtREV, MtMam, MtArt, HIVb, HIVw), each of which comes in eight flavours: on its own, with invariant sites (+I), with gamma distributed rates across sites (+G), with amino acid frequencies estimated from the data (+F), and with combinations of two or more of these options (+I+G, +G+F, +I+F, +I+G+F).

`models = raxml; models = mrbayes; models = beast;` tells PartitionFinder to use only the nucleotide models available in RaxML, MrBayes3.1.2, or BEAST 5.7.2 (and earlier versions) respectively. This can be particularly useful if you intend to use one of these programs for your phylogenetic analysis, as it restricts the models that are compared to only those that are implemented in the particular programs. This is not only the most appropriate thing to do, but also saves a lot of computational time.

`models = <list>;` This can be any list of models appropriate for the data type. In PartitionFinder this is anything from the Nucleotide Models list (below). In PartitionFinderProtein it is anything from the Amino Acids model list. Each model in the list should be separated by a comma. For example, if I was only interested in a few nucleotide models in PartitionFinder, I might do this:

```
models = JC, JC+G, HKY, HKY+G, GTR, GTR+G;
```

Or, for protein models in PartitionFinderProtein I might do this:

```
models = LG, LG+G, LG+G+F, WAG, WAG+G, WAG+G+F;
```

Note that in this list you can specify either nucleotide models, or amino acid models, but not a mixture of both. If you have a mixed dataset (i.e. some data blocks are amino acid, some are nucleotides, you have to run PartitionFinder on the nucleotide data, then PartitionFinder protein on the amino acid data.

Here are lists of all of the models implemented in PartitionFinder and PartitionFinderProtein. It's easy for us to implement new models, so if you'd like us to do so, please get in touch either by emailing Rob Lanfear, or by posting on the PartitionFinder google group.

Nucelotide Models in PartitionFinder (56 in total)

+I: include a proportion of invariant sites
+G: include gamma distributed rates across sites (with 4 categories)
JC, K80, TrNef, K81, TVMef, TIMef, SYM, F81, HKY, TrN, K81uf, TVM, TIM, GTR, JC+I, K80+I, TrNef+I, K81+I, TVMef+I, TIMef+I, SYM+I, F81+I, HKY+I, TrN+I, K81uf+I, TVM+I, TIM+I, GTR+I, JC+G, K80+G, TrNef+G, K81+G, TVMef+G, TIMef+G, SYM+G, F81+G, HKY+G, TrN+G, K81uf+G, TVM+G, TIM+G, GTR+G, JC+I+G, K80+I+G, TrNef+I+G, K81+I+G, TVMef+I+G, TIMef+I+G, SYM+I+G, F81+I+G, HKY+I+G, TrN+I+G, K81uf+I+G, TVM+I+G, TIM+I+G, GTR+I+G

Amino Acid Models in PartitionFinderProtein (112 in total)

+I: include a proportion of invariant sites
+G: include gamma distributed rates across sites (with 4 categories)
+F: include amino acid frequencies estimated from the alignment
LG, WAG, mtREV, Dayhoff, DCMut, JTT, VT, Blosum62, CpREV, RtREV, MtMam, MtArt, HIVb, HIVw, LG+F, WAG+F, mtREV+F, Dayhoff+F, DCMut+F, JTT+F, VT+F, Blosum62+F, CpREV+F, RtREV+F, MtMam+F, MtArt+F, HIVb+F, HIVw+F, LG+I, WAG+I, mtREV+I, Dayhoff+I, DCMut+I, JTT+I, VT+I, Blosum62+I, CpREV+I, RtREV+I, MtMam+I, MtArt+I, HIVb+I, HIVw+I, LG+G, WAG+G, mtREV+G, Dayhoff+G, DCMut+G, JTT+G, VT+G, Blosum62+G, CpREV+G, RtREV+G, MtMam+G, MtArt+G, HIVb+G, HIVw+G, LG+I+G, WAG+I+G, mtREV+I+G, Dayhoff+I+G, DCMut+I+G, JTT+I+G, VT+I+G, Blosum62+I+G, CpREV+I+G, RtREV+I+G, MtMam+I+G, MtArt+I+G, HIVb+I+G, HIVw+I+G, LG+I+F, WAG+I+F, mtREV+I+F, Dayhoff+I+F, DCMut+I+F,

JTT+I+F, VT+I+F, Blosum62+I+F, CpREV+I+F, RtREV+I+F, MtMam+I+F, MtArt+I+F, HIVb+I+F, HIVw+I+F, LG+G+F, WAG+G+F, mtREV+G+F, Dayhoff+G+F, DCMut+G+F, JTT+G+F, VT+G+F, Blosum62+G+F, CpREV+G+F, RtREV+G+F, MtMam+G+F, MtArt+G+F, HIVb+G+F, HIVw+G+F, LG+I+G+F, WAG+I+G+F, mtREV+I+G+F, Dayhoff+I+G+F, DCMut+I+G+F, JTT+I+G+F, VT+I+G+F, Blosum62+I+G+F, CpREV+I+G+F, RtREV+I+G+F, MtMam+I+G+F, MtArt+I+G+F, HIVb+I+G+F, HIVw+I+G+F

model_selection: AIC | AICc | BIC

Sets which metric to use for model selection. It also defines the metric for comparing partitioning schemes if you use search=greedy (see below).

The AIC, AICc, and BIC are similar in spirit – they all reward models that fit the data better, but penalise models that have more parameters. The idea is include parameters that help the model fit the data more than some specified amount, but to avoid including too many parameters (overparameterisation). The BIC penalises extra parameters the most, followed by the AICc, and then the AIC. Which model_selection approach you use will depend on your preference. There are lots of papers comparing the merits of the different metrics, and based on those papers my own preference is to use the BIC (see especially Minin et al Syst. Biol. 52(5):674–683, 2003; and Adbo et al Mol. Biol. Evol. 22(3):691–703. 2004).

[data_blocks]

On the lines following this statement you define the starting subsets for your analysis (we call these data blocks). Each data block has a name, followed by an "=" and then a description. The description is built up as in most Nexus formats, and tells PartitionFinder which sites of your original alignment correspond to each data block. The best way to understand this it to look at a couple of examples.

Imagine a DNA sequence alignment with 1000bp of protein-coding DNA, followed by 1000bp of intron DNA. Let's imagine that some of the intron was unalignable too, so we don't want that included in our analysis, but we don't want to cut it out of our alignment file. Your data block definitions might look like this:

```
Gene1_codon1 = 1-1000\3;                         ❶
Gene1_codon2 = 2-1000\3;                         ❷
Gene1_codon3 = 3-1000\3;                         ❸
intron       = 1001-1256 1675-2000;              ❹
```

❶-❸ are typical of how you might separate out codon positions for a protein coding gene. The numbers either side of the dash define the first and last sites in the data block, and the number after the backslash defines the spacing of the sites. Every third site will define a codon position, as long as your alignment stays in the same reading frame throughout that gene.

❹ shows how you can include ranges of sites without backslashes, and demonstrates that you can combine more than one range of sites in a single data block. Here, we excluded sites 1257-1674 because they were unalignable.

The total list of data blocks does not have to include all the sites in your original alignment. For instance, you might exclude some sites you're not interested in, or that were unalignable. You'll get a warning from PartitionFinder if all of the sites in the original alignment are not included in the data blocks you've defined. Also, note that data blocks cannot be overlapping. That is, each site in the original alignment can only be included in a single data block.

To help with cutting and pasting from Nexus files (like those used by MrBayes) you can leave "charset" at the beginning of each line. So, the following would be treated exactly the same as the example above:

```
charset Gene1_codon1 = 1-1000\3;
charset Gene1_codon2 = 2-1000\3;
charset Gene1_codon3 = 3-1000\3;
charset intron       = 1001-1256 1675-2000;
```

`[schemes]`

On the lines following this statement, you define how you want to look for good partitioning schemes, and any user schemes you want to define. You only need to define user schemes if you choose search=user.

`search: all | greedy | rcluster | hcluster | user`

This option defines which partitioning schemes PartitionFinder will analyse, and how thorough the search will be. In general 'all' is only practical for analyses that start with 12 or fewer data blocks defined (see below). In general, the algorithms earlier in this list give better answers, but require longer to run. So, use the earliest algorithm in the list that is practical for your data (roughly, 'all' for very small datasets, 'greedy' for datasets of ~10 loci, 'clustering' for datasets of 100's of loci).

`search = all` Tells PartitionFinder to analyse all possible partitioning schemes. That is, every scheme that includes all of your data blocks in any combination at all. Whether you can analyse all schemes will depend on how much time you have, and on what is computationally possible. **If you have any more than 12 data blocks to start with you should not choose 'all'.** This is because the number of possible schemes can be extremely large. For instance, with 13 data blocks there are almost 28 million possible schemes, and for 16 data blocks the number of possible schemes is over 10 billion. It's just not possible to analyse that many schemes exhaustively. For 12 data blocks, the number of possible schemes is about 4 million, so it might be possible to analyse all schemes if you have time to wait, and a fast computer with lots of processors.

`search = greedy` Tells PartitionFinder to use a greedy algorithm to search for a good partitioning scheme. This is a lot quicker than using search=all, and will often give you the same answer. However, it is not 100% guaranteed to give you the best partitioning scheme. The algorithm is described in the PartitionFinder paper (see Citation, below). When you use `search=greedy`, PartitionFinder has to compare partitioning schemes using an information-theoretic metric (AIC, AICc, or BIC). Which metric it uses is defined using the **`model_selection`** option (see above).

`search = rcluster` Tells PartitionFinder to use a relaxed hierarchical clustering algorithm to search for a good partitioning scheme. This option only works with the **--raxml** commandline option (see above). It works by measuring the similarity of different subsets, then looking at schemes that combine the most similar subsets. It usually performs worse than the greedy search option, and better than the hcluster option. You can control this algorithm using the --rcluster-percent and --weights command line options (see below). The rcluster algorithm is a very efficient way to search, and can be used even on large phylogenomic datasets with 1000's of loci. It's designed for use with datasets that are too large to analyse with the greedy algorithm.

`search = hcluster` Tells PartitionFinder to use a strict hierarchical clustering algorithm to search for a good partitioning scheme. This option only works with the --raxml commandline option. This algorithm is the fastest, but usually the worst performing, of all the search algorithms. If your dataset is huge, and it's just not possible to use any of the other algorithms, this one will still do a reasonable job, usually much much better than trying to choose a partitioning scheme by hand. You can control this algorithm using the --weights command line options (see below).

`search = user` Use this option to compare partitioning schemes that you define by hand. User-defined schemes are listed, one-per-line, on the lines following "search=user". A scheme is defined by a name, followed by an "=" and then a definition. To define a scheme, simply use parentheses to join together data blocks that you would like to combine. Within parentheses, each data block is separated by a comma. Between parentheses, there is no comma. All user schemes must contain all of the data blocks defined in [data_blocks].

Here's an example. If I'm working on my one protein-coding gene plus intron alignment above, I might want to try the following schemes: (i) all data blocks analysed together; (ii) intron analysed separately from protein coding gene; (iii) intron separate, 1st and 2nd codon positions analysed separately from 3rd codon positions; (iv) all data blocks analysed separately. I could do this as follows, with one scheme on each line:

```
together    = (Gene1_codon1, Gene1_codon2, Gene1_codon3, intron);
intron_123  = (Gene1_codon1, Gene1_codon2, Gene1_codon3) (intron);
intron_12_3 = (Gene1_codon1, Gene1_codon2) (Gene1_codon3) (intron);
separate    = (Gene1_codon1) (Gene1_codon2) (Gene1_codon3) (intron);
```

user_tree_topology

This is an additional option which can be added into the .cfg file after the 'alignment' line. It's used if you'd like to supply PartitionFinder with a fixed topology, rather than relying on the neighbour joining topology that the program estimates by default. This might be useful if you know ahead of time what the true tree is, for instance when doing simulations. To use the option, just add in an extra line to the .cfg file like this:

```
# ALIGNMENT FILE #
alignment = test.phy;
user_tree_topology = tree.phy;
```

Where "tree.phy" is the name of the file containing a newick formatted tree topology (with or without branch lengths). The file name can be anything – it doesn't have to be 'tree.phy'. The tree file must be in the same folder as the alignment and the .cfg file. When you use this option, the topology you supply in the tree file will be fixed throughout the analysis. Branch lengths will be re-estimated using a GTR+I+G model on the whole dataset, as in a standard analysis.

If you don't want to use this option, you can just leave out the user_tree_topology line from the .cfg file.

Configuration File options specific to using the --raxml option

When using the --raxml option, the configuration file is exactly the same, but the models that are available differ.

```
models (PartitionFinder): all | <list>
    (PartitionFinderProtein): all_protein | all_protein_gamma
                              | all_protein_gammaI | <list>
```

The details of how these options work is the same as above, but because RAxML has different models available, setting 'models=all' when using the --raxml option will imply different lists of models.

`models = all;` Both available models in RAxML are used, GTR+G and GTR+I+G

`models = all_protein;` all 44 available models of protein substitution in RAxML are used (see list below). NB RAxML does not include any models without gamma distributed rates across sites. This is probably not a bad thing!

`models = all_protein_gamma; models = all_protein_gammaI;` use only models that have +G but not +I from the list below; or use only models that have both +G and +I from the list below. These lists are included in case you are intending to run your final analyses in RAxML (i.e. estimate your tree). In RAxML you cannot have a model with +G in one partition, and another with +I+G in another partition (although you can in PartitionFinder, even using the --raxml option, because we process the data very differently). So, these options might help here.

`models = <list>;` Details are the same as above, but the available model lists are different when using the --raxml option. Here they are:

Nucelotide Models in PartitionFinder when using --raxml (2 in total)

GTR+G, GTR+I+G

Amino Acid Models in PartitionFinderProtein when using --raxml (44 in total)

BLOSUM62+G, CPREV+G, DAYHOFF+G, DCMUT+G, JTT+G, LG+G, MTMAM+G, MTREV+G, RTREV+G, VT+G, WAG+G, BLOSUM62+G+F, CPREV+G+F, DAYHOFF+G+F, DCMUT+G+F, JTT+G+F, LG+G+F, MTMAM+G+F, MTREV+G+F, RTREV+G+F, VT+G+F, WAG+G+F, BLOSUM62+I+G, CPREV+I+G, DAYHOFF+I+G, DCMUT+I+G, JTT+I+G, LG+I+G, MTMAM+I+G, MTREV+I+G, RTREV+I+G, VT+I+G, WAG+I+G, BLOSUM62+I+G+F, CPREV+I+G+F, DAYHOFF+I+G+F, DCMUT+I+G+F, JTT+I+G+F, LG+I+G+F, MTMAM+I+G+F, MTREV+I+G+F, RTREV+I+G+F, VT+I+G+F, WAG+I+G+F

Output files

All of the output is contained in a folder called "analysis" which appears in the same file as your alignment. There is a lot of output, but in general you are likely to be interested in four things, maybe this order:

best_schemes.txt

has information on the best partitioning scheme found. This includes a detailed description of the scheme, as well as the model of molecular evolution that was selected for each subset in the scheme. It also contains a description of the each scheme in RAxML format (for use with the –q option in RAxML).

subsets folder

is a folder which contains detailed information on the model selection performed on each subset. This output is very similar to what you would get from any model-selection program. Each model tested is listed, in order of increasing BIC score (i.e. best model is at the top). This folder also contains alignments for each subset, and a .bin file which allows PartitionFinder to re-load information from previous analyses.

schemes folder

is a folder which contains detailed information on the schemes that were analysed, each in a separate .txt file. For the greedy and clustering algorithms, this folder contains only the starting scheme and the best scheme that was found at each step of the algorithm.

Command line options

There are a number of additional commands you can pass to PartitionFinder from the comandline. These can be used to fine-tune your analyses.

--raxml

This tells PartitionFinder and PartitionFinderProtein to use RAxML rather than PhyML (the default). For reasons why you might do this, read the section “Using PartitionFinder with RAxML”, above. Because of the nature of RAxML, we can’t guarantee that the RAxML executables we have provided in the ‘programs’ folder will work on all Windows and Mac machines. So if you use this option and RAxML doesn’t work, you’ll need to download and compile RAxML yourself, on your own computer. Instructions on how to do that are here: https://github.com/stamatak/standard-RAxML

-p N, --processors N

Default – use all available processors.
N is the number of processors you want PartitionFinder to use. This controls the number independent PhyML or RAxML runs that PartitionFinder will run at any one time. The default is for PartitionFinder to use all of the available processors (look for this message at the start of the run, to see how many it found: “You appear to have N cpus”). However, if you don’t want it to use all the processors, control with this option. E.g. –p 5 would tell PartitionFinder to use up to 5 processors at once.

--force-restart

This will delete all previous workings (by deleting the ‘analysis’ folder) before restarting a run. The default is not to do this, so PartitionFinder can use results that it has already calculated.

--weights “W_{rate}, W_{base}, W_{model}, W_{alpha}”

Default: --weights “1, 0, 0, 0”
A list of weights to use in the clustering algorithms (NB, this only works in combination with the --raxml option and either the hcluster or rcluster search options). This list allows you to assign different weights to the overall rate for a subset, the base/amino acid frequencies, the model parameters, and the alpha parameter (which describes gamma distributed rates across sites). This will affect how subsets are clustered together. For instance:

--weights '1, 1, 1, 0.1'

would weight the subset rate, base frequencies, and the model parameters equally, but the alpha parameter as 10x less important. You can play around with these parameters to try and find the best scheme that you can.

--rcluster-percent N

Default: --rcluster-percent 10
This option controls the thoroughness of the relaxed clustering algorithm. By default, this algorithm will, at each iteration, search through the 10% of partitioning schemes that lump together the most similar subsets (where similarity is determined by --weights above). You can make it more thorough by increasing this percentage, or faster and less thorough by decreasing it. In general, if you have a huge dataset you’re better off using the rcluster algorithm with a very low percentage (e.g. --rcluster-percent 0.1) than using

the hcluster algorithm (we have a paper in prep on this: Lanfear, Calcott, Kainer, Mayer, and Statmatakis “Selecting optimal partitioning schemes for phylogenomic datasets: a comparison of clustering methods.”)

Credits

PartitionFinder relies heavily on the following things.

PhyML

PhyML does most of the sums performed by PartitionFinder. PhyML is described in this paper: New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. Guindon S., Dufayard J.F., Lefort V., Anisimova M., Hordijk W., Gascuel O. Systematic Biology, 59(3):307-21, 2010.

RAxML

RAxML allows PartitionFinder to work with large datasets. It is a fantastically fast and efficient piece of software developed by Alexis Stamatakis, the latest version is described here: A. Stamatakis, RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models, *Bioinformatics* **22**, 2688–2690 (2006).

PyParsing

PyParsing is a great Python module that we use for parsing input files.
http://pyparsing.wikispaces.com/

Python

PartitionFinder is written in Python. http://www.python.org/

Helpful People

A few people helped a lot in testing PartitionFinder and making helpful suggestions. In alphabetical order, these wonderful people are: Matt Brandley, Renee Catullo, Karen Meusemann, Bernhard Misof, Ainsley Seago, and Jessica Thomas. Thanks.