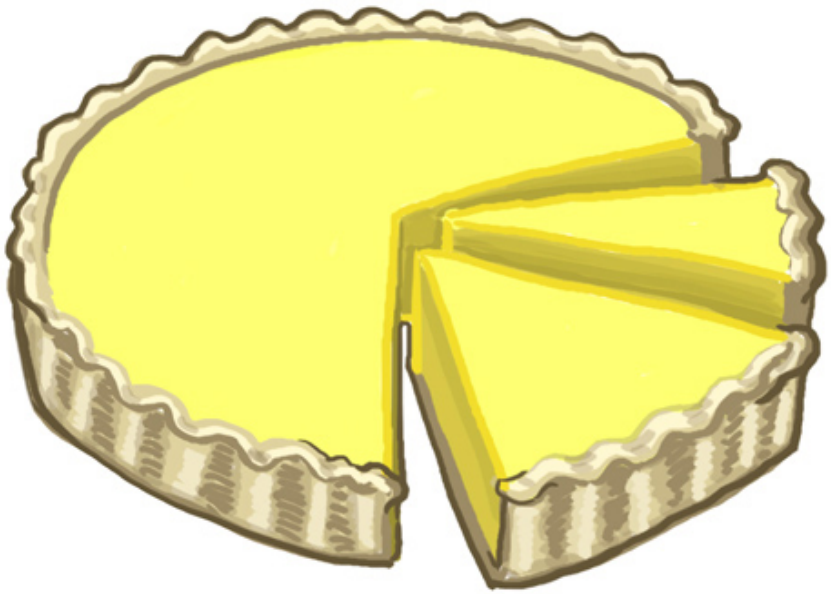
PartitionFinder v1.0

Rob Lanfear, August 2011
Last updated, April 2012

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/



If you use PartitionFinder for your published work, please cite the following paper:

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* in press.
http://dx.doi.org/10.1093/molbev/mss020

Disclaimer

What PartitionFinder is for

PartitionFinder is a program for selecting best-fit partitioning schemes and models of molecular evolution for DNA and amino acid alignments. 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). PartitionFinder then finds 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 PartitionFinder:

1. Find the best-fit partitioning scheme from all possible schemes
2. Find a good partitioning scheme using a heuristic search
3. Compare user-defined partitioning schemes
4. Find best-fit models of molecular evolution for partitioned datasets

PartitionFinder is 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. PartitionFinder implements 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 PartitionFinder run, you are given output files that tell you the best scheme that PartitionFinder could find, 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.

What PartitionFinder is not for

PartitionFinder will not divide up a dataset into subsets from scratch, with no information from the user. That is, PartitionFinder will not try to subdivide any of your data blocks (see [data_blocks], below).

Operating system (Mac & Windows will work, Linux might)

Partitionfinder will run on Mac OSX and Windows. The code was written with Linux in mind too, so if you are interested in getting it running on Linux it's probably just a case of building a new version of PhyML. Get in touch if you'd like to try.

Overview

A lot of people want to estimate phylogenetic trees (and other things like dates of divergence) from DNA and protein sequence alignments. To do this, it is necessary to make assumptions about the way that sequences have evolved. Partitioning allows independent assumptions to be made for different sites in a sequence alignment.

Partitioning involves splitting an alignment into subsets of sites. In this manual, we define two types of sets of sites: a **data block** is a set of sites defined by the user; a **subset** is a collection of one or more data blocks. For instance, it's common to define separate data blocks for the three codon positions of a protein-coding gene (see Figure 1). The point of partitioning is to account for differences in evolutionary processes experienced by the different sites in the alignment. For instance, sites might have evolved at different rates or have experienced different mutational or substitutional processes. Accounting for these differences will improve phylogenetic analyses.

There are two things that make partitioning difficult. The first problem is that it's tricky to compare one partitioning scheme to another on the same data (for instance, comparing schemes a, b, and c in figure 1). Typically, it involves running separate analyses for each scheme you want to consider. This can be arduous, long-winded, and error prone. The second problem is there are A LOT of possible partitioning schemes, so it's hard to know if the few that you chose a-priori are sensible. PartitionFinder is designed to solve both of these problems.

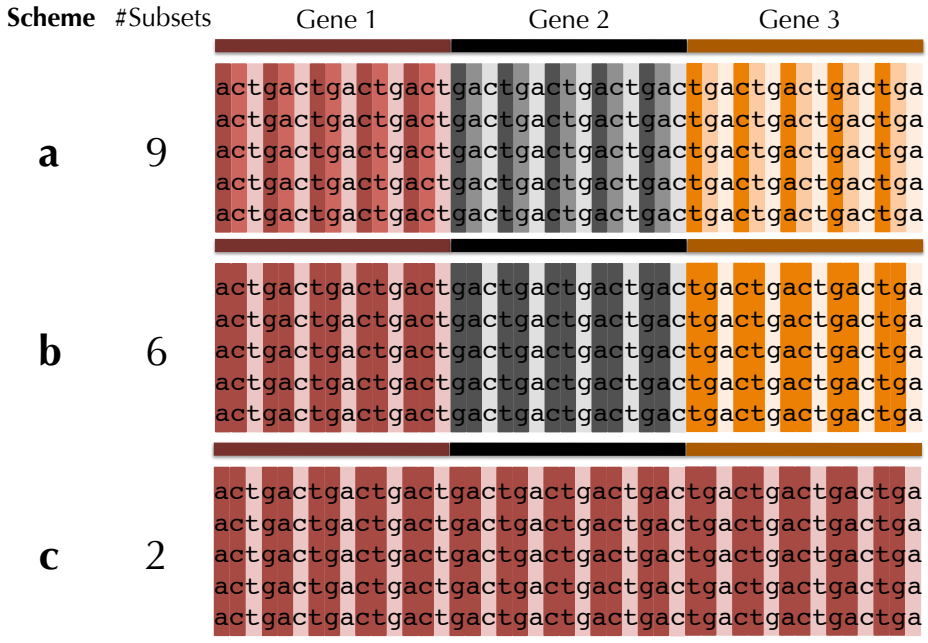


Figure 1. Three commonly employed partitioning schemes for DNA alignments of three protein-coding genes.

Figure 1 shows a typical partitioning problem. 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 data into 9 sets of sites for this alignment – one for each codon position in each gene (scheme a, Figure 1). 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 (scheme b, Figure 1). 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 (scheme c, Figure 1). 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 PartitionFinder run you are told not only which partitioining scheme is the best, but 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 its own model comparisons). 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`, below). You can also tell PartitionFinder exactly which models of molecular evolution to consider (see `models`, below). And you can define how it should 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 availalbe, so it's fast too.

Running PartitionFinder on Mac OSX

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 paragraph. 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.

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

Running PartitionFinder on Macs

Once you have Python 2.7 installed, download the latest version of PartitionFinder from here: www.robertlanfear.com/partitionfinder

Once you have your input files set up (see below), follow these steps to run PartitionFinder.

1. Open Terminal (on most Macs, this is found in Applications/Utilities)
2. In the terminal, you need to type the line below, where `<PartitionFinder.py>` is the full path to PartitionFinder.py file and `<InputFoldername>` is the full path to your input folder.

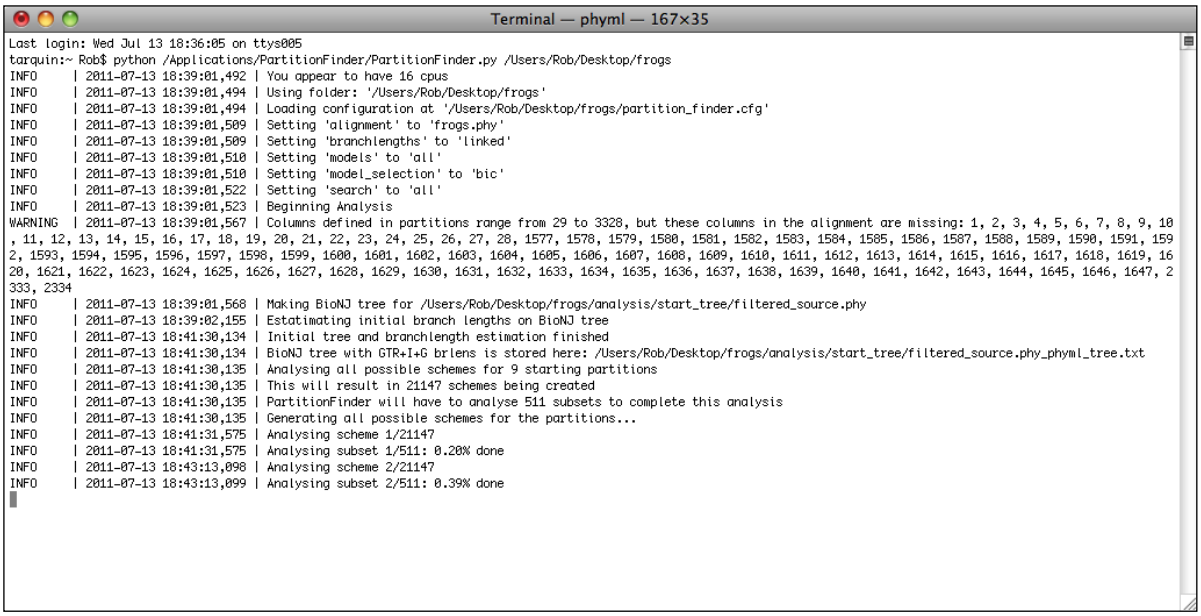
```
python <PartitionFinder.py> <InputFoldername>
```

For example, if I'd downloaded PartitionFinder and put it into my 'Applications' folder, and I had an analysis on my Desktop in a folder called 'frogs', I would type this at the command line, and then hit Enter:

```
python  /Applications/PartitionFinder/PartitionFinder.py Users/Rob/Desktop/frogs
```

There's a trick that makes this very easy – you can drag and drop files or folders onto the terminal and it will fill out the whole filepath for you. So, once you've typed "python" followed by a space, you can just use Finder to navigate to PartitionFinder.py and drag and drop it onto the terminal, then navigate to your analysis folder and drag and drop the blue folder icon onto the terminal. Once that's done, just hit 'Enter' to start PartitionFinder.

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.



Running PartitionFinder on Windows

Installing Python on Windows

Partitionfinder works fine on Windows. The first thing you'll need to install python to get it to work. You can get it 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 ensure that update your "PATH", so that your computer can find it. To do this, follow these steps:

1. Open up the "Control Panel" (under settings in the Start Menu).
2. Choose the "System" icon, and go to the "Advanced" Tab.
3. Click the button called "Environment Variables", This brings up a dialog box.
4. Edit the "path" entry in the System variables. It will contain lots of entries separated by semicolons. Go right to the end, and add a semicolon, and the path where python is found. If Python is in its default directory, you'll be adding this text: ";c:\Python27". So before it might look like this:

   ```
   c:\Windows;c:\Program Files\Some Program;
   ```

   After you're done it will look like this:

   ```
   c:\Windows;c:\Program Files\Some Program;c:\Python27
   ```

A video of how to do this is online here:
http://showmedo.com/videotutorials/video?name=960000&fromSeriesID=96

Running PartitionFinder on Windows

Once you have Python 2.7 installed, download the latest version of PartitionFinder from here: www.robertlanfear.com/partitionfinder

Once you have your input files set up (see below), follow these steps to run PartitionFinder.

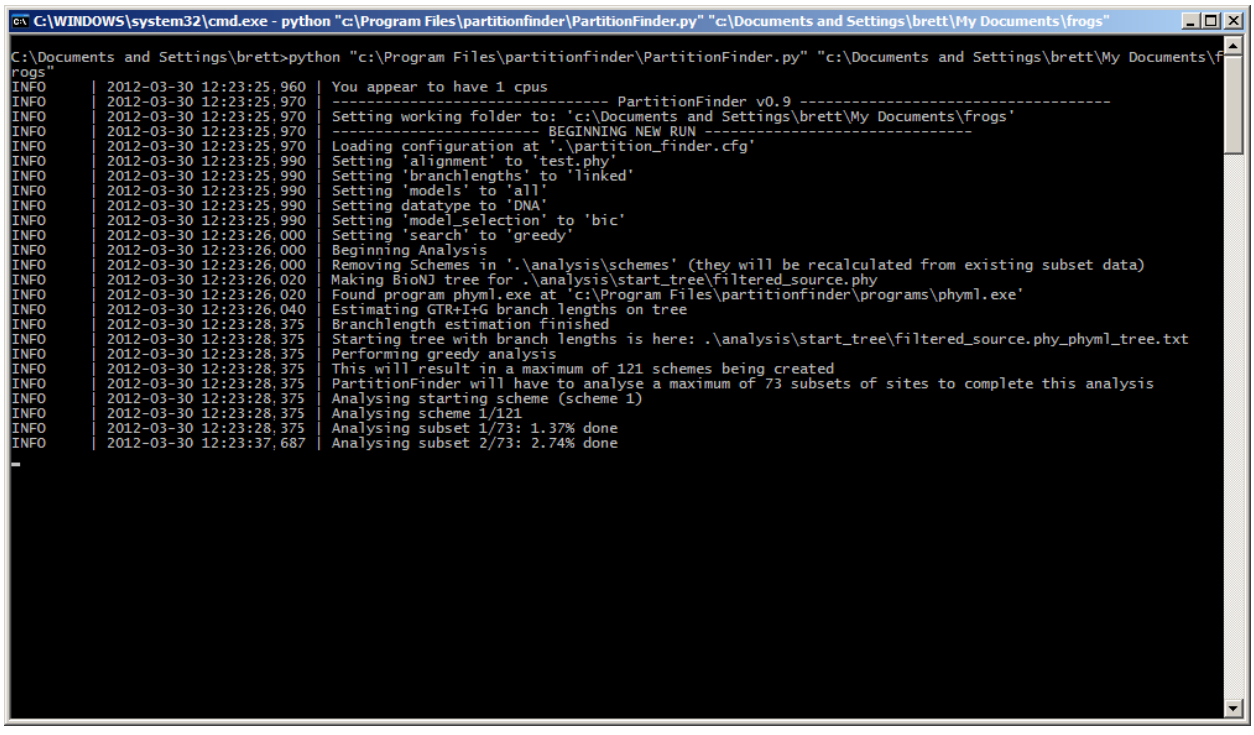
1. Open a command prompt. To do this, click on the Start Menu, then navigate to the command prompt like this: Programs/Acessories/Command Prompt.
2. In the command prompt, you need to tell the computer where to find PartitittionFinder, and where to find your input files. To do that, you'll enter a line that looks like the line below, where `<PartitionFinder.py>` is the full path to PartitionFinder.py file and `<InputFoldername>` is the full path to your input folder. Remember to use quotes around the two names, as shown below.

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

For example, if I'd downloaded PartitionFinder and unzipped it into my 'Program Files' folder, and I had an analysis in a folder in 'My Documents' called 'frogs', I would type this at the command line, and then hit Enter:

```
python  "c:\Program Files\partitionfinder\PartitionFinder.py" "c:\Documents and Settings\brett\My Documents\frogs"
```

Once that's done, just hit 'Enter' to start PartitionFinder. 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.



Input Files

PartitionFinder needs 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 example folder. There is also an online tutorial at www.robertlanfear.com/partitionfinder/tutorial.

Alignment

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.

Changing alignment 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, open up your alignment file, then go the 'File' menu, click 'Export', then 'Selected documents...'. You'll get a list of options for the file format. Scroll down and choose 'Phylip (*.phy)'. Click 'OK', then you'll get an option box which asks how long the names should be in the exported file, choose the 'Export full length' option. Done.

Configuration File

PartitionFinder gets all of its information on the analysis you want to do from a configuration file. This file should always be called "partition_finder.cfg". 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 "|"

alignment

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

branchlengths: linked | unlinked

This setting tells PartitionFinder how to treat branch lengths of the subsets. How you set this will depend to some extent on which program you intend to use for you final phylogenetic analysis. 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

changes all the branch lengths at once, 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: all | all_protein | raxml | mrbayes | <list>

This setting tells PartitionFinder which models of molecular evolution to consider during model selection. PartitionFinder performs 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. For most people, models=all will be the most useful setting.

models = all; 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; 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; tells PartitionFinder to use only the nucleotide models available in RaxML or MrBayes3.1.2 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>;
If you want to restrict the list of models considered, you can do that by specifying any list of models from either the nucleotide or amino acid models. Each model in the list should be separated by a comma. For example, if I was only interested in a few nucleotide models, I might do this:

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

Or, for protein models 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 two separate PartitionFinder analyses to find the best partitioning scheme – one in which you analyse just the nucleotide data blocks, and another for the amino acid data blocks).

In case it's helpful, here are lists of all of the models implemented in PartitionFinder.

Nucelotide Models (56 in total)
+I: estimate a proportion of invariant sites
+G: estimate 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 (112 in total)
+I: estimate a proportion of invariant sites
+G: estimate gamma distributed rates across sites (with 4 categories)
+F: estimate amino acid frequencies from the alignment, rather than the model
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

This setting tells PartitionFinder which method to use for model selection. It also defines the metric that is used 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 | user | greedy

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).

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 = 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 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.

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(s) found. This includes a detailed description of the schemes, as well as the model of molecular evolution that was selected for each subset in the scheme.

If you search among all schemes (`search=all`) or some pre-defined user schemes (`search=user`) then this file will contain information on the best scheme under each of the three information-theory metrics: the Akaike Information Criterion (AIC), the corrected Akaike Information Criterion (AICc), and the Bayesian Information Criterion (BIC).

If you use the greedy algorithm (`search=greedy`), there will only be a single scheme in best_schemes.txt. This is because the greedy algorithm searches among partitioning schemes using one of the information-theory metrics to guide it (defined using `model_selection`, see above). Because of this, you can only find the best scheme for the metric you've used, and not for all three metrics at once.

all_schemes.txt

contains most of the same information as best_schemes.txt, but organised in spreadsheet format, and for all schemes that were compared during the search. This is probably only useful if you're interested in working on methods of finding good partitioning schemes.

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 all the schemes that were analysed, each in a separate .txt file which has the same name as the scheme. Most of this information is contained in all_schemes.txt.

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.

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, Ainsley Seago, and Jessica Thomas. Thanks.