A quick guide to PartitionFinder

Rob Lanfear, June 2011

Overview

PartitionFinder is a Python program for simultaneously choosing partitioning schemes and models of molecular evolution for DNA sequence data. It is useful to use before running a phylogenetic analysis of DNA sequence data, in order to decide how to divide up your sequence data into separate blocks before analysis, and to simultaneously perform model selection on each of those blocks. Currently, PartitionFinder will only work on Intel Macs. We hope to extend this to PPC Macs and to Windows machines soon.

A lot of people want to estimate phylogenetic trees (and a host of other things like dates of divergence) from DNA sequence data. In order to go from a DNA sequence alignment to a phylogenetic tree, one needs to make some assumptions about the way DNA sequences have evolved. Partitioning allows those assumptions to vary for different sites in the DNA sequence alignment.

Partitioning is just the definition of a series of partitions on a DNA sequence aligment. A **partition** is defined here as the thing that tells me how to divide up a DNA sequence alignment. Partitions in turn define **subsets** of sites. The point of partitioning is to define subsets that you suspect may have evolved in different ways. This might include subsets that you think might have evolved at different rates, or that might have evolved under different mutational or substitutional processes. The point of partitioning is to try and capture more of the variation in molecular evolution that occurs in an alignment.

A good example of partitioning, and of what PartitionFinder is designed to do, is to imagine an alignment of three protein-coding genes. I might suspect that each gene has been evolving differently, maybe because they exist on different bits of the genome, or because they have experienced different evolutionary constraints. So I might want to define some partitions that separate out the three genes. But in addition, I might think that each codon position in each gene has been evolving differently – the different codon positions are known to evolve at different rates, and to experience different substitutional processes because of the effect that a mutation at each position can have on the encoded amino acid. So, I could partition out 9 subsets for my three gene alignment – one for each codon position in each gene. But, I could also assume that the first and second codon positions within each gene are evolving in a similar way, so I could partition our 6 subsets. The trouble is, I could carry on like this for a long time, defining lots of different partitioning schemes that are sensible. In fact, if I am willing to define up to 9 subsets, there are 21,147 different partitioning schemes that define 9 or less than 9 subsets. My biological intuition can only take me so far.

PartitionFinder solves this problem by finding optimal partitioning schemes for you. All you need to do is define the first partitioning scheme (the one with 9 partitions in this case), and PartitionFinder will do the rest. At the end of a PartitionFinder run you are told which partitioining scheme is the best, and also which model of molecular evolution you should apply to each subset of sites in that scheme. You can then go straight on to performing your phylogenetic analysis, without any additional model-testing or partition analysis.

Quick Start

If these instructions don't make sense, there are much more detailed instructions and usage examples below. Have a look at the example file to get your input files in order. Note, **all lines in the .cfg file (except comments) must end with semi-colons.**

1. Open Terminal and cd to the directory with PartitionFinder in it
2. Run PartitionFinder by typing at the command prompt:
   `python PartitionFinder.py <foldername>`
   where '<foldername>' is the full file path to a folder containing a Phylip alignment and a PartitionFinder configuration file called 'partition_finder.cfg'. E.g.
   `python PartitionFinder.py /Users/Rob/new_analysis`

The results will be in new subfolder called "analysis" within the original folder. The most important files are "best_schemes.txt" and "all_schemes.txt".

Before You Start

Make sure you have Python 2.7 or later installed. To check which version you have, open Terminal and type "`python`". It will tell you the version you have. If you have anything before 2.7, update python using the appropriate installer from here: http://www.python.org/getit/

Input Files

There are two input files, a Phylip alignment and a configuration file.

Alignment

Your DNA alignment needs to be in non-interleaved Phylip format. We use the same version of Phylip that PhyML uses, which is described in detail here http://www.atgc-montpellier.fr/phyml/usersguide.php?type=phylip

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. The order of variables in the file is also important, and must be kept the same as in this example.

The configuration file looks like this (note the semi-colons!):

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

# BRANCHLENGTHS #
branchlengths = linked;

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

# PARTITIONS #
[partitions]
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 DNA sequence alignment. If the alignment is in the same file as the partition_finder.cfg file, all you need is the name here, e.g. "`alignment = alignment.phy`". If you're alignment is in a different file you will need to supply a full filepath, e.g. "`alignment = Users/Rob/analysis/alignment.phy`".

branch_lengths: linked | unlinked

This setting tells PartitionFinder how to treat branchlengths in each subset of sites. 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 branchlenghts, but only some support unlinked branchlenghts (e.g. MrBayes, BEAST, and RaxML). If you are not sure, the best thing to do is try two independent runs of PartitionFinder (set up two separate analysis folders for this) one with branchlengths linked, and another with branchlengths unlinked. You should then compare the best schemes from both analyses, and choose the overall winner.

`branch_lengths = linked;` means that only one underlying set of branch lengths is estimated. Each subset then has its own scaling parameter, which can stretch or shrink or all the branchlengths together, but won't change the length of one branch relative to another. The total number of branchlength parameters here is quite small. If there are N species in your dataset, then there are 2N-3 branchlengths in your tree, and each subset after the first one adds an extra scaling parameter. So if there are S subsets in a given scheme, the total number of branchlength parameters is:

$$P_{linked} = (2N - 3) + (S - 1)$$

For instance, if you had a scheme with 10 subsets of sites and a dataset with 50 species, you would have 106 parameters for branchlengths.

`branch_lengths = unlinked;` means that each subset of sites has its own totally independent set of branchlenghts. In this case, branchlengths are totally unrelated between subsets, and so each subset has it's own set of 2N-3 branch length parameters that need to be estimated. With this setting, the number of branchlength paramters can be quite large. If there are S subsets in a given scheme, and N species in your dataset, the number of parameters is:

$$P_{unlinked} = S(2N - 3)$$

In the same situation as above (a scheme with 10 subsets of sites and a dataset with 50 species), the number of parameters when branch_lengths=unlinked would be 970. That's a lot more parameters than the 106 when branch_lengths = linked, but it doesn't necessarily mean that it will give you worse scores.

models: all | raxml | mrbayes | <list>

This setting tells PartitionFinder which models of molecular evolution to consider during model selection. PartitionFinder performs model selection (in much the same way as other programs like ModelTest, MrModelTest, or ModelGenerator) on each subset of sites. Your results therefore tell you not only the best partitioning scheme, but also which model of molecular evolution is most appropriate for each subset of sites in that scheme. So 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; tells PartitionFinder to compare 56 models of molecular evolution for each subset. These comprise the usual 12 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 = raxml; models = mrbayes; tells PartitionFinder to use only the 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 final analysis, as it restricts the number of models that are compared to only those available in the final programs. This is not only the most appropriate thing to do, but also saves a lot of computational time.

models = <list>, e.g. **models = GTR+G, GTR+I+G;**
If you want to restrict the list of models considered, you can do that by specifying any particular list of models from the above 56 (use the terminology as given above). Each model in the list should be separated by a comma. For instance, the example given above implements only the models found in RaxML (GTR+G and GTR+I+G), so is equivalent to using the 'raxml' command.

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 partition selection if you use search=greedy (see below). If you're looking for a sensible value here, my own preference is to use the BIC (see below).

The AIC, AICc, and BIC are all quite similar in spirit – they reward models which fit the data better, but also penalise models depending on how many parameters they have. The idea is find the right balance between including parameters that allow us to model evolution, but avoiding overparameterisation. In general, the BIC penalises parameters the most, followed by the AICc, and then the AIC is most lenient. 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).

[partitions]

On the lines following this statement you define the starting partitions for your analysis. Each partition 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 partition. 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 intronic DNA. Let's imagine that some of the intron was unalignable too, so we don't want that included in our analysis.

```
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 start and end of the gene, 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 >1 range of sites in a single partition. Here, we excluded sites 1257-1674 because they were unalignable.

There are a couple of general points about partitions. First, the total list of partitions 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. However, you'll get a warning from PartitionFinder if not all of the sites in the original alignment are included in the partitions you've defined. Second, partitions cannot be overlapping. That is, each site in the original alignment can only be included in a single partition.

To help with cutting and pasting from Nexus files (like MrBayes) you can leave "charset" at the beginning of each line, and a semi-colon at the end of each line. Both of these things are ignored. 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;
```

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 10 or fewer partitions defined (see below).

search = all Tells PartitionFinder to analyse all possible partitioning schemes. That is, every scheme that includes all of your partitions 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 partitions to start with you should not choose 'all'.** This is because the number of possible schemes can be extremely large. For instance, with 13 starting partitions there are almost 28 million possible schemes, and for 16 partitions the number of possible schemes is over 10 billion. It's just not possible to analyse that many schemes exhaustively. For 12 starting partitions, 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 = user` Use this option to compare only the partitioning schemes that you define by hand. User-defined schemes are simply 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 partitions that you would like to combine. Within parentheses, each partition is separated by a comma. Between parentheses, there is no comma.

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 partitions 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 partitions 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)
```

`search = greedy` Tells PartitionFinder to implement 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 optimal scheme. However, it is not 100% guaranteed to give you the optimal partitioning scheme. When you implement this option, PartitionFinder will start by looking at the most partitioned scheme (e.g. 'separate' in the example above. It will then try all possible ways of joining together two partitions of that scheme, and see if any of them improve on the fully-partitioned model. It will carry on in this way until it can't find a scheme that improves on the current 'best' scheme.

When you use `search=greedy`, PartitionFinder has to compare models using a scoring system. Which scoring system it uses is defined using the **`model_selection`** option (see above). Because the greedy algorithm follows a path through the partitions, it will only find a single best-scheme, which will be output to the 'best_schemes.txt' file (see below).

Running PartitionFinder

Once you have your input files set up, to run PartitionFinder you:

1. Open Terminal (on most Macs, this is found in Applications/Utilities)
2. cd to the directory with PartitionFinder in it. The easiest way to do this is to type "cd " (the space is important) at the command prompt, and then drag-and-drop the PartitionFinder folder icon into terminal. This will fill out all of the filepath for the PartitionFinder file for you. Now hit enter.
3. Run PartitionFinder by typing at the command prompt:

   ```
   python PartitionFinder.py <foldername>
   ```

   where '<foldername>' is the full file path to a folder containing a Phylip alignment and a PartitionFinder configuration file called 'partition_finder.cfg'. E.g.

   ```
   python PartitionFinder.py /Users/Rob/new_analysis
   ```

Note that you don't have to be in the PartitionFinder folder to run PartitionFinder. In fact, you can just open up terminal and run it. To do that, all you need to do is change "PartitionFiner.py" above to the full filepath for PartitionFinder.py. For instance, it might look something like this:

```
python Users/Rob/Applications/PartitionFinder/PartitionFinder.py –p -1
/Users/Rob/new_analysis
```

There are a couple of additional commandline options that can be useful:

`--force-restart`
: delete all previous output and start afresh. Useful if you have had to quit an anlaysis halfway through, and the program won't restart.

`-p N`
: Define the number processors to use, where 'N' is an number. Use -1 to match the total number of cpus on your machine (this is the default), or set N to any number to use that many processors. The more processors you can use, the faster PartitionFinder will be.

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 partition 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.

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 partition, or combination of partitions (which we call a '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). There will be some minor differences from other model selection programs, due to the implementation we use, but the results should be broadly similar. 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, apart from the results of the model-selection on each partition in a scheme.

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 for parsing input files.
http://pyparsing.wikispaces.com/

Python

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