PartitionFinder manual

Rob Lanfear, July 2011

Disclaimer

What PartitionFinder is for

PartitionFinder is a Mac OSX program for selecting best-fit models of molecular evolution and best-fit partitioning schemes for DNA sequence data. The user provides a DNA sequence alignment with some pre-defined partitions (e.g. 9 partitions defining the 1st, 2nd and 3rd codon positions of 3 protein-coding genes, see Figure 1), and optionally some pre-defined partitioning schemes (see e.g. figure 1). PartitionFinder can then calculate:

1. The optimal partitioning scheme from all possible schemes
2. The best-fit partitioning scheme from a heuristic search of schemes
3. The best-fit partitioning scheme from any set of user-defined schemes
4. The best-fit model of molecular evolution for each subset of sites in each partitioning scheme

PartitionFinder is designed to take the hard work out of comparing partitioning schemes, and to help users select a scheme that maximises the fit of the data to the model, without including more parameters than are necessary. PartitionFinder is useful in any situation where you wish to compare partitioning schemes for molecular sequence data, and it also does all of the model-selection into the bargain. 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).

Overview

A lot of people want to estimate phylogenetic trees (and 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 (see Figure 1).

Partitioning involves definition of a series of partitions on a DNA sequence aligment. A **partition** is something that divides up a DNA sequence alignment into **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 idea is to try to model the variation in rates and forms of molecular evolution among different subsets, in the hopes that it improves the accuracy of subsequent phylogenetic inference.

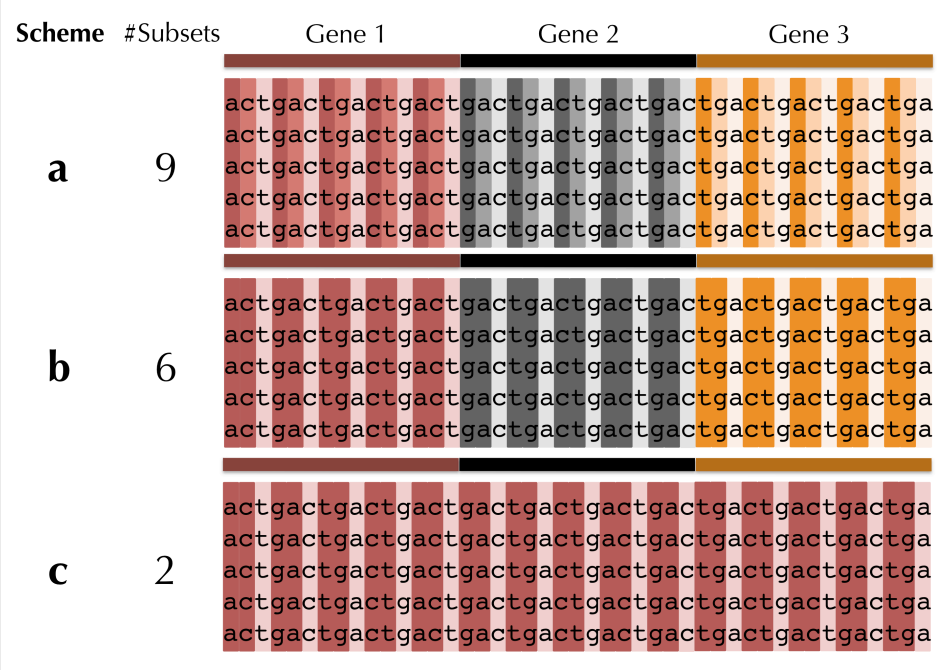


Figure 1. Three commonly employed partitioning schemes for DNA alignments of three protein-coding genes.

There are two problems with partitioning. The first problem is how to know whether one partitioning scheme is better than some other partitioning scheme on the same data. The second problem is how to practically compare partitioning schemes, without having to run thousands of separate analyses, and perform several thousand runs of model-selection programs. PartitionFinder is designed to solve both of these problems.

A good example of partitioning is to imagine an alignment of three protein-coding genes (Figure 1). You might suspect that each gene 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 use partitions to define 9 subsets of sites for this alignment – one for each codon position in each gene (scheme a, Figure 1). The question is whether 9 subsets is too many? Perhaps a better partitioning scheme would be to join together the 1st and 2nd codon sites of each gene, so defining 6 subsets (scheme b, Figure 1). Or perhaps it would be better to forget the divisions between genes, and define only 2 subsets – 1st and 2nd codon sites versus 3rd codon sites (scheme c, Figure 1). The trouble is that if you start with 9 possible partitions, 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 starting partitions (those in scheme a, Figure 1, in this case) 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 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 comparisons of partitioning schemes analysis.

PartitionFinder is also very flexible. If you don't want to compare all possible schemes, you can define exactly the schemes you do want to compare (see `search=user`, below), or use a heuristic search algorithm (see `search=greedy`, below). You can also tell PartitionFinder exactly which models of molecular evolution to consider (e.g. HKY, GTR, etc), or just let it consider the usual 56 models (see `models`, below). And you can define how it should compare partitioning schemes and models (see `model_selection`, below). Finally, PartitionFinder is fast – it uses a number of methods to reduce the complexity of comparing partitioning schemes, and it automatically runs on all the processors as you have available (but you can stop this if you want, see '`-p`', below).

Running PartitionFinder

Before you start

Make sure you have Python 2.7 or later installed (but avoid installing Python 3.0 or above). Installing Python is really easy. 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/

Starting your analysis

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

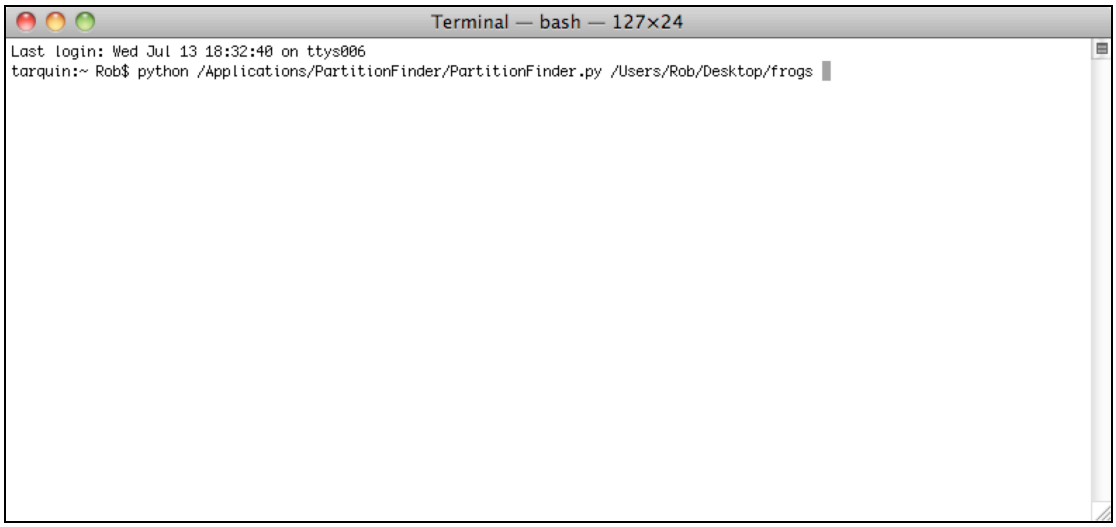
1. Open Terminal (on most Macs, this is found in Applications/Utilities)
2. In the terminal, you need to type something of the following form, where `<PartitionFinder.py>` is the full location of the PartitionFinder.py file and `<InputFoldername>` is the full location of your input folder.

```
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 ~/Desktop/frogs
```

There's a neat trick to make this easy – you can drag-and-drop files onto the terminal and it will type out the whole filepath for you. So, here you could type "python" followed by a space. Then drag-and-drop the PartitionFinder.py file into terminal, and it will fill out the location. Finally drag-and-drop your analysis folder into terminal, and it will fill out the location of that too. Once that's done, just hit 'Enter' to start PartitionFinder. Here's what it looks like in terminal, when I do that on my computer:



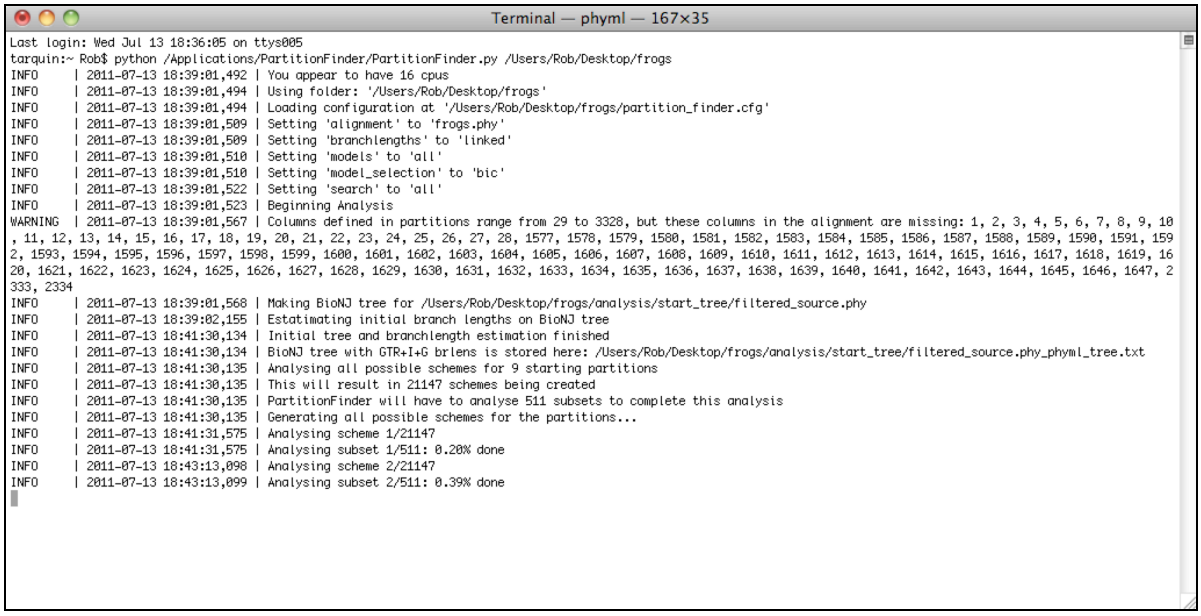
There are a couple of additional commandline options that can be useful. You add these options after you've typed in the rest of the commandline.

`--force-restart`
delete all previous output and start afresh. Useful if you made a mistake in your input files, and want to start again. Using this option is the same as manually deleting the 'analysis' folder before starting a new PartitionFinder analysis.

`-p N`
Define the number processors to use, where 'N' is an number. If you don't specify this number, PartionFinder will by default use all the processors it can find. If you don't want to do this, set N to the number of processors that you want to use. The more processors you can use, the faster PartitionFinder will be.

Output on the screen

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

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. 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 each line.

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. This file should be in the same file as the .cfg file.

branchlengths: linked | unlinked

This setting tells PartitionFinder how to treat branch lengths 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 is to do two runs of PartitionFinder (set up two separate analysis folders for this) one with branch lengths linked, and another with branch lengths unlinked. You should then compare the best schemes from both analyses, to find the best possible scheme, and the best possible settings to use in subsequent phylogentic analyses.

branchlengths = linked; means that only one underlying set of branch lengths is estimated. Each subset has its own scaling parameter that 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. So if there are S subsets in a given scheme, the total number of branch length 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 branch lengths.

branchlengths = unlinked; means that each subset of sites has its own totally independent set of branch lengths. In this case, branch lengths 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 branch length parameters 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 branchlengths=unlinked would be 970. That's a lot more parameters than the 106 when branchlengths = linked, but it doesn't necessarily mean that it will give you worse scores – that depends on the fit of your data to each different model.

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 jModelTest, 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. 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;` 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 comparing partitioning schemes if you use search=greedy (see below).

The AIC, AICc, and BIC are all quite similar in spirit – they all reward models that fit the data better, but penalise models that have more parameters. The idea is find the right balance between including parameters that allow us to model evolution, but at the same time to avoid including too many paramteres (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).

`[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;
```

[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 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).

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/