**PartitionFinder2**

**Manual**

Rob Lanfear, April 2015

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/>

FAQs: [http://www.robertlanfear.com/partitionfinder/faq/](http://www.robertlanfear.com/partitionfinder/news/)

**Citations**

PartitionFinder2 incorporates many years of hard work from many people, presented in various different papers. Citations are the only way that we know people are using our methods. So if you use it PartitionFinder2 in your published work please cite the appropriate papers (there will often be more than one). The correct citations are listed in the ‘Citations’ section, below, and will also included at the bottom of the best\_scheme.txt.

Disclaimer 3

What PartitionFinder and PartitionFinderProtein are for 3

Operating systems (Mac, Windows and Linux work) 4

QuickStart – simple use cases 5

Overview 6

Running PartitionFinder on Macs 7

Installing Python on Macs (most Macs already have it) 7

Installing PartitionFinder on Macs 7

Running PartitionFinder and PartitionFinderProtein on Macs 8

Running PartitionFinder on Windows 9

Installing Python on Windows 9

Installing PartitionFinder on Windows 9

Running PartitionFinder and PartitionFinderProtein on Windows 10

Using PartitionFinder with RAxML (the --raxml option). 11

Why bother? 11

Advanced tips 11

Input Files 12

Alignment File in phylip format 12

Configuration File 13

alignment 13

branchlengths 14

models 14

model\_selection 16

[data\_blocks] 16

[schemes] 17

search 17

user\_tree\_topology 18

Configuration File options specific to using the --raxml option 19

models 19

Output files 20

best\_schemes.txt 20

subsets folder 20

schemes folder 20

Command line options 21

--raxml 21

-p N, --processors N 21

--force-restart 21

--weights “Wrate, Wbase, Wmodel, Walpha” 21

--rcluster-percent N 21

Credits 23

PhyML 24

RAxML 24

PyParsing 24

Python 24

Helpful People 24

# Disclaimer

*Copyright (C) 2011-2015 Robert Lanfear, Paul Frandsen, and Brett Calcott*

*This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. PartitionFinder also includes the PhyML program, the RAxML program, and the PyParsing library all of which are protected by their own licenses and conditions, using PartitionFinder implies that you agree with those licences and conditions as well.*

# What PartitionFinder2 is for

PartitionFinder2 is a program for selecting best-fit partitioning schemes and models of molecular evolution for nucleotide and amino acid alignments. The user provides an alignment, and optionally 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 program 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 the programs:

1. Find the best-fit partitioning scheme for any 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 PartitionFinder2 helps make the model as good as it can be.

PartitionFinder2 can be downloaded from [www.robertlanfear.com/partitionfinder](http://www.robertlanfear.com/partitionfinder). It is designed to take the hard work out of comparing partitioning schemes, and to help find a scheme that maximises the fit of the model to your data, without including more parameters than are necessary. PartitionFinder2 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). We recommend that you use the AICc, see below.

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.

# What PartitionFinder2 is not for

Here are a few things you should be aware that PartitionFinder2 does not do. These are particularly important to know if you are just getting started in molecular phylogenetics:

* PartitionFinder2 does not estimate a phylogenetic tree for you. It estimates a best-fit partitioning scheme that you can use to estimate a phylogenetic tree with downstream programs like RAxML, PhyML, GARLI, IQTree, etc.
* PartitionFinder2 does not estimate models in a Bayesian framework. This is an important distinction. PartitionFinder selects a single ‘best fit’ model in an information theoretic framework. Any analyses that use this model assume that this is the correct model. Usually this isn’t a problem (i.e. this assumption doesn’t usually change any inferences we might make about the phylogeny in ways that are important to most biologists), but in some cases that is not a good assumption. For example if there are lots of almost equally good models, and some models give different trees than others, assuming that a single model is true might give you a phylogeny that would be biologically misleading. In a Bayesian framework one would co-estimate the model and the tree, and integrate out uncertainty in the model of evolution while estimating your tree. Bayesian phylogenetics programs like MrBayes and BEAST and include various options for doing model selection in a full Bayesian framework. It’s important to note that it’s much better for most people to use Bayesian approaches to model selection if possible, precisely because you then get an estimate of your phylogenetic tree (strictly, a posterior distribution) that integrates out many aspects of uncertainty about the model of evolution. So, if you can use whatever the latest methods are in BEAST or MrBayes, you certainly should prefer them over using PartitionFinder. Note that partitionfinder is perfectly appropriate to use with likelihood-based tree searching programs like RAxML, PhyML, GARLI, IQTree, and many others. And it’s also true that for larger datasets, many of the full Bayesian approaches to model selection work too slowly to be practical. In these cases, it is pragmatic to use PartitionFinder to select a single ‘best fit’ model of evolution, and then fix that model in a program like MrBayes or BEAST.

# Operating systems (Mac, Windows and Linux work)

Mac OSX and Windows are supported. For Linux, we do our best to help out. Details of what to do to get up and running are provided in the FAQs on the website: [http://www.robertlanfear.com/partitionfinder/faq/](http://www.robertlanfear.com/partitionfinder/news/)

# 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 = aicc;

search=greedy;

1. 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;

1. Run PartitionFinder using RAxML, as follows:

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

This implements the relaxed clustering algorithm described in Lanfear et al 2014 (Selecting optimal partitioning schemes for phylogenomic datasets. BMC evolutionary biology, 14(1), 82). If the default behaviour is still too slow on your dataset, reduce rcluster-max from the default of 1000 to e.g. 100 like this (see below for an explanation of what this does):

python “<PartitionFinder.py>” “<InputFoldername>” --raxml --rcluster-max 100

**For datasets of any size where you can’t define meaningful data blocks:**

1. Define a single data block with all of your sites.
2. Set the .cfg file options as above, except:

search=kmeans;

1. Run PartitionFinder using PhyML or RAxML (depending on what models you are interested in, and the size of your dataset), e.g. with RAxML:

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

This implements the kmeans partitioning algorithm described in Frandsen et al 2015 (Automatic selection of partitioning schemes for phylogenetic analyses using iterative k-means clustering of site rates. *BMC Evolutionary Biology*, *15*(1), 13), with one important difference – we use the entropy (instead of TIGER rates) to group similar sites together. This is faster and gives better results (we are preparing a paper on this). If you want to use TIGER rates, see instructions below.

# Overview

Partitioning involves splitting sites in your alignment into sets that have evolved under similar models. For example, if you have a dataset of 3 protein-coding genes 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 substitution processes thanks to the triplet structure of the genetic code. Because of this, you might split your data into 9 sets of sites (we call those data blocks) 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?

Even worse, what if you have a dataset (like one made up of UCE’s, for example) that doesn’t have convenient features like codon positions? How do you partition that?

PartitionFinder2 (PF2) solves these problems in one of two ways. If you have pre-defined data blocks, PF2 can quickly and efficiently search for the best scheme from the set of all possible schemes, by trying lots of combinations of data blocks (this is what the greedy, rcluster, and hcluster algorithms do). In the example above, all you would 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 PF2 will do the rest. If you do not have pre-defined data blocks, you can specify a single set of sites and use the K-means algorithm in PF2, which will attempt to find a good partitioning scheme by splitting your alignment into sets of sites which evolve in similar ways.

At the end of a run you are told the best partitioning scheme that PF2 could find (in the best\_scheme.txt file), 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.

PF2 is pretty flexible, and should be able to accommodate most of the popular kinds of partitioned analyses people like to do. This includes options for how to treat branch lengths between subsets, which models of molecular evolution to consider, and many other things. This manual describes in detail all of these steps.

# Running PartitionFinder2 on a Mac

## 1. Install python using Anaconda

PartitionFinder needs Python 2.7.x and some additional libraries to run. By far the simplest way to set this up is to install the Anaconda Python distribution, which is a simple point-and-click installer which can be downloaded from here:

<http://continuum.io/downloads>

Follow the link for the Python 2.7 graphical installer, then open it and follow the prompts.

## 2. Install PartitionFinder2

1. Download the latest version of PartitionFinder2 from [www.robertlanfear.com/partitionfinder](http://www.robertlanfear.com/partitionfinder)
2. Double-click the .zip file, and it will automatically unzip. You will get a folder called something like ‘PartitionFinder2.0.0’
3. Move it to wherever you want to store PartitionFinder2 (e.g. in /Applications)

## 3. Run PartitionFinder2

These instructions describe how to run the ‘example/nucleotide’ analysis using PartitionFinder. This is a DNA alignment with 9 data blocks. To use partitionfinder with amino acid alignments, 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 Python, PartititionFinder2, and your input files. The easiest way to do this is as follows:
   1. Type “python“ followed by a space
   2. 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.
   3. Type another space
   4. Drag and drop the blue ‘example/nucleotide’ folder (in the PartitionFinder folder) onto the command prompt
3. Hit Enter/Return to run PartitionFinder2

That’s it!

More generally, you run PartitionFinder2 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 PartitionFinder2 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.

output

# Running PartitionFinder on Windows

## 1. Install python using Anaconda

PartitionFinder needs Python 2.7.x and some additional libraries to run. By far the simplest way to set this up is to install the Anaconda Python distribution, which is a simple point-and-click installer which can be downloaded from here:

<http://continuum.io/downloads>

Follow the link for the Python 2.7 graphical installer, then open it and follow the prompts.

## 2. Install PartitionFinder2

1. Download the latest version of PartitionFinder2 from [www.robertlanfear.com/partitionfinder](http://www.robertlanfear.com/partitionfinder)
2. Double-click the .zip file, and it will automatically unzip. You will get a folder called something like ‘PartitionFinder2.0.0’
3. Move it to wherever you want to store the PartitionFinder2

## 3. Run PartitionFinder2

These instructions describe how to run the ‘example/nucleotide’ analysis using PartitionFinder. This is a DNA alignment. To use partitionfinder with amino acid alignments, 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 Python, PartititionFinder2, and your input files. The easiest way to do this is as follows:
   1. Type “python“ followed by a space
   2. 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.
   3. Type another space
   4. 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 PartitionFinder2 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.



# Input Files

PartitionFinder2 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 examples we’ve provided in the ‘example’ folder. There is also an online tutorial at [www.robertlanfear.com/partitionfinder/tutorial](http://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/](http://www.geneious.com/default,28,downloads.sm). 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 PartitionFinder2 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 to only focus on the data you want PartitionFinder2 to analyse, more instructions below.

## Configuration File

PartitionFinder2 gets most of its information on the analysis you want to do from a configuration file. This file should always be called “partition\_finder.cfg”. The easiest thing to do is to base your own .cfg on one the examples 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 = aicc;

# DATA BLOCKS #

[data\_blocks]

Gene1\_pos1 = 1-789\3;

Gene1\_pos2 = 2-789\3;

Gene1\_pos3 = 3-789\3;

# SCHEMES #

[schemes]

search = greedy;

# user schemes

123 = (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, and must be in the correct phylip format (see above).

### branchlengths: linked | unlinked

This determines how branch lengths of will be estimated during your analysis. 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, where S is the number of species in your alignment). So, a scheme with 10 subsets and a dataset with 50 species would have 970 branch length parameters.

### models all | allx | beast | mrbayes | gamma | gammai <list> models

Most people will just want to set models to ‘**all**’. Below is a very long description of what all of the models are, and what the other options do.

The ‘models’ option sets which models of molecular evolution to consider during model selection. The models that are available for any particular analysis will depend on two things: your data type (e.g. nucleotides or amino acids) and the phylogeny program you are using (i.e. PhyML, which is the default, or RAxML which you can specify using the --raxml command line option). There are many models you can specify, and although lists are provided below, perhaps the best way to understand what is available is to look at the models.csv file, which is located in the /partfinder folder. This lists all available models, as well as details of the models themselves, and whether each model is implemented in PhyML or RAxML.

Whatever you set for ‘models’, PF2 will print out the list of models it will use at the start of the analysis. You should check this and make sure it is what you want. For every subset of sites that PF2 looks at, it will fit every single model in your list of models, and pick the best one according to your chosen information theoretic score (AIC, AICc, or BIC). 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 PartitionFinder2 is done.

**A very short primer on models in phylogenetics**

Over the years I’ve had a lot of questions about models in phylogenetics. So here’s a primer if you’re interested. There are two important things to understand: what the difference between the models is, and how the models are named.

The difference between all of the models is in which parameters are fixed a-priori, which are estimated from the data (we call these free parameters), and how the free parameters are estimated. In general, all models (both amino acid and nucleotide models) have three components: the frequencies of nucleotides or amino acids (e.g. the proportion of A, C, T, and G in your data), the relative rates at which the different nucleotides or amino are replaced by each other (e.g. the rate at which A replaces G in your data), and the distribution of rates of evolution among sites in your dataset. There are many many possible models of evolution for nucleotide and amino acid alignments. All of the possible models (plus a few that are not possible) are listed in the models.csv file, which is in the partitionfinder folder, under /partfinder/models.csv. As you read the next paragraph, it might help to open up this file and take a look at the models to familiarise yourself with what’s possible.

Frequencies of nucleotides can be fixed in advance, estimated from the data by simply counting the proportions observed in your alignment, or estimated using maximum likelihood. In the models.csv file, this is described in the base\_frequencies column. There are four possible types of base frequency: ‘**equal**’ is when all frequencies are set to equal (e.g. proportions of 0.25 for A, C, T, and G in the JC model) and is only implemented for nucleotide models in PF2; ‘**model**’ is when frequencies are determined a-priori from some other dataset, which is common for amino acid models (e.g. the JTT model); ‘**empirical**’ is when the frequencies are determined by just counting the proportions of nucleotides or amino acids in your data; ‘**ML**’ is when the frequencies are determined using maximum likelihood. The last two options tend to give very very similar answers (ML is guaranteed to be better, but the difference is usually so marginal that it makes no difference to the phylogeny) but ‘empirical’ is much faster, and so most people never use ML base frequencies. One more important thing – thanks to historical naming conventions, the way that these models are named differs for nucleotide and amino acid models. For amino acid models, it’s easy, we’ll use the LG model as an example. If you specify ‘LG’, then amino acid frequencies will be estimated from the model (the LG model comes with some best-guess amino acid frequencies estimated from a large collection of datasets). If you specify ‘LG+F’ then the amino acid frequencies are estimated empirically from your data. If you specify ‘LG+X’ then the amino acid frequencies are estimated with maximum likelihood from your data. For nucleotide models, it’s different. Take a look at rows 2-22 of the models.csv file. First, no nucleotide models have a priori base frequencies included in the model (i.e. the ‘model’ option), so your only options are to have them all equal, estimate them empirically, or estimate them with maximum likelihood. Historically, the first two of these options were given completely different model names, rather than using the ‘+F’ notation that we use for amino acid models. Let’s take the JC model as an example. If you specify ‘JC’, that assumes all base frequencies are equal. But there is no JC+F model, it does exist, but it’s just called the F81 model. This is the reason why there are no ‘+F’ models in the list of possible models for DNA sequence alignments. Rows 2-22 spell this out: each model with ‘equal’ base frequencies is paired with a model with empirical base frequencies. For example, the model in row 1 (JC) matches the model in row 8 (F81), row 2 (K80) matches row 9 (HKY), and so on. To complete the picture, we can specify that we want our base frequencies for nucleotide models estimated with ML (few people use these models though). To do that, we just specify the name of the empirical model with a ‘+X’, e.g. ‘F81+X’. Note that most of these kinds of nucleotide models are only implemented in PhyML, not RAxML.

Relative rates (this is usually called the rate matrix) can be fixed in advance, or estimated from the data. For DNA models, there are really just 6 types of relative rate matrix. JC and F81 have all rates equal (no free parameters), other models set certain parameters to be equal (1-4 free parameters, depending on the model), and at the other end of the scale the SYM and GTR models allow all six reversible rate parameters to differ (5 free parameters, since one is set arbitrarily to 1.0). In the amino acid models, almost all models use pre-estimated rate matrices. The matrices tend not to be estimated from the data, because most datasets do not contain enough information to estimate so many parameters (189 free parameters in an amino acid replacement matrix). These matrices have names like WAG, LG, JTT, etc. There is one exception – one can estimate a GTR model for amino acid data – this has a LOT of parameters (189 free parameters), but for very big datastets it might be appropriate, and so might be worth including in your list of possible models (note that only this model is only supported when you are using the ‘--raxml‘ commandline option (see below).

**models = all;** Chooses the largest sensible set of models possible, depending on the analysis being conducted (i.e. whether you are using DNA or amino acid alignments, and whether you are using PhyML or RAxML).

For example, if you are analysing DNA sequences with the default options in PF2, then models = all will 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). If you are analysing DNA sequences with the --raxml option, then models = all will compare the 3 models of nucleotide evolution available in RAxML: GTR, GTR+G, GTR+I+G.

**models = mrbayes; models = beast;** tells PartitionFinder to use only the nucleotide models available in MrBayes3.2, or BEAST2 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 (see above), but can save 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; ➊

Gene1\_codon2 = 2-1000\3; ➋

Gene1\_codon3 = 3-1000\3; ➌

intron = 1001-1256 1675-2000;  ➍

➊-➌ 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.

➍ 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> mod(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 “Wrate, Wbase, Wmodel, Walpha”

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.”)

# Citations

Depending on your analysis, you may need to cite up to three papers. One for PartitionFinder2, one for the algorithm you use (if you use the rcluster, hcluster, or kmeans options), and one for either PhyML or RAxML.

## PartitionFinder2

If you are reading this manual, then you are probably using ParititionFinder2. The paper describing PF2 will be out (we hope) in 2015, so please take a quick look here to see if you can see it: [www.robertlanfear.com/publications](http://www.robertlanfear.com/publications). If the paper is not yet out, please cite the PF1 paper:

Lanfear, R., Calcott, B., Ho, S. Y., & Guindon, S. (2012). PartitionFinder: combined selection of partitioning schemes and substitution models for phylogenetic analyses. *Molecular biology and evolution*, *29*(6), 1695-1701.

## Using search = ‘rcluster’ or search = ‘hcluster’

These algorithms are described in the following paper, if you use them please cite:

Lanfear, R., Calcott, B., Kainer, D., Mayer, C., & Stamatakis, A. (2014). Selecting optimal partitioning schemes for phylogenomic datasets. *BMC evolutionary biology*, *14*(1), 82.

## Using search = ‘kmeans’

This algorithm is described in the following paper, if you use it please cite:

Frandsen, P. B., Calcott, B., Mayer, C., & Lanfear, R. (2015). Automatic selection of partitioning schemes for phylogenetic analyses using iterative k-means clustering of site rates. *BMC Evolutionary Biology*, *15*(1), 13.

## PhyML

If you use PF2 without the --raxml command line option, PF2 relies heavily on PhyML version 3.0, so please cite:

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.

## Using the --raxml command line option

If you use the --raxml commandline option, PF2 uses RAxML v8.0 for calculations. If you use it, please cite:

A. Stamatakis, RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models, *Bioinformatics* **22**, 2688–2690 (2006)