BAli-Phy User's Guide v2.1.0

Benjamin Redelings

Table of Contents

1. Pre-requisites 2
 1.1. Linux and UNIX 2
 1.2. Windows 2
 1.3. Mac OS X 2
2. Compiling BAli-Phy 2
 2.1. Software requirements 3
 2.2. Quick Start 3
 2.3. Options to **configure** 3
 2.4. Generating the `configure` script and Makefiles (git only) 4
 2.5. Installing when compiling from source 5
3. Installation 5
 3.1. Recommended Additional Software 5
 3.2. Installing precompiled executables 5
 3.3. Setting the PATH 6
4. Running the program 6
 4.1. Quick Start 6
 4.2. Command line options 7
 4.3. Multiple genes or data partitions 7
 4.4. Option files (Scripts) 8
 4.5. Examples 8
 4.6. Command-line options: An overview 9
5. Input 10
 5.1. Sequence formats 10
 5.2. Is my data set too large? 10
6. Output 11
 6.1. Output directory 11
 6.2. Output files 11
 6.3. Summarizing the output 11
 6.4. Summarizing the output - scripted 13
7. Models 14
 7.1. Substitution models 14
 7.2. Insertion/deletion models 18
 7.3. Genetic Codes 19
 7.4. Alignment constraints 19
8. Convergence and Mixing: Is it done yet? 20
 8.1. Definition of Convergence 20
 8.2. Definition of Mixing 20
 8.3. Diagnostics and multiple independent chains 20
 8.4. Diagnostics: Variation in split frequencies across runs 20
 8.5. Diagnostics: Potential Scale Reduction Factors (PSRF) 21
 8.6. Diagnostics: Effective sample sizes (ESS) 21
 8.7. Diagnostics: Stabilization 22
9. Tuning the Markov Chain 23
 9.1. Parameters 23
10. Auxiliary tools 23

10.1. alignment-find 23
10.2. alignment-draw 23
10.3. alignment-thin 23
10.4. alignment-chop-internal 24
10.5. alignment-info 24
10.6. alignment-indices 24
10.7. alignment-cat 24
10.8. trees-consensus 24
10.9. trees-bootstrap 24
10.10. trees-to-SRQ 24
11. Frequently Asked Questions (FAQ) 25
11.1. Running **bali-phy**. 25
11.2. Run-time error messages 25
11.3. Stopping **bali-phy**. 25
11.4. Interpreting the results. 26
11.5. How do I... 26

1. Pre-requisites

BAli-Phy is a Unix command line program that is developed primarily on Linux. BAli-Phy also runs on Windows and Mac OS X, but it is not a GUI program and so you must run it in a terminal.

You may need a 64-bit executable and a 64-bit OS version of your OS to be run large data sets.

We typically run BAli-Phy on Core2 processors with 8Gb of RAM.

1.1. Linux and UNIX

On Linux, you should be able to install and run BAli-Phy without any modification. For other UNIX operating systems, you may need to compile BAli-Phy before running it.

1.2. Windows

On Windows, it is highly recommended that you first install Cygwin [http://www.cygwin.com]. Cygwin is a Unix/Linux command-line environment for Windows. You can access the Cygwin command line (not the normal windows command line) through the Start menu.

Installing Cygwin should not be strictly necessary to run the **bali-phy** program, but it may be necessary for running the scripts that analyze the output. Furthermore, we recommend installing the binary executables from the Cygwin prompt, in order to make the files easily accessible from inside Cygwin.

1.3. Mac OS X

We recommend Mac OS X version 10.4 (or higher).

2. Compiling BAli-Phy

Most users will not need to compile BAli-Phy and can skip this section, because they can use the precompiled executables from the official website for Linux, Mac, and Windows. However, compiling BAli-Phy is intended to be a relatively painless process.

If you are compiling "live" source code that you checked out using GIT (and you probably aren't) then you need to follow the directions in Section 2.4, “Generating the `configure` script and Makefiles (git only)” before you start compiling.

2.1. Software requirements

The following software packages are required for compiling BAli-Phy.

- The GNU C++ Compiler (GCC [http://gcc.gnu.org]) version 3.4 (or higher).

  ### Mac OS X issues:

  Apple's XCode software works, but only if you use OS X 10.4 (Tiger) or higher, and install XCode 2.2 or higher.

- The GNU Scientific Library (GSL [http://www.gnu.org/software/gsl/]) version 1.8 (or higher).

- The Cairo graphics library (Cairo [http://www.cairographics.org/]) version 1.6 (or higher). (Cairo is not strictly necessary, but is a requirement for building the tool draw-tree that is used to draw consensus trees.)

See also Section 3.1, “Recommended Additional Software”.

2.2. Quick Start

In order to compile the program on UNIX, first extract the source code archive, using a graphical archive manager, or the command-line tool **tar**:

```
% tar -zxf bali-phy-2.1.0.tgz
```

Then create a *separate* build directory, enter it, and run the configure command:

```
% mkdir build
% cd build
% ../bali-phy-2.1.0/configure
```

If this command succeeds, then you can simply type

```
% make
% make install
```

to build and install **bali-phy** and its associated tools and install it in `/usr/local/`. (This requires the GNU version of **make**.) To customize the compilation and installation process, read the following sections on supplying arguments to the **configure** script.

2.3. Options to configure

2.3.1. Installing to a location besides `/usr/local/`

The configure script chooses to install **bali-phy** in the directory `/usr/local/` by default. You can install executables to another directory *`dir`* by passing **`--prefix=`*`dir`***. For example, in order to install BAli-Phy under `~/local`, you can enter:

```
% ../bali-phy-2.1.0/configure --prefix=%HOME/local
```

This is recommended if you do not have permission to install to `/usr/local/`.

2.3.2. Specifying where to find libraries and header files (e.g. GSL)

You can instruct the compiler to look for include files in directory *`dir`* by passing **`--with-extra-includes=`*`dir`*** to the **configure** script.

You can instruct the compiler to look for libraries files in directory *`dir`* by passing **`--with-extra-libs=`*`dir`*** to the **configure** script.

For example, if your system has GSL installed in `/usr/local/`, then you might need to add "**`--with-extra-includes=/usr/local/include --with-extra-libs=/usr/local/lib`**" to the configure script arguments so that the compiler can find the GSL include files and libraries.

2.3.3. Selecting a non-default C++ compiler

The default C++ compiler is **g++**. On some systems, **g++** invokes GCC version 3.3, and the correct compiler is called something else, such as **g++-4.2**. To use **g++-4.2** as the C++ compiler when compiling BAli-Phy, you would set the CXX environment variable as follows:

```
% ../bali-phy-2.1.0/configure CXX=g++-4.2
```

2.3.4. Optimizing for a specific architecture

You can specify optimizing for a specific brand of CPU, by specifying the **`CHIP`** variable to **configure**, as follows:

```
% ../bali-phy-2.1.0/configure CHIP=cpu
```

You can set CHIP to any of **`pentium3`**, **`pentium4`**, **`nocona`**, **`core2`**, **`G3`**, **`G4`**, or **`G5`**. (On recent versions of GCC, you can set **`CHIP=native`** to auto-detect the type of CPU you have.) This may produce faster executables, but at the cost of producing executables that may not run on a different kind of chip.

2.3.5. Statically linked executables

Call **configure** with the flag **`--enable-static`** to build static executables. Static executables will be able to run on other computers with the with the same type of CPU but slightly different versions of the operating system.

2.3.6. Example

All these options to **configure** can be combined, as follows:

```
% ../bali-phy-2.1.0/configure --prefix=$HOME/local --enable-static CXX=g++-4.5 CHIP=pentiu
```

This example uses **g++-4.5** to build a pentium4-optimized version of **bali-phy** with static linkage.

2.4. Generating the `configure` script and Makefiles (git only)

Skip this step unless you are compiling a snapshot of the source code that you checked out using GIT. If you downloaded an official tar.gz archive of the source from the website, then it already includes these files.

To generate these files, you need automake 1.8 (or higher) and autoconf 2.59 (or higher). Run these commands in the top level directory of the repository that you checked out.

```
% autoheader
% aclocal -I m4
% automake -a
% autoconf
```

If your system has multiple versions of automake, then you may have to type e.g. **`automake-1.11 -a`** and **`aclocal-1.11`** instead in order to specify which version to use.

2.5. Installing when compiling from source

After compiling BAli-Phy, you can simply type **make install**. This will copy the compiled executables to the installation directory (See Section 2.3.1, “Installing to a location besides `/usr/local/`”).

3. Installation

3.1. Recommended Additional Software

We recommend that you install some additional software in order to graphically view the simple text files that BALi-Phy outputs:

- Cygwin (on windows) - to give you a UNIX prompt (Tracer [http://www.cygwin.com])
- Tracer - to analyze MCMC diagnostics (Tracer [http://tree.bio.ed.ac.uk/software/tracer/])
- PERL - to script an analysis of BALi-Phy output files
- The plotting program gnuplot (gnuplot [http://www.gnuplot.info/]) - to script an analysis of BALi-Phy output files
- Mozilla or Mozilla/Firefox - to view the math in the XHTML documentation. (Firefox [http://www.mozilla.org/products/firefox/])

On Mac OS X, PERL should already be installed, and you should be able to install gnuplot using MacPorts [http://www.macports.org]. On Windows, you can install PERL and gnuplot using the Cygwin [http://www.cygwin.com] installer. On Linux, simply use your distribution's package manager.

We additionally recommend the program seaview [http://pbil.univ-lyon1.fr/software/seaview.html] to graphically view sequence alignments, and the program FigTree [http://tree.bio.ed.ac.uk/software/figtree/] to graphically view phylogenies.

3.2. Installing precompiled executables

You can extract the compressed archive on the Unix (or Cygwin) command line using the **tar** command:

```
% tar -zxf bali-phy-version.tgz
```

On Windows, we recommend that you first install Cygwin, and then extract the compressed archive from the Cygwin prompt (e.g. instead of simply installing to `Program Files/`.) While it is not strictly necessary to run **bali-phy** from inside Cygwin, this will make it easier to analyze the output files. Also, the manual assumes that you are using a Unix-type console, like Cygwin.

From the Unix (or Cygwin) prompt, you might choose to extract the compressed archive in the directory `/usr/local` if you have root access. If you do not have root access, you could make a directory `~/local` in your home directory and then extract the files there.

3.3. Setting the PATH

If you installed BAli-Phy to the directory `/usr/local/`, then you can run bali-phy by typing **`/usr/local/bin/bali-phy`**. However, it would be much nicer to simply type **`bali-phy`** and let the computer find the executable for you. This can be achieved by putting the directory that contains the `bali-phy` executables into your "path". (The directory that contains the executables will be `/usr/local/bin` if you extracted the bali-phy archive to the directory `/usr/local`, but will be somewhere else if you extracted the bali-phy archive somewhere else.)

The "path" is a colon-separated list of directories that is searched to find program names that you type. It is stored in a variable called PATH. You can examine it the current value of this variable by

```
% echo %PATH
```

We will assume that you extracted the bali-phy archive in `/usr/local` and so you want to add `/usr/local/bin` to your PATH. The commands for doing depend on what "shell" you are using. Type **`echo %SHELL`** to find out.

If your shell is **sh** or **bash** then the command looks like this:

```
% PATH=/usr/local/bin:%PATH
```

If your shell is **csh** or **tcsh**, then the command looks like this:

```
% setenv PATH /usr/local/bin:%PATH
```

Unfortunately, the effects of this will affect only the window you are typing in, and will vanish when you reboot. You can put these command into the files `.profile` (for the Bourne shell **sh**), `.bash_profile` (for BASH), or `.login` (for tcsh). However, those details are beyond the scope of this document.

4. Running the program

Here are some examples and explanations of how to run **bali-phy**. You can get an overview of command line options (see Section 4.6, “Command-line options: An overview”) by running **bali-phy --help**.

We recommend running multiple chains in parallel for each command, because

1. You can combine the samples, leading to faster run times.

2. You can compare the runs to determine if the chains have converged.

4.1. Quick Start

The simplest way to run **BAli-Phy** is to type all the arguments on the command line:

```
% bali-phy sequence-file
```

Here *`sequence-file`* is a FastA or PHYLIP file containing the sequences you wish to analyze. The filename should end in **`.fasta`** or **`.phy`** to indicate which format it is using.

In this simple example, **bali-phy** automatically detects whether *`sequence-file`* contains DNA, RNA, or Amino-Acids and uses default values for several command line options. Thus, if *`sequence-file`* contains DNA, then this is equivalent to the more verbose command line

```
% bali-phy sequence-file --alphabet DNA --smodel TN --imodel RS07 --iterations=100000
```

Here the substitution model is Tamura-Nei, the insertion/deletion model is RS07, and the number of iterations is 100,000. If *sequence-file* contains amino acids, then the defaults will be:

```
% bali-phy sequence-file --alphabet Amino-Acids --smodel LG --imodel RS07 --iterations=100
```

4.2. Command line options

You can specify a more complex substitution model as follows (See Section 7.1, “Substitution models”):

```
% bali-phy sequence-file --smodel WAG+gamma+INV
```

You may specify an indel model of **none** to fix the alignment to its initial value, and ignore information in shared insertions or deletions.

```
% bali-phy sequence-file --imodel none
```

If you desire to use a codon model, you must specify the alphabet:

```
% bali-phy sequence-file --smodel M0 --alphabet Codons
```

4.3. Multiple genes or data partitions

You may analyze multiple genes by putting each one it its own data partition:

```
% bali-phy sequence-file1 sequence-file2
```

You should put the data from the first gene in *sequence-file1* and the second gene in *sequence-file2*. In this scenario, both genes share the same tree, but their alignments vary independently. Furthermore, the branch lengths for each gene are scaled by an independent factor. By default, each partition will have its own default alphabet, substitution model, insertion/deletion model, and tree length.

By default, each partition will recieve an independent copy of the model, and will not share parameter values:

```
% bali-phy sequence-file1 sequence-file2 --smodel TN --imodel RS07
```

However, you can select partition-specific values for 5 options: **--smodel**, **--imodel**, **--alphabet**, **--same-scale**, and **--align-constraint**. For example, to specify different substitution models but the same alphabet:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR --alphabet 1,2:DNA
```

You can fix the alignment and ignore insertion/deletion information in one partition, while allowing the alignment to vary and using insertion/deletion information in another partition:

```
% bali-phy sequence-file1 sequence-file2 --imodel 1:RS07 --imodel 2:none
```

You can also specify that two partitions share a single copy of a single substitution model or indel model. This reduces the number of parameters and also pools information between the partitions:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1,2:TN --imodel 1,2:RS07
```

By default each partition has a separate scale, but you can force groups of partitions to share a scale. The name of the groups for the scale are not currently used, but may be used in later versions of the software:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR --same-scale 1,2:gro
```

Finally, you may specify the option **`--traditional`**, or its short form **`-t`**. This is the same as **`--imodel none`** and affects all partitions:

```
% bali-phy sequence-file1 sequence-file2 --smodel 1:TN --smodel 2:GTR -t
```

4.4. Option files (Scripts)

In addition to using the command line, you may also specify options in a file. Using an option file can be more convenient if you are going to run the same analysis many times, or if the number of options is large. Furthermore, the option file may contain comments and blank lines. Option files are a good to record what options you used in an analysis, and why.

An option file is specified with the command line option **`--config file`** or **`-cfile`**. If values for an option are given both on the command line and in an option file, then the command line value overrides the value in the option file.

4.4.1. Syntax

Option files use the same option names as the command line. However, the syntax is different: each option is given on its own line using the syntax "**`option = value`**" instead of the syntax "**`--option value`**". If the option has no value then it is given using the syntax "**`option = option`**".

4.4.2. Example

For example, consider the following option file:

```
#select a data set to analyze
align = examples/EF-Tu/5d.fasta

#select an substitution model
smodel = log-normal+INV

#fix the alignment and do not model indels
traditional = traditional
```

The first option, **`align`**, is the name of the sequence file, which has no name on the command line. Lines that begin with # are comments, and blank lines are ignored. The option **`--traditional`** uses the option name as the value, because it does not take a value. Thus, this configuration file corresponds to the command line

```
%bali-phy examples/EF-Tu/5d.fasta --smodel log-normal+INV --traditional
```

4.4.3. The configuration file

The file `~/.bali-phy` is a special option file called the *configuration file.* If it exists, it is always loaded. Options given on the command line or an option file override values given in `~/.bali-phy`.

4.5. Examples

Here are some examples which demonstrate how to run BAli-Phy. In order to run these examples, you must find the `examples/` directory which contains the example files. Typically, the `examples/` directory will be found at *prefix*`/share/bali-phy/examples/` if you installed bali-phy in directory *prefix*.

Also note that **bali-phy** does *not* run until it is "finished", but continues to gather samples until the user determines that enough samples have been gathered, and stops it. Thus, it is useful to continually examine the output files while the program is running.

Example 1. No frills

Here we analyze the EF-Tu 5-taxon data set provided with the software.

```
% bali-phy somewhere/examples/EF-Tu/5d.fasta
```

Example 2. Multiple-Rate Substitution Model

We now modify the previous example by changing the substitution model to allow log-normal-distributed rate variation and invariant sites. The amount of rate variation and the fraction of invariant sites are estimated

```
% bali-phy somewhere/examples/EF-Tu/5d.fasta --smodel log-normal+INV --randomize-alignmen
```

Example 3. Fixed alignment

Here we use the 5S rRNA 5-taxon data set provided with the software. The alignment is fixed and the traditional likelihood model is used, making indels non-informative. In addition, the transition kernel which samples nucleotide frequencies is disabled, thus fixing the nucleotide frequencies to empirical values estimated from the input sequences.

```
% bali-phy somewhere/examples/5S-rRNA/5d.fasta --smodel F=constant --traditional
```

4.6. Command-line options: An overview

You can get an up-to-date overview of these command line options by running **bali-phy --help**.

4.6.1. General options

`-h, --help`	Show help message.
`-v, --version`	Show version information.
`-c file, --config file`	Option file to read.
`--show-only`	Analyze initial values and exit.
`--seed seed`	Use the specified seed to initialize the random number generator.
`--name string`	Specify the name for the analysis directory.
`-t, --traditional`	Fix the alignment and don't model indels.

4.6.2. MCMC options

`-i, --iterations number=100000`	Specify the number of iterations to run.
`--pre-burnin iterations=3`	Iterations to refine initial tree.
`--subsample factor=1`	Specify a factor by which to subsample.
`--enable move`	Enable a comma-separated list of transition kernels.

`--disable` *move*	Disable a comma-separated list of transition kernels.

4.6.3. Parameter options

`--randomize-alignment`	Randomly re-align sequences before use.
`--tree` *file*	Specify file with initial tree.
`--set` *parameter=value*	Specify initial value of *parameter*.
`--fix` *parameter[=value]*	Mark *parameter* fixed, and optionally specify a value.
`--unfix` *parameter[=value]*	Mark *parameter* not fixed, and optionally specify an initial value.
`--frequencies` *frequencies*	Specify initial frequencies: 'uniform','nucleotides', or a comma-separated list of frequencies.

4.6.4. Model options

`--alphabet` *name*	Specify the alphabet: DNA, RNA, Amino-Acids, Amino-Acids+stop, Triplets, Codons, or Codons + stop.
`--smodel` *name*	Specify the substitution model.
`--imodel` *name*	Specify the indel model.
`--branch-prior` *name*	Exponential or Gamma.
`--same-scale` *specification*	Which partitions have the same scale?
`--align-constraint` *filename*	File with alignment constraints.

5. Input

5.1. Sequence formats

BAli-Phy can read in sequences and alignments in both FastA and PHYLIP formats. Filenames for FastA files should end in **`.fasta`**, **`.mpfa`**, **`.fna`**, **`.fas`**, **`.fsa`**, or **`.fa`**. Filenames for PHYLIP files should end in **`.phy`**. If one of these extensions is not used, then BAli-Phy will attempt to guess which format is being used.

5.2. Is my data set too large?

Large data sets run more slowly than small data sets. We recommend a conservative starting point with few taxa and short sequence lengths. You can then increase the size of your data set until a balance between speed and size is reached.

The number of samples that you need depends on whether you are primarily interested in obtaining a point estimate or in obtaining detailed measures of confidence and uncertainty. For detailed measures of confidence and uncertainty you should obtain a minimum of 10,000 samples after the Markov chain converges. For an estimate, you don't need very many samples after convergence. (But you may need many samples to be sure that you've converged!)

5.2.1. Too many taxa?

BAli-Phy is quite CPU intensive, and so we recommend using 50 or fewer taxa in order to limit the time required to accumulate enough MCMC samples. We recommend initially pruning as many taxa as possible from your data set, then adding some back if the MCMC is not too slow.

5.2.2. Sequences too long?

Aligning just a pair of sequences takes $O(L^2)$ time and memory, where L represents the sequence length. Therefore sequences longer than (say) 1000 letters become increasingly impractical. However, you might try to see how long you can make your sequences before you run out of memory, or the program becomes too slow.

For multi-gene analyses, two separate data partitions (i.e. genes) of 500 letters will be twice as fast to align as one data partition of 1000 letters. So, it may be possible to analyze several genes as long as each gene individually is not too long.

You can speed up alignment for long genes by specifying alignment constraints (See Section 7.4, “Alignment constraints”). Ideally, 10 evenly spaced constraints should reduce the cost of re-aligning a sequence by a factor of 10.

Also, note that you can sometimes speed up the analysis of protein sequences by coding them as amino acids or codons, rather than nucleotides. This is because it decreases the sequence length.

6. Output

6.1. Output directory

BAli-Phy creates a new directory to store its output files each time it is run. By default, the directory name is the name of the sequence file, with a number added on the end to make it unique. BAli-Phy first checks if there is already a directory called *file*`-1/`, and then moves on to *file*`-2/`, etc. until it finds an unused directory name.

You can specify a different name to use instead of the sequence-file name by using the **`--name`** option.

6.2. Output files

BAli-Phy writes the following output files inside the directory that it creates:

C1.out	Iteration numbers, probabilities, success probabilities for transition kernels, etc..
C1.P*p*.fastas	Sampled alignments for partition *p*
C1.err	You can ignore this file.
C1.MAP	Successive estimates of the MAP point (if you have only one partition).
C1.p	Scalar parameters: indel and substitution parameters, etc.
C1.trees	Tree samples

For the last two files, each line in these files corresponds to one iteration.

6.3. Summarizing the output

This section is primarily oriented to extracting estimates from output files. See Section 8, “Convergence and Mixing: Is it done yet?” for methods of determine effective sample sizes, and for checking mixing and convergence.

6.3.1. Finding the consensus tree (`C1.trees`)

To compute the majority consensus tree do:

```
% trees-consensus --skip=burnin C1.trees --consensus-PP=0.5:c50.PP.tree
```

Here the **`0.5`** indicates the PP threshold -- it specifies a consensus tree containing only splits with probability 0.5 or higher. The option **`:c50.PP.tree`** indicates the output file to store the tree. Leaving off the ":*`filename`*" or specifying ":-" sends the output to the terminal. You can supply multiple levels and filenames separated by commas.

The program FigTree [http://tree.bio.ed.ac.uk/software/figtree/] allows you to view the consensus tree graphically.

6.3.2. Finding the M.A.P. topology (`C1.trees`)

To compute the *maximum a posteriori* tree topology do:

```
% trees-consensus --skip=burnin C1.trees --map-tree=MAP.tree
```

The MAP topology may be used instead of a consensus tree when a fully resolved (e.g. bifurcating) tree is required. However, when the topology has many tips, each topology may be sampled only once, leading to low quality estimates of the MAP topology.

The program FigTree [http://tree.bio.ed.ac.uk/software/figtree/] allows you to view the consensus tree graphically.

6.3.3. Checking topology convergence (`C1.trees`)

```
% trees-bootstrap dir1/C1.trees dir2/C1.trees
```

This command computes the effective sample size for the posterior probability of each split. It also computes the Average Standard Deviation of Split Frequencies (ASDSF) between two or more independent runs.

See Section 8, “Convergence and Mixing: Is it done yet?” for more information.

6.3.4. Summarizing numerical parameters (`C1.p`)

This command gives a median and confidence interval, ESS, and a stabilization time:

```
% statreport C1.p > Report
```

This command compares multiple runs to give PSRF and joint ESS values as well:

```
% statreport C1.p C2.p > Report
```

The program Tracer [http://tree.bio.ed.ac.uk/software/tracer/] allows you to view the same summaries graphically.

See Section 8, “Convergence and Mixing: Is it done yet?” for more information.

6.3.5. Computing an alignment using Posterior Decoding (`C1.P`*`p`*`.fastas`)

```
% cut-range --skip=burn-in < C1.Pp.fastas | alignment-max > Pp-max.fasta
```

You can use the program seaview [http://pbil.univ-lyon1.fr/software/seaview.html] to view the alignment graphically.

6.3.6. Find the alignment from the maximum a posterior (MAP) point (`C1.MAP`)

```
% alignment-find < C1.MAP > P1-MAP.fasta
```

6.3.7. Create an Au (Alignment Uncertainty) plot (`C1.P`*p*`.fastas`)

To annotate a specific alignment *`alignment`*.fasta, choose a fully resolved tree estimate *`tree`*:

```
% cut-range --skip=burn-in < C1.Pp.fastas | alignment-gild alignment.fasta tree  > alignme
% alignment-draw alignment.fasta --AU alignment-AU.prob > alignment-AU.html
```

The majority consensus tree is usually not fully resolved, so we recommend using the MAP topology instead.

6.4. Summarizing the output - scripted

Instead of manually running each of the steps to analyze the output files, you may instead run the PERL script **bp-analyze.pl** to execute these commands. You may run it inside the output directory, like this:

```
% bp-analyze.pl --burnin=iterations
```

The script will create an HTML page `Results/index.html` that summarizes the posterior distribution.

You may also run it with one or more output directories as arguments, like this:

```
% bp-analyze.pl --burnin=iterations directory-1/ directory-2/
```

In this case, output from multiple runs will be used to assess convergence and mixing, as well as to increase the precision of the estimates.

6.4.1. Meaning of generated files

The `Results/` directory will contain the following useful files:

Report	A summary of numerical parameters: credible intervals and mixing.
consensus	A summary of supported splits (clades).
c-levels.plot	The number of splits (clades) supported at each LOD level.
c50.tree	The majority consensus topology + branch lengths (Newick format)
c50.PP.tree	The majority consensus topology + branch lengths + Posterior Probabilities (Newick format)
MAP.tree	An estimate of the MAP topology + branch lengths (Newick format)

The following files will be generated to summarize alignment uncertainty, unless the analysis uses a fixed alignment.

MAP.fasta	An estimate of the MAP alignment.
P*p*-max.fasta	An estimate of the alignment for partition *p* using maximum posterior decoding.
MAP-AU.html	An AU plot of the MAP alignment (AA/DNA color-cheme).

P*p*-max-AU.html — An AU plot of the maximum posterior decoding alignment for partition *p* (AA/DNA color-cheme).

The following files describe convergence and mixing:

partitions.bs — Confidence intervals on the support for partitions, generated using a block bootstrap.

partitions.SRQ — A collection of SRQ plots for the supported partitions.

c50.SRQ — An SRQ plot for the majority consensus tree.

The SRQ plots can be viewed by typing "**`plot 'file' with lines`**" in gnuplot.

6.4.2. `Mixing/partitions.bs`: partition mixing

This file reports the quality of estimates of support for each partition in terms of the posterior probability (PP) and log-10 odds (LOD). It also reports the auto-correlation time (ACT), the effective sample size (Ne), the number of samples that support (1) or do not support (0) the partition, and the number of regenerations. Only partitions with PP > 0.1 are shown by default.

7. Models

7.1. Substitution models

Substitution models in BAli-Phy are specified using a stack, as follows: **`Model[arg]+Model[arg]+...+Model[arg]`** where each model uses the previous models as input. For example, **`WAG+gamma[4]+INV`**. Arguments are optional.

> **Note**
>
> If you are using the C-shell command line shell (**csh** or **tcsh**), then it will try to interpret each argument as an array reference, giving the error message "bali-phy: Not found." To avoid this you may need to insert backslashes before the left square brackes, like this: **`Model\[arg]+Model\[arg]+...+Model\[arg]`**.

7.1.1. Default substitution models

If the substitution model is not specified, then the default model for the alphabet is used. For DNA or RNA, the default model is TN. For Triplets, the default is TNx3. For Codons, the default model is M0. For Amino-Acids, the default model is LG.

7.1.2. Basic CTMC models

The basic substitution models in BAli-Phy are continuous-time Markov chains (CTMC). CTMC models can be characterized by transition rates Q_{ij} from letter i to letter j. After a given time t the probability for transition from state i to state j is given by

$$P(t)_{ij} = e^{Q_{ij} \times t}$$

using a matrix exponential. Becase the CTMC models used in BAli-Phy are all reversible, the rate matrix for these reversible models can be decomposed into a symmetric matrix S and equilibrium frequencies π as follows:

$Q_{ij} = S_{ij} \times \pi_j$

The matrix S is called the exchangability matrix, and represents how exchangeable letters i and j are, independent of their frequencies.

The basic CTMC models are EQU, HKY, TN, GTR, HKYx3, TNx3, GTRx3, JTT, WAG, LG, and M0. Each of these models is a way of specifying the exchangeability matrix S_{ij}.

Table 1. Substitution Models

Model	Alphabet	Parameters	Description
`EQU`	any	none	$S_{ij} = 1$ for every i and j.
`HKY` Hasegawa, Kishino, Yano (1985)	DNA or RNA	κ: the ts/tv ratio.	$S_{ij} = 1$ for transversions. $S_{ij} = \kappa$ for transitions.
`TN` Tamura, Nei (1993)	DNA or RNA	κ_1: the purine ts/tv ratio. κ_2: the pyrimidine ts/tv ratio.	$S_{ij} = 1$ for transversions. $S_{ij} = \kappa_1$ for purine transitions. $S_{ij} = \kappa_2$ for pyrimidine transitions.
`GTR` General Time-Reversible Tavare (1986)	DNA or RNA	$S_{i \neq j}$	$\sum_{i \neq j} S_{ij} = 1$. (5 degrees of freedom).
`JTT` Jones, Taylor, Thornton (1992)	Amino-Acids	none.	
`WAG` Whelan and Goldman (2001)	Amino-Acids	none.	
`LG` Le and Gascuel (2008)	Amino-Acids	none.	
`Empirical[`*`file`*`]`	Amino-Acids	none.	A user-specified empirical exchangeability matrix may be used. The lower-triangular part of the symmetric matrix is given, followed by the estimated equilibrium frequencies.
`HKYx3` **`TNx3`**	Triplets	*`nuc-model`* parameters.	If the nuc-model has transition matrix S_{ij} on nucleotides, then:

Model	Alphabet	Parameters	Description
GTRx3			$S_{\alpha\beta}=0$ for changes of more than one nucleotide. $S_{\alpha\beta}=S'_{ij}$ for single nucleotide changes $i \to j$.
M0 Nielsen and Yang (1998)	Codons	κ: the ts/tv ratio. ω: the dN/dS ratio.	$S_{\alpha\beta}=0$ for changes of more than one nucleotide. $S_{\alpha\beta}=1$ for synonymous transversions. $S_{\alpha\beta}=\omega$ for non-synonymous transversions. $S_{\alpha\beta}=\kappa$ for synonymous transitions. $S_{\alpha\beta}=\omega\kappa$ for non-synonymous transitions.
M0[*nuc-model=HKY*] Nielsen and Yang (1998)	Codons	*nuc-model* parameters. ω: the dn/ds ratio	If the nuc-model has transition matrix S'_{ij} on nucleotides, then: $S_{\alpha\beta}=0$ for changes of more than one nucleotide. $S_{\alpha\beta}=S'_{ij}$ for synonymous changes. $S_{\alpha\beta}=\omega S'_{ij}$ for non-synonymous changes.

7.1.3. Substitution Frequency models

The rate matrix Q_{ij} can be more generally expressed as

$$Q_{ij}=S_{ij}\times\frac{\pi_j^f}{\pi_i^{1-f}},$$

where f ranges from 0 to 1. Here the parameter f specifies the relative importance of unequal conservation ($f=0$) and unequal replacement ($f=1$) in maintaining the equilibrium frequencies π.

In fact, this can be generalized even further to

$$Q_{ij}=S_{ij}\times R(\pi)_{ij}$$

where

$\pi_i \times R_{ij} = \pi_j \times R_{ji}$.

These models can therefore be expressed as a combination of an "exchange model" (for S) and a "frequency model" (for R).

Table 2. Frequency Models

Model	Alphabet	Parameters	Description
F Simple frequency model	any	f (1) π ($\|\alpha\|$)	$R_{ij} = \frac{\pi_j^f}{\pi_i^{1-f}}$.
F=nucleotides Independent nucleotide frequency model	Triplets	f (1) π_N (4)	$\pi_\alpha = \pi_i \pi_j \pi_k$ $R_{\alpha\beta} = \frac{\pi_\beta^f}{\pi_\alpha^{1-f}}$.
F=amino-acids Amino-acid based codon frequencies. (no codon bias)	Codons	f (1) π_{AA} (20)	$R_{ij} = \frac{\pi_j^f}{\pi_i^{1-f}}$.

7.1.4. Substitution Mixture Models

Complex substitution models in BAli-Phy are constructed as mixtures of reversible CTMC models (see Section 7.1, “Substitution models”) that run at different rates (e.g. $\Gamma_4 + \mathrm{INV}$) or have different parameters (e.g. an M2 codon model).

Model modifiers are gamma, log-normal, INV, M2, M3, and M7.

Table 3. CTMC Mixture Models

Model	Alphabet	Parameters	Description
***sm* + INV**	*sm* alphabet	p: invariant fraction.	A fraction p of sites do not allow substitutions.
***sm* + gamma[*n*]** Yang (1994)	*sm* alphabet	σ/μ: noise to signal ratio for Γ.	rate $\sim \Gamma(\mu = 1, \sigma)$. A discrete approximation to the Γ with n bins is used.
***sm* + log-normal[*n*]**	*sm* alphabet	σ/μ: noise to signal ratio for $\mathrm{logNormal}$.	rate $\sim \mathrm{logNormal}(\mu = 1, \sigma)$. A discrete approximation to the $\mathrm{logNormal}$ with n bins is used.
M2 ***sm* + M2** Yang, et. al. (2000)	Codons	κ: the ts/tv ratio p_1, p_2, p_3: bin frequencies. ω_3: value of ω in bin 2.	$\Omega = \omega_i$ with probability p_i. $\omega_1 = 0,\ \omega_2 = 1$. The default for *sm* is M0.
M3[*n*]	Codons	κ: the ts/tv ratio	$\Omega = \omega_i$ with probability p_i.

Model	Alphabet	Parameters	Description
***sm* + M3[*n*]** Yang, et. al. (2000)		$\rho_1, \ldots, \rho_n$: bin frequencies. $\omega_1, \ldots, \omega_n$: values of ω.	
M7[*n*] ***sm* + M7[*n*]** Yang, et. al. (2000)	Codons	μ: mean of the Beta distribution. σ/μ: noise to signal ratio for Beta.	$\Omega \sim \text{Beta}(\mu, \sigma)$. A discrete approximation to the Beta with n bins is used.

7.1.5. Substitution model examples

Example: --smodel WAG+F+log-normal+INV

Example: --smodel WAG+log-normal+INV (same as above)

Example: --smodel EQU --alphabet Triplets

Example: --smodel HKY

Example: --smodel TN+F=constant

Example: --smodel M0 --alphabet Codons

Example: --smodel M0+F=nucleotides --alphabet Codons

Example: --smodel M2 --alphabet Codons

Example: --smodel M0[HKY]+M2 --alphabet Codons (same as above)

Example: --smodel M0[TN]+M2 --alphabet Codons

7.2. Insertion/deletion models

The current models are RS05, RS07, and **none**. The default is RS07. Each of these models is a probability distribution on pairwise alignments. The probability distribution on multiple sequence alignments $\Pr(A|T, \tau, \Lambda)$ is constructed by factoring the multiple sequence alignment into pairwise alignments along each branch of the tree, as described in Redelings and Suchard (2005).

Table 4. Substitution Models

Model	Parameters	Description
RS05 Redelings and Suchard (2005)	δ: the gap-opening probability ϵ: the gap-extension probability	Gap lengths are geometrically distributed with extension probability ϵ. This indel model is independent of the branch length connecting the ancestor and descent sequences.
RS07 Redelings and Suchard (2007)	λ: the insertion and deletion rate ϵ: the gap-extension probability	Gap lengths are geometrically distributed with extension probability ϵ.

Model	Parameters	Description
		This probability of an indel event depends on the branch length in this model.
none		Indicates the lack of a model.

Specifying an indel model of **none** for a given partition results in fixing the alignment for that partition to its initial value, and ignoring information in shared insertions or deletions.

7.3. Genetic Codes

When using a codon-based substitution model like **M0**, you may select the genetic code by specifying **--alphabet Codons[*genetic-code*]**. Available genetic codes are **standard**, **mt-vert**, **mt-invert**, **mt-yeast**, **mt-protozoan**.

If the genetic code is not specified, then the standard code is used:

```
% bali-phy sequence-file --smodel M0 --alphabet Codons
```

This example specifies the vertebrate mitochondrial code:

```
% bali-phy sequence-file --smodel M0 --alphabet Codons[mt-vert]
```

7.4. Alignment constraints

To fix specific columns of the alignment, you may specify alignment constraints in a file as follows:

1. Use the argument **--align-constraint *filename***

2. The filename refers to a file in which each line represents a constraint.

7.4.1. Syntax

The first line of the file is a header consisting of an ordered list of sequence names separated by spaces. Each subsequent line consists of a space-separated list of sequence positions, with the first position corresponding to the first leaf sequence, the second position corresponding to the second leaf sequence, etc. Thus, if there are *n* leaf taxa, then each line corresponds to a space-separated list of *n* integers.

7.4.2. Examples

For example, the file

```
A B C
1 2 2
```

implies that position 1 of leaf sequence A is aligned to position 2 of leaf sequences B and C. Note that the first position in a sequence is position 0.

Optionally, one may use a '-' instead of an integer, which denotes a lack of constraint for that sequence. This can be useful as follows:

```
A B C D
2 2 - -
- - 2 2
```

The above constraints force alignment between position 2 of sequences A and B, and between position 2 of sequence C and D.

7.4.3. Computing the constraints

The program **alignment-indices** may be used to aid in computing a constraint file from an input alignment. See Q: 11.5.3.

8. Convergence and Mixing: Is it done yet?

When using Markov chain Monte Carlo (MCMC) programs like MrBayes, BEAST or BAli-Phy, it is hard to determine in advance how many iterations are required to give a good estimate. The number depends on the specific data set that is being examined. As a result, BAli-Phy relies on the user to analyze the data in a running chain periodically in order to determine when enough samples have been obtained. This section describes a number of techniques to diagnose when more samples must be taken.

Some of the better diagnostics for lack of convergence rely on running at least 4 independent copies of the Markov chain (preferably 10) from different random starting points to see if the sampled posterior distributions for each chain are the same. Unfortunately, when the distributions all seem to be this same, this doesn't *prove* that they have all converged to the equilibrium distribution. However, if the distributions are different then you can reject either convergence or good mixing.

8.1. Definition of Convergence

Convergence refers to the the tendency of a Markov chain to to "forget" its starting value and become typical of its equilibrium distribution. Note that convergence is a property of the Markov chain itself, not of individual runs of the Markov chain. Ideally a number of individual runs should be examined in order to determine how many initial iterations to discard as "burnin".

8.2. Definition of Mixing

In MCMC, each sample is not fully independent of previous samples. In fact, even after a Markov chain has convergence, it can get "stuck" in one part of the parameter space for a long time, before jumping to an equally important part. When this happens, each sample contributes very little new information and we need to obtain many more samples to obtain good precision on our estimates. In such a case, we say that the chain isn't "mixing" well.

8.3. Diagnostics and multiple independent chains

Many of the following diagnostic measures require that you run the MCMC a number of different times. In the text that follows, we refer to the different parameter files as `C1.p`, `C2.p`, ..., `Cn.p` and the different tree samples as `C1.trees`, `C2.trees`, ..., `Cn.trees`. Substitute the actual names of the files.

8.4. Diagnostics: Variation in split frequencies across runs

8.4.1. ASDSF and MSDSF

To calculate the ASDSF and MSDSF run:

```
% trees-bootstrap C1.trees C2.trees ... Cn.trees > partitions.bs
```

For each split, the SDSF value is just the standard deviation across runs of the Posterior Probabilities for that split. By averaging the resulting SDSF values across splits, we may obtain the ASDSF value (Huelsenbeck and Ronquist 2001). This is commonly considered acceptable if it is < 0.01.

However, it is also useful to consider the maximum of the SDSF values (MSDSF). This represents the range of variation in PP across the runs for the split with the most variation.

8.4.2. Split-frequency comparison plot

To generate the split-frequency comparison plot, you must have R installed. Locate the script `compare-runs.R`. Then run:

```
% trees-bootstrap C1.trees C2.trees ... Cn.trees --LOD-table=LOD-table > partitions.bs
% R --slave --vanilla --args LOD-table compare-SF.pdf < compare-runs.R
```

Following Beiko et al (2006) [http://dx.doi.org/10.1080/10635150600812544], this displays the variation in estimates of split frequencies across runs. Splits are arranged on the x-axis in increasing order of Posterior Probability (PP), which is obtained by averaging over runs. We then plot a vertical bar from the minimum PP to the maximum PP.

8.5. Diagnostics: Potential Scale Reduction Factors (PSRF)

Potential Scale Redution Factors check that different runs have similar posterior distributions. Only numerical variables may have a PSRF. To calculate the PSRF for each numerical parameter, you may run:

```
% statreport C1.p C2.p ... Cn.p > Report
```

The PSRF is a ratio of the width of the pooled distribution to the average width of each distribution, and should ideally be 1. The PSRF is customarily considered to be small enough if it is less than 1.01.

We compare the PSRF based on the length of 80% credible intervals (Brooks and Gelman 1998) and report the result as PSRF-80%CI. For integer-valued parameters, we avoid excessively large PSRF values by subtracting 1 from the width of the pooled CI.

We also report a new PSRF that is more sensitive for integer distributions. For each individual distribution, we find the 80% credible interval. We divide the probability of that interval (which may be more than 80%) by the probability of the same interval under the pooled distribution. The average of this measure over all distributions gives us a PSRF that we report as PSRF-RCF.

This convergence diagnostic gives a criterion for detecting when a parameter value has stabilized at different values in several independent runs, indicating a lack of convergence. This situation might occur if different runs of the Markov chain were trapped in different modes and failed to adequately mix between modes.

8.6. Diagnostics: Effective sample sizes (ESS)

8.6.1. ESS for numerical values

To calculate the split ESS values, run:

```
% statreport C1.p C2.p ... Cn.p > Report
```

We calculate effective sample sizes based on integrated autocorrelation times. This method has the nice property that simply doubling every sample does not increase the ESS.

The program Tracer [http://evolve.zoo.ox.ac.uk/software/tracer/] also computes ESS values.

8.6.2. ESS for split frequencies

To calculate the split ESS values, run:

```
% trees-bootstrap C1.trees C2.trees ... Cn.trees > partitions.bs
```

To compute the ESS for a split, we consider the presence or absence of a split in each iteration as a series of binary values. We compute the integrated autocorrelation time for this binary sequence, which leads to an ESS. This approach is similar to dividing the iterations into blocks and computing the ESS on the PP estimates in the blocks. It is also similar to estimating the variance reduction under a block bootstrap.

8.7. Diagnostics: Stabilization

8.7.1. Stabilization of numerical values

To obtain estimates of the stabilization time for each numerical parameter, you may run:

```
% statreport C1.p > Report
```

Each series of values is counted as having stabilized after the series crosses its upper and then lower 95% confidence bounds twice (if the initial value is below the median) or crosses its lower and then upper confidence bounds twice (if the initial value is above the median). The confidence bounds are those based on its equilibrium distribution as calculated from the last third of the values in the sequence.

8.7.2. Stabilization of tree topologies and tree distances

In addition to examining convergence diagnostics for continuous parameters, it is important to examine convergence diagnostics for the topology as well (Beiko et al 2006 [http://dx.doi.org/10.1080/10635150600812544]). In theory, we recommend the web tool Are We There Yet (AWTY) [http://ceb.csit.fsu.edu/awty/] (Wilgenbush et al, 2004). However, AWTY gives incorrect results if you upload plain NEWICK tree samples -- which is what BAli-Phy outputs. Therefore, if you wish to use AWTY, you must convert the tree samples files to NEXUS before you upload them to AWTY in order to get correct results.

It is also be possible to assess stabilization of tree topologies using tools distributed with bali-phy by using commands like the following. Here, sub-sampling and burnin does not apply to the equilibrium tree files. Also, note that you need to manually construct the equilibrium samples, which we recommend to contain at least 500 trees; you might do this by sub-sampling using the BAli-Phy tool **sub-sample**.

1. To report the average distances within and between two tree samples:

   ```
   % trees-distances --skip=burnin --sub-sample=factor compare C1.trees C2.trees
   ```

2. To compute the distance from each tree in C1.trees to all trees equilibrium.trees, as a time series:

   ```
   % trees-distances --skip=burnin --sub-sample=factor convergence C1.trees equilibrium.tre
   ```

3. To assess when the above time series stabilizes:

   ```
   % trees-distances --skip=burnin --sub-sample=factor converged C1.trees equilibrium.trees
   ```

   The stabilization criterion is the same one described above for numerical values.

Note that the running time is the product of the number of trees in the two files. Therefore, comparing two complete tree samples without sub-sampling will take too long.

9. Tuning the Markov Chain

The Markov chain should be largely self-tuning, since all numerical parameters are now sampled using self-tuning slice samplers. However, the following parameters still affect the size of Metropolis-Hastings proposals. You can modify them using the command line syntax "**`--set parameter=value`**".

9.1. Parameters

Table 5. Tunable Parameters

Name	Variable	Default	Meaning
log_branch_sigma	branch lengths	0.6	Scale of log-proposal.
branch_sigma	branch lengths	0.6	Scale of non-log-proposal.
mu_scale_sigma	mu	0.6	Width of proposal on log scale.
kappa_scale_sigma	HKY::kappa, TN::kappa(pur), TN::kappa(pyr)	0.3	Width of proposal on log scale.
omega_scale_sigma	M0::omega, M2::omega	0.3	Width of proposal on log scale.
beta::mu_scale_sigma	beta::mu	0.2	Width of proposal.
INV::p_shift_sigma	INV::p	0.03	Width of proposal.
gamma::sigma_scale_sigma	gamma::sigma/mu	0.25	Width of proposal of log scale.
pi_dirichlet_N	pi*	1.0	Tightness of dirichlet proposal for frequencies.
lambda_shift_sigma	delta, lambda	0.35	Width of proposal.
epsilon_shift_sigma	epsilon	0.15	Width of proposal.

10. Auxiliary tools

Most of these tools will describe their options if given the "**`--help`**" argument on the command line.

10.1. alignment-find

Usage: alignment-find [OPTIONS] < *`alignments-file`*

Find the last (or first) FastA alignment in a file.

10.2. alignment-draw

alignment-draw *`alignment-file`* [*`AU-file`*] [OPTIONS]

Draw an alignment to HTML, optionally coloring residues by AU.

10.3. alignment-thin

alignment-thin *`alignment-file`* *`tree-file`* [OPTIONS]

Remove taxa from an alignment to preserve the most sequence diversity, as measured by the total length of the tree for the remaining taxa.

10.4. alignment-chop-internal

alignment-chop-internal *alignment-file* [OPTIONS]

Remove ancestral sequences from an alignment. (This probably only works for alignments output by bali-phy.)

10.5. alignment-info

alignment-info *alignment-file* *tree-file* [OPTIONS]

Display basic information about the alignment, including its length, the number of sequences, columns that are constant or informative, letter frequencies, etc.

If a tree is supplied, then the unweighted parsimony score is given as well.

10.6. alignment-indices

alignment-indices *alignment-file* [OPTIONS]

Show the alignment in terms of the index of each character in its sequence. Each line in this file corresponds to one alignment column. This can be useful in producing alignment constraint files.

Also, you can specify which columns to keep using the --columns option.

10.7. alignment-cat

alignment-cat *file1* [*file2* ...]

Concatenate several alignments (with the same sequence names) end-to-end.

10.8. trees-consensus

Usage: trees-consensus *file* [OPTIONS]

This program analyzes the tree sample contained in *file*. It reports the MAP topology, the supported taxa partitions (including partial partitions), and the majority consensus topology.

10.9. trees-bootstrap

Usage: trees-bootstrap *file1* [*file2* ...] --predicates *predicate-file* [OPTIONS]

This program analyzes the tree samples contained in *file1*, *file2*, etc. It gives the support of each tree sample for each predicate in *predicate-file*, and reports a confidence interval based on the block bootstrap.

Each predicate is the intersection of a set of partitions, and is specified as a list of partitions or (multifurcating) trees, one per line. Predicates are separated by blank lines.

10.10. trees-to-SRQ

Usage: trees-to-SRQ *predicate-file* [OPTIONS] *trees-file*

This program analyzes the tree samples contained in ***trees-file***. It uses them to produce an SRQ plot for each predicate in ***predicate-file***. Plots are produced in gnuplot format, with one point per line and with plots separated by a blank line.

If **--mode sum** is specified, then a "sum" plot is produced instead of an SRQ plot. In this plot, the slope of the curve corresponds to the posterior probability of the event. If the **--invert** option is used then the slope of the curve correspond to the probability of the inverse event. This is recommended if the probability of the event is near 1.0, because the sum plot does not distinguish variation in probabilities near 1.0 well.

11. Frequently Asked Questions (FAQ)

11.1. Running bali-phy.

11.1.1. Can I fix the alignment and ignore indel information, like MrBayes and other MCMC programs?

Yes. Add **-t** or **--traditional** on the command line. This has the same effect as **--imodel none**.

11.1.2. Can I fix the tree topology, while allowing the alignment to vary?

Yes. Add **--disable=topology --tree=*treefile*** on the command line.

11.1.3. Can I fix the tree topology and *relative* branch lengths, while allowing the alignment to vary?

Yes. Add **--disable=tree --tree=*treefile*** on the command line.

11.1.4. Can I fix the tree topology and *absolute* branch lengths *in all data partitions*, while allowing the alignment to vary?

Yes. Add **--disable=tree --tree=*treefile* --fix=mu1 ... --fix-mu*n*** on the command line.

11.2. Run-time error messages

11.2.1. I tried to use **--smodel gamma[6]** and I got an error message "bali-phy: No match." What gives?

You are probably using the C-shell as your command line shell. It is trying to interpret **gamma[6]** as an array before running the command, and it is not succeeding. Therefore, it doesn't even run **bali-phy**.

To avoid this, put a backslash in front of the first "[" and write **--smodel gamma\[6]**. This will keep the C-shell from interfering with your command.

11.3. Stopping bali-phy.

11.3.1. Why is **bali-phy** still running? How long will it take?

It runs until you stop it. Stop it when its done.

11.3.2. So, how can I know when to stop it?

You can stop when it has both converged and also run for long enough to give you >1000 effectively independent samples.

11.3.3. How can I tell when the chain has converged?

See section Section 8, “Convergence and Mixing: Is it done yet?”.

11.3.4. How can I check how many iterations the chain has finished?

Run **wc -l C1.p** inside the output directory, and subtract 2.

11.4. Interpreting the results.

11.4.1. How do I compute the clade support?

Actually, BAli-Phy uses unrooted trees, so it only estimates bi-partition support. A bi-partition is a division of taxa into two groups, but it does not specify which group contains the root.

11.4.2. How do I compute the split/bi-partition support?

After you analyze the output (Section 6.4, “Summarizing the output - scripted”), the partition support is indicated in `Results/consensus` and in `Results/c50.PP.tree`.

11.5. How do I...

11.5.1. How do I concatenate alignments?

```
%alignment-cat filename1.fasta filename2.fasta > result.fasta
```

The alignments must have the same sequence names, but the names need not be in the same order.

11.5.2. How do I select columns from an alignment?

```
%alignment-cat -c1-10,50-100,600- filename.fasta > result.fasta
```

The resulting alignment will contain the selected columns in the order you specified.

11.5.3. How do I create an alignment-constraint file from an alignment?

To constrain the alignment to match some alignment file *`filename.fasta`* in columns 100, 200-250, and 300, run:

```
%alignment-indices -c100,200-250,300 filename.fasta > filename.constraint
```