`BAli-Phy` User's Guide v4.0

Benjamin Redelings

Table of Contents

1. Introduction

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. Therefore, you might want to keep a Unix tutorial or Unix cheat sheet handy while you work.

BAli-Phy analyses have two phases. (This structure is common to all Bayesian analyses.) First the **bali-phy** program generates *posterior samples* of trees, alignments and parameters. Second, the **bp-analyze** script creates *posterior summaries* that collapse the collection of posterior samples down to single trees, alignments, and parameter estimates. It also diagnoses *lack of convergence*.

In addition to the main **bali-phy** executable, *BAli-Phy* comes with a collection of small command-line utilities such as **alignment-cat**, **trees-consensus**, etc. These utilities can be used to process alignments, assemble data sets, and summarize the results of MCMC.

2. Installation

2.1. Hardware requirements

We typically run *BAli-Phy* on workstations with at least 16Gb of RAM and 4 cores. More cores will allow you to run more MCMC chains at once, and more RAM will allow you to run larger data sets. However, it is often easier and faster to run BAli-Phy on a (Linux) computing cluster, if you have one available.

2.2. Upgrades

If you have previously installed bali-phy, you do not have to remove the old version before installing the new version. Simply follow the installation instructions for the new version. If you are manually adding the new version of bali-phy to your PATH, just make sure that the new version comes before the old version in the PATH, or remove the old version from the PATH.

In order to remove an older version, simply delete the directory `bali-phy-`*`oldversion`*. This will completely uninstall the old version from the system. BAli-Phy does not create hidden files that will remain after you remove its directory.

2.3. Install on MS Windows

2.3.1. Install a Unix command line: WSL2 (recommended)

We recommend installing WSL2. Then you can install bali-phy inside the WSL2 Linux installation and follow the instructions for Linux users.

2.3.2. Install a Unix command line: Cygwin

It is also possible to install native windows executables. However, you will still need a Unix command-line environment. We recommend installing Cygwin:

- Run the Cygwin installer `setup-x86_64.exe`.
- In the installer, add the following extra packages: **`R`**, **`gnuplot`**, **`perl`**, **`python3`**, **`wget`**, and **`nano`**.

Its easiest to find extra packages if you set the View to "Full" and enter each package name in the Search box. After you run the installer, you can access the Unix command line environment by running the Cygwin shell (not the normal windows command line). You can run the installer again to add more packages.

The native windows version of BAli-Phy uses Windows-style filenames (such as `C:\`). However, the Cygwin shell uses UNIX-style filenames, so you will need to keep the difference in mind:

UNIX-style	Windows-style
/home/*username*	C:\cygwin64\home\username
~/file	C:\cygwin64\home\username\file

/cygdrive/c/file	C:\file

You can use the `cygpath` program to convert between UNIX and Windows filenames:

```
% cygpath -w ~/Applications
C:\cygwin64\home\username\Applications\
```

Note

If you supply UNIX-style filenames to BAli-Phy, then it will complain "**file does not exist!**".

2.4. Install on Mac OS X

2.4.1. Install BAli-Phy using homebrew (recommended)

First install the XCode (version 15 or higher) command line tools:

```
% xcode-select --install
```

Then install homebrew and use homebrew to compile and install **bali-phy**:

```
% brew tap brewsci/bio
% brew install bali-phy
```

Check that the executable runs:

```
% bali-phy --version
```

If you install with homebrew, you don't need to do anything extra to put bali-phy in your PATH.

2.4.2. Install BAli-Phy using executables from website (alternative)

Open a windows in the Terminal app to access the UNIX command line. Then download and extract the executables. If you have an Apple Silicon computer, type:

```
% mkdir -p ~/Applications
% cd ~/Applications
% curl -LO https://github.com/bredelings/BAli-Phy/releases/download/4.0/bali-phy-4.0-mac-arm64.tar.gz
% tar -zxf bali-phy-4.0-mac-arm64.tar.gz
```

If you have an older Intel computer, type:

```
% mkdir -p ~/Applications
% cd ~/Applications
% curl -LO https://github.com/bredelings/BAli-Phy/releases/download/4.0/bali-phy-4.0-mac-intel64.tar.gz
% tar -zxf bali-phy-4.0-mac-intel64.tar.gz
```

Check that the executable runs:

```
% ~/Applications/bali-phy-4.0/bin/bali-phy --version
```

You still need to add it to your PATH as described in Section 2.6, “Add BAli-Phy to your PATH”.

2.4.3. Install programs used by bp-analyze using homebrew

You can install `gnuplot` via homebrew:

```
%  brew install gnuplot
```

You can install *R* via homebrew:

```
%  brew tap caskroom/cask
%  brew cask install xquartz
%  brew install r
```

However, note that this might conflict with R installed from other places, such as MRAN.

2.4.4. Install some of the programs used for viewing the results using homebrew

You can install Figtree with homebrew:

```
% brew install --cask figtree
```

However, Seaview and Tracer don't have homebrew packages at the moment.

2.5. Install on Linux

2.5.1. Download BAli-Phy executables from website

First install **wget**. If you have Debian or Ubuntu Linux, type:

```
% sudo apt-get install wget
```

Then download and extract the executables. If you have a recent Linux installation, type:

```
% mkdir -p ~/Applications
% cd ~/Applications
% wget https://github.com/bredelings/BAli-Phy/releases/download/4.0/bali-phy-4.0-ubuntu-24.04.tar.gz
% tar -zxf bali-phy-4.0-ubuntu-24.04.tar.gz
```

If you have an older Linux installation, type:

```
% mkdir -p ~/Applications
% cd ~/Applications
% wget https://github.com/bredelings/BAli-Phy/releases/download/4.0/bali-phy-4.0-ubuntu-22.04.tar.gz
% tar -zxf bali-phy-4.0-ubuntu-22.04.tar.gz
```

Second, check that the executable runs:

```
% ~/Applications/bali-phy-4.0/bin/bali-phy --version
```

Third, add BAli-Phy to your PATH as described in Section 2.6, “Add BAli-Phy to your PATH”.

Fourth, test the software as described in Section 2.7, “Test the installed software”.

2.5.2. Install BAli-Phy using apt-get (alternative)

BAli-Phy packages exists for Ubuntu ("Cosmic Cuttlefish" or later), and Debian (testing and unstable). However, these may not be up-to-date.

```
% sudo apt-get install bali-phy
```

Check that the executable runs:

```
% bali-phy --version
```

If you install with **apt-get**, you don't need to do anything extra to put bali-phy in your PATH.

2.5.3. Install programs used by bp-analyze

If you have Debian or Ubuntu Linux, you can install other recommended programs by typing:

```
% sudo apt-get install gnuplot
% sudo apt-get install r-base
```

2.5.4. Install programs used to view the results

```
% sudo apt-get install seaview
% sudo apt-get install figtree
```

However, there isn't a Debian or Ubuntu package for Tracer at the moment.

2.6. Add BAli-Phy to your `PATH`

2.6.1. Is bali-phy in your PATH already?

First check if the executable is in your PATH.

```
% bali-phy --version
```

If this shows version info, then **bali-phy** is already in your PATH and you can skip this section. This should be true if you installed **bali-phy** using a package manager such as homebrew or apt, or if you've already added it to your PATH.

If bali-phy is not in your path, then you should see:

```
% bali-phy --version
bali-phy: command not found.
```

If bali-phy is not in your PATH, then continue with this section.

2.6.2. Quick version

Add **bali-phy** to your PATH, so that the shell knows where to find it. This command only affects the terminal in which it is typed, and will not affect new terminals:

```
% export PATH=~/Applications/bali-phy-4.0/bin:$PATH
```

To set the PATH automatically for new terminals, type:

```
% test -r ~/.bash_profile && echo 'export PATH=~/Applications/bali-phy-4.0/bin:$PATH' >> ~/.bash_profile
% echo 'export PATH=~/Applications/bali-phy-4.0/bin:$PATH' >> ~/.profile
```

This will affect new terminals only after you log out and log back in though.

Now check that the executable runs:

```
% bali-phy --version
```

If it does, then your PATH is set up correctly, and you can probably skip the rest of this section.

2.6.3. I have a path?

If you installed *`BAli-Phy`* to the directory `~/Applications`, then you can run bali-phy by typing **~/Applications/bali-phy-4.0/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 "path" is a colon-separated list of directories that is searched to find program names that you type. It is stored in an environment variable called `PATH`.

Setting your `PATH` is also a pre-requisite for running the **bp-analyze** script to summarize your MCMC runs.

2.6.4. Examining your `PATH`

You can examine the current value of this environment variable by typing:

```
% echo $PATH
```

We will assume that you extracted the bali-phy archive in `~/Applications` and so you want to add `$HOME/Applications/bali-phy-4.0/bin` to your `PATH`. (If you installed to another directory, replace `$HOME/Applications/bali-phy-4.0/` with that directory.)

2.6.5. Adding BAli-Phy to your `PATH`

The commands for doing this 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=$HOME/Applications/bali-phy-4.0/bin:$PATH
```

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

```
% setenv PATH $HOME/Applications/bali-phy-4.0/bin:$PATH
```

Note that these commands will only affect the window you are typing in, and will vanish when you reboot.

2.6.6. Making the change stick

To make this change survives when you logout or reboot, open your shell configuration file in a text editor, and add the command on a line by itself. This will ensure that it is run every time you log in.

To find the right configuration file, look in your $HOME directory for `.profile` (for the Bourne shell **sh**), `.bash_profile` (for BASH), or `.login` (for tcsh). You may have to create the file if it is not present. On Cygwin, you should put the change in the file `.bashrc`.

If you do not know which directory is your home directory, you can find its full name by typing:

```
% echo $HOME
```

2.7. Test the installed software

In order to determine that the software has been correctly installed, and the `PATH` has been correctly set, run the following commands:

```
% cp ~/Applications/bali-phy-4.0/share/doc/bali-phy/examples/sequences/5S-rRNA/25.fasta .
% bali-phy --version
% bali-phy help
% bali-phy 25.fasta --iter=200
% bali-phy 25.fasta --iter=200
% bp-analyze 25-1 25-2
```

Then check that the file `Results/index.html` exists and can be opened in a web browser.

2.8. Install programs used for viewing the results

- Tracer : MCMC parameter/diagnostic viewer.

  Check by opening: `25-1/C1.log` and `25-2/C1.log`

- FigTree : Phylogeny Viewer

  Check by opening: `Results/c50.PP.tree`

- SeaView and/or AliView : Alignment viewers.

  Check by opening: `Results/P1.max.fasta`

3. Running the program

BAli-Phy analyses have two phases. (This structure is common to all Bayesian analyses.) First the **bali-phy** program generates *posterior samples* of trees, alignments and parameters. Second, the **bp-analyze** script creates *posterior summaries* that collapse the collection of posterior samples down to single trees, alignments, and parameter estimates. It also diagnoses *lack of convergence*.

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

```
% bali-phy sequences.fasta
```

You can run a traditional fixed-alignment Bayesian tree inference by adding `-I none`:

```
% bali-phy sequences.fasta -I none
```

You can also specify a character set for analysis:

```
% bali-phy sequences.fasta:1-30,90-100
```

3.1. Quick Start

Let's run an example analysis using the 5S-rRNA 25-taxon data set.

```
% cp ~/Applications/bali-phy-4.0/share/doc/bali-phy/examples/sequences/5S-rRNA/25.fasta .
```

We will now start 4 simultaneous runs:

```
% bali-phy 25.fasta -S 'gtr +> Rates.free +> Covarion.hb02' --iter=1000 &
% bali-phy 25.fasta -S 'gtr +> Rates.free +> Covarion.hb02' --iter=1000 &
% bali-phy 25.fasta -S 'gtr +> Rates.free +> Covarion.hb02' --iter=1000 &
% bali-phy 25.fasta -S 'gtr +> Rates.free +> Covarion.hb02' --iter=1000 &
```

These runs will all execute at the same time because of the "`&`". Each run will create a unique directory of the form `25-`*`number`* to store its results.

You can use the program `top` to verify that four copies of `bali-phy` are running:

```
% top                                    # use q to exit
```

We can use program Tracer to assess the progress of the runs by loading the files `25-1/C1.log`, `25-2/C1.log`, `25-3/C1.log`, and `25-4/C1.log`. All four log files should be loaded simultaneously in order to compare them.

When enough iterations have finished, we then run the script `bp-analyze` to summarize the results:

```
% bp-analyze 25-1/ 25-2/ 25-3/ 25-4/
Creating new directory 'Results' for summary files.
Summarizing distribution of numerical parameters: done.
Analyzing scalar variables: done.

Summarizing topology distribution:  done.
Drawing trees: c50 c66 c80 c90 c95 c99 c100 greedy MAP . done.

Generate mixing diagnostics for topologies ... done.
Generate SRQ plot for partitions: done.
Generate SRQ plot for c50 tree: done.

Generate MDS plots of topology burnin:  done.
Computing initial alignments:  done.
Computing WPD alignments:  done.
Computing ancestral state alignment:  done.
Drawing alignments: **** done.
Generating AU values for 'P1.initial'... done.
Generating AU values for 'P1.max'... done.

NOTE: burnin (scalar) <= Not Converged!
NOTE: min_ESS (scalar)    = 96.73
NOTE: min_ESS (partition)    = 96.278
NOTE: ASDSF = 0.035
NOTE: MSDSF = 0.117
NOTE: PSRF-80%CI = 1.077
NOTE: PSRF-RCF = 1.256

RUN 1: directory = 25-1     iterations = 1000     burnin = 100
RUN 2: directory = 25-2     iterations = 1000     burnin = 100
RUN 3: directory = 25-3     iterations = 1000     burnin = 100
RUN 4: directory = 25-4     iterations = 1000     burnin = 100

Report written to 'Results/index.html'
```

Load the file `Results/index.html` in a web browser to view the results. On Linux you can type:

```
% firefox Results/index.html
```

The tree estimate, alignment estimate, mixing diagnostics and other information will be displayed in the HTML report. The HTML report contains links to FASTA and Newick files in the `Results/` directory.

We can also view the tree and alignment estimates directly:

- The majority consensus tree is in the file `Results/c50.PP.tree`. It can be viewed with *`Figtree`*.
- The consensus alignment is in the file `Results/P1.max.fasta`. It can be viewed with *`Seaview`* or *`Aliview`*.

See section Section 4.2, “Posterior summaries” for further description of the files in the `Results/` directory.

3.2. Input

`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.

FASTA format prefixes sequence names with ">":

```
>human       this is a comment and is not part of the sequence name
CTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAA
GTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTT
>tarsier     this is also a comment
CTGACTGCTGAAGAGAAGGCCGCCGTCACTGCCCTGTGGGGCAAGGTAGACGTGGAAGAT
GTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTCGTCTACCCATGGACCCAGAGGTTCTTT
>bushbaby
CTGACTCCTGATGAGAAGAATGCCGTTTGTGCCCTGTGGGGCAAGGTGAATGTGGAAGAA
GTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTTGTCTACCCATGGACCCAGAGGTTCTTT
>hare
CTGTCCGGTGAGGAGAAGTCTGCGGTCACTGCCCTGTGGGGCAAGGTGAATGTGGAAGAA
GTTGGTGGTGAGACCCTGGGCAGGCTGCTGGTTGTCTACCCATGGACCCAGAGGTTCTTC
```

If the sequence-name line contains a space, BAli-Phy treats everything after the space as a comment.

The sequences in the file do not need to be aligned unless you fix the alignment with **`-I none`**.

3.3. Command line options

Sensible defaults are supplied for command line options that are not specified. For example, if `sequences.fasta` contains DNA sequences, then

```
% bali-phy sequences.fasta
```

is equivalent to

```
% bali-phy sequences.fasta -A DNA -S tn93 -I rs07
```

Default values that are used will always be displayed on the screen and in the output files so that you do not have to guess. You can specify a more complex substitution model using the `-S` option. You will generally need to write the substitution model inside single quotes unless it is just a single word.

```
% bali-phy sequences.fasta -S 'lg08 +> Rates.gamma +> inv'
```

Every short option like `-S` has an equivalent long option like `--smodel`. To see the most frequently-used command-line options, you can run

```
% bali-phy help
```

3.4. Option files (Scripts)

In addition to using the command line, you may also specify options in a file. Option files also use the long form of command line options. Each option is given on its own line using the syntax "`:option value`" instead of the syntax "`--option value`". The value can be blank if the option does not take an argument. The `align` option indicates sequence files. Lines that begin with # are comments, and blank lines are ignored.

For example, consider the following option file:

```
# sequence data for 3 genes/partitions
:align ITS1.fasta
:align 5.8S.fasta
:align ITS2.fasta

# linked substitution model for 1st and 3rd partition
:smodel 1,3:tn93  +>  Rates.free(n=3)

# substitution model for 2nd partition
:smodel 2:tn93

# indel model for second partition
:imodel 2:none

# linked scale for 1st and 3rd partition
:scale 1,3:

# choose a name for output directories
:name ITS-analysis1
```

Options files are specified with the `-c option_file` option:

```
% bali-phy -c analysis1.txt                        # run the analysis
% bali-phy -c analysis1.txt --name ITS-analysis1b     # override the name
```

Options given on the command line will override values given in the option file.

3.5. Running on computing clusters

Running **bali-phy** on a computing cluster is not necessary, but can speed up the analysis dramatically. This is because a cluster allows you to run several *independent* MCMC chains simultaneously and pool the resulting samples. You can run multiple chains simultaneously simply by starting several different instances of **bali-phy**. Each instance of bali-phy runs only one chain and does not require using MPI or special command-line options.

This approach to parallel computation is sometimes more efficient than MCMCMC-based parallelism involving heated chains. It is equivalent to running MCMCMC with no temperature difference between chains, with the exception that it allows results from *all* chains to be used, instead of just results from the single "cold" chain. Thus, if you run 10 independent chains in parallel, then you may gather samples 10 times faster than a single chain.

3.6. Is my data set too large?

Bayesian inference programs must run for many iterations to complete an analysis. A data set is considered "too large" if waiting for it to complete takes "too long".

3.6.1. Too many sequences?

Bayesian phylogenetics analyses require more iterations to converge as the number of sequences increases. Additionally, the computing time for each iteration increases with the number of sequences.

BAli-Phy has been successfully used to compute the full posterior with up to 150 sequences. Additionally, it has been used with up to 500 sequences to obtain alignment estimates that are more accurate than alignments from other software, but without measures of uncertainty.

If you have many sequences, we recommend using the tool *`alignment-thin`* with the **`--down-to=`*`n`*** option to construct a preliminary data set of 30-60 sequences. It is described in section Section 11, “Alignment utilities: brief overview” . Analyzing such a data set can complete much more quickly. You can then increase the size of your data set until a balance between speed and usefulness is reached.

3.6.2. 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 slow.

One solution to this problem is to divide a long gene into multiple partitions. Dividing a long gene into *`n`* partitions will be roughly *`n`* times as fast as a single partition. The downside of this approach is that it requires performing a preliminary alignment, perhaps with a different aligner, in order to identify the partition boundaries.

When the multiple partitions can be categorized as introns or exons, then this approach allows treating intron and exon regions differently. It is possible to link all the intron partitions and link all the exon partitions, so that introns have one evolutionary rate and exons have another. Likewise, it is possible to fix the alignment for the exons, but infer the alignment for the introns. A similar approach can be taken with RNA stem and loop regions.

4. Output

BAli-Phy analyses have two phases. (This structure is common to all Bayesian analyses.) First the **bali-phy** program generates *posterior samples* of trees, alignments and parameters. Second, the **bp-analyze** script creates *posterior summaries* that collapse the collection of posterior samples down to single trees, alignments, and parameter estimates. It also diagnoses *lack of convergence*.

4.1. Posterior samples

4.1.1. Output directories

`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.

4.1.2. Output files

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

C1.P*n*.fastas	Sampled alignments for partition *n* including ancestral sequences.
C1.MAP	Successive estimates of the MAP alignment, tree and parameters.
C1.log	Numeric parameters: indel and substitution rates, etc. (*One sample per line.*)
C1.trees	Tree samples in Newick format. (*One sample per line.*)
C1.run.json	JSON file containing information about the command line, models, hostname, start time, etc.

4.1.3. Field names in `C1.log`

This section explains the meaning of the various field names in the file `C1.log`.

prior	The log prior probability.
likelihood	The log likelihood.
posterior	The log of the posterior probability. (*The posterior probability is the product of the prior and the likelihood*).
prior_A	The log-probability of the alignments in all partitions.
|A|	The total number of alignment columns across all partitions.
#indels	The total number of indel events across all partitions. (*Adjacent indels that occur on the same branch are merged*).
|indels|	The total length of indel events across all partitions. (*Adjacent indels that occur on the same branch are merged*).
#substs	The total unweighted parsimony score for substitutions across all partitions.
P*n*/likelihood	The substitution log-likelihood for partition *n*.
P*n*/prior_A	The log-probability of the alignment for partition *n*.
P*n*/|A|	The length of the alignment in the *n*th partition.
P*n*/#indels	The number of indel events in partition *n*, if we group adjacent indels that occur on the same branch.
P*n*/|indels|	The length of indel events in partition *n*, if we group adjacent indels that occur on the same branch.
P*n*/#substs	The unweighted parsimony score for substitutions in partition *n*.
Scale[*m*] * |T|	The *scaled* branch lengths for partition group *m*.
|T|	The *unscaled* tree length. (This will probably be around 1.0).
Scale[*m*]	The average number of substitutions per site on the entire tree for partitions in the *m*th scale group.
S*n*/*name*	Parameter *name* in the *n*th substitution model.
I*n*/*name*	Parameter *name* in the *n*th insertion/deletion model.

The "prior" field includes the probability of the alignment, since the alignment is not observed.

The likelihood is the probabilistic analogue to summed mismatch penalties.

The prior_A is the probabilistic analogue to summed gap penalties.

The prefixes "S*n*/" and "I*n*/" will be dropped if not necessary to disambiguate parameters with the same name in different sub-models.

4.2. Posterior summaries

The **bp-analyze** script summarizes the posterior samples to create posterior summaries for the alignment, tree, and parameters. It creates an HTML page `Results/index.html` that summarizes the posterior distribution.

You may run **bp-analyze** inside the output directory, like this:

```
% bp-analyze --skip=iterations
```

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

```
% bp-analyze --skip=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.

All the commands that are executed by **bp-analyze** will be logged to `Results/commands.log`. You can also see these commands as they are executed by supplying the **--verbose** option:

```
% bp-analyze --skip=iterations --verbose
```

4.2.1. 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.

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

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`*.

4.2.2. `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.

4.3. Posterior summaries (Advanced)

This section is primarily about summarizing the posterior to extract estimates from posterior samples, not about assessing convergence. See Section 10, “Convergence and Mixing: Is it done yet?” for methods of determining effective sample sizes, and for checking mixing and convergence.

4.3.1. Finding the majority consensus tree

To compute the majority consensus tree, do the following. (The program FigTree allows you to view the resulting tree file graphically.)

```
% trees-consensus dir-1/C1.trees dir-2/C1.trees > c50.PP.tree
```

By default, the first 10% of tree samples are skipped as burn-in (**`--skip=10%`** or **`-s 10%`**) and every generation is analyzed (**`--subsample=1`** or **`-x 1`**). To discard the first 1000 tree samples and analyze every 10th sample:

```
% trees-consensus -s 1000 -x 10 dir-1/C1.trees dir-2/C1.trees > c50.PP.tree
```

By default, splits are included in the consensus tree if they have a PP greater than 0.5. You can specify a more stringent level (e.g. 0.66) by adding the option **`--consensus-PP=0.66`** as follows:

```
% trees-consensus -s20% -x10 --consensus-PP=0.66 dir-1/C1.trees dir-2/C1.trees > c66.PP.tree
```

You may also make the program write directly to the output file (e.g. `c66.PP.tree`) by using the more general form **`--consensus-PP=0.66:c66.PP.tree`**. Leaving off the "**`:c66.PP.tree`**" part (as we did above) or specifying "**`:-`**" sends the output to the standard output (e.g. the terminal, if not redirected).

```
% trees-consensus -s20% -x10 dir-1/C1.trees dir-2/C1.trees --consensus-PP=0.66:c66.PP.tree
```

You can supply multiple levels and filenames separated by commas. This is faster than running the program multiple times with different consensus levels.

```
% trees-consensus -s20% -x10 dir-1/C1.trees dir-2/C1.trees --consensus-PP=0.5:c50.PP.tree,0.66:c66.PP.tree
```

Finally, you may use the option **`--consensus=`** instead of the option **`--consensus-PP=`** if you do not wish the resulting tree to contain embedded posterior probabilities on branches, as well as branch lengths.

```
% trees-consensus -s20% -x10 dir-1/C1.trees dir-2/C1.trees --consensus=0.5:c50.PP.tree,0.66:c66.PP.tree
```

Both the **`--consensus=`** and **`--consensus-PP=`** options may be given simultaneously.

See **`trees-consensus --help`** for a complete list of options.

4.3.2. Finding the greedy consensus tree

The greedy consensus tree may be used instead of a majority-consensus tree when a fully resolved (e.g. bifurcating) tree is required. When the topology has many tips and each topology may be sampled only once, the greedy consensus should be higher quality than the estimate of the MAP topology. To obtained a fully resolved tree, the greedy consensus strategy starts with the majority consensus and then adds the highest-supported split that does not conflict.

To compute the *greedy consensus* tree do:

```
% trees-consensus --skip=burnin dir-1/C1.trees dir-2/C1.trees --greedy-consensus=greedy.tree
```

4.3.3. Finding the M.A.P. tree

To compute the *maximum a posteriori* tree do:

```
% trees-consensus --skip=burnin dir-1/C1.trees dir-2/C1.trees --map-tree=MAP.tree
```

When the tree has many tips, each topology may be sampled only once, leading to low quality estimates of the MAP topology. As a result, when you need a bifurcating tree you should probably use the greedy consensus instead.

4.3.4. Checking topology convergence

```
% trees-bootstrap dir-1/C1.trees dir-2/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 10, "Convergence and Mixing: Is it done yet?" for more information.

4.3.5. Summarizing numerical parameters

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

```
% statreport dir-1/C1.log dir-2/C1.log > Report
```

When multiple runs are analyzed, this command gives PSRF and joint ESS values. The program Tracer allows you to view the same summaries graphically.

See Section 10, "Convergence and Mixing: Is it done yet?" for more information.

4.3.6. Computing an alignment using Posterior Decoding

To construct an alignment estimate via posterior decoding, select any tree file *tree* that corresponds to your alignment. It does not need to be fully resolved.

```
% cut-range dir-1/C1.Pp.fastas dir-2/C1.Pp.fastas --skip=burn-in | alignment-chop-internal --tree tree | alignment-max
> Pp-max.fasta
```

You can optionally replace **`--tree tree`** with **`-N n_sequences`**, where *n_sequences* is the number of non-ancestral sequences in your alignment.

You can use the program SeaView to view the alignment graphically.

4.3.7. Create an Au (Alignment Uncertainty) plot

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

```
% cut-range dir-1/C1.Pp.fastas dir-2/C1.Pp.fastas --skip=burn-in | alignment-chop-internal --tree tree  | alignment-
gild alignment.fasta tree  > alignment-AU.prob
% 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 greedy consensus instead.

5. Substitution models

5.1. DNA and RNA models

The default substitution model for DNA and RNA is tn93.

5.1.1. Substitution rates

All the DNA models are special cases of the GTR model.

Model	d.f.	Summary

`jc69`	0	Equal rates and equal base frequencies. (Jukes and Cantor, 1969)
`k80`	1	Unequal transition & transversion rates, equal base frequencies. (Kimura, 1980)
`f81`	3	Equal exchangeabilities, unequal frequencies. (Felsenstein, 1981)
`hky85`	4	Unequal Transition & transversion rates, unequal base frequencies. (Hasegawa, Kishino, and Yano, 1985)
`tn93`	5	Unequal rates for transitions (purines), transitions (pyrimidines) and transversions, unequal base frequencies. (Tamura and Nei, 1993)
`gtr`	8	Unequal exchangeabilities, unequal frequencies. (Tavare, 1986)

5.1.2. Frequencies

Frequencies are estimated by default. Frequencies can be fixed by setting the `pi` parameter to a constant value, if the model allows unequal frequencies.

Constant frequencies are specified as a list of pairs that associates each letter with its frequency:

```
gtr(pi={"A":0.1, "C":0.2, "T":0.3, "G":0.4})
```

Frequencies can also be specified using functions:

```
gtr(pi=Frequencies.uniform)
```

Model	d.f.	Summary
`Frequencies.uniform`	0	Equal frequencies

5.2. Protein models

The default substitution model for proteins is lg08.

5.2.1. Substitution rates

Model	d.f.	Summary
`jc69`	0	Equal rates and equal frequencies. (Jukes and Cantor, 1969)
`f81`	19	Equal exchangeabilities, unequal frequencies. (Felsenstein, 1981)
`jtt +> f`	19	Empirical exchange rates, all proteins. (Jones, Taylor, and Thornton, 1992)
`wag +> f`	19	Empirical exchange rates, all proteins. (Whelan and Goldman, 2001)
`lg08 +> f`	19	Empirical exchange rates, all proteins.

		(Le and Gascuel, 2008)
empirical(*file*) +> f	19	
gtr	208	Unequal exchangeabilities, unequal frequencies. (Tavare, 1986)

5.2.2. Frequencies

Frequencies are estimated by default. Frequencies can be fixed by setting the `pi` parameter to a constant value, if the model allows unequal frequencies.

Constant frequencies are specified as a list of pairs that associates each letter with its frequency:

```
wag +> f({"A":0.047, "R":0.19,...})
```

Frequencies can also be specified using functions:

```
wag +> f(pi=Frequencies.uniform)
```

Model	d.f.	Summary
Frequencies.uniform	0	Equal frequencies
wag_freq	0	The constant amino-acid frequencies from the WAG paper.
lg08_freq	0	The constant amino-acid frequencies from the LG08 paper.

The `+> fe` model is shorthand for `+> f(pi=Frequencies.uniform)`:

```
wag +> fe
```

5.3. Doublet models (RNA stems)

The doublets alphabet consists of 16 RNA dinucleotides. It is used to model RNA stems, where two nucleotides matched in the RNA secondary structure are highly correlated.

The default substitution model for doublets is `tn93_sym +> x2_sym +> f`.

5.3.1. Doublet data

As of version 3.4, BAli-Phy does not yet allow specifying which nucleotides are paired either with a string like `((.))` or with a "pairs" file. Instead you must manually extract the paired nucleotides and put them in their own partition (for stems), and then manually extract each loop and put it in its own partition.

The stems should be arranged so that paired nucleotides are adjacent. For example, suppose the sequence `AGGCT` was paired according to `((.))`. Then the input file for the stems should contain a sequence of doublets that looks like `ATGC`, where `AT` is the first pair, and `GC` is the second pair. Later versions of the software should allow extracting stems and loops from nucleotide sequences using parenthesis notation or a "pairs" file.

5.3.2. Substitution rates

Model	d.f.	Summary
***nuc_model* +> x2**	df(nuc_model)	The the same as *`nuc_model`*, but on dinucleotides instead of nucleotides. Simultaneous changes of both letters are *not*

		allowed. Dinucleotide frequencies are the product of independent nucleotide frequencies.
***nuc_model* +> x2 +> mut_sel**	df(nuc_model)+15	Mutation-selection model: neutral mutation follows *nuc_model* and scaled selection coefficients 2Ns on dinucleotides. Simultaneous changes of both letters are *not* allowed.
***nuc_model* +> x2_sym +> f**	df(nuc_model)+15	This model has separate frequencies for each dinucleotide. Simultaneous changes of both letters are *not* allowed.
RNA.m16a	19	This model has separate frequencies for each dinucleotide, and distinguishes between transitions and transversion between match states (including GU/UG). Simultaneous changes of both letters *are* allowed, but only between match states. (Savill et al., 2001)
gtr	134	Unequal exchangeabilities, unequal frequencies. It is unlikely that you would want to use this model, since it has so many parameters. (Tavare, 1986)

5.3.3. Frequencies

Frequencies are estimated by default. Frequencies can be fixed by setting the **pi** parameter to a constant value, if the model allows unequal frequencies.

Constant frequencies are specified as a list of pairs that associates each letter with its frequency.

```
hky85(pi={"A":0.1, "C":0.2, "T":0.3, "G":0.4}) +> x2

hky85_sym +> x2_sym +> f({"AA":0.01, "AC":0.01, "AG":0.01, "AU":0.22, "CA":0.01, "CC":0.01, "CG":0.22, "CU":0.01,
"GA":0.01, "GC":0.22, "GG":0.01, "GU":0.01, "UA":0.22, "UC":0.01, "UG":0.01, "UU":0.01})
```

Frequencies can also be specified using functions:

Model	d.f.	Summary
Frequencies.uniform	0	Equal frequencies on dinucleotides

5.3.4. Branch lengths

BAli-Phy interprets branch lengths for doublet models as 1/2 the number of substitutions per doublet. Thus, they should be comparable to branch lengths under DNA/RNA nucleotide models.

5.4. Triplet models

The triplets alphabet is similar to the codons alphabet, except that stop codons are included. Unlike the codons alphabet, the triplets alphabet has no knowledge of the genetic code.

The default substitution model for triplets is tn93 +> x3.

5.4.1. Substitution rates

Model	d.f.	Summary
***nuc_model* +> x3_sym +> f**	df(*nuc_model*)+63	GY94-style rate matrix constructed from nucleotide exchangeability matrix.
***nuc_model* +> x3**	df(*nuc_model*)	MG94-style rate matrix constructed from nucleotide rate matrix. This model should give the same likelihood as *nuc_model* on triplets, but not on codons.
***nuc_model* +> x3 +> mut_sel**	df(*nuc_model*)+63	Mutation-selection model with neutral mutation following *nuc_model* and scaled selection coefficients 2Ns *for each codon.*

5.4.2. Frequencies

Frequencies are estimated by default. Frequencies can be fixed by setting the `pi` parameter to a constant value, if the model allows unequal frequencies.

Constant frequencies are specified as a list of pairs that associates each letter with its frequency.

```
hky85(pi={"A":0.1, "C":0.2, "T":0.3, "G":0.4}) +> x3
```

Frequencies can also be specified using functions:

```
hky85_sym +> x3_sym +> f(pi=f1x4)           // nucleotide frequencies are estimated
```

Model	d.f.	Summary
Frequencies.uniform	0	Equal frequencies
f1x4	3	Constructs triplet frequencies from independent nucleotide frequencies.
f3x4	9	Constructs triplet frequencies from independent nucleotide frequencies for each codon position.

The `+> fe` model is shorthand for `+> f(pi=Frequencies.uniform)`:

```
hky85_sym +> x3_sym +> fe
```

BAli-Phy interprets branch lengths for codon models as 1/3 the number of substitutions per triplet. Thus, they should be comparable to branch lengths under DNA/RNA nucleotide models.

5.5. Codon models

The default substitution model for codons is gy94.

5.5.1. Standard models

Model	d.f.	Summary
gy94	62	Model of dN/dS with a separate frequency for each codon. Rate for changing a nucleotide depends on neighboring nucleotides. (Goldman and Yang, 1994)

`gy94(pi=f1x4)`	5	The GY94 model with codon frequencies constructed from nucleotide frequencies. (Goldman and Yang, 1994)
`gy94(pi=f3x4)`	11	The GY94 model with codon frequencies constructed from nucleotide frequencies for each codon position. (Goldman and Yang, 1994)
`gy94_ext(`*`nuc_model`*`)`	df(*nuc_model*)+61	GY94 model extended with a generic nucleotide exchangeability matrix. (Goldman and Yang, 1994)
`mg94`	4	Model of dN/dS with f81 as the neutral model. Rate for changing a nucleotide depends only on that nucleotide. (Muse and Gaut, 1994)
`mg94k`	5	Model of dN/dS with hky85 as the neutral model. (Muse and Gaut, 1994)
`mg94_ext(`*`nuc_model`*`)`	df(*nuc_model*)+1	Model of dN/dS with *`nuc_model`* as the neutral model. (Muse and Gaut, 1994)
`fMutSel`	65	MG94-like model with fitnesses for each codon. (Yang and Nielsen, 2008)
`fMutSel0`	24	MG94-like model with fitnesses for each amino-acid. (Yang and Nielsen, 2008)
`busted`	24	MG94-like model where each branch and site has a randomly chosen dN/dS category. (Murrell et al, 2015)
`busted_s`	13	BUSTED model with synonymous rate variation. (Wisotsky et al, 2020)

BAli-Phy interprets branch lengths for codon models as 1/3 of the number of substitutions per codon. Thus, they should be comparable to branch lengths under DNA/RNA models.

5.5.2. Constructing and tweaking codon models

Sometimes the researcher wants to use an existing codon model, but with a slight tweak. For example, you might want to use an M3 model that is based on GTR instead of HKY. This section provides building blocks to construct codon models from nucleotide models. This allows you to change the underlying nucleotide model or add other modifications.

Here are some examples for how to construct classic codon models piecewise:

- `mg94` is equivalent to `f81 +> x3 +> dNdS`.
- `mg94k` is equivalent to `hky85 +> x3 +> dNdS`.
- `gy94` is equivalent to `hky85_sym +> x3_sym +> f +> dNdS`.
- `fMutSel` is equivalent to `gtr +> x3 +> dNdS +> mut_sel`.
- `fMutSel0` is equivalent to `gtr +> x3 +> dNdS +> mut_sel_aa`.
- `m3` is equivalent to `|w: gy94(w)| +> m3`.
- `m3_test` is equivalent to `|w: gy94(w)| +> m3_test`.
- `busted` is equivalent to `|w:gtr +> x3 +> dNdS(omega=w)| +> m3_test(n=2) +> branch_site_mixture`.
- `busted_s` is equivalent to `|w:gtr +> x3 +> dNdS(omega=w)| +> m3_test(n=2) +> branch_site_mixture +> Rates.gamma(n=3)`.

Model	d.f.	Summary

***nuc_model* +> x3_sym +> f**	df(nuc_model)+60	GY94-style rate matrix constructed from nucleotide exchangeability matrix (dN/dS = 1). This model should give the same likelihood as *nuc_model* on codons only if the frequency of stop codons is zero.
***nuc_model* +> x3**	df(nuc_model)	MG94-style rate matrix constructed from nucleotide rate matrix (dN/dS = 1). Constructs either a codon model or a triplet model, depending on the alphabet.
***nuc_model* +> mnm**	df(nuc_model)	Like **x3**, but with multi-nucleotide mutations.
***codon_model* +> dNdS(omega)**	df(*codon_model*)+1	Scales non-synonymous rates by *omega*.
***codon_model* +> mut_sel**	df(*codon_model*)+60	Mutation-selection model with neutral mutation following *codon_model* and scaled selection coefficients 2Ns *for each codon*.
***codon_model* +> mut_sel_aa**	df(*nuc_model*)+19	Mutation-selection model with neutral mutation following *nuc_model* and scaled selection coefficients 2Ns *for each amino acid*.
***codon_mixture_model* +> branch_site_mixture**	df(*codon_mixture_model*)	Codon model where each the rate matrix for each branch and site is chosen from the mixture distribution.

5.5.3. Frequencies

Frequencies are estimated by default. Frequencies can be fixed by setting the `pi` parameter to a constant value, if the model allows unequal frequencies.

Constant frequencies are specified as a list of pairs that associates each letter with its frequency.

```
gy94(pi={"AAA":0.01, "C":0.02,...})
mg94(pi={"A":0.1, "C":0.2, "T":0.3, "G":0.4})
```

Frequencies can also be specified using functions:

```
gy94(pi=f1x4)              // nucleotide frequencies are estimated
```

Model	d.f.	Summary
Frequencies.uniform	0	Equal frequencies
f1x4	3	Constructs codon frequencies from independent nucleotide frequencies.
f3x4	9	Constructs codon frequencies from independent nucleotide frequencies for each codon position.

5.5.4. Genetic Codes

When using a codon-based substitution model like `gy94`, you may select the genetic code by specifying **`-A "Codons(,`*`genetic-code`*`)"`**. Available genetic codes are:

Name	Number	Description

standard	1	Standard
mt-vert	2	Mt: Vertebrate
mt-yeast	3	Mt: Yeast
mt-protozoa	4	*: Mold, Protozoan and Coelenterate Mitochondrial Code and Mycoplasma/Spiroplasma
mt-invert	5	Mt: Invertebrate
nuc-ciliate	6	Nuc: Ciliate, Dasycladacean and Hexamita
mt-echinoderm	9	Mt: Echinoderm and Flatworm
nuc-euplotid	10	Nuc: Euplotid
bacteria	11	*: Bacterial, Archaeal and Plant Plastid
nuc-yeast-alt	12	Nuc: Alternative Yeast
mt-ascidian	13	Mt: Ascidian
mt-flatworm-alt	14	Mt: Alternative Flatworm
nuc-blepharisma	15	Nuc: Blepharisma Nuclear Code
mt-chlorophycean	16	Mt: Chlorophycean
mt-trematode	21	Mt: Trematode
mt-scenedesmus-obliquus	22	Mt: Scenedesmus obliquus
mt-thraustochytrium	23	Mt: Thraustochytrium
mt-rhabdopleuridae	24	Mt: Rhabdopleuridae
bacteria-sr1	25	*: Candidate Division SR1 and Gracilibacteria
nuc-pachysolen-tannophilus	26	Nuc: Pachysolen tannophilus
nuc-karyorelict	27	Nuc: Karyorelict
nuc-condylostoma	28	Nuc: Condylostoma
nuc-mesodinium	29	Nuc: Mesodinium
nuc-peritrich	30	Nuc: Peritrich
nuc-blastocrithidia	31	Nuc: Blastocrithidia
mt-cephalodiscidae	33	Mt: Cephalodiscidae UAA-Tyr

Genetic codes may be specified by name or by **`code`*n*** where *n* is the code number. For example **`code1`** is the standard code. If the genetic code is not specified, then the standard code is used:

```
% bali-phy sequence-file -S gy94 -A Codons
% bali-phy sequence-file -S gy94 -A "Codons(RNA)"
```

These examples specify the vertebrate mitochondrial code:

```
% bali-phy sequence-file -S gy94 -A "Codons(DNA,mt-vert)"
% bali-phy sequence-file -S gy94 -A "Codons(,mt-vert)"
```

5.5.5. Heterogeneous dN/dS and tests for positive selection

Model	d.f.	Summary
`m1a`	df(*submodel*)+2	A mixture of conserved and neutral sites. (Wong et al., 2004)
`m2a`	df(*submodel*)+4	A mixture of conserved, neutral, and positively-selected sites. (Wong et al., 2004)
`m2a_test`	df(*submodel*)+4	A Bayesian test for positive selection that compares M2a with M1a. (Wong et al., 2004)
`m3`	df(*submodel*)+2*n-1	An free mixture of n categories of conserved dN/dS values. (Yang et al., 2000)
`m3_test`	df(*submodel*)+2*n+1	A Bayesian test for positive selection based on the M3 model extended with an extra category of either neutral of positively-selected sites.
`m7`	df(*submodel*)+2	The M7 model places a beta distribution on dN/dS. (Yang et al., 2000)
`m8a`	df(*submodel*)+3	The M8a model adds a category of *neutral* sites to the M7 model. (Swanson et al., 2003)
`m8`	df(*submodel*)+4	The M8 model adds a category of *positively-selected* sites to the M7 model. (Yang et al., 2000)
`m8a_test`	df(*submodel*)+4	A Bayesian test for positive selection that compares the M8 to the M8a model. (Swanson et al., 2003)
`branch_site`	df(*submodel*)+4	A Bayesian test for positive selection that on some (unknown) sites and some (known) branches. (Zhang et al., 2005)

5.5.6. The branch-site substitution model

In order to use the branch-site substitution model, the user needs to specify an unrooted tree topology and fix the topology:

```
% bali-phy alignment.fasta -S branch_site --fix topology=treefile
```

The tree file should be in Newick format, with foreground branches labelled using & attributes. The attribute must be applied to the branch, not the node, so it must occur after a colon.

Example 1. An tree with a foreground branch

(((A1, B1),(C1, D1)),((E1:[&foreground=1], F1:[&foreground=1]),(G1, H1)),(((A2, B2),(C2, D2)),((E2, F2),(G2, H2))));

The posterior probability of positive selection is the posterior mean of the posSelection parameter. This may be computed using the statreport program with the `--mean` option. In case this probability is extremely close to 1 or 0, you may wish to add the option `--Rao-Blackwellize branch_site:posSelection`. This will report the log-probability of positive selection each iteration. The user may exponentiate the reported values and then average them (using R, for example) in order to compute a more accurate estimate of the posterior probability of positive selection.

5.6. Heterogenous Rates across Sites

Complex substitution models in *BAli-Phy* are constructed as mixtures of reversible CTMC models that run at different rates (e.g. $\Gamma_4 + \mathrm{INV}$) or have different parameters (e.g. an M2a codon model).

Model	d.f.	Summary
`submodel +> Rates.gamma`	df(*submodel*)+1	Site rates follow a discrete approximation to the Gamma distribution (Yang, 1994)
`submodel +> Rates.logNormal`	df(*submodel*)+1	Site rates follow a discrete approximation to the logNormal distribution
`submodel +> Rates.free`	df(*submodel*)+2(*n*-1)	Sites fall in one of *n* categories. Each category has its own rate. (Yang, 1995)
`submodel +> multi_rate(dist)`	df(*submodel*)+df(*dist*)	Site rates follow a discrete approximation to the distribution *dist*.
`submodel +> inv`	df(*submodel*)+1	Some fraction inv:p_inv of sites are invariable.

5.7. Heterotachy models

These models attempt to model the fact that evolutionary rates may change over time within a single column. These models are sometimes called "covarion" models, based on the idea that changes in rate might be caused by changes in an unspecified covarying site.

These models are "Markov modulated" models that create multiple different states for each letter by augmenting each letter with some unobserved hidden state. They attempt to model the fact that substitution processes might not be Markov on the letters, but might become more Markov given the hidden state.

Model	d.f.	Summary
`Q +> Covarion.ts98`	df(*submodel*)+2	Each state in rate matrix Q is split into an ON and OFF variant. Models burstiness. (Tuffley and Steel, 1998)

`Q +> Rates.gamma +> Covarion.hb02`
`submodel +> Covarion.hb02`
	df(*Q+Rates.gamma*)+2
	df(*submodel*)+2
		Combines Gamma (or other) rate heterogeneity with the Tuffley-Steel model. (Huelsenbeck, 2002)

`Q +> Rates.gamma +> Covarion.gt01`
`submodel +> Covarion.gt01`
	df(*Q+Rates.gamma*)+2
	df(*submodel*)+2
		Allows switching between Gamma (or other) rate classes over time. Models changes in conservation. (Galtier, 2001)

`Q +> Rates.gamma +> Covarion.wssr07`
`submodel +> Covarion.wssr07`
	df(*Q+Rates.gamma*)+4
	df(*submodel*)+4
		Allows switching between ON/OFF states and *also* between Gamma (or other) rate classes over time. Models both burstiness and changes in conservation. (Wang et al., 2007)

Note

Note that the obvious way to combine the Tuffley-Steel model with rate heterogeneity is wrong:

- `Q +> Covarion.ts98 +> Rates.gamma`: This is incorrect. Under this model, sites with faster substitution rates will switch between the ON/OFF states faster.
- `Q +> Rates.gamma +> Covarion.hb02`: This is correct. Sites switch between ON/OFF states independent of the speed of substitution.

6. Insertion/deletion models

Each of these models is a probability distribution on pairwise alignments. The probability distribution on multiple sequence alignments $\Pr(A \mid T, \tau, \Lambda)$ is constructed by factoring the multiple sequence alignment into pairwise alignments along each branch of the tree, as described in Redelings and Suchard (2005).

The default insertion/deletion model is `rs07`.

Model	d.f.	Summary
`rs05`	3	A symmetric insertion-deletion model with geometrically-distributed indel lengths. Indels occur on all branches with the same probability, regardless of branch length. (Redelings and Suchard, 2005)
`rs07`	2	A symmetric insertion-deletion model with geometrically-distributed indel lengths. Longer branches have more indels. (Redelings and Suchard, 2007)
`none`		No indel model for the partition, indels uninformative. Fixed alignment for the partition.

The user can specify priors and parameters for indel models (See section Section 7, “Models and Priors”):

```
rs07(log_rate~logLaplace(-4,0.707),mean_length=2)
```

7. Models and Priors

7.1. Models and distributions are functions

Models, probability distributions, and functions are treated the same in BAli-Phy because all of them have parameters or arguments. Parameters have names in BAli-Phy. Parameter values are specified using square brackets as follows:

```
hky85(kappa=2)             // model
log(x=2)                   // function
normal(mean=0,sigma=1)     // probability distribution
```

It is possible to specify parameter values by position instead of by name:

```
hky85(2)
log(2)
normal(0,1)
```

It is even possible to mix positional and named arguments, as long as all the positional arguments come before all the named arguments:

```
normal(0,sigma=1)   // OK
normal(mean=0,1)    // not OK
```

The order and type of parameters for a function can be found with the **`help`** command. For example,

```
% bali-phy help hky85
```

A value must be given for each parameter, unless the parameter has a default value (See Section 7.4, “Default values and default priors”).

Note

You need to put single quotes around terms with parenthesis or square brackets on the command-line:

```
% bali-phy file.fasta -S 'hky85(kappa=2)'
% bali-phy file.fasta -S 'mixture([tn93,hky85(2)])'
```

If you do not add quotes, the shell will try to interpret the parentheses or square brackets and give an error message without running bali-phy. For example, "`-bash: syntax error near unexpected token `('`" (for **bash**) or "`Badly placed ()'s`" (for **csh**) or "`zsh: no matches found: mixture([tn93,hky85(2)])`" (for **zsh**).

7.2. Model stacking and '+>' notation

Models in phylogenetics literature are often combined using **`+`**. For example, the model **`WAG + F + G4 + I`** starts with the WAG amino-acid model, and places several modifiers, like " + G4" on the right.

BAli-Phy follows this convention by treating **`A +> B`** as an abbreviation for **`B(A)`**. When there are multiple '**`+>`**' symbols they associate to the left, so that **`A +> B +> C`** is understood to mean **`(A +> B) +> C`**, which is equivalent to **`C(B(A))`**. For example:

```
hky85 +> Rates.gamma              // rewritten to Rates.gamma(hky85)
hky85 +> inv                      // rewritten to inv(hky85)
wag +> f                          // rewritten to f(wag)
wag +> f +> Rates.gamma +> inv    // rewritten to inv(Rates.gamma(f(wag)))
```

This allows a simple method for combining models, when one model is an argument to another model.

7.3. Priors

7.3.1. Specifying priors

Priors on model parameters are specified by giving a random value. Random values can be obtained from distributions using the function **`sample`**. For example, this places a log-normal prior on the parameter **`kappa`** of the **`hky85`** model:

```
hky85(kappa=sample(logNormal(1,1)))
```

You can write **`~Dist`** as a shorthand for **`sample(Dist)`**:

```
hky85(kappa = ~logNormal(1,1))
```

The **`=~`** can be further shortened to just **`~`**:

```
hky85(kappa ~ logNormal(1,1))
```

7.3.2. Random function arguments

It also is possible to use random values as inputs to other functions. For example:

```
1.0 + ~exponential(10)
```

In such cases the parameter value should be specified with `=`, as in the following example:

```
rs07(mean_length=1.0 + ~exponential(10))
```

7.3.3. Distributions are not random values

Random values and distributions have different types. For example, the following is of type `Distribution<Double>`:

```
uniform(0,1)
```

In contrast, the following are both of type `Double`:

```
sample(uniform(0,1))
~uniform(0,1)
```

This is important when passing distributions as arguments to other distributions and functions. For example, the distribution `iid` is used to generate a specific number of samples from another distribution. Thus, it needs to receive a distribution as an argument:

```
~iid(4, normal(0,1))      // OK    : 4 samples from the normal(0,1) distribution
~iid(4, ~normal(0,1))     // not OK: 4 samples from ... a random number?
```

(See Section 7.5, “Argument and result types”.)

7.4. Default values and default priors

Some function arguments have default values. For example, the `Rates.gamma` parameter `n` has a default value of 4. Thus the following are equivalent:

```
hky85 +> Rates.gamma(n=4) +> inv
hky85 +> Rates.gamma +> inv
```

When the default value is random, then the argument has a default prior. For example, the `kappa` parameter of `hky85` has a default value of `~logNormal(log(2),0.25)`, so the following are equivalent:

```
hky85(kappa~logNormal(log(2),0.25))
hky85
```

The `help` command can be used to determine the default value for a parameter, if there is one.

7.5. Argument and result types

Every function has a *result type*, as well as an *argument type* for each argument. The argument type specifies what kind of arguments are acceptable, and the result type specifies what kind of result the function produces. Types include `Int` for integers, `Double` for double-precision floating point numbers, and `String` for text strings. Integer arguments are implicitly converted to `Double` when the argument type is `Double`.

Some types contain parameters. For example `List<Int>` indicates a list of integers and `List<Double>` indicates a list of real numbers. In order to indicate a list of unknown type, we use a *type variable* `a` and write `List<a>`. Type variables always begin with a lower-case letter. They are able to match any specific type, and their value is found by pattern-matching. For example, the function `x+y` takes two arguments of type `a` and has a result of type `a`. Thus:

```
1 + 2        // arguments are a=Int, so result is of type Int
1.0 + 2.0    // arguments are a=Double, so result is of type Double
```

`(a,b)` is a parameterized type that can be specialized to (for example) `(String,Double)` and `(Int,Int)`.

Types for components of substitution models are often parameterized by type of the alphabet. For example, hky85 has a result type of `RevCTMC<a>`, where `a` could be `DNA` or `RNA`. The use of alphabet types in substitution models prevents combining substitution models with mismatched alphabets.

8. Partitioned data sets

8.1. Partitions

You should analyze multiple genes under different evolutionary models by putting each one it its own data partition. Placing different genes in different partitions means that their alignments vary independently. It also prevents sequences in one gene from being aligned against sequences in another gene.

Different partitions share the same tree topology and a common set of unscaled branch lengths. However, branch lengths are scaled by a different factor in each partition, since some genes may evolve faster than others.

To put different genes in different partitions, you can place the sequences from each partition in a different FASTA or Phylip file. The sequence names in files for all partitions should be the same.

```
% bali-phy gene1.fasta gene2.fasta
```

You can also select different sites from a single larger file:

```
% bali-phy sequences.fasta:3-350 sequences.fasta:351-570
```

8.2. Unlinked models

By default, each partition will have its own substitution model, insertion/deletion model, and scaled tree length. For example, even if all partitions are assigned a `tn93` substitution model, their base frequencies will all be estimated independently. When parameters are estimated separately for two partitions, we say that the parameters for those partitions are "unlinked".

A substitution model or insertion-deletion model that is specified without qualification will apply to every partition. However, each partition will recieve its own copy of each model with unlinked parameter values:

```
% bali-phy sequence-file1 sequence-file2 -S tn93 -I rs07
```

You can select partition-specific values for 4 options: `-S`, `-I`, `-A`, and `--scale`. For example, to specify different substitution models but the same alphabet:

```
% bali-phy sequence-file1 sequence-file2 -S 1:tn93 -S 2:gtr -A DNA
```

8.3. Fixing the alignment in some partitions

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 -I 2:none
```

Since alignments are estimated by default, the alignment will be estimated in the first partition, but fixed in the second partition.

Specifying specify `-I none` fixes the alignment in all partitions:

```
% bali-phy sequence-file1 sequence-file2 -I none
```

8.4. Linked models

You can also specify that two partitions share a single copy of a single substitution model or indel model. For example, if two partitions both have a `tn93` model, linking these models would force the partitions to have the same nucleotide frequencies and substitution rates. Linking partitions reduces the number of parameters that need to be estimated, and also pools information between the partitions:

```
% bali-phy sequence-file1 sequence-file2 -S 1,2:tn93 -I 1,2:rs07
```

By default each partition has a separate scale, but you can force groups of partitions to share a scale as follows:

```
% bali-phy sequence-file1 sequence-file2 --scale 1,2:
```

8.5. Linking models via the `link` command

The `--link` command is provided to allow specifying a model for each partition separately, and then afterwards choose which partitions to link.

```
% bali-phy sequence-file1 sequence-file2 -S 1:tn93 -S 2:tn93 --link=1,2 -t
% bali-phy sequence-file1 sequence-file2 -S tn93             --link=1,2 -t
```

If the linked partitions are given different models, BAli-Phy will give an error and refuse to run:

```
% bali-phy sequence-file1 sequence-file2 -S 1:tn93 --link=1,2 -t
bali-phy: Error! Partitions 1 and 2 cannot be linked because they have differing values 'tn93' and ''
```

You can also specify which of the 3 attributes "smodel", "imodel", and "scale" are being linked:

```
% bali-phy sequence-file1 sequence-file2 --link=1,2:smodel,scale -t    // Don't link the indel model
```

9. Ancestral sequence reconstruction

9.1. Ancestral sequences with gaps

BAli-Phy can reconstruct ancestral sequences for all internal nodes. Unlike some programs, BAli-Phy explicitly infers the presence and absence of characters in ancestral sequences. This means that if the ancestral sequence has no character for a column, the reconstructed ancestor will have a gap there. BAli-Phy reconstructs ancestors for fixed-alignment partitions as well as variable-alignment partitions, but it won't write out fixed alignment samples unless you add the flag `--set write-fixed-alignments=true`. Additionally, if you have an ambiguous character such as `N` in an observed sequence BAli-Phy will impute this character.

9.2. Generating a consensus alignment with ancestral sequences

BAli-Phy can reconstruct ancestral sequences for a given tree topology and (leaf sequence) alignment. This is similar to the ancestor-reconstruction that is usually done for fixed-alignment analyses. However, it is not quite the same, because of uncertainty in the tree and the alignment. When computing the probability of an ancestral residue, this summary averages over uncertainty in the topology, the alignment, and the ancestral state itself.

```
# Construct the leaf sequence alignment to annotate using posterior decoding
% cut-range dir-1/C1.P1.fastas dir-2/C1.P1.fastas --skip=burn-in | alignment-chop-internal --tree c50.tree | alignment-max > P1-max.fasta
# Construct the tree topology to annotate
```

```
% trees-consensus dir-1/C1.trees dir-2/C1.trees | tree-tool - --strip-internal-names --name-all-nodes > c50.tree
# Reconstruct ancestral sequences on the given tree and alignment
% summarize-ancestors P1.max.fasta -A dir-1/C1.P1.fastas -T dir-1/C1.trees -A dir-2/C1.P1.fastas -T dir-2/C1.trees -n c50.tree -g c50.tree > P1.ancestors.fasta
```

BAli-Phy uses an alignment estimate (here, `P1-max.fasta`) as a template to construct a consensus alignment with ancestral sequences. BAli-Phy doesn't condition on the alignment columns, because (i) many columns occur only once in a posterior sample and (ii) conditioning on the column gives too much weight to the template alignment.

Because the alignment is uncertain, residues in the same column of the template alignment may end up in different columns in an MCMC sample. Therefore, in a given MCMC sample, different leaf residues in the same column may have different ancestors at the same internal node! However, in our ancestral reconstruction, a given column may only display a single ancestral residue.

BAli-Phy addresses this problem by averaging across the different ancestral residues in each column of the template alignment. When identifying the ancestral character to column C from a sampled alignment A, we random select a residue in C and use it to select a column from A. This procedure has the nice property that it will yield the traditional ancestral residue prediction if the alignment column is fixed.

9.3. Sampled alignments contain ancestral sequences

Ancestral sequences are written as part of the alignment matrix in each iteration. Ancestral sequences are given names starting with the letter **`A`**. For example, in the following alignment, the sequences **`A5`**, **`A6`**, and **`A7`** are reconstructed ancestors:

```
>Halobacterium
-T-TAAGGCGGCCATAGCGGTGGGGTTACTCCCGTAC
>Pyrococcus
GG-TACGGCGGTCATAGCGGGGGGGCCACACCCGGTC
>Sulfolobus
GC-CCACCCGGTCACAGTGAGCGGGCAACACCCGGAC
>Homo
GTCTACGGC---CATACCACCCTGAACGCGCCCGATC
>Escherichia
TG-CCTGGCGGCCGTAGCGCGGTGGTCCCACCTGACC
>A5
GG-CAAGGCGGCCATAGCGGGGGGGCCACACCCGGCC
>A6
GT-CAAGGCGGCCATAGCGGGGGGGCTACACCCGGTC
>A7
GT-CAAGGCGGCCATAGCGGGGGGGCTACACCCGGTC
```

Sampled alignments for the *n*th partition are in the file. `C1.Pn.fastas`.

Ancestral states in these alignments are randomly sampled from their joint posterior and do *not* represent the most probable ancestral state. The alignment of ancestral sequences is also inferred, so these sequences may contain gaps. The length of ancestral sequences may vary between samples when the length of the ancestral sequence is uncertain.

9.4. Sampled alignments correspond to specific sampled trees

Each sampled alignment matrix corresponds to a tree in the file `C1.trees` that is written in the same iteration. This tree specifies the phylogenetic location of each ancestral sequence by labelling the internal nodes of the tree. For example, the tree below shows where the internal nodes **`A5`**, **`A6`**, and **`A7`** are located on the tree:

```
(Halobacterium:0.213240,
((Escherichia:0.435762,Pyrococcus:0.122678)A5:0.114725,Sulfolobus:0.427210))A6:0.042527,Homo:0.427026))A7;
```

While the tree is written every iteration, the alignment is only written every 10 iterations (by default) in order to save disk space. One method for extracting the trees that correspond to saved alignments is to extract every 10th tree with the program **`bali-subsample`**:

```
% bali-subsample 10 < C1.trees > C1.10.trees
```

9.5. Using the sampled alignments instead of a consensus

Instead of constructing consensus ancestral sequences, you can also analyze the sampled alignments and their ancestral sequences directly. This approach involves performing a downstream analysis on *each* sampled (alignment,tree) pair, yielding the posterior distribution of the downstream analysis. Averaging these results then yields the posterior mean analysis result.

When this approach is feasible, it is more statistically rigorous than analyzing the consensus ancestral sequence alignment. The consensus ancestral sequence alignment does not account for uncertainty in the ancestral sequences, and is not a joint reconstruction. In contrast, analyzing the posterior samples accounts for uncertainty in the ancestral sequences, the alignment, and the tree. Furthermore, it also analyzes joint reconstructions instead of a marginal reconstruction.

9.6. Tree uncertainty in ancestral sequence reconstruction.

In Bayesian phylogenetic analyses, the tree is not fixed. Therefore the internal node corresponding to the ancestral sequence you wish to reconstruct may not exist in every posterior sample. The standard Bayesian approach to tree uncertainty is to reconstruct the ancestor for each node by conditioning on the existence of that node in the tree. This allows the reconstructed ancestor for each node to average over uncertainty about the existence of other nodes.

BAli-Phy additionally allows the researcher to condition on branches, since a branch condition is less restrictive. BAli-Phy does not run a separate MCMC chain with a tree constraint for each node, but instead performs conditioning by selecting samples from a single run that satisfy the condition.

9.6.1. Extracting and naming sequences that satisfy a query

Note that the sequence names (e.g. A6) for internal nodes may change over time. Therefore, you cannot simply extract ancestral sequences with a given name. To extract ancestral sequences for a given node, you need to specify a method of identifying that node on a tree, and a name to give to the sequence at that node. This is called a *query*.

For example, you might specify how to identify the ancestor node of Eukaryotes, and the name "Eukaryotes" to use for the sequence there. You can then use the program `extract-ancestors` to extract ancestral sequences from the sampled trees and alignment, and label them with useable names.

```
% trees-consensus dir-1/C1.trees dir-2/C1.trees | tree-tool - --strip-internal-names --name-all-nodes > c50.tree
% extract-ancestors -A dir-1/C1.P1.fastas -T dir-1/C1.trees -A dir-2/C1.P1.fastas -T dir-2/C1.trees -n c50.tree -g
c50.tree > P1.ancestors.fastas
```

Here the options `-n c50.tree` and `-g c50.tree` specify node-based queries and branch-based queries.

9.6.2. Conditioning on a node: node-based queries

A node exists in a sampled tree if every branch connected to that node exists in the sampled tree. A node-based query asks for the reconstructed ancestral sequence only from samples where every branch connected to that node exists. A node-based query is more stringent than a branch-based query, since it requires multiple branches to exist.

BAli-Phy allows constructing node-based queries by passing in a Newick tree with labelled internal nodes. A node-based query is automatically constructed from each internal node that is labelled.

9.6.3. Conditioning on a branch: branch-based queries

A branch-based query requires only that a single branch exist in a sampled tree. The branch-based query asks for the reconstructed ancestral sequence on one endpoint of that (directed) branch. When the focus is on changes that occur on a particular branch, this makes more sense than a node-based query.

BAli-Phy allows constructing branch-based queries from a file where every line is either a Newick tree *or* a named group of taxa. For each line that contains a Newick tree, a branch-based query is automatically constructed from each branch where *both* endpoints are labelled. For a branch from `node1` to `node2`, the query is named `"node2<=node1"`.

Branch-based query files can also contain lines of the form

```
name = taxon1 taxon2 ... taxonN
```

This matches branches that separate the listed taxa from all other taxa, and points toward the listed taxa.

10. 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 output of 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 2 independent copies of the Markov chain (preferably 4-10) 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.

10.1. 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".

10.2. Definition of Mixing

In MCMC, each sample is not fully independent of previous samples. In fact, even after a Markov chain has converged, 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 new sample contributes very little new information, and we need to obtain many more samples to get good precision on our parameter estimates. In such a case, we say that the chain isn't "mixing" well.

10.3. Diagnostics: Variation in split frequencies across runs (ASDSF/MSDSF)

10.3.1. ASDSF and MSDSF

To calculate the ASDSF and MSDSF run:

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.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.

10.3.2. 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 dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.trees --LOD-table=LOD-table > partitions.bs
% R --slave --vanilla --args LOD-table compare-SF.pdf < compare-runs.R
```

Following Beiko et al. (2006), 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.

10.4. Diagnostics: Potential Scale Reduction Factors (PSRF)

Potential Scale Reduction 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 dir-1/C1.log dir-2/C2.p ... dir-n/C1.log > 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.

10.5. Diagnostics: Effective sample sizes (ESS)

10.5.1. ESS for numerical values

To calculate the split ESS values, run:

```
% statreport dir-1/C1.log dir-2/C1.log ... dir-n/C1.log > Report
```

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

The program Tracer also computes ESS values.

10.5.2. ESS for split frequencies

As desribed in Gaya et al. (2011), we can also compute ESS values for splits on the tree:

```
% trees-bootstrap dir-1/C1.trees dir-2/C1.trees ... dir-n/C1.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.

10.6. Diagnostics: Stabilization

10.6.1. Stabilization of numerical values

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

```
% statreport C1.log > 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.

10.6.2. 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). In theory, we recommend the web tool Are We There Yet (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 --subsample=factor compare dir-1/C1.trees dir-2/C1.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 --subsample=factor convergence C1.trees equilibrium.trees
   ```

3. To assess when the above time series stabilizes:

   ```
   % trees-distances --skip=burnin --subsample=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.

11. Alignment utilities: brief overview

This section gives a brief overview showing *some* of the things that can be done with the included alignment utilities. It is intended to be helpful, but not exhaustive. To see the full set of options for each tool, give the argument "**`--help`**" on the command line.

11.1. alignment-info

Show basic information about the alignment:

```
% alignment-info file.fasta
% alignment-info file.fasta file.tree
```

11.2. alignment-cat

To select columns from an alignment:

```
% alignment-cat -c1-10,50-100,600- file.fasta > result.fasta
% alignment-cat -c5-250/3 file.fasta > first_codon_position.fasta
% alignment-cat -c6-250/3 file.fasta > second_codon_position.fasta
```

To concatenate two or more alignments:

```
% alignment-cat file1.fasta file2.fasta > all.fasta
```

11.3. alignment-thin

Remove columns without a minimum number of letters:

```
% alignment-thin --min-letters=5 file.fasta > file-thinned.fasta
```

Remove sequences by name:

```
% alignment-thin --remove=seq1,seq2 file.fasta > file2.fasta
```

Remove short sequences:

```
% alignment-thin --longer-than=250 file.fasta > file-long.fasta
```

Remove sequences with <= 5 differences from the closest other sequence:

```
% alignment-thin --cutoff=5 file.fasta > more-than-5-differences.fasta
```

Like `--cutoff`, but stop when we have the right number of sequences:

```
% alignment-thin --down-to=30 file.fasta > file-30taxa.fasta
```

Protect some sequences from being removed:

```
% alignment-thin --down-to=30 file.fasta --protect=seq1,seq2 > file-30taxa.fasta
```

Remove sequences that are missing conserved columns:

```
% alignment-thin --remove-crazy=10 file.fasta > file2.fasta
```

11.4. alignment-draw

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

```
%  alignment-draw file.fasta --show-ruler --color-scheme=DNA+contrast > file.html
%  alignment-draw file.fasta --show-ruler --AU=file-AU.prob --color-scheme=DNA+contrast+fade+fade+fade+fade > file-
AU.html
```

11.5. alignment-find

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

```
% alignment-find --first < file.fastas > first.fasta
% alignment-find < file.fastas > last.fasta
```

11.6. alignment-indices

Turn columns from a template alignment into alignment constraints:

```
% alignment-indices template.fasta > constraints.txt
% alignment-indices -c100-110,200,300- template.fasta > constraints.txt
```

Each line in this file corresponds to one alignment column.

11.7. alignment-chop-internal

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

```
% alignment-chop-internal file.fasta > file-chopped.fasta
```

12. Tree utilities: brief overview

This section gives a brief overview showing *some* of the things that can be done with the included tree utilities. It is intended to be helpful, but not exhaustive. To see the full set of options for each tool, give the argument "`--help`" on the command line.

12.1. trees-consensus

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.

12.2. 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.

12.3. 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.

13. Compiling *`BAli-Phy`*

Compiling *`BAli-Phy`* is intended to be a relatively painless process. However, most people will want to use the pre-compiled binaries as described in the standard installation instructions at Section 2, “Installation” instead of compiling BAli-Phy themselves. You might want to compile BAli-Phy yourself if you want to

- run BAli-Phy on a non-Intel CPU (such as ARM64 or Alpha).
- run BAli-Phy on a computing cluster.
- test an unreleased version of bali-phy.
- change the optimization options used to compile BAli-Phy in the pre-compiled binaries.
- compile with debugging options to find the cause of a bug, and maybe fix it.
- modify the source code and submit a patch with new functionality.

Otherwise, the pre-compiled binaries will be fine.

13.1. Setup

In order to compile BAli-Phy, you need

- a C++20 compiler
- meson (version >= 1.1)

We recommend the GNU C++ Compiler (GCC) version 12.0 (or higher) or the Clang compiler version 17 or higher. The Cairo graphics library is optional, but if it is missing, the **drawtree** tool that is used to draw consensus trees won't be built. See also Section 2.8, “Install programs used for viewing the results”.

13.1.1. Linux

On Debian and Ubuntu, you can type:

```
% sudo apt-get install g++ git libcairo2-dev pandoc libboost-all-dev
```

If your version of Debian or Ubuntu is recent enough to contain meson version 1.1 or higher, you can install meson with apt-get:

```
% sudo apt-get install meson
% meson --version
1.6.0
```

On computing clusters, you might want to use miniconda to install the build tools.

```
% conda create -n devel -c conda-forge --strict-channel-priority
% conda activate devel
% conda install meson gxx boost-cpp cmake pkg-config cairo
% export BOOST_ROOT=$CONDA_PREFIX
```

Otherwise you can install meson through pip3:

```
% sudo apt-get install python3 python3-pip ninja
% python3 -m venv meson
% source meson/bin/activate
% pip3 install meson
```

13.1.2. Mac

On Mac OS X, the simplest way to get a compiler is to install XCode version 15 (or newer) command line tools, which come with clang.

```
% xcode-select --install
```

To get the other tools, first install homebrew, and then type:

```
% brew install git meson cairo pandoc
```

13.1.3. Windows (native)

The MSYS2 project provides an MINGW64 compiler that can create native windows executables. MSYS2 itself is actually non-native (it is derived from cygwin), and therefore the MSYS2 shell refers to drives as `/c/` instead of `C:/`.

```
% pacman --needed --noconfirm -Sy pacman-mirrors
% pacman -Sy
% pacman -S mingw-w64-x86_64-ninja
% pacman -S mingw-w64-x86_64-toolchain
% pacman -S mingw-w64-python3-pip
% PATH=/c/msys64/mingw64/bin:$PATH # Put the mingw64 executables into your path
% pip3 install meson
```

Keep in mind that MSYS2 keeps its (non-native) executables in `C:/msys64/usr/bin`, while it keeps the (native) MINGW executables in `C:/msys64/mingw64/bin`. If you want to use the native MINGW executables, you need to make sure that `/c/msys64/mingw64/bin/` is in your PATH. If you forget to put the MINGW executables in the path, some of the installed MINGW programs (such as pip3 above) will show up as missing when you try to run them.

13.2. Clone, Configure, Compile

First check out the code using git:

```
% git clone https://github.com/bredelings/BAli-Phy.git
% cd BAli-Phy
```

Then run meson to configure the build process:

```
% meson setup build --prefix=$HOME/Applications/bali-phy-4.0/ --buildtype=release
```

Finally, build and install the software:

```
% ninja -C build test
% ninja -C build install
```

The command **bali-phy** and its associated tools should then be located in `~/Applications/bali-phy-4.0/bin/`. To install to another directory *`dir`*, specify --prefix=*`dir`* to **meson**.

13.3. Options: compiler and linker flags

You can select the C++ compiler by setting the CXX variable. A useful example of this is to use **g++-14** on systems where **g++** invokes a compiler that is too old:

```
% CXX=g++-14 meson setup build --prefix=$HOME/Applications/bali-phy-4.0 --buildtype=release
```

You may also set compiler and linker options using the CPPFLAGS, CXXFLAGS, and LDFLAGS variables. For example, you can instruct the compiler to use all the features of your chip, instead of producing generic code that will run anywhere:

```
% CXXFLAGS="-mtune=native -march=native" meson setup --prefix=$HOME/Applications/bali-phy-4.0
```

For example, you can set the CPPFLAGS and LDFLAGS variables to instruct the compiler where to look for libraries, such as cairo:

```
% CPPFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib" meson setup build --prefix=$HOME/Applications/bali-phy-4.0
--buildtype=release
```

Another useful example of this is to produce an OS X executable on that can run on older versions of OS X:

```
% CXXFLAGS="-mmacosx-version-min=10.9" LDFLAGS="-mmacosx-version-min=10.9" meson setup build --
prefix=$HOME/Applications/bali-phy-4.0 --buildtype=release
```

14. Frequently Asked Questions (FAQ)

14.1. Input files

14.1.1. Does BAli-Phy accept the wildcard characters "N" or "X"? How does it treat them?

Yes, BAli-Phy accepts the wildcard characters "N" (for DNA) and "X" (for proteins). These characters indicate that some letter is present (as opposed to a gap), but that you don't know *which* letter it is.

14.1.2. Does BAli-Phy accept "?" characters?

No. "?" characters are often used to indicate *either* letter presence (e.g. "N", "X") *or* absence (e.g. "-"). BAli-phy will insist that you replace each "?" with either "N"/"X" or "-" to indicate which one you mean.
(Most programs ignore indels and consider only substitutions, and in that case "N" and "-" have the same effect on the likelihood or parsimony score. However, since BAli-Phy takes indels into account, these two alternatives are quite different.)

14.1.3. Does BAli-Phy accept the characters "R" and "Y", etc.?

Yes. BAli-Phy accepts the characters Y, R, W, S, K, M, B, D, H, and V for DNA, RNA, and Codon alphabets. BAli-Phy

also accepts the characters B, Z, and J for amino acids. These characters indicate partial knowledge about a letter. For example, R indicates that a nucleotide is present, and is a puRine (A or G). J indicates that an amino acid is present and is either I or L.
(Note that sequences sometimes contain such ambiguity codes because the DNA that was sequenced contains *both* values. This might occur when sequencing a heterozygote or when sequencing pooled DNA from several individuals. However, the model in BAli-Phy (and other phylogeny inference programs) is that only one letter is correct, but we do not know which one it is. This is probably not problematic when dealing with pooled sequences, but should be considered.)

14.2. Running bali-phy.

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

Yes. Add `-Inone` or `-I none` on the command line.

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

Yes. Add `--fix topology=`***`treefile`*** on the command line.

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

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

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

Yes. Add `--fix tree=`***`treefile`*** `'--scale=~gamma(0.5,2)'` on the command line.

14.3. Run-time error messages

14.3.1. I tried to use `-S lg08+>Rates.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 `lg08+>Rates.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 quotes around the substitution model, like this: `-S 'lg08 +> Rates.gamma(6)'`. This will keep the C-shell from interfering with your command.

14.4. Stopping bali-phy.

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

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

The longer answer is that is is hard to predict how long MCMC will take to converge, since it depends on each data set in complex ways. Automatic rules for determining when to stop an MCMC chain can be difficult to get right. BAli-Phy does not contain an automatic stopping rule yet, so it relies on the user to run convergence diagnostics and determine when to stop the run.

14.4.2. How do I stop a **bali-phy** run on my personal computer?

Simply kill the process -- there is no special command to stop **bali-phy**. If you are running it on your personal workstation, then you can use the command **kill**. To do that, you need to find the PID (process ID) of the running program. You can find this by examining the beginning of the file `C1.run.json`. For example:

```
% less 5d-1/C1.run.json
    ...
    "partitions": [
        {
            "alphabet": "DNA",
            "filename": "5d-muscle.fasta",
            "imodel": 0,
            "range": "",
            "scale": 0,
            "smodel": 0
        }
    ],
    "pid": 549319,
    "program": {
        "arch": "linux x86_64",
        "build-date": "Mar  2 2024 11:27:23",
        "compiler": "gcc 13.2.0 x86_64",
        "name": "bali-phy",
        "revision": "[HEAD -> master, origin/master, origin/HEAD commit d394a4fb6]  (Mar 02 2024 11:11:25)",
        "version": "4.0-beta9-preview"
    },
    ...
```

Here the PID is 549319. Therefore you can type:

```
% kill 549319
```

On some operating systems you can also type:

```
% killall bali-phy
```

However, be aware that this will terminate *all* of your **bali-phy** runs on that computer.

14.4.3. How do I stop a **bali-phy** run on a computing cluster?

Simply terminate the submitted job. The specific command to terminate a job will depend on the queue manager that is installed on your cluster. Examine the documentation for your cluster, or ask your cluster support staff how to delete running jobs on your cluster.
As an example, if the SLURM software is used to submit jobs, then the command **squeue** should list your jobs and their job ID numbers (which is different than the process ID number). You can then use the command **scancel** to delete jobs by ID number. The SLURM documentation describes how to use these commands.

14.4.4. 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.

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

See section Section 10, "Convergence and Mixing: Is it done yet?".

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

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

14.5. Running bp-analyze.

14.5.1. Why does **bp-analyze** say "Program 'draw-tree' not found. Tree pictures will not be generated"?

The program **draw-tree** was not distributed on this platform (Windows, Mac). This is not a fatal error message, it just means that a pretty picture of the tree will not be generated automatically. You can still view the tree with *`FigTree`*, for example.

14.5.2. Why does **bp-analyze** say "Program 'gnuplot' not found. Trace plots will not be generated"?

This is because you have not installed *`gnuplot`*. This is not a fatal error message, it just means that pictures of partition support, and SRQ plots will not be generated automatically.

14.5.3. Why does **bp-analyze** say "Program 'R' not found. Some mixing graphs will not be generated"?

This is because you have not installed *`R`*. This is not a fatal error message, it just means that a plot showing differences in clade probabilities between runs will not be generated.

14.5.4. Why is **bp-analyze** stopping early, or failing to generate some files?

Look in the file `Results/commands.log`. This should contain the specific tool commands that were run, along with error message from these commands. Identify the first tool command that fails, and read the error message.

14.6. Interpreting the results.

14.6.1. 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.

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

After you analyze the output (Section 4.2, “Posterior summaries”), the partition support is indicated in `Results/consensus` and in `Results/c50.PP.tree`.

14.7. How do I...

14.7.1. 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.

14.7.2. How do I select columns from an alignment?

You can select columns for analysis by specifying a range:

```
% bali-phy sequences.fasta:1-200,401-600 sequences.fasta:201-400
```

You can create a new alignment from selected columns using **`alignment-cat`**:

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

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