
BAlI-Phy User's Guide v2.0.2

Benjamin Redelings

Table of Contents

1. Pre-requisites	2
1.1. Hardware Requirements	2
1.2. Operating System requirements	2
1.3. Software Requirements	3
2. Compiling BAlI-Phy	3
2.1. Quick Start	4
2.2. Installing to a location besides /usr/local/	4
2.3. Specifying where to find libraries and header files (e.g. GSL)	4
2.4. Selecting a non-default C++ compiler	4
2.5. Optimizing for a specific architecture	5
2.6. Statically linked executables	5
2.7. Generating Autoconf/Automake files	5
2.8. Examples	5
3. Installation	5
3.1. Installing when compiling from source	5
3.2. Installing precompiled binaries	5
3.3. Setting the PATH	6
4. Running the program	6
4.1. The simple way	6
4.2. Locating the Data/ directory	6
4.3. Option files (Scripts)	7
4.4. Running long simulations	8
5. Command-line options	8
5.1. Usage	8
5.2. General options	9
5.3. MCMC options	9
5.4. Parameter options	9
5.5. Model options	10
6. Examples	10
7. Input	11
7.1. Sequence formats	11
7.2. Is my data set too large?	11
8. Models	11
8.1. Substitution models (Basic)	11
8.2. Indel models	14
8.3. Alignment constraints	14
9. Substitution Models (Advanced)	14
9.1. Specifying a substitution model	14
9.2. CTMC Frequency models	15
9.3. Frequency Models	15
9.4. CTMC Mixture Models	16
9.5. Examples	16
10. Output	17
10.1. Output directory	17
10.2. Output files	17

11. Convergence and Mixing: Is it done yet?	17
11.1. Convergence	17
11.2. Mixing	18
12. Analyzing the output	18
12.1. How to run the full analysis	18
12.2. Results/	19
12.3. Mixing/	19
13. Tuning the Markov Chain	20
13.1. Parameters	20
14. Auxiliary tools	20
14.1. alignment-find	20
14.2. alignment-draw	21
14.3. alignment-thin	21
14.4. alignment-chop-internal	21
14.5. alignment-info	21
14.6. alignment-indices	21
14.7. alignment-cat	21
14.8. trees-consensus	21
14.9. trees-bootstrap	21
14.10. trees-to-SRQ	22
15. Frequently Asked Questions (FAQ)	22
15.1. Running bali-phy	22
15.2. Figuring out run-time errors	22
15.3. Stopping bali-phy	22
15.4. Interpreting the results.	23

1. Pre-requisites

1.1. Hardware Requirements

BAlI-Phy should be able to run on any hardware with a modern operating system, including Linux or UNIX, Windows, and Mac OS X.

We recommend at least 512Mb of RAM and a 1.5GHz processor. We typically run BAlI-Phy on Pentium 4 processors with 1Gb of RAM and a 3 GHz processor.

1.2. Operating System requirements

Command line ahead!

BAlI-Phy is not a graphical application. While there are Mac and Windows versions available, these assume that you understand some UNIX commands, or at least are not intimidated by them. BAlI-Phy is (hopefully) usable, but it is not "user-friendly"!

1.2.1. Linux and UNIX

A Linux or UNIX system should be able to install and run BAlI-Phy without any modification.

1.2.2. Windows

On Windows you must first install cygwin [<http://www.cygwin.com>]. Cygwin is a Linux environment for Windows. You can use the cygwin installer to install other software packages that you need (See Section 1.3, "Software

Requirements”). You can access the Cygwin command line (not the normal windows command line) through the Start menu.

1.2.3. Mac OS X

A Mac OS X system is based on UNIX, and should be able to compile and run BAlI-Phy. The version of OS X must be 10.1 or higher. If you want to use the development tools from Apple to compile BAlI-Phy, the system must be 10.4 or higher, and you must upgrade to XCode 2.2 or higher (see Software Requirements).

You can use Fink [<http://fink.sourceforge.net>] to install other useful applications, such as gnuplot.

1.3. Software Requirements

1.3.1. Compilation Requirements

The following software packages are required for compiling BAlI-Phy. (Note that precompiled binaries are available on the website, so that you may not need to do any compilation. In that case these requirements are not necessary.)

- A modern C++ compiler (GCC [<http://gcc.gnu.org>] version 3.4, 4.0, 4.1, and 4.2 are known to work)

Mac OS X issues:

If you want to use the development tools shipped by Apple (e.g. GCC) then you need OS X version 10.4 (Tiger) or higher, and you need to install XCode 2.2 or higher. (You can use a GCC compiler built from the non-apple FSF sources, but installing XCode is much simpler.)

- GNU make version 3.80 or higher (GNU make [<http://www.gnu.org/software/make/>]).
- The GNU Scientific Library (GSL [<http://sources.redhat.com/gsl/>]) version 1.8 or higher.

1.3.2. Requirements for analyzing BAlI-Phy output

After you run **balI-phy** you will need to run an analysis of the resulting files (See Section 12, “Analyzing the output”). This analysis depends on the following software packages:

- GNU make version 3.80 or higher (GNU make [<http://www.gnu.org/software/make/>]).
- PERL

1.3.3. Recommendations for analysis and documentation

In addition, the following software packages are strongly recommended. These packages allow you to view plots and diagnostics for BAlI-Phy, or to view the documentation correctly.

- Tracer - to analyze MCMC diagnostics (Tracer [<http://evolve.zoo.ox.ac.uk/software/tracer/>])
- The plotting program gnuplot (gnuplot [<http://www.gnuplot.info/>])
- Mozilla or Mozilla/Firefox - to view the math in the XHTML documentation. (Firefox [<http://www.mozilla.org/products/firefox/>])

2. Compiling BAlI-Phy

Note that precompiled binaries are available on the web for Linux, Mac, and Windows so that compilation may not be necessary. However, in case you want to compile your own binaries for some reason, compiling BAlI-Phy is intended to be a relatively painless process.

Also note that in case you are compiling "live" source code that you checked out using subversion (and you probably aren't) then you need to follow the directions in Section 2.7, "Generating Autoconf/Automake files" before you start compiling.

2.1. Quick Start

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

```
% tar -zxf bali-phy-2.0.2.tar.gz
```

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

```
% mkdir build
% cd build
% ../bali-phy-2.0.2/configure
```

If this command succeeds, then you can simply type

```
% make
% make install
```

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

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

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

```
% ../bali-phy-2.0.2/configure --prefix=$HOME/local
```

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

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

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

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

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

2.4. Selecting a non-default C++ compiler

The default C++ compiler is **g++**. On some systems, **g++** invokes GCC version 3.3, and the correct compiler is called something else, such as **g++-4.0**. To use **g++-4.0** as the C++ compiler when compiling BAli-Phy, you would set the **CXX** environment variable as follows:

```
% ../bali-phy-2.0.2/configure CXX=g++-4.0
```

Alternatively, on Macintosh systems, you can run the command `gcc_select 4.0` to choose the correct compiler.

2.5. Optimizing for a specific architecture

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

```
% ../bali-phy-2.0.2/configure CHIP=cpu
```

You can set `CHIP` to any of `pentium3`, `pentium4`, `nocona`, `G3`, `G4`, or `G5`.

2.6. Statically linked executables

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

2.7. Generating Autoconf/Automake files

This step is *only* necessary if you are using the subversion (SVN) source. It is *not* necessary if you are compiling from a distributed tar.gz archive of the source, because these archives already include the files that this step will generate, such as the `configure` script and some Makefiles.

To generate these files, you need automake 1.8 or higher and autoconf 2.59 or higher. Run these commands inside the `trunk/` directory that you checked out.

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

If your system has multiple versions of automake, then you may have to type e.g. `automake-1.8 -a` and `aclocal-1.8` instead in order to specify which version to use.

2.8. Examples

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

```
% ../bali-phy-2.0.2/configure --prefix=$HOME/local --enable-static CXX=g++-4.1 CHIP=pentiu
```

This example uses `g++-4.1` to build a `pentium4`-optimized version of `bali-phy` with static linkage.

3. Installation

3.1. Installing when compiling from source

After compiling BAli-Phy, you can simply type `make install`. This will copy the compiled binaries to the directory chosen when you ran `configure`.

3.2. Installing precompiled binaries

To install pre-compiled binaries, simply extract the compressed archive in the directory of your choice. If you have root access you might extract in `/usr/local`; if not, you might make a directory `~/local` in your home directory and extract there.

You can extract the compressed archive on the command line using the **tar** command:

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

3.3. Setting the PATH

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

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

```
% echo $PATH
```

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

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

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

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

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

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

4. Running the program

4.1. The simple way

The simplest way to run BALi-Phy is to invoke the **bali-phy** executable directly:

```
% bali-phy sequence-file --data-dir prefix/share/bali-phy/Data/
```

Here *sequence-file* is a FastA or PHYLIP file containing the sequences you wish to analyze. It should end in **.fasta** or **.phy** to indicate which format it is using. Here *prefix* stands for the directory where you extracted the bali-phy archive.

4.2. Locating the Data/ directory

BALi-Phy stores important information in a "Data" directory, and needs to know where to find this information. This information includes the genetic code and the WAG rate matrix. If you installed BALi-Phy in a directory called *prefix*, then the `Data/` directory will be at `prefix/share/bali-phy/Data/`.

If you specify the location of the `Data/` directory in a configuration file, then you do not need to specify it on the command line every time. Then, in the future you can just type:

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

If you don't specify where the data directory is, then BAli-Phy will look for it at `./Data`.

4.2.1. Put the location in the configuration file

Create a new text file at `~/ .bali-phy` and add the following line:

```
data-dir = somewhere/Data
```

Then **baali-phy** will read this file every time and know where to find the Data directory. This is the recommended method.

4.2.2. Making a link/shortcut to the Data directory

Instead of specifying the location of the data directory, you can make a shortcut to the Data directory:

```
% ln -s somewhere/Data
```

The shortcut will then be at `./Data`. Then **baali-phy** will find the data directory at the default location `Data/`, but only if you run it in the same directory as the shortcut.

4.2.3. Making your own copy of the Data directory

Instead of specifying the location of the data directory, you can make a local copy of the Data directory that is also called `Data/`:

```
% cp -r somewhere/Data .
```

Then **baali-phy** will find the data directory at the default location `Data/`, but only if you run it in the directory as the local copy.

4.3. Option files (Scripts)

Putting the analysis options in an option file instead of on the command line can be more convenient if you are going to run the same analysis many times, or if you have a large number of options. In addition, the option file may contain comments and blank lines, and can be a good way to record what options you used in an analysis.

4.3.1. Syntax

An option file is specified the option "`--config file`" and uses the same option names as the command line. However, the syntax is different. In the option file, each given option is on its own line using the syntax "`option = value`" instead of the syntax "`--option value`". If the option has no value then it is given using the syntax "`option = option`". If values for an option are given both on the the command line and in an option file, then the command line value overrides the value in the option file.

4.3.2. Example

For example, consider the following option file:

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

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

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

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

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

4.3.3. The configuration file

The file `~/.bali-phy` is a special option file called the *configuration file*. If it exists, it is always loaded. If options are given on the command line or an option file they always override values given in the configuration file. You should use your configuration file to set global values that are the same in all your analyses, such as the location of the Data directory. Thus, you probably would not want to set **align** in this file. However, setting **smodel** would change the default substitution model while allowing a specific analysis to override the default.

4.4. Running long simulations

In order to run **bali-phy** for long periods of time, some modifications to the above commands may be necessary.

4.4.1. CPU time limits

If **bali-phy** is terminating prematurely, CPU time limits may be the cause. In the BASH shell you can find the soft and hard time limits (in seconds) using the commands **ulimit -St** and **ulimit -Ht** respectively. BAli-Phy ignores the soft limit, but cannot ignore the hard limit. If the hard limit is not long enough, you must ask your system administrator to change it for you.

4.4.2. Hangup (HUP) Signals

Most programs will terminate when you log out, or when you close the terminal that started them. The **nohup** command is often used to avoid this. BAli-Phy ignores the HUP (hangup) signal in versions 2.0.0 and newer, making **nohup** unnecessary.

4.4.3. Submitting jobs on an SGE cluster

You can use the wrapper script **bali-phy-sge** to submit a **bali-phy** job on an SGE cluster. It takes the same arguments as **bali-phy**, and submits a job for you using the **qsub** command.

5. Command-line options

5.1. Usage

The syntax for the program is:

```
bali-phy {sequence-file} [OPTIONS]
```

The sequence file is the only required argument. It can be either a FastA (*.fasta) file or a PHYLIP (*.phy) file. The file must end in one of these two suffixes or BAlI-Phy won't know how to read it. In addition your FastA files should not contain any blank lines.

The optional arguments are described below. They can also be found by typing **bali-phy --help** on the command line.

5.2. General options

<code>--help</code>	Show help message.
<code>--version</code>	Show version information.
<code>--config <i>file</i></code>	Option file to read.
<code>--show-only</code>	Analyze initial values and exit.
<code>--seed <i>seed</i></code>	Use the specified seed to initialize the random number generator.
<code>--data-dir <i>directory</i></code>	Specify the data directory.
<code>--name <i>string</i></code>	Specify the name for the analysis directory.
<code>--align-constraint <i>file</i></code>	Specify a file with alignment constraints
<code>--traditional</code>	Fix the alignment and don't model indels.
<code>--letters <i>star</i></code>	Use a star tree for the substitution model.

5.3. MCMC options

<code>--iterations <i>number=100000</i></code>	Specify the number of iterations to run.
<code>--subsample <i>factor=1</i></code>	Specify a factor by which to subsample.
<code>--T <i>temperature=1</i></code>	Specify a temperature for MCMCMC.
<code>--enable <i>move</i></code>	Enable a comma-separated list of transition kernels.
<code>--disable <i>move</i></code>	Disable a comma-separated list of transition kernels.

5.4. Parameter options

<code>--randomize-alignment</code>	Randomly re-align sequences before use.
<code>--internal +</code>	Set all internal node entries wildcards initially.
<code>--tree <i>file</i></code>	Specify file with initial tree.
<code>--set <i>parameter=value</i></code>	Specify initial value of <i>parameter</i> .
<code>--fix <i>parameter[=value]</i></code>	Mark <i>parameter</i> fixed, and optionally specify a value.

`--unfix` Mark *parameter* not fixed, and optionally specify an initial value.
`parameter[=value]`

`--frequencies` Specify initial frequencies: 'uniform', 'nucleotides', or a comma-separated list of
`frequencies` frequencies.

5.5. Model options

`--alphabet name` Specify the alphabet: DNA, RNA, Amino-Acids, Amino-Acids+stop, Triplets, Codons, or Codons + stop.

`--genetic-code` Specify alternate genetic code file in the data directory.
`file=standard-code.txt`

`--smodel name` Specify the substitution model.

`--imodel name` Specify the indel model.

6. Examples

Here are some examples which demonstrate how to run BAlI-Phy. In order to run these examples, you must first do two things. Firstly, you must specify the location of the `Data/` directory is specified as in Section 4.2, “Locating the `Data/` directory”.

Secondly, you must find the `examples/` directory which contains the example files. Typically, the `examples/` directory will be found at `prefix/share/bali-phy/examples/` if you installed `bali-phy` in directory `prefix`.

Also note that **bali-phy** does run until it is "finished", but continues to gather samples until the user stops it. Thus, it is useful to continually examine the output files while the program is running.

Example 1. No frills

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

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

Example 2. Multiple-Rate Substitution Model

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

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

Example 3. Fixed alignment

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

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

7. Input

7.1. Sequence formats

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

Note that blank lines are now allowed in FastA files in version 2.0.1 and later.

7.2. Is my data set too large?

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

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

7.2.1. Too many taxa?

BAlI-Phy is quite CPU intensive, and so we recommend using 12 or fewer taxa in order to limit the time required to accumulate enough MCMC samples. We note that many phylogenetic hypotheses can be reduced to hypotheses concerning only four taxa. We recommend initially removing as many taxa as possible from your data set. You can then add taxa back into the data set after you have determined how much this will increase the duration of the run.

7.2.2. Sequences too long?

Aligning sequences takes $O(L^2)$ time and RAM, where L represents the sequence length. Thus, doubling the length of the input sequences increases RAM requirements and run time by a factor of 4. Therefore sequence lengths beyond 750 letters may prove impractical due to limited RAM or the speed of the calculation. For protein sequences, we recommend coding the sequences in terms of amino acids or codons, rather than nucleotides.

However, you might try to see how long you can make your sequences before the attempt becomes impractical.

8. Models

8.1. Substitution models (Basic)

The basic substitution models in BAlI-Phy are continuous-time Markov chains (CTMC). More advanced models such as the Γ^4 + INV model or the M2 codon model are described in Section 9, "Substitution Models (Advanced)". CTMC models can be characterized by transition rates Q_{ij} from letter i to letter j . After a given time t the probability for transition from state i to state j is given by

$$P_{ij}(t) = e^{Q_{ij} t}$$

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

$$Q^{ij} = S^{ij} \pi_j$$

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

8.1.1. Basic CTMC models

The basic CTMC models are EQU, HKY, TN, GTR, HKYx3, TNx3, GTRx3, Empirical, and M0. Each of these models is a way of specifying the exchangeability matrix S .

8.1.2. Default substitution models

If the substitution model is not specified, then the default model for the alphabet is used. For DNA or RNA, the default model is HKY. For Triplets, the default is HKYx3. For Codons, the default model is M0. For Amino-Acids, the default model is Empirical[WAG].

Table 1. Substitution Models

		κ^1	$S_{ij} = 1$ for transitions.
TN	DNA or RNA	Ali-Phy User's Guide v2.0.2	$S_{ij} = \kappa^1$ for transversions.
Tamura, Nei (1993)		κ^2 : the pyrimidine ts/tv ratio.	$S_{ij} = \kappa^2$ for purine transitions. $S_{ij} = 1$ for pyrimidine transitions.
8.1.3. Extended model descriptions		$S_{i \rightarrow j}$	
GTR	DNA or RNA	\neq	$\sum_{\neq} =$. (5 degrees of freedom).
General Time-Reversible Tavare (1986)			
Empirical[WAG]	Amino-Acids	none.	
Whelan and Goldman (2001)			
HKYx3	Triplets	<i>nuc-model</i> parameters.	If the <i>nuc-model</i> has transition matrix S_{ij} on nucleotides, then: $S_{ab} = 0$ $S_{ab} =$ for changes of more than one nucleotide. $S_{ab} =$ for i single nucleotide changes \rightarrow .
TNx3			
GTRx3			
M0	Codons	κ : the ts/tv ratio. ω : the dn/ds ratio.	$S_{ab} = 1$ for changes of more than one nucleotide. $S_{ab} = \omega$ for synonymous transversions. $S_{ab} = \kappa$ for non-synonymous transversions. $S_{ab} = \omega\kappa$ for synonymous transitions. $S_{ab} =$ for non-synonymous transitions.
Nielsen and Yang (1998)			
M0 [nuc-model=HKY]	Codons	<i>nuc-model</i> parameters. ω : the dn/ds ratio	If the <i>nuc-model</i> has transition matrix S_{ij} on nucleotides, then: $S_{ab} = 0$ $S_{ab} =$ for changes of more than one nucleotide. $S_{ab} = \omega S_{ij}$ for synonymous changes. $S_{ab} =$ for non-synonymous changes.
Nielsen and Yang (1998)			

8.2. Indel models

The current models are RS05, RS07-no-T, and RS07. The default is RS07.

8.3. Alignment constraints

To pin columns of the alignment, specify alignment constraints in a file as follows:

1. Use the argument `--align-constraint filename`
2. The filename refers to a file in which each line represents a constraint.

8.3.1. Syntax

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

8.3.2. Examples

For example, the file

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

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

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

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

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

8.3.3. Computing the constraints

The program **alignment-indices** may be used to aid in computing a constraint file from an input alignment. Each line in the resulting file is a constraint corresponding to one column of the input alignment. Simply remove the constraints that you do not want to keep.

9. Substitution Models (Advanced)

Advanced substitution models in BAli-Phy are constructed as mixtures of the basic CTMC models (see Section 8.1, “Substitution models (Basic)”) run at different rates (e.g. Γ^4 + INV) or parameters (e.g. an M2 codon model).

9.1. Specifying a substitution model

Substitution models are specified using a stack, as follows: `Model[arg]+Model[arg]+...+Model[arg]` where each model uses the previous models as input. Arguments are optional.

Note

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

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

9.2. CTMC Frequency models

The above decomposition can be generalized slightly to yield the following decomposition, where f ranges from 0

$$Q_{ij} = S_{ij} \times \frac{\pi_j}{f},$$

Here the parameter f specifies the relative importance of unequal conservation ($f = 0$) and unequal replacement ($f = 1$) in maintaining the equilibrium frequencies π_j .

In fact, this can be generalized even further to

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

where

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

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

9.3. Frequency Models

Table 2. Frequency Models

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

9.4. CTMC Mixture Models

Table 3. Extended Model Descriptions

sm + INV n	sm alphabet	p : invariant fraction. σ μ	A fraction p of sites do not allow substitutions.
sm + gamma[] Yang (1994) n	sm alphabet	Γ : noise to signal ratio for . σ μ	rate \sim (= ,). A discrete approximation to the with bins is used.
sm + log-normal[]	sm alphabet	logNormal noise to signal ratio for . κ	rate logNormal μ 1 σ \sim (= ,). A discrete approximation to the with bins is used.
M2 sm + M2 Yang, et. al. (2000) n	Codons	: the ts/tv ratio p^1 p^2 p^3 , , : bin frequencies. ω^3 ω κ : value of in bin 2.	p^i = with probability . ω^1 0 ω^2 1 = , = . The default for sm is M0. Ω ω^i p^i
M3[] n sm + M3[] Yang, et. al. (2000) n	Codons	: the ts/tv ratio p^1 p^n , ..., : bin frequencies. ω^1 ω^n ω μ , ..., : values of .	= with probability . Ω Beta μ σ
M7[] n sm + M7[] Yang, et. al. (2000)	Codons	: mean of the Beta distribution. σ μ Γ : noise to signal ratio for Beta.	\sim (,). A discrete approximation to the Beta with bins is used.

9.5. Examples

Example: `--smodel EQU --alphabet Triplets`

Example: `--smodel HKY`

Example: `--smodel TN+pi=constant`

Example: `--smodel Empirical[WAG]+log-normal+INV`

Example: `--smodel M0 --alphabet Codons`

Example: `--smodel M0+pi=nucleotides --alphabet Codons`

Example: `--smodel M2 --alphabet Codons`

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

10. Output

10.1. Output directory

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

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

10.2. Output files

BAlI-Phy produces the following output files inside the directory that it creates:

l.out	Iteration numbers, alignments, and probabilities.
l.err	Success probabilities for transition kernels.
l.MAP	Successive estimates of the MAP point
l.p	Scalar parameters: indel and substitution parameters, etc.
l.trees	Tree samples

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

11. Convergence and Mixing: Is it done yet?

When using Markov chain Monte Carlo (MCMC) programs like MrBayes, BEAST or BAlI-Phy, one of the key questions is how many iterations are required to give a good estimate. Unfortunately, this question is complicated and depends upon both the problem and the specific data set that is being examined. As a result, BAlI-Phy relies on the user to determine when enough samples have been obtained - and to terminate the program when this happens.

In general, we recommend that you analyze the data from a running chain periodically and repeatedly in order to determine when to stop it.

To inspect the Markov chain generated by BAlI-Phy, we recommend the program Tracer [<http://evolve.zoo.ox.ac.uk/software/tracer/>]. You can open the file `l.p` in Tracer to view traceplots and to estimate the effective sample size. However, these techniques that work well for numerical parameters do not work well for objects such as trees and alignments. Therefore, the full analysis is necessary as well (Section 12, "Analyzing the output").

11.1. Convergence

11.1.1. Definition

What is convergence, and how can you check if a Markov chain has "converged"? We first note that "convergence" refers to the tendency of a Markov chain to converge to the desired distribution (its equilibrium distribution). The Markov chain starts at a specific tree and alignment, which may not be representative of the equilibrium distribution, and so convergence represents the process of "forgetting" the starting value. If we represent the *i*-th iteration of the Markov chain as $X[i]$, then we desire rapid convergence of the distribution of $X[i]$ to the equilibrium distribution.

11.1.2. Burn-in

In practice, most MCMC analyses designate some number of samples at the beginning of each chain as "burn-in". These samples are discarded because they represent the initial starting point, but not the equilibrium distribution.

Determining the number of samples to discard as burn-in is usually rather ad-hoc. MCMC practitioners often run the chain for a "long" time, and then visually inspect the samples, perhaps using Tracer [<http://evolve.zoo.ox.ac.uk/software/tracer/>]. After some number of iterations, the samples become "typical" of later values, and that number is chosen as the burn-in. Unfortunately, it is always possible that if you run the Markov chain even longer, then the values will change again. Therefore, this method is not strong evidence that a chain converges before a given burn-in time.

11.1.3. Testing Convergence

A better (but not perfect) test of convergence is to run several independent chains starting from many different places. It is possible that each Markov chain will get stuck in local optima near its own starting value, but that the movement between the different local optima will be exceedingly slow. However, if the posterior distributions from each Markov chain seem to be the same, then that gives some evidence that the MCMC procedure is truly converging to a global equilibrium.

11.2. Mixing

11.2.1. Definition

In MCMC, each sample is not fully independent of previous samples. In fact, a Markov chain can get "stuck" in one part of the parameter space for a long time, before jumping to an equally important part. When this happens, we say that the chain isn't "mixing" well.

11.2.2. Measures: how good is the mixing?

For continuous parameters, one common measure is the effective sample size (ESS). Given a collection of *dependent* samples from a Markov chain, the effective sample size is the equivalent number of *independent* samples. However, there are many ways one could measure "equivalence". The usual way is to compare samples based on the variance of the sum of the values.

12. Analyzing the output

In addition to running the full analysis, it will be helpful to analyze the numerical parameters using Tracer (Section 11, "Convergence and Mixing: Is it done yet?"). The full analysis should be run repeatedly as the simulation progresses, to determine when enough samples have been collected.

12.1. How to run the full analysis

First, enter the output directory. Then make a local copy of the the analysis makefile:

```
% cp prefix/share/doc/bali-phy/GNUMakefile .
```

Here, *prefix* is the directory where you installed BAli-Phy.

After you have installed the analysis makefile, run the command:

```
% make SKIP=burn-in
```

or

```
% make SKIP=burn-in SIZE=iterations
```

Here, *burn-in* represents the number of iterations to skip as burn-in, and *iterations* represents the maximum number of iterations to use in the analysis. Extra samples are discarded. In order for this to work, the directory containing the BAlI-Phy tools must be in your PATH.

Running the analysis will produce the three directories: *Results/*, *Mixing/*, and *Work/*. The *Results/* directory contains tree and alignment estimates and confidence levels. The *Mixing/* directory contains various measures of whether the chain ran long enough given its *Mixing*. The *Work/* directory contains various intermediate files that you probably do not need.

12.2. Results/

Running the above command will produce the following files in the *Results/* directory:

analysis	A description of the analysis: burn-in, length, time completed.
Report	A summary of results: posterior means and credible intervals, etc..
consensus	Summary of MAP and consensus topologies, as well as supported partitions (clades).
c-levels.plot	The number of partitions (clades) supported at each LOD level.
clevel.topology	The consensus topology at level <i>level</i> (Newick format)
clevel.tree	The majority consensus topology + branch lengths (Newick format)
MAP.topology	An estimate of the MAP tree (Newick format)
MAP.tree	An estimate of the MAP topology + branch lengths (Newick format)
MAP.fasta	An estimate of the MAP alignment, w/ taxa sorted by similarity.
MAP.phy	An estimate of the MAP alignment, w/ taxa sorted by similarity.

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

MAP-AU.html	An AU plot of the MAP alignment (rainbow color-scheme).
MAP-AU2.html	An AU plot of the MAP alignment (AA/DNA color-scheme).
MAP-AU.prob	The probabilities for each letter in the MAP alignment AU plot.
consensus.fasta	A consensus alignment, representing information shared by most alignment samples.
consensus-AU.html	An AU plot of the consensus alignment (rainbow color-scheme).
consensus-AU2.html	An AU plot of the MAP alignment (AA/DNA color-scheme).
consensus-AU.prob	The probabilities for each letter in the consensus alignment AU plot.

12.2.1. Results/consensus: partition support

This file reports the support of each bi-partition in terms of its posterior probability (PP) and posterior log-10 odds (LOD). Only partitions with PP>0.5 are shown by default.

12.3. Mixing/

In addition, the following files will be produced in the *Mixing/* directory:

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.

12.3.1. Mixing/partitions.bs: partition mixing

This file reports the quality of estimates of support for each partition, in terms of the auto-correlation time (ACT), effective sample size (Ne), and a 95% confidence interval for posterior probability (PP) and log-10 odds (LOD). It also reports the number of samples that support (1) or do not support (0) the partition, as well as the number of regenerations. Only partitions with PP>0.5 are shown by default.

13. Tuning the Markov Chain

These parameters can be set using the command line syntax "`--set parameter=value`".

13.1. Parameters

Table 4. Tunable Parameters

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

14. Auxiliary tools

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

14.1. alignment-find

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

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

14.2. alignment-draw

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

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

14.3. alignment-thin

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

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

14.4. alignment-chop-internal

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

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

14.5. alignment-info

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

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

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

14.6. alignment-indices

alignment-indices *alignment-file* [OPTIONS]

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

14.7. alignment-cat

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

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

14.8. trees-consensus

Usage: trees-consensus *file* [OPTIONS]

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

14.9. trees-bootstrap

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

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

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

14.10. trees-to-SRQ

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

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

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

15. Frequently Asked Questions (FAQ)

15.1. Running bali-phy.

15.1.1 Can I fix the alignment?

Yes. Add `-t` or `--traditional` on the command line.

15.2. Figuring out run-time errors

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

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

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

15.3. Stopping bali-phy.

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

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

15.3.2 So, how can I know when to stop it?

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

15.3.3 How can you tell when the chain has converged?

It depends on the data set. To test convergence, run at least 4 independent chains (preferably 10) from different random starting points and see if the 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.

15.3.4How can I check how many iterations the chain has finished?

Run `wc -l 1.p` inside the output directory, and subtract 2.

15.4. Interpreting the results.

15.4.1Where can I find 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.

15.4.2Where can I find the partition support?

After you analyze the output (Section 12, “Analyzing the output”), the partition support is indicated in `Results/consensus`. (see Section 12.2.1, “`Results/consensus`: partition support”).