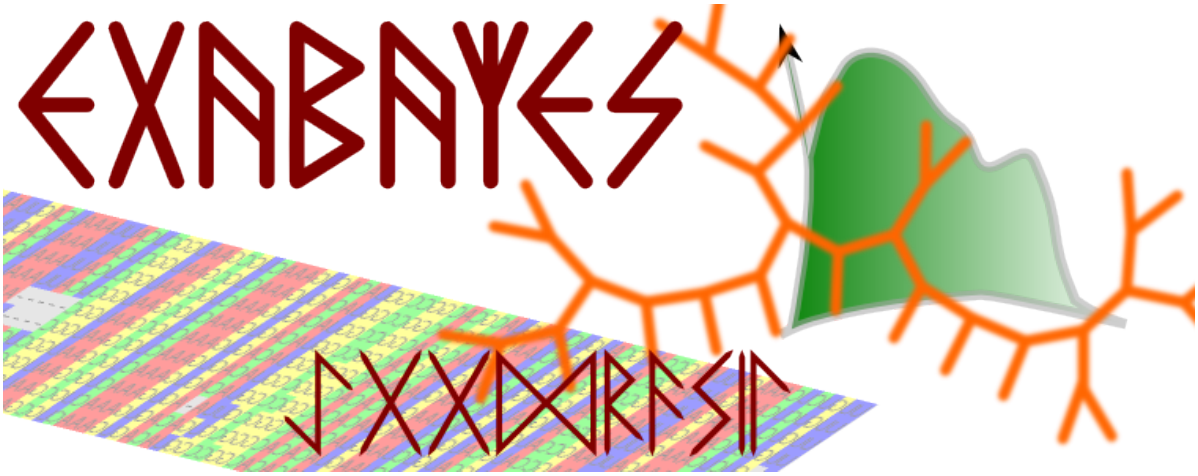

ExaBayes User's Manual

for support, please contact exabayes-at-googlegroups-dot-com

February 19, 2014

Contents

1 Quick Start

Installation of `ExaBayes` requires basic proficiency with using a terminal (for help, consider this tutorial).

For impatient users who want to give `ExaBayes` a quick try, we recommend:

1. Download and unpack the software package. You find executables (sequential: `yggdrasil`, parallel: `exabayes`) in the `./bin` folder. The `-h` flag provides you with an overview of options. If you downloaded the plain source, executing the `./build.sh` script is sufficient to build the sequential version.

2. If you have convinced yourself, that the executables are in place, consider running the examples in the `./examples` directory (call the `./call.sh` or `./call-parallel.sh` scripts there). If you want to run your own dataset, convert your alignment file into phylip-format (e.g., using `seaview`).

3. If you have prepared your dataset, run the sequential version of ExaBayes, where alignmentFile.phy is the alignment file (use -m PROT for protein data) and `$RANDOM` could be any random number seed.

   ```
   $ ./bin/bin/yggdrasil -f alignmentFile.phy -m DNA -s $RANDOM
   ```

4. After some time, ExaBayes should finish or you may abort at any time. You can now examine the two output files ExaBayes_topology* and ExaBayes_parameters* using the post-processing tools `consense`, `postProcParam`, `credibleSet` and `extractBips` (see Sect. 7). Call the respective programs with `-h` for an overview of functions.

2 Scope of ExaBayes: What is it? What is it not?

ExaBayes is a tool for Bayesian phylogenetic analyses. It implements a Markov chain Monte Carlo sampling approach that allows to determine the posterior probability of a tree (resp., topology) and various evolutionary model parameters, for instance, branch lengths or substitution rates. Similar approaches are implemented in BEAST [1] or MrBayes [4]. ExaBayes has heavily drawn inspiration specifically from the latter one.

ExaBayes comes with the most commonly used evolutionary models, such as the generalized time reversible model (GTR) of character substitution, the discretized Γ model of among site rate heterogeneity and estimates trees with unconstrained branch lengths. For clocked tree models or less parameter-rich substitution models, we refer you to the established tools.

The distinguishing feature of ExaBayes is its capability to handle enormous datasets efficiently. ExaBayes provides an implementation of data parallelism using the *Message Passing Interface* (MPI). This means, that if you conduct your analysis on a computing cluster composed of several machines (a.k.a. nodes), the memory needed to evaluate the

likelihood of trees and parameters given a large alignment can be spread out across multiple computing nodes. In conclusion, the size of the concatenated alignment ExaBayes can handle is only limited by the combined main memory of your entire computing cluster.

Aside from that ExaBayes also implements

- chain-level and run-level parallelism,
- techniques to trade runtime for reduced memory footprint,
- a subtree equality vector approach that reduces memory without loss of runtime,
- a native AVX implementation for evaluating likelihood and parsimony scores (i.e., ExaBayes makes full use of your cutting-edge CPU),
- techniques to efficiently handle an arbitrary number of partitions.

We use the highly efficient parsimony and likelihood implementation of RAxML [6]. Many of the techniques described above are adapted from or inspired by our experiences with large-scale maximum likelihood inferences using RAxML-Light/ExaML [7, 5].

The ExaBayes package contains all tools necessary for post-processing your sampled chains. For visualization of parameter distributions, we recommend Tracer and FigTree (for which ExaBayes parameter files are compatible).

3 Installation

For ExaBayes, we provide pre-compiled binaries that run on a wide range of systems (limited to Linux and MacOS though, we will not be able to support Windows in the foreseeable future). For optimal efficiency or if you want to use the parallel version, it is highly recommendable to compile ExaBayes from source. This should be straightforward on a computer center, requires a bit of work on a Linux system and unfortunately is not entirely trivial if you have a MacOS system and no experience with the command line.

3.1 Executing downloaded executables

After you have downloaded the appropriate package, you find all executables in the `./bin` folder. Just execute these only using the `-h` flag and you will be given a help page. If the executables produce error messages (indicating that some library was not found), you either downloaded the wrong package or you have to compile from source.

3.2 Compiling ExaBayes from source

Download and extract the source archive (sources are also included in all binary distributions).

3.2.1 Prerequisites for Linux systems

ExaBayes requires a relatively recent `c/c++` compiler that supports `c++11` features. ExaBayes is confirmed to work with (only one required):

1. GCC, version 4.6 or greater
2. Clang, version 3.2 or greater

For running ExaBayes in parallel, you need a working MPI installation. If you want to make most of your local multi-core machine (e.g., laptop), simply install OpenMPI or Mpich2. On Debian/Ubuntu, this should be as simple as (choose one):

```
$ sudo apt-get install mpich2 libmpich2-dev
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

For checking, if MPI already is installed on your machine, try to enter `mpirun` or `mpiexec` in a terminal. If it is not installed, you will receive a message "command not found".

3.2.2 Prerequisites for Mac OS X

For installation on an Apple system, you ideally should have set up an environment that allows you to compile (MPI-)applications in the terminal.

First, you need to download and install Xcode (also available in the AppStore) and MacPorts. You only need MacPorts, if you want to build the parallel version. Open a terminal and use MacPorts to install an MPI implementation (either openmpi or mpich2):

```
$ sudo port install openmpi-default
$ sudo port install mpich-default
```

After the installation, a message will suggest that you set the MPI installation as default. You now have MPI compiler wrappers available (mpicc-mpich-mp, mpicxx-mpich-mp *or* mpicc-mpich-mp, mpicxx-mpich-mp *or* mpicc, mpicxx).

For installing MPI, also consider HomeBrew which is an alternative to MacPorts that does not require `sudo`.

3.2.3 Automated build

We provide a small script that compiles ExaBayes and dumps all binaries into the `./bin` folder. If you want to build the parallel version (and you already have installed MPI as described in the previous section), then you have to provide the names of the mpi compiler wrappers. If you just want the sequential version, just execute the script.

Sequential only:

```
$ ./build.sh
```

Sequential and parallel version (where mpicc and mpicxx are the names of the mpi compilers):

```
$ ./build.sh CC=mpicc CXX=mpicxx
```

If you build on MacOS, an additional option will be necessary **before** you call the build script:

```
$ export CXXFLAGS="-stdlib=libc++"
```

Once everything has compiled, check, if all binaries are there as expected:

```
$ ls bin
    consense credibleSet exabayes extractBips parser postProcParam sdsf yggdrasil
```

3.2.4 Customized build

If you have a bit of experience with the usual Linux way to install programs, you may want to read this section for additional advice on the build process. This section also helps, if the automated build fails for whatever reason (although preferably you can contact our googlegroup).

Essentially, you have to build the parallel and sequential executables separately. Since it is hard to guess the mpi compiler correctly, you have to explicitly set the MPI C and MPI C++ compiler.

The following commands first build the sequential version, then the parallel version. For MacOS, Remember to set CXXFLAGS as mentioned in the previous apple-specific section.

```
$ ./configure
$ make
$ ./configure --enable-mpi CC=mpicc CXX=mpicxx
$ make clean
$ make
```

Tip: Use make with `-j n`, where `n` is the number of cores on your system.

4 Tutorial and Workflow

This is a basic tutorial for how to conduct Bayesian tree inference with `ExaBayes` (specifically useful, if you do not have much experience with Bayesian tree inference).

Assume you want to analyze an alignment file that contains several partitions. Create a folder and copy `aln.phy` and `aln.part` from the folder `examples/dna-partitioned` into that folder. Copy `./examples/configFile-all-options.nex` as well and rename it to `config.nex`. We will assume that all executables are in this working folder (otherwise, modify the path to the executable accordingly).

4.1 Basic Workflow

Let's at first simply run a single chain for some time:

```
$ ./yggdrasil -f aln.phy -m DNA -n myRun -s $RANDOM
```

`ExaBayes` will print output that is separated into various sections:

1. after the header, you find a section (divided by '=') that re-iterates the alignment (number of unique patterns and type of partitions).
2. In the next section, `ExaBayes` lists the parameters to be integrated. For instance, the tree topology is considered a parameter. It also displays the (default) prior for parameters and initial values.
3. The 3^{rd} section contains the proposals that are instantiated for integrating over the aforementioned parameters.

During the MCMC simulation, ExaBayes prints the log-likelihoods (lnl) of the (sole) chain. You'll find that after some time the lnl will reach a plateau. Stop the run after something like 50,000 generations (using Control-c).

Now examine the output files created by ExaBayes (we neglect the binary alignment file and the checkpoint files):

- ExaBayes_info.myRun
  Contains the same information also printed to the screen.
- ExaBayes_topologies.myRun.0
  Contains all sampled topologies in nexus format.
- ExaBayes_parameters.myRun.0
  Contains values sampled for all non-topological, non-branch length values.
- ExaBayes_diagnostics.myRun
  Contains chain diagnostics (e.g., acceptance ratios for all proposals or topological convergence in form of asdsf).

Now use the post-processing tools to examine the result. First, we create a consensus tree:

```
$ ./consense -f ExaBayes_topologies.myRun.0 -n myCons
```

Now, open a tree viewer of your choice (e.g., FigTree, Archaeopteryx or Dendroscope) and have a look at the consensus tree. If you ran just 50,000 generations, you will probably find the confidence in most branches is pretty low.

Let's inspect the 50% credible set of trees:

```
$ ./credibleSet -f ExaBayes_topologies.myRun.0 -n cred
```

The output file `ExaBayes_credibleSet.tmp` contains all sampled trees ordered by the frequency of their occurrence. You probably will find that no tree occurred more than once.

Finally, let's check, how well the parameters are sampled:

```
$ ./postProcParam   -f ExaBayes_parameters.myRun.0  -n params
```

Alternatively, you could also open the parameter file with `Tracer` and visualize the distributions. If you do not have `Tracer` installed, have a look at `ExaBayes_parameterStatistics.params` (spreadsheet tools like Excel are helpful). You'll find summary statistics for each parameter. Specifically, check out the effective sampling size (ESS) value for each parameter. Since samples in a chain are correlated, they are less informative, than if you had drawn the values independently from the original distribution. The ESS of samples indicates the number of samples your samples corresponds to, if they were drawn independently.

You will find that most ESS values are in the range between 30 and 80. This is not bad for the low number of generations, but to assure that each parameter has been sampled adequately, values should be > 100 or even better > 200. High ESS values indicate that your chain has explored this parameter sufficiently.

4.2 Partitioned Alignment

Now remove the files from the initial run (otherwise ExaBayes will complain). We will now conduct a proper analysis using several chains on a partitioned alignment.

You have to modify `config.nex` (remove the square brackets to uncomment). Uncomment `numGens` and set it to 100000 (or 1e5). Uncomment `numRuns` and set it to 4. This means that `ExaBayes` will conduct 4 independent analysis (starting in 4 different random trees). If all runs yield the same split frequencies (i.e., the same confidence in a split), we can be relatively sure that we have sampled the topology parameter sufficiently (while it is still possible that simply all 4 chains got stuck in the same local optimum).

Check out the partitions file (`aln.part`): it contains 4 partitions. By default, ExaBayes assumes that partitions have distinct branch lengths. Let's link all partitions into a single branch length parameter. To do so, add `"brlens = (0-3)"` in the `params` section.

Use this command line to start the analysis:

```
$ ./yggdrasil -f aln.phy -q aln.part  -n myRun -s $RANDOM -c config.nex
```

By default, ExaBayes will compute the average standard deviation of split frequencies (asdsf) every 5,000 generations and stops the analysis once it the asdsf is better than 5% (and once we have at least 100,000 generations for each chain).

The analysis may take a while. If you have a cluster (with for instance 16 cores) at your disposal, consider running the parallel version:

```
$ mpirun -np 16  ./exabayes -f aln.phy -q aln.part  -n myRun -s $RANDOM -c config.nex -R 4
```

With `-R 4`, ExaBayes runs all 4 chains in parallel. In an additional section, ExaBayes will inform you which chains and how many unique site patterns are assigned to each process.

You will notice after 100,000 generations, that the 4 chains converge rather slowly. Thus, enable Metropolis-Coupling to speed up the convergence. Set `numCoupledChains` to 2 and (for purely computational reasons in this example), reduce the `numRuns` to 2. Also, set `parsimonyStart` to true, such that your chains start from a parsimony tree instead of a random tree. Using parsimony trees as initial topologies saves you some time, however theoretically it also increases the probability that all your independent chains become stuck in the same local optimum and you obtain incorrect estimates for posterior probabilities.

If you have many cores, to run the two independent runs as well as the coupled chains in parallel:

```
$ mpirun -np 16 ./exabayes -f aln.phy -q aln.part -n myRun -s $RANDOM -c config.nex -R 2 -C 2
```

After $\approx$ 150,000 generations, the ASDSF should have fallen below 5%, which is acceptable. A look at the consensus tree reveals that this dataset in fact contains several low confidence branches. While in the first run the reason for low confidence branches was due to the low number of generations, we can be more certain now that that low confidence branches are a result of the phylogenetic signal in the alignment. Since you ran 2 independent analyses, you obtain 2 sets of output files (parameter and topologies). You can feed both files into the consensus tree (for both files the initial 25% of samples will be discarded).

```
$ ./consense -f ExaBayes_topologies.myRun.* -n myCons
```

Analyze both parameter files using

```
$ ./postProcParam -f ExaBayes_parameters.myRun.* -n params
```

Since we carried out two partitioned analyses, we have a larger number of parameters (where e.g., `r{2}(C<->T)` is the substitution probability of C to T in the third partition). We find that nearly all parameters have an ESS > 100. The final column now contains the potential scale reduction factor (PSRF). It indicates, whether within-chain variance (of a parameter) is similar to between-chain variance. For this convergence statistic, values should be close to 1, but lower than 1.2 or 1.1 (probably the case after 150,000 generations).

So far, we have neglected the branch length parameter. You may already have noticed, that the consensus tree has been annotated with branch lengths. To extract ESS and PSRF values for each branch length, run:

```
$ ./extractBips -f ExaBayes_topologies.myRun.* -n bls
```

Thus, you obtain the following files:

- ExaBayes_bipartitions.tmp
  contains an identifier for each branch/split (that is explicitly printed)
- ExaBayes_fileNames.tmp
  lists the file names used as input (and assigned an id to them)
- ExaBayes_bipartitionBranchLengths.tmp
  contains all raw branch lengths sampled and lists split id and file ids
- ExaBayes_bipartitionStatistics.tmp
  contains statistics for each branch (specifically the ESS and PSRF)

Notice that for low confidence branches you are likely to encounter poor ESS/PSRF values. This means that you substantially have to increase the number of generations in order to obtain accurate estimates of branch lengths distributions.

5 Command Line Options

Command line options specify, **how** ExaBayes will carry out the analyses. In contrast, the a config file specifies which kind of analyses will be executed.

5.1 Mandatory Arguments

- **-f alignmentFile**

  provides an alignment file. If this file is the binary output produced by `parser` (see Section 7.1), then no further arguments are required. If you provide a plain (un-processed) Phylip file, then either `-m` (single partition model) or `-q` (model file) are mandatory.

- **-m DNA | PROT**

  specifies the data type used, when a Phylip-formatted alignment has been passed via `-f`. This way, the alignment is parsed as a single partition with either DNA or amino acid (`PROT`) data.

- **-q modelFile**

  specifies a raxml-style partitioning/model scheme for the alignment. For this option, a Phylip-formatted alignment must be passed via `-f`. See Section 9 for a description of the file format.

- **-s seed**

  provides a random seed. This number makes the run reproducible. The same seed, data set configuration file will result in the exact same result (apart from limitations given in Section 8.5). If you restart from a checkpoint file, this option will be ignored.

- **-n id**

  provides a run id used for naming output files

- **-r runid**

  restarts your run from a previous run id. If your previous ExaBayes-run did not finish (because of a manual abort or walltime restrictions), this option can be used for continuing the run. It is essential, that you pass the same configuration and alignment file. Some adaptions to the configuration file are possible (e.g., larger number of generations to be run, lower topological convergence threshold). Furthermore, all files that carry the previous runid in their name must be located in the current folder.

  Example:

```
$ mpirun -np 8 ./exabayes -s $RANDOM -n myId -c myConfig -f myBinaryAlnFile.bin
$ [runnig....] -> aborted!
$ mpirun -np 2  ./exabayes -r myId -n myIdContinued -c myConfig -f myBinaryAlnFile.bin -S
```

5.2 Optional Arguments

- **-d**

  carries out a dry-run. Only checks your config and alignment file and does not compute anything. Very recommendable, before submitting a large run to a cluster.

- **-c configFile**

  passes a configuration file that specifies how the MCMC will be carried out (see ./examples/configFile-all-options.nex and Section 6 for details)

- **-w workDir**

  specifies a location for output files

- **-R num**

  (`exabayes`-only) specifies the number of runs (i.e., independent chains) to be executed in parallel. Large runs should be carried out as separate runs, see Section 8 for further details.

- **-C num**

  (`exabayes`-only) specifies the number of chains (i.e., coupled chains per independent run) to be executed in parallel. Employing this option may be less efficient in terms of runtime and memory than data-level parallelism, see Section 8 for further details.

- **-Q**

  (`exabayes`-only) enables per-partition data distribution. This option assigns entire partitions to processors. Thus, If your alignment comprises more partitions

than you have processors available, this option is likely to speed up calculations substantially. You should check the print-out right before the start of MCMC sampling about whether load is distributed equally.

- **-S**

  try to save memory using the SEV-technique for gap columns on large gappy alignments Please refer to this paper. On very gappy alignments this option yields considerable runtime improvements.

- **-M mode**

  specifies the memory versus runtime trade-off. <mode> is a value between 0 (fastest, highest memory consumption) and 3 (slowest, least memory consumption). See Section 8.3 for details.

6 Configuration File

In this Section, we describe all available options of the configuration file in detail. The configuration file is a file in nexus-format that is divided into sections. See examples/all-options-documented.nex for a complete version (and maybe copy and customize this file). None of the following blocks in mandatory. The parameter file itself is not mandatory and the default values mentioned below are used instead. The nexus-syntax for declaring a block is (here declaring a `run` block).

```
begin run;
   option value
end;
```

6.1 Declaring and Linking Parameters

keyword: `params`

This section allows to declare and link parameters (e.g., branch lengths) across partitions. You should have declared partitions in the partition file (passed via `-q`). If you provided a partition file to the `parser` tool, then the binary output file already contains information about partitions. Partition ids start with 0 and refer to the order provided in the partition file.

Currently the following keywords can be used to specify a parameter linking scheme (keywords are case-insensitive):

param	explanation
`stateFreq`	link the equilibrium state frequencies (4 for DNA, 20 for AA) for partitions
`rateHet`	link the alpha parameter of the Γ distribution of rate heterogeneity among sites
`revMat`	link the substitution rates in the GTR matrix (DNA:6, AA:190) across partitions
`brlens`	link branch lengths across partitions
`aaModel`	link the fixed rate substitution matrix across partitions (if applicable)

Note that, by default all parameters are unlinked for all partitions. Specifically regarding branch lengths, most people only want one global branch length parameter. If a partition id is omitted from the scheme, the default behaviour of ExaBayes is to instantiate a new parameter for this partition (i.e., it is unlinked).

You have the following options for specifying linkage (here demonstrated for the branch length parameter):

- use *comma* to declare partitions as separate parameters
  example: `brlens = (0,1,2,3)`
  result: v{0}, v{1}, v{2}, v{3}

- use *plus* to link two partitions into one parameter
  example: `brlens = (0 + 1 , 2 , 3)`
  result: v{0,1}, v{2}, v{3}

- use *colon* to declare a range of unlinked partitions (each one parameter)
  example: `brlens = (0:3)`
  result: v{0}, v{1}, v{2}, v{3}

- use *dash* to declare a range partitions linked into one parameter
  example: `brlens = (0-3)`
  result: v{0,1,2,3}

For most use cases, you probably will only want to link all branch lengths. However, in case you work with protein partitions, please consider:

- By default ExaBayes creates one `aaModel` parameter for each of your amino acid partitions. As state frequencies, ExaBayes uses the empirical frequencies provided by the respective amino acid substitution matrix.

- Instead of using the empirical frequencies, you may want to let ExaBayes integrate over these state frequencies. For doing so, you simply have to declare one of the respective partitions when specifying the `stateFreq` parameter scheme. If you have two AA partitions, then `stateFreq = (0)` instructs ExaBayes to integrate over the state frequencies of the first amino acid model parameter.

- As an alternative to proposing fixed-rate AA substitution matrices for AA partitions, you can use ExaBayes to integrate over amino acid GTR matrices (189 free parameters). For doing so, declare (and link) the respective AA partitions in the `revMat` linking scheme (e.g., `revMat = (0+1)` for 1 shared GTR matrix across 2 AA partitions).

6.2 Declaring Priors for Parameters

keyword: `priors`

The prior block let's you declare your prior belief regarding the values of parameters ExaBayes integrates over. This affects parameters implicitly instantiated by ExaBayes or explicitly defined in a `params` block (see Section 6.1).

By default priors specifications are applied to all matching parameters. You can overwrite these *general* priors by specifying parameter-specific priors. For doing so, list all at least one partition that is assigned to your target parameter in curly brackets after the prior keyword. For instance:

```
brlenPr exponential(10)
brlenPr{0,2,10} uniform(1e-6,10)
```

applies a uniform prior with $[1e-6, 10]$ to all branch length parameters that contain the partitions 0,2 or 10 and applies an exponential prior with $\lambda = 10$ to all remaining branch length parameters.

6.2.1 Topology Prior

keyword: `topoPr`,
default: `topoPr uniform()`
valid values:

- `fixed()`
  topology is kept fixed
- `uniform()`
  all topologies have the same prior probability

6.2.2 Branch Lengths Prior

keyword: `brlenPr` ,
default: `brlenpr exponential(10)`
valid values:

- `exponential(` λ `)`
  exponential prior with parameter λ,
- `uniform(start,end)`
  uniform probability in the range $[start, end]$
- `fixed(` *val* `)`
  all branch lengths will be assigned the value *val* that is kept fixed during the analysis

- `fixed()`
  all branch lengths keep original branch length provided via a starting tree. If no starting tree is available, a default value (currently 0.1) is assigned and kept fix during MCMC sampling.

6.2.3 Reversible Matrix Prior

keyword: `revMatPr`
default: `revMatPr dirichlet(1,...,1)`
valid values:

- `dirichlet(` $x_1, x_2, \ldots, x_n$ `)`
  where for a dirichlet prior x_i are the substitution rates in a GTR matrix and thus $n = 6$ for DNA GTR matrices and $n = 190$ (use with care) for AA GTR matrices.

- `fixed(` $x_1, x_2, \ldots, x_n$ `)`
  fixed rates are assigned to the matrix and kept fix during MCMC sampling. The values x_i may be expressed as relative rates (i.e, ExaBayes will normalize the rates, s.t. they sum up to 1.0)

6.2.4 Rate Heterogeneity Prior

keyword: `shapePr`,
default: `shapePr uniform(0,200)`
valid values:

- `exponential(` λ `)`
  prior probability of α values have an exponential distribution with parameter λ

- `uniform( start, end )`
  α values have uniform prior probability in the range $[start, end]$

6.2.5 State Frequencies Prior

keyword: `stateFreqPr` ,
default: dirichlet(1,1,...,1)
valid values:

- `dirichlet(` $x_1, x_2, \ldots, x_n$ `)`
  where for a dirichlet prior x_i are the state frequencies in a GTR matrix and thus $n = 4$ for DNA and $n = 20$ in a protein GTR matrix.

- `fixed(` $x_1, x_2, \ldots, x_n$ `)`
  fixed values are assigned to the state frequencies and not changed during MCMC sampling. x_i can be expressed as relative rates (i.e., if the sum is ≥ 1, ExaBayes does the normalizing for you)

6.2.6 Amino Acid Model Prior

keyword: `aaPr`,
default: `aaPr disc(remainder=1.0)`
valid values:

- `disc(` $m_1 = w_1$, $m_2 = w_2$, ..., $m_n = w_n$ `)`
  a discrete probability distribution assigning weights w_i to protein substitution matrices m_i. If only one model is specified, this is equivalent to a fixed prior.

  m may be one of the following models: DAYHOFF, DCMUT, JTT, MTREV, WAG, RTREV, CPREV, VT, BLOSUM62, MTMAM, LG, MTART, MTZOA, PMB, HIVB, HIVW, JTTDCMUT, FLU.

  By default, if a model is not mentioned in the list, then its prior probability is 0 and thus is not considered during MCMC sampling.

  Additionally, you can include remainder value (i.e., `remainder=` w_i). This means that all matrices not mentioned have a prior probability of w_i.

- `fixed(` m `)`
  fix the value of the parameter to one of the models listed above

6.3 Configuring the Run

All of the following options need to be enclosed within a block featuring the keyword `run`.

6.3.1 General Options

The following options allow you to exactly configure what kind of Bayesian sampling is performed. Keywords and default values are mentioned along the description of the options.

The most important settings are, how many independent runs (**numRuns**, default: 1) you want to run for how many generations (**numGen**, default: 1,000,000). If you execute exactly 1 run, then ExaBayes will terminate after `numGen` generations. For more than 1 run, ExaBayes will terminate once `numGen` generations have passed and one of the following topological convergence diagnostics are below a specified threshold.

By default, ExaBayes draws a sample from every cold chain (i.e., for each independent run) every 500 generations (can be changed via **sampleFreq**). To change the print frequency (informing you about the likelihood state of each chain), modify **printFreq**.

ExaBayes updates a checkpoint file at regular intervals (1,000 generations by default), the respective variable for changing the frequency is **checkPointInterval**.

If you do not specify starting trees, then ExaBayes uses random trees as initial topology. If you set **parsimonyStart** to `true`, a parsimony starting tree will be used instead (recommendable).

Some proposals (e.g., the branch length multiplier) can be tuned for achieving good acceptance ratios. ExaBayes tunes proposal parameters, once a proposal has been drawn 100 times (use **tuneFreq** to change this, set it to 0 to disable tuning).

If you are running a dataset in parallel that comprises many partitions (possibly using the -Q option), it is advisable to group the proposals per partition into proposal sets (i.e., set **proposalSets** to `true`, this is the default). If proposal sets are enabled and you have for instance multiple substitution matrix parameters, then ExaBayes will propose new substitution parameters for each substitution matrix parameter one after another (instead of only drawing one of the parameters at random).

6.3.2 Options regarding convergence

ExaBayes implements the same diagnostics for topological convergence as MrBayes and BEAST. These are either the maximum or the average deviation of split (i.e., bipartition) frequencies (MSDSF/ASDSF). By default, ExaBayes employs the ASDSF. You can change to MSDSF by setting **convergenceCriterion** to `max`. For disabling the convergence detection, set it to `none`.

The convergence threshold for either of these statistics can be specified via **sdsfConvergence** (default: 0.05, i.e., the respective statistic must be $\leq 5\%$). Usually, splits that exhibit a low posterior probability are excluded from this statistic, since it is hard to determine their probability accurately. You can specify the exclusion threshold for the ASDSF/MSDSF via **sdsfIgnoreFreq** (default: 0.1, i.e., splits that do not occur in at least 10% of the trees of a run are ignored).

Also relevant for the convergence statistic is how many samples are discarded by ExaBayes as burn-in. By default, the initial 25% of all sampled trees are discarded (change this via **burninProportion**). If you want to use an absolute burn-in, specify **burninGen** (e.g., `"burninGen 1e4"`) instead. In this case, all trees sampled prior to generation 10,000 are discarded.

ExaBayes checks for topological convergence once every run has proceeded by $5,000$ generations (set the **diagFreq** variable to change this value).

6.3.3 MC3 options

If you sample a rough likelihood landscape, you may want to employ Metropolis-coupled MCMC (MC3, turned off by default). In very brief terms, this means that a number of heated chains are coupled to the cold chain (from which samples are drawn). All coupled chains attempt to swap their states at regular intervals. Thus, the cold chain can be enabled to reach regions of the parameter space (potentially separated by values with low posterior probability) that are otherwise very unlikely to be sampled.

The total number of coupled chains can be specified via **numCoupledChains**. This number includes the cold chain, so if you want to add three heated chains, `"numCoupledChains 4"` is the correct statement.

The chains are heated incrementally, so the more chains you added, the hotter the hottest chain will get. The heat β for the i-th heated chain (where $i = 0$ for the cold chain) is defined as

$$\beta = \frac{1}{1 + i \cdot \delta}. \tag{1}$$

When deciding upon acceptance of a new state, the likelihood and prior ratio are exponentiated with β (thus increasing the acceptance probability for heated chains). By default, the heat constant δ is set to 0.1. The value changed by setting the variable **heatFactor**.

The expected number of swap attempts between chains per generation (i.e., after each chain has proceeded this many generations) can be specified via **numSwapPerGen** (default: 1). This is a very important variable that affects both the performance of the MC3 mechanism as well as the your parallel runtime performance (if applicable).

The reason for this is, that an increase of number of coupled chains will not directly translate into more efficient sampling. If the number of swap attempts is kept constant, then it becomes increasingly unlikely that any change is propagated to the cold chain as you increase the number of heated chains. On the other side, if you run coupled chains in parallel (-R argument), then more swapping attempts will lead to increased waiting times. This is, because processes computing the chain will have to wait for processes that compute the likelihood of the other chain involved in a swap attempt.

If you want heated chains to start from the same topology as the cold chain, set **heatedChainsUseSame** to `true`.

6.4 Configuring Proposals

ExaBayes allows you to configure proposals that are used to move your chains through the parameter space. For each proposal, a relative weight governs, how often a specific proposal is drawn. You can customize your proposal mixture by modifying these weights. A proposal provides values for a single parameter only, so a change of the relative weight affects all related proposals. Specifically the topological proposals are described in detail in [3].

Using proposal sets does not change how often a proposal is drawn relative to runtime (so no modifications are necessary).

keyword	full name	affected parameters	default weight
nodeSlider	node slider	branch lengths	5
treeLengthMult	tree length multiplier	branch lengths	1
branchMulti	multiplier on branch lengths	branch lengths	15
eTBR	extending tree bisection and reconnection (eTBR)	topology	5
eSPR	extending subtree pruning and regrafting (eSPR)	topology	5
parsimonySPR	parsimony-biased subtree pruning and regrafting	topology	5
stNNI	stochastic nearest neighbor interchange	topology	5
rateHetMulti	multiplier on α	rate heterogeneity	1
revMatSlider	sliding window	rev. matrix (DNA,AA)	1
revMatDirichlet	dirichlet proposal	rev. matrix (DNA,AA)	1
RevmatRateDirich	partial dirichlet proposal	rev. matrix (AA)	1
frequencySlider	sliding window	state frequencies	0.5
frequencyDirichlet	dirichlet proposal	state frequencies	0.5
aaModelJump	fixed AA matrix	amino acid model	1

Moreover, the behaviour of the topological proposals can be customized. The eSPR prunes a subtree, follows down a random path (starting with the original pruning position) and chooses the current branch as re-grafting position with a certain stopping probability (keyword: `eSprStopProb`). In case of the eTBR, the tree is bisected at a branch and the bisected branch traverses the tree on both ends as described for the eSPR (keyword for the stopping probability is `eTbrStopProb`).

The parsimony-biased SPR (parsSPR) move prunes a subtree and proposes a regraft position proportionally to the parsimony score of the resulting tree. The parsSPR evaluates the parsimony score for regrafting positions that are no more than n steps (keyword: `parsSPRRadius`) apart (i.e., it considers branches within a specified radius for re-insertion). Computing the parsimony score is extremely fast and parallelized in ExaBayes. If you are dealing with large trees, consider increasing the radius. It may not increase mixing, but definitely will reduce the burn-in time and the increase in runtime should not be problematic. The default value depends on the logarithm of the number of taxa (a reasonable assumption, if we do not expect comb-like trees).

Similar to MrBayes, parsimony scores are *heated* (i.e., exponentiated) using the value of `parsimonyWarp`. If this value is decreased, the probability that trees with low parsimony score are proposed will get higher.

keyword	default value
eSprStopProb	0.5
eTbrStopProb	0.5
parsimonyWarp	0.10
parsSprRadius	$\lfloor 2 \cdot log(n) \rfloor$

7 Pre-/post-processing utilities

For all utilities, please use the -h option, the documentation is mostly sufficient to execute the programs. In this section, we provide additional hints and caveats about employment of these tools.

7.1 parser

This utility parses an phylip-formatted alignment and creates a binary representation of this alignment. You either have to indicate the data type of a single partition alignment (via `-m`) or provide a model file via `-q` (see Section 9).

Parsing large alignment can take a considerable amount of time that is lost manifold when ExaBayes is executed in parallel.

7.2 postProcParam

This utility can be used to summarize (similar to sump in MrBayes or the summary statistics in Tracer) all sampled parameters.

This is straight-forward for continuous parameters (such as substitution rates). If you integrate over fixed protein model matrices (e.g, WAG, LG,...), you are integrating over a discrete parameter. The output in the ExaBayes_parameters* will list the respective matrices. In this case, postProcParam will create an extra column that contains the discrete distribution.

You should check, if all ESS values are greater than 100 and (if available) PSRF values are close to 1 (< 1.1 is considered good convergence).

7.3 sdsf

This utility computes deviations of split frequencies (either maximum or average, abbrev. as ASDSF/MSDSF). If you are integrating over topologies (you usually are), ASDSF/MSDSF are an essential convergence criterion. Usually an ASDSF of $0.5 - 1\%$ is considered "excellent convergence" and values between $1 - 5\%$ are considered to be acceptable.

You will encounter strongest deviations for branches with low posterior probability.

The stand-alone `sdsf` computes the same result that is also calculated, if you run multiple independent analysis with convergence criterion. If you run an exceptionally large analysis with multiple independent runs and plan on sampling a very large number of trees, it is recommendable to launch each independent run as a distinct ExaBayes session. You could have a master-script that launches the independent runs (to be run for e.g., 2 h), then checks for convergence and restarts the runs from the respective checkpoints, if not converged yet. If an immense number of processes is involved and your cpu-h budget is tight, this saves you sequential overhead.

7.4 credibleSet

This utility computes the credible set of topologies (up to a specified percentile) in one or many tree sets. Use it for post-analyses of your tree samples.

7.5 extractBips

This utility extracts bipartitions (AKA splits or edges) from tree sets and the branch lengths associated with these bipartitions. Note that, this utility also examines trivial bipartitions (these correspond to outer branches in a tree).

extractBips produces the following files:

- **ExaBayes_bipartitions.*** lists the smaller partition of a bipartition (i.e., all taxa omitted are in the complementary partition) and assigns a unique identifier to the bipartition.
- **ExaBayes_fileNames.*** lists the file names of the input topology files and assigns a for reference in the remaining two files.
- **ExaBayes_bipartitionBranchLengths.*** contains all unique branch lengths samples associated with a specific bipartition in a specific file. The file id and bipartition id from the previous two files are used for that.
- **ExaBayes_bipartitionStatistics.*** contains summary statistics for the branch lengths associated with bipartitions (similar to the output of postProcParam). The ESS value indicates, whether you have sufficiently sampled the branch length associated with a branch and the PSRF value can be used to judge, if the samples from different chains converged against the same distribution. You have sufficiently sampled a parameter, if the ESS is > 100 and a PRSF < 1.1 is an indicator of good convergence.

If a bipartition occurs only in one chain, extractBips will produce `-nan`-values.

7.6 consense

This utility allows to build consensus trees from one or more tree sets. If computing the consensus tree (specifically the extended MR consensus) becomes computationally challenging, you may want to give the parallelized consensus tree algorithm in RAxML a try (use `-J MRE`).

`consense` will produces two output files (one in Newick format, one in Nexus format). Nodes are annotated with marginal probability (i.e., confidence), median, mean and 5%/95% quantile values.

If you ran analyses with unlinked branch lengths (i.e., you had multiple partitions with a branch length parameter each), then you should create one consensus tree for each branch length parameter. ExaBayes writes one topology file per parameter and per analysis. So you can consense all files that have a "tree-x" (where x is the id of the parameter) in their name.

8 ExaBayes on Clusters/Supercomputers

The striking feature of ExaBayes is its capability to execute standard analyses on clusters and super-computers efficiently. This section goes through various aspects worth considering.

On clusters you often have to load a MPI module first. If you downloaded binaries, try to execute ExaBayes using `n` processes and using either `mpirun` or `mpiexec` as follows:

```
$ mpirun -np n ./exabayes <further args>
```

The exact invocation may vary depending on the MPI installation. If this does not work, make sure you downloaded the corrected package or consider compiling ExaBayes from source. On clusters, you usually have to provide a batch script that is committed to the scheduler.

In ExaBayes, you may have several computing nodes working on a chain in parallel. We refer to the entirety of nodes computing a chain as *parallel unit*.

8.1 TL;DR summary

In most cases, if you have x processes available (e.g., 4 machines with each 12 cores $\Rightarrow$ 48 processes):

- if you run n independent analyses, use `-R n`.
- if you want to run m coupled chains, check, if ExaBayes becomes faster, if you increase from `-C 1` (default) over `-C 2` to `-C 4` up to `-C m` (often the optimum is 2 or 4). For each increment (default: 500 generations), ExaBayes prints the time required for the last increment (use this for benchmarking).
- check the load balance output. For good parallel efficiency, each process should at least have $\approx$ 100 patterns. Otherwise, less processes may be sufficient, but more do not hurt, if you are not under a budget constraint.

8.2 Choosing the right kind of parallelism

ExaBayes implements three levels of parallelism (in descending order of granularity):

- runs-level parallelism,
- chain-level parallelism,
- data parallelism.

For optimal performance, please consider the following example. Assume, you run m coupled chains and n independent runs, while you specify that m_p coupled chains and n_p independent runs are run in parallel (via `-R` and `-C`). For reasons of load balance, m

should be a multiple of m_p (analog for n). Assume each of your computing nodes has k cores and you want to use l computing nodes for each parallel working unit (working on one coupled chain in an independent run that is executed in parallel). Thus, you will obtain optimal performance, if you execute ExeBayes with a total number of processes of

$$processes = m_p \cdot n_p \cdot l \cdot k. \tag{2}$$

If the number of cores k is divisible by 2, $l = 2^i$ (where $i < 0$) works as well. This way several parallel working units fit on a node.

8.2.1 On Run-level Parallelism

Obviously, run-level parallelism is the most efficient kind of parallelism. Processes working on different runs rarely have to communicate with each other (except for writing a checkpoint, so make sure your checkpointing frequency is not too low). If you instruct ExaBayes to execute 2 runs parallel, then using twice as many processes should result in an optimal speedup of two.

Alternatively, you can commit each independent run separately to the cluster and naturally get the same parallel speedup this way. You will save computational time, if you regularly check for topological convergence (using the `sdsf` tool). So one possibility is to commit several runs for which you specify a large number of generations in the config file and a relatively short walltime (maybe 2h). After the scripts have finished, another script checks the ASDSF and recommits the runs (using the checkpointing functionality `-r`). Or you could commit several jobs for each of your run and each job has to wait for the previous job to finish (e.g., using `-hold_jid` with Grid Engine).

The optimal strategy depends on the configuration of your cluster/supercomputer. In some instances a single large run parallelized via `-R` allows your job a higher priority in the queue, in other instances smaller jobs that run for a short period will allow you to get the results as quickly as possible.

If you sample an immense number of trees using an immense number of processes, we recommend to choose a non-monolithic (e.g., the second) strategy. The `sdsf` requires a bit of runtime on its own that increases with the number of trees and that is lost manifold, if many processes have to wait before they continue.

8.2.2 On Chain-level Parallelism

Employing chain-level parallelism in Bayesian analyses comes with some caveats. The speedup you can achieve with coupled chains strongly depends on how often an individual coupled chain is involved in a swapping attempt. Each time two chains a and b swap, all processes working on a have to wait for chain b to reach the respective generation and vice versa. Reducing the number of swap attempts (via `numSwapPerGen`) will improve your parallel efficiency, but probably reduces your mixing between coupled chains (e.g., it is less likely that the cold chain benefits from the hotter ones).

So while runtime efficiency probably is the weakest argument for employing chain-level parallelism, memory is a point to consider (also see section 8.3). Likelihood computation is the single dominating factor of memory consumption. The formula for computing memory requirements (in Byte) of a single chain in one run is

$$mem = 4 \cdot 8 \cdot r \cdot p \cdot (n-2), \tag{3}$$

where r is 4 for DNA and 20 for AA data, p is the number of unique site patterns in your alignment and n is the number of taxa.

For executing m coupled chains (efficiently), you require $m+1$ sets of likelihood arrays, thus $mem \cdot (m+1)$ byte. Even if data parallelism is favorable for your dataset, memory requirements may become prohibitive. If you employ more processes, you will also increase the amount of memory that is at your disposal. However, depending on the size of your dataset, parallel efficiency of data parallelism will decrease at some point. This is where chain-level parallelism should be considered.

Using chain-level will allow you to increase the number of processes, while still enough work load is assigned to each process.

As described above you need an additional set of likelihood arrays. Unfortunately, this rule still holds, when chain-level parallelism is employed. Assuming, you run m_p coupled chains in parallel, you will need $mem \cdot (m + m_p)$ byte. For a discussion on how to reduce m_p, please see Section 8.3.

8.2.3 On Data Parallelism

Data parallelism means that the unique site patterns of your alignment are spread out across processes. As discussed at the beginning of this section you should choose the number of processes such that the processes involved in computing the likelihood of a single tree are distributed across as few computing nodes as possible.

Since it takes longer to compute the likelihood of a larger pattern (i.e., your alignment contains more taxa), it is hard to say until which point data parallelism can be employed efficiently. As a rule of thumb each process should at least be responsible for at least 100 sites. If a parallel run of ExaBayes is started, ExaBayes prints the load distribution (i.e., how many pattern are assigned to each process) before starting the computation.

By default, ExaBayes uses a cyclic distribution scheme. This means that if you have partitions with less patterns than processes in a single parallel working unit, then some processes will be idle and you would waste computational resources. Using the `-Q` option distributes entire partitions to processes and solves this problem. If the load distribution is still even with `-Q`, you can expect additional performance improvements.

8.3 Saving Memory

As mentioned earlier, with ExaBayes you can do Bayesian MCMC on alignments of which the size is only limited by the total memory you have available in your computing center.

In addition to that, ExaBayes implements techniques to reduce the overall memory footprint.

The `-M x` option allows you to trade runtime for reduced memory consumption. The higher `x`, the slower but less memory-intensive are the likelihood computations. Remember, that for any `-M x` ExaBayes will yield the exact same results with the limitations described in section 8.5.

This is particularly relevant, if you use chain-level parallelism, since increased parallelism also increases the memory-overhead as explained in section 8.2.2. Recall that for `x=0`, you need the $(m + m_p)$ sets of likelihood arrays (where `m` is the number of coupled chains and m_p the number of coupled chains executed in parallel). This is, because ExaBayes uses an additional set of likelihood arrays to evaluate the likelihood of a new proposal and saves the previous likelihood arrays for the case of rejection of the proposal.

With `-M 1`, you can instruct ExaBayes to not save likelihood arrays for arrays for inner nodes that are (recursively) computed from two leave (resp. tip) nodes. These nodes are particularly fast to compute, so you will not loose too much runtime. For a balanced binary tree (best case), the memory consumption of the saved likelihood arrays (adding the m_p to the equation of memory consumption) is reduced by more than 50%. In the worst case (a comb-/caterpillar-like tree), the memory consumption is merely reduced by 1 array.

When run with `-M 2`, ExaBayes will only save likelihood arrays for the most expensive kind of nodes. These are nodes that have two inner nodes as their descendants. In terms of memory consumption, a balanced binary tree is the worst case (saving only $> 50\%$ of the additional likelihood arrays). In the best case (here the comb-like tree), ExaBayes will not save any addition likelihood arrays.

For `-M 3`, ExaBayes by default does not save any likelihood arrays. The run will be executed substantially slower (but still less than a factor of 2), but specifically if you run a lot of chains in parallel, the factor $(m + m_p)$ in the memory consumption formula is reduced to `m`.

Aside from that, ExaBayes implements a subtree equality vector-technique, that allows you to save memory for dataset that contain many gaps or undetermined characters (see [2]). The amount of memory you save is proportional to the amount of missing data and the runtime penalty should be negligible (resp., there are instances where this actually increases runtime performance).

8.4 Highly Partitioned Runs

In a parallel setting, it is less straight-forward to efficiently execute an analysis, if the alignment is highly partitioned. The issue of load distribution in case of data parallelism is discussed in section 8.2.3. The upshot is that you should, whether all processes of a parallel unit have about the same portion of the overall data. If you achieve good load distribution using the `-Q` option, your run may be executed much more efficiently (if you are interested in the technical reasons for this, consider [8]). Furthermore, for efficiency it is important that the option `proposalSets` is by default set to `true` (default).

8.5 Note on Reproducibility

ExaBayes comes with a strong guarantee of reproducibility.

Ideally, the same seed, configuration file and alignment file have to result in the exact same outcome (e.g., topology/parameter samples) regardless whether `yggdrasil` or `exabayes` were employed. This should hold for any kind of command line parameter governing the specifics of how calculations are to be performed. Furthermore, repeated continuations from a checkpoint file should not influence the output either.

Any change in the configuration file potentially interferes with perfect reproducibility (e.g., increasing the checkpoint frequency).

When parallelism is involved, this guarantee does not hold necessarily. The reason for this is indeterminism in the calculation of the likelihood, when conducted on multiple processors. Compensating for this problem comes at the cost of runtime performance, thus this has not been implement in ExaBayes.

In other words: running ExaBayes with a different number of processes theoretically may yield different results (however, we have not observed this for any MPI implementations yet).

All of the above does not influence the correctness of the results, however it limits the guarantee that the chain is in the exact same state.

9 File Format: Model/Partitioning file

If you want to partition your data, you have to provide a model file either to the `parser` utility or to `yggdrasil/exabayes` (via `-q`). In brief, this format is identical to the raxml model-file format, except that instead of specifying specific protein substitution matrices, you must identify a protein partition with **PROT** instead of a matrix name such as **LG**.

The example file below demonstrates the syntax of this file format:

```
DNA, gene1=1-300
DNA, gene2-codonPos1=301-500\3
DNA, gene2-codonPos2=302-500\3
DNA, gene2-codonPos3=303-500\3
PROT, protId=501-800
DNA, composit=801-1000,1101-1200
DNA, gene3=1000-1100
```

The bottom line is:

- data type identifier: **DNA** or **PROT** (followed by comma)
- partitionName (followed by equal sign)
- alignment positions:
  - range component, see "gene1"

- strided range (useful for codon positions), see "gene2-codonPos1", "gene2-codonPos2" and "gene2-codonPos3". Notice that the starting position of the range is incremented for the second and third codon position.
- combining elements, see "composit". You can combine any element using a comma.

For concatenating a large number of alignments efficiently, we distribute this tool separately from ExaBayes. It automatically creates the appropriate model file, although you will have to manually set the data type for amino acid partitions. Please use with caution.

A side note on efficiency: partitioning your data makes likelihood calculation less efficient. If for instance you partition your data and link all parameters across all partitions, then you could have provided an unpartitioned alignment and the MCMC sampling would require less computational resources.

10 References

References

[1] Alexei J Drummond, Marc a Suchard, Dong Xie, and Andrew Rambaut. Bayesian phylogenetics with BEAUti and the BEAST 1.7. *Molecular biology and evolution*, 29(8):1969–73, August 2012.

[2] Fernando Izquierdo-Carrasco, Stephen a Smith, and Alexandros Stamatakis. Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees. *BMC bioinformatics*, 12(1):470, January 2011.

[3] Clemens Lakner, Paul van der Mark, John P Huelsenbeck, Bret Larget, and Fredrik Ronquist. Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics. *Systematic biology*, 57(1):86–103, February 2008.

[4] Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L Ayres, Aaron Darling, Sebastian Höhna, Bret Larget, Liang Liu, Marc a Suchard, and John P Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic biology*, 61(3):539–42, May 2012.

[5] A. Stamatakis and A.J. Aberer. Novel parallelization schemes for large-scale likelihood-based phylogenetic inference. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1195–1204, 2013.

[6] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, November 2006.

[7] Alexandros Stamatakis, Andre J Aberer, Christian Goll, Stephen A Smith, Simon A Berger, and Fernando Izquierdo-Carrasco. RAxML-Light: a tool for computing terabyte phylogenies. *Bioinformatics (Oxford, England)*, 28(15):2064–6, August 2012.

[8] J Zhang and A Stamatakis. The multi-processor scheduling problem in phylogenetics. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 691–698. IEEE, 2012.
