The RAxML 7.0.4 Manual

Alexandros Stamatakis
The Exelixis Lab[1]
Teaching & Research Unit Bioinformatics
Department of Computer Science
Ludwig-Maximilians-Universität München
stamatakis@bio.ifi.lmu.de

1 About RAxML

RAxML (Randomized Axelerated Maximum Likelihood) is a program for sequential and parallel Maximum Likelihood [1] based inference of large phylogenetic trees. It has originally been derived from `fastDNAml` which in turn was derived from Joe Felsentein's `dnaml` which is part of the PHYLIP [2] package.

1.1 What's new in version 7.0.4?

- Added taxon-name error checking
- Increased allowed taxon-name length to 256 characters
- Amended constraint and backbone tree options `-r` and `-g` to work under rapid bootstrapping option
- Added option to compute pair-wise ML distances between taxa

1.2 RAxML 7.0.4

In addition to the sequential version, RAxML offers two ways to exploit parallelism: *fine-grained parallelism* that can be exploited on shared memory machines or multi-core architectures and *coarse-grained parallelism* that can be exploited on Linux clusters.

The current version of RAxML is a highly optimized program, which handles DNA and AA alignments under various models of substitution and several distinct methods of rate heterogeneity.

In addition, it implements a significantly improved version (run time improvement of factor 2.5) of the fast rapid hill climbing algorithm [3] compared to the algorithm described in [4]. At the same time these new heuristics yield qualitatively comparable results.

In addition to this, it also offers a novel—unpublished—rapid Bootstrapping [5] algorithm that is faster by at least one order of magnitude than all other current implementations (RAxML 2.2.3, GARLI [6], PHYML [7]). Once again, the results obtained by the rapid bootstrapping algorithm are qualitatively comparable to those obtained via the standard RAxML BS algorithm and, more importantly, the deviations in support values between the rapid and the standard RAxML BS algorithm are smaller than those induced by using a different search strategy, e.g., GARLI or PHYML. This rapid BS search can be combined with a rapid ML search on the original alignment and thus allows users to conduct a **full ML analysis** within one single program run.

Some data-structures have been changed and functions re-written. Those technical changes yield an additional run time improvement of around 5%.

The program has been developed to be able to handle extremely large datasets, such as a single–gene 25,000-taxon alignment of protobacteria (length approximately 1,500 base pairs, run time on a single CPU: 13.5 days, memory consumption: 1.5GB) or a large multi-gene alignment of 2,100 mammals with a

[1] Exelixis is the Greek word for evolution

length of over 50,000 base pairs. We also inferred trees for a dataset of 250 taxa and about 500,000 base-pairs, the—to the best of the author's knowledge—largest dataset analyzed under ML to date, on 1,024 processors of a Blue-Gene supercomputer [8]. The BlueGene version is a specialized unreleased RAxML version (available upon request), but the concepts developed in this paper are currently being integrated into the standard RAxML release.

Finally RAxML, despite being developed for handling large datasets, also does fine on smaller to medium–sized datasets (see [9] for a respective performance study on datasets up to 150 taxa).

1.3 RAxML Community Contributions

Several people have contributed to make RAxML easier to use and make it available on more platforms. I would like to express my gratitude to all of them.

My colleague Frank Kauff (now at University of Kaiserslautern, `fkauff@rhrk.uni-kl.de`, previously at Duke University) has written a *cool* biopython wrapper called PYRAXML2. This is a script that reads NEXUS-style data files and prepares the necessary input files and command-line options for RAxML.You can download the Beta-version at `http://www.lutzonilab.net/downloads/`.

My colleague Olaf Bininda-Emonds (`olaf.bininda@uni-jena.de`) has written a perl script that provides a wrapper around RAxML to easily analyze a set of data files according to a common set of search criteria. It also organizes the RAxML output into a set of subdirectories. You can download it at `http://www.personal.uni-jena.de/~b6biol2/ProgramsMain.html`.

James Munro (`munroj01@student.ucr.edu`) at UCR has put up a web-site that provides a guide for installing RAxML on MACs: `http://hymenoptera.ucr.edu/index.php?option=com_content&task=view&id=62&Itemid=8`.

Dave Carmean (`carmean@sfu.ca`) at Simon Fraser University has kindly assembled a RAxML executable for MACs and put up a web-site entitled "Installing and running RAxML on a Mac in less than a minute": `http://www.sfu.ca/biology/staff/dc/raxml/`.

Graham Jones (`http://www.sightsynthesis.co.uk/`) has provided invaluable help by contributing the Windows executable of RAxML.

Finally, Andreas Tille at the Robert Koch-Institute (`tillea@rki.de`) has pushed forward the integration of RAxML and AxParafit (another open-source Bioinformatics code I have developped, see [10]) into the Debian-med package (for details on this project see: `http://www.debian.org/devel/debian-med/`).

1.4 RAxML Web-Servers

Together with Jacques Rougemont (formerly at the Vital-IT Unit of the Swiss Institute of Bioinformatics, now at EPFL, `jacques.rougemont@epfl.ch`) and Paul Hoover at the San Diego Supercomputer Center (`phoover@sdsc.edu`) we have developed two RAxML Web-Servers that offer the novel rapid RAxML Bootstrapping algorithm and thorough ML searches on the original alignments. The one in Switzerland is located at the Vital-IT unit of the SIB: `http://phylobench.vital-it.ch/raxml-bb/` and the one at SDSC runs on the CIPRES project cluster: `http://8ball.sdsc.edu:8889/cipres-web/Bootstrap.do`.

In addition, RAxML is currently being integrated into the Distributed European Infrastructure for Supercomputing Applications system (`http://www.deisa.org/`), but I am not directly involved in this, and only provide some occasional support. The RAxML-DEISA integration is currently supposed to be in the beta-testing phase.

1.5 Citing RAxML

If you use RAxML please **always cite the following paper:** Alexandros Stamatakis : "RAxML-VI-HPC: Maximum Likelihood-based Phylogenetic Analyses with Thousands of Taxa and Mixed Models", *Bioinformatics* 22(21):2688–2690, 2006 [4].

In additon, **when using the Web-Servers or the rapid Bootstrapping algorithm please also cite**: Alexandros Stamatakis, Paul Hoover, and Jacques Rougemont: "A Rapid Bootstrap Algorithm for the RAxML Web-Servers", to be published.

In case you use the **parallel Pthreads-based version please also cite** Michael Ott, Jaroslaw Zola, Srinivas Aluru, Alexandros Stamatakis: "Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L", in *Proceedings of ACM/IEEE Supercomputing conference 2007* [8]. While this paper does not really describe the Pthreads-based version (information on Pthreads: `https://computing.llnl.gov/tutorials/pthreads/`, manuscript in preparation) an analogous parallelization scheme is used which is more efficient than the previous OpenMP-based shared memory implementation described in [11].

Finally, if you used the **CAT approximation of rate heterogeneity (see Section 2.2)** in your analyses, please **also cite** Alexandros Stamatakis: "Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective", in *Proceedings of IPDPS2006* [12].

In case that you use RAxML as a component of larger software packages or Bioinformatics pipelines, I would greatly appreciate if you could add a text box or analogous appropriate information that RAxML should also be cited separately, when used.

If you want RAxML to be further maintained and extended in the future it is in your own interest to properly cite the program!

2 IMPORTANT WARNINGS

2.1 RAxML Likelihood Values

It is **very important** to note that the likelihood values produced by RAxML **can not be directly compared** to likelihood values of other ML programs. However, the likelihood values of the current version are much more similar to those obtained by other programs with respect to previous releases of RAxML (usually between $+/-$ 1.0 log likelihood units of those obtained e.g. by PHYML,IQPNNI [13], or GARLI). Note, that the deviations between PHYML/RAxML and GARLI likelihood values can sometimes be larger because GARLI uses a slightly different procedure to compute empirical base frequencies (Derrick Zwickl, personal communication) while the method in RAxML is exactly the same as implemented in PHYML. These deviations between RAxML/PHYML on the one side and GARLI on the other side appear to be larger on long multigene alignments. Also note, that likelihood values obtained by different RAxML versions, especially those prior to version 2.1.0 should not be directly compared with each other either. The same holds for comparisons of likelihood values between RAxML-VI-HPC v2.2.3 and RAxML 7.0.4! This is due to frequent code and data structure changes in the likelihood function implementation and model parameter optimization procedures!

Thus, if you want to compare topologies obtained by distinct ML programs make sure that you optimize branch lengths and model parameters of final topologies with **one and the same program**. This can be done by either using the respective RAxML option (`-f e`) or, e.g., the corresponding option in PHYML [7].

PERSONAL OPINION: Differences in Likelihood scores:

In theory all ML programs implement the same mathematical function and should thus yield the same likelihood score for a fixed model and a given tree topology. However, if we try to implement a numerical function on a finite machine we will unavoidably obtain rounding errors. Even if we change the sequence (or if it is changed by the compiler) of some operations applied to floating point or double precision arithmetics in our computer we will probably get different results [2]. In my experiments I have observed differences among final likelihood values between GARLI, IQPNNI, PHYML, RAxML (every program showed a different value). You can also experiment with this by removing the `gcc` optimization flag `-O3` in the RAxML `Makfile`. This will yield much slower code, that is in theory mathematically equivalent to the optimized code, but will yield slightly different likelihood scores, due to re-ordered floating point operations.

My personal opinion is that the topological search (number of topologies analyzed) is much more important than exact likelihood scores to obtain "good" final ML trees. Especially on large trees with more than 1,000 sequences the differences in likelihood scores induced by the topology are usually so large, that a very rough parameter optimization with an ϵ of 1 log likelihood unit (i.e., if the difference ϵ between two

[2] As an example for this you might want to implement a dense matrix multiplication on doubles and then re-order the instructions

successive model parameter optimization iterations is $\leq$ 1.0 we stop the optimization) will already clearly show the differences.

Note that, if you perform a bootstrap analysis you don't need to worry too much about likelihood values anyway, since usually you are only interested in the bootstrapped topologies.

2.2 The GTRCAT Mystery

WARNING: It is not a good idea to use the CAT approximiation for datasets with less than 50 taxa, in general there is not enough data per alignment column available to reliably estimate the per-site rate parameters. CAT has been designed to accelerate the computations on large datasets with many taxa! Please read the respective paper [12] to understand how CAT works, what the rate categories are (they are conceptually different from the discrete rate categories of the Γ model), and what the limitations of this method are.

The `GTRCAT` approximation is a computational work–around for the widely used (see [14] for interesting usage statistics) General Time Reversible (GTR [15]) model of nucleotide substitution under the Γ model of rate heterogeneity [16, 17]. CAT is used in an analogous way to accomodate searches with rate heterogeneity in the AA substitution models.

There is a paper available [12] which describes what `GTRCAT` is and why I don't like `GTRGAMMA` despite the fact that Γ is a beautiful Greek letter. The main idea behind `GTRCAT` is to allow for integration of rate heterogeneity into phylogenetic analyses at a significantly lower computational cost (about 4 times faster) and memory consumption (4 times lower). Essentially, `GTRCAT` represents a rather un-mathematical "quick & dirty" approach to rapidly navigate into portions of the tree space, where the trees score well under `GTRGAMMA`. However, due to the way individual rates are optimized and assigned to rate categories in `GTRCAT` (for details on this please read the paper [12]), the **likelihood values computed by `GTRCAT` are completely meaningless.** This means: NEVER COMPARE ALTERNATIVE TREE TOPOLOGIES USING THEIR CAT-based LIKELIHOOD VALUES! You will probably obtain a biased assessment of trees. This is the reason why `GTRCAT` is called approximation instead of model. The same applies to the CAT approximation when used with AA data.

Finally, note that, in the few real-world phylogenetic studies I have worked on so far in collaboration with Biologists, we never received "nasty" reviewer comments for using the CAT approximation. A very recent phylogenetic analysis with RAxML in *Nature* also used the CAT approximation [18].

3 Installation, Compilers, Platforms

RAxML 7.0.4 can be download at `icwww.epfl.ch/~stamatak` as open source code under the GNU General Public Licence (GPL). To install RAxML 7.0.4 download the `RAxML-7.0.4.tar.gz` archive and uncompress it.

This version comes in three flavors:

1. `raxmlHPC` just the standard sequential version, compile it with `gcc` by typing `make -f Makefile.gcc` for LINUX and MAC.

2. `raxmlHPC-PTHREADS` the Pthreads-parallelized version of RAxML which is intended for shared-memory and multi-core architectures. It is compiled with the `gcc` compiler by typing `make -f Makefile.PTHREADS` or `make -f Makefile.PTHREADS.MAC` on MACs.

3. `raxmlHPC-MPI` the MPI-parallelized version for all types of clusters to perform parallel bootstraps, rapid parallel bootstraps, or multiple inferences on the original alignment, compile with the `mpicc` (MPI) compiler by typing `make -f Makefile.MPI`.

Other compilers: It might make sense to use the now much improved Intel-compiler `icc` instead of `gcc` on some systems. The `icc` version 10.0 I have on my laptop produces 20-30% faster code than `gcc`.

IMPORTANT WARNING FOR MPI and PTHREADS VERSIONS: If you want to compile the MPI or PTHREADS version of RAxML but have previously compiled the sequential version, make sure to remove

all object files of the sequential code by typing “rm *.o”, everything needs to be re-compiled for MPI and PTHREADS!

3.1 When to use which Version?

The use of the sequential version is intended for small to medium datasets and for initial experiments to determine appropriate search parameters. However, by using the rapid BS algorithm, you can conduct a full ML analysis with RAxML on single-gene datasets up to 2,000 taxa within 2-3 days on your desktop!

The Pthreads version will work well for very long alignments, but performance is extremely hardware-dependent! It currently appears to scale best on AMD (shared memory nodes as well as the recent multi-core platforms) and the new SUN x4600 systems, while scalability is significantly worse on current Intel architectures. It also scales well on the SGI Altix, which is very large shared-memory supercomputer architecture.

Even for short alignments (1,900 taxa, 1,200bp, DNA data) we observed speedups of around factor 6.5 on an 8-way shared memory Opteron processor on the CIPRES (CyberInfrastructure for Phyligenetic RESearch `http://www.phylo.org`) cluster. For a long alignment (125 taxa, 20,000 base-pairs, DNA) we observed significant super-linear speedups of around 10-11 on the 8-way CIPRES SMP nodes (those are traditional shared-memory nodes, not multi-cores). In general, the Pthreads version is more efficient, i.e., yields higher parallel efficiency and better speedups, than the previous OpenMP-based [11] version. In addition, it is easier to compile, because you do not need an OpenMP compiler any more, just the Pthreads library which is pretty much available by default on all Linux and MAC-based systems. WARNING: Make sure to specify the exact number of CPUs available on your system via the -T option, if you start more threads than you have CPUs available, there will be a significant performance decrease!

The MPI-version is for executing really large production runs (i.e. 100 or 1,000 bootstraps) on a LINUX cluster. You can also perform multiple inferences on larger datasets in parallel to find a best-known ML tree for your dataset. Finally, the novel rapid BS algorithm and the associated ML search have also been parallelized with MPI.

WARNING: REDUCED FUNCTIONALITY OF MPI-VERSION: The current MPI-version only works properly if you specify the “-#” or “-N” option in the command line, since it has been designed to do multiple inferences or rapid/standard BS searches in parallel! For all remaining options, the usage of this type of coarse-grained parallelism does not make much sense!

The best hardware to run RAxML on is currently the AMD Opteron [11] architecture.

3.2 Processor Affinity and Thread Pinning with the Pthreads Version

An important aspect if you want to use the Pthreads version of the program is to find out how your operating system/platform handles processor affinity of threads. Within the shared-memory or multi-core context processor affinity means that if you run e.g. 4 threads on a 4-way CPU or 4 cores each individual thread should always run on the same CPU, i.e. `thread0` on `CPU0`, `thread1` on `CPU1` etc. This is important for efficiency, since cache entries can be continuously re-used if a thread, which works on the same part of the shared memory space, remains on the same CPU. If threads are moved around, e.g.,`thread0` is initially executed on `CPU0` but then on `CPU4` etc. the cache memory of the CPU will have to be re-filled every time a thread is moved. With processor affinity enabled, performance improvements of $\approx$ 5% have been measured on sufficiently large and thus memory-intensive datasets.

On multi-core systems the analysis of memory access patterns and cache congestion is more complicated, we are currently looking at this and will provide additional information and hopefully appropraite solutions soon.

Version 7.0.4 now contains a function that will automatically pin threads to CPUs, i.e., enforce thread affinity, under LINUX/UNIX. This function might occasionally cause some error messages during compilation of RAxML. If this happens please send an email to `stamatakis@bio.ifi.lmu.de` and `ottmi@in.tum.de`.

4 The RAxML Formats, Options & Output Files

4.1 Input Alignment & Input Tree Formats

The input alignment format of RAxML is **relaxed interleaved or sequential PHYLIP**. "Relaxed" means that sequence names can be of variable length between 1 up to 256 characters. If you need longer taxon names you can adapt the constant `#define nmlngth 256` in file `axml.h` appropriately. Moreover, RAxML should be less sensitive with respect to the formatting (tabs, insets, etc) of interleaved PHYLIP files.

The input tree format is Newick (see `http://evolution.genetics.washington.edu/phylip/newicktree.html`), the **RAxML input trees must not be comprehensive**, i.e., need not contain all taxa.

4.2 Alignment Error Checking

I recently noticed that a lot of alignments should be checked for the following errors/insufficiencies before running an analysis with RAxML or any other phylogenetic inference program.

RAxML will now analyze the alignment and check for the following errors:

Identical Sequence name(s) appearing multiple times in an alignment, this can easily happen when you export a standard PHYLIP-file from some tool which truncates the sequence names to 8 or 10 characters.

Identical Sequence(s) that have different names but are exactly identical. This mostly happens when you excluded some hard-to-align alignment regions from your alignment.

Undetermined Column(s) that contain only ambiguous characters that will be treated as missing data, i.e. columns that entirely consist of `X, ?, *, -` for AA data and `N, O, X, ?, -` for DNA data.

Undetermined Sequence(s) that contain only ambiguous characters (see above) that will be treated as missing data.

Prohibited Character(s) in taxon names taxon names that contain any form of whitespace character, like blanks, tabulators, and carriage returns, as well as one of the following prohibited characters: `:,();[]`.

In case that RAxML detects Identical Sequences and/or Undetermined Columns and was executed, e.g., with `-n alignmentName` it will automatically write an alignment file called `alignmentName.reduced` with Identical Sequences and/or Undetermined Columns removed. If this is detected for a multiple model analysis a respective model file `modelFileName.reduced` will also be written. In case RAxML encounters identical sequence names or undetermined sequences or illegal characters in taxon names it will exit with an error and you will have to fix your alignment.

4.3 Program Options

```
raxmlHPC[-MPI|-PTHREADS] -s sequenceFileName
                         -n outputFileName
                         -m substitutionModel
                         [-a weightFileName]
                         [-b bootstrapRandomNumberSeed]
                         [-c numberOfCategories]
                         [-d]
                         [-e likelihoodEpsilon]
                         [-E excludeFileName]
                         [-f a|b|c|d|e|g|h|i|j|m|n|o|p|s|t|w|x]
                         [-g groupingFileName]
                         [-h]
                         [-i initialRearrangementSetting]
```

```
[-j]
[-k]
[-l sequenceSimilarityThreshold]
[-L sequenceSimilarityThreshold]
[-M]
[-o outGroupName1[,outGroupName2[,...]]]
[-p parsimonyRandomSeed]
[-P proteinModel]
[-q multipleModelFileName]
[-r binaryConstraintTree]
[-t userStartingTree]
[-T numberOfThreads]
[-u multiBootstrapSearches]
[-v]
[-w workingDirectory]
[-x rapidBootstrapRandomNumberSeed]
[-y]
[-z multipleTreesFile]
[-#|-N numberOfRuns]
```

Depending on the compiler you used and the platforms that are at your disposal, you will have three alternative executables:

1. `raxmlHPC` is just the sequential version.
2. `raxmlHPC-MPI` is the parallel coarse-grained version. It can be used if you have a LINUX cluster available and want to perform multiple analysis or multiple (rapid) bootstraps, i.e. in combination with the `-#|-N` *or* `-#|-N` *and* `-b`,`-x` or `-f a -x` options. Note, that if you do not specify `-#|-N` the parallel MPI code will not work properly!
3. `raxmlHPC-Pthreads` only makes sense if you have access to a shared–memory or multi-core machine. Note that, `-N` can be used as an alternative to `-#` since the `#` character seems to cause problems with some parallel job submission systems, because it is sometimes used to start comments.

The options in brackets `[]` are optional, i.e., must not be specified, whereas RAxML *must* be provided the sequence file name with `-s` and the output file(s) name appendix with `-n` and the desired model of DNA or AA substitution with `-m`.

Let's have a look at the individual options now:

`-a weightFileName`

This option specifies the name of a column weight file, which allows you to assign individual weights to each column of the alignment. The default is that each column has the weight 1. The weights in the weight file must be integers separated by any type and number of whitespaces within a separate file. In addition, there must of course be as many weights as there are columns in your alignment.

The contents of an example weight file would look like this:

```
5 1 1 2 1 1 1 1 1 1 1 2 1 1 3 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 4 1 1
```

Example: `raxmlHPC -a wgtFile -s alg -m GTRCAT -n TEST.`

`-b bootstrapRandomNumberSeed`

This option allows you to turn on non-parametric bootstrapping [5]. To allow for reproducibility of runs in the sequential program, you have to specify a random number seed, e.g. `-b 123476`. Note however, that parallel bootstraps with the parallel version `raxmlHPC-MPI` are not reproducible despite the fact that you specify a random number seed. They are also not reproducible for the sequential version in case you do not provide a fixed starting tree with `-t` or a parsimony random seed via `-p`.

Example: `raxmlHPC -b 12345 -# 100 -s alg -m GTRCAT -n TEST.`

`-c numberOfCategories`

This option allows you to specify the number of distinct rate categories, into which the individually optimized rates for each individual site are "thrown" under `-m GTRCAT`. The results in [12] indicate that the default of `-c 25` works fine in most practical cases.
Example: `raxmlHPC -c 40 -s alg -m GTRCAT -n TEST`.

`-d`

This option allows you to start the RAxML search with a complete random starting tree instead of the default Maximum Parsimony starting tree. On smaller datasets (around 100–200 taxa) it has been observed that this might sometimes yield topologies of distinct local likelihood maxima which better correspond to empirical expectations.
Example: `raxmlHPC -d -s alg -m GTRGAMMA -n TEST`.

`-e likelihoodEpsilon`

This allows you to specify up to which likelihood difference, i.e., ϵ, the model parameters will be optimized when you use either the `GTRGAMMA` or `GTRMIX` models or when you just evaluate final trees with the `-f e` option. This has shown to be useful to quickly evaluate the likelihood of a bunch of large final trees of more than 1,000 taxa because it will run much faster. I typically use e.g. `-e 1.0` or `-e 2.0` in order to rapidly compare distinct final tree topologies based on their likelihood values. Note that, topology-dependent likelihood-differences are typically far larger than `1.0` or `2.0` log likelihood units. The default setting is `0.1` log likelihood units which proves to be sufficient in most practical cases.
Example: `raxmlHPC -e 0.00001 -s alg -m GTRGAMMA -n TEST`.

`-E`

Used to specify an exclude file name, that contains a specification of alignment positions you wish to exclude from your analysis. The format is similar to Nexus, the file shall contain entries like `100-200 300-400` to exclude, e.g. all columns between positions 100 and 200 as well as all columns between positions 300 and 400. Note that, the bounds, i.e., positions 100, 200, 300, and 400 will also be excluded. To exclude a single column write, e.g., `100-100`. This option will just make RAxML write a reduced alignment file without the excluded columns that can then be used for the real analysis. If you use a mixed model, an appropriately adapted model file will also be written.
Example: `raxmlHPC -E excludeFile -s alg -m GTRCAT -q part -n TEST`.
In this case the files with columns excluded will be named `alg.excludeFile` and `part.excludeFile`.

`-f algorithm`

This option allows you to select the type of algorithm/function you want RAxML to execute.

-f a: tell RAxML to conduct a rapid Bootstrap analysis and search for the best-scoring ML tree in one single program run.
Example: `raxmlHPC -f a -s alg -x 12345 -# 100 -m GTRCAT -n TEST`.

-f b: when this is specified RAxML will draw the bipartitions using a bunch of topologies (typically bootstrapped trees) specified with `-z` (see below) onto a single tree topology specified by `-t` (typically the best-scoring ML tree).
Example: `raxmlHPC -f b -t ref -z trees -m GTRCAT -s alg -n TEST`.

-f c: just checks if RAxML can read the alignment.
Example: `raxmlHPC -f c -t -m GTRCAT -s alg -n TEST`.

-f d: DEFAULT, RAxML will execute the new (as of version 2.2.1) and significantly faster rapid hill-climbing algorithm [3].

-f e: RAxML will optimize the model parameters and branch lengths of a topology provided via the `-t` option under `GTRGAMMA` or the respective AA substitution model under `GAMMA`.
Example: `raxmlHPC -f e -t ref -m GTRGAMMA -s alg -n TEST`

-f g: used to compute the per–site log Likelihoods for one ore more trees passed via `-z`. They will be written to a Treepuzzle-formatted file [19], that can be read by CONSEL [20].
Example: `raxmlHPC -f g -s alg -m GTRGAMMA -z trees -n TEST`.

-f h: RAxML will compute a log likelihood test (SH-test [21]) between a best tree passed via `-t` and a bunch of other trees passed via `-z`.
Example: `raxmlHPC -f h -t ref -z trees -s alg -m GTRGAMMA -n TEST`.

-f i: performs a really thorough standard bootstrap (in combination with `-b` option DOES NOT WORK with `-x`). RAxML will refine the final BS tree under GAMMA and a more exhaustive algorithm.
Example: `raxmlHPC -f i -b 12345 -# 100 -s alg -m GTRCAT -n TEST`.

-f j: generates a bunch of bootstrapped alignment files from an original alignment file.
Example: `raxmlHPC -f j -b 12345 -# 100 -s alg -m GTRCAT -n TEST`.

-f m: RAxML will compare bipartitions between two bunches of trees passed via `-t` and `-z` respectively. The program will return the Pearson correlation between all bipartitions found in the two tree files. A file called `RAxML_bipartitionFrequencies.outpuFileName` will be printed that contains the pair-wise bipartition frequencies of the two sets.
Example: `raxmlHPC -f m -t trees1 -z trees2 -s alg -m GTRCAT -n TEST`.

-f n: computes the log likelihood score of all trees contained in a tree file provided by `-z` under GAMMA or GAMMA+P-Invar.
Example: `raxmlHPC -f n -z trees -s alg -m GTRGAMMA -n TEST`.

-f o: RAxML will execute the slower old search algorithm of version 2.1.3 [4], this is essentially just for backward compatibility.

-f p: performs just pure stepwise MP addition of new sequences to an incomplete starting tree.
Example: `raxmlHPC -f p -t ref -s alg -m GTRCAT -n TEST`.

-f s: option can be used to split a multi-gene alignment into individual genes, provided a model file with `-q`. This might be useful to select best-fitting models for individual partitions of an AA multi-gene alignment or to infer per-partition trees in order to analyze tree compatibility.
Example: `raxmlHPC -f s -q part -s alg -m GTRCAT -n TEST`.

-f t: will perform `-#|-N` randomized tree searches, that always start from one fixed starting tree.
Example: `raxmlHPC -f t -t ref -# 100 -s alg -m GTRCAT -n TEST`.

-f w: will perform an ELW-test [22] on a bunch of input trees passed via `-z`. You will also need to specify a BS seed via `-b` and and the number of replicates you want to compute via `-#|-N`. This test does obvisouly not work under the CAT approximation.
Example: `raxmlHPC -f w -z trees -# 100 -b 12345 -s alg -m GTRGAMMA -n TEST`.

-f x: will compute ML-based pair-wise distances between all sequences in an alignment. RAxML will optimize ML model parameters on a user-defined tree provided via `-t` or simply compute and use a Maximum Parsimony starting tree if no user-defined tree is provided. This option only works for the Γ-based models of rate heterogeneity.
Example: `raxmlHPC -f x -t tree -m GTRGAMMA -n TEST`.

`-g groupingFileName`

This option allows you to specify an incomplete or comprehensive multifurcating constraint tree for the RAxML search in NEWICK format. Initially, multifurcations are resolved randomly. If the tree is incomplete

(does not contain all taxa) the remaining taxa are added by using the MP criterion. Once a comprehensive (containing all taxa) bifurcating tree is computed, it is further optimized under ML respecting the given constraints. Important: If you specify a non-comprehensive constraint, e.g., a constraint tree that does not contain all taxa, RAxML will assume that the remaining taxa, that are not contained in the constraint topology are **unconstrained**, i.e., these taxa can be placed in any part of the tree. As an example consider an alignment with 10 taxa: `Loach, Chicken, Human, Cow, Mouse, Whale, Seal, Carp, Rat, Frog`. If for example you would like `Loach, Chicken, Human, Cow` to be monophyletic you would specify the constraint tree as follows: `((Loach, Chicken, Human, Cow),(Mouse, Whale, Seal, Carp, Rat, Frog));`. Moreover, if you would like `Loach, Chicken, Human, Cow` to be monophyletic and in addition `Human, Cow` to be monophyletic within that clade you could specify: `((Loach, Chicken, (Human, Cow)),(Mouse, Whale, Seal, Carp, Rat, Frog));` If you specify an incomplete constraint: `((Loach, Chicken, Human, Cow),(Mouse, Whale, Seal, Carp));`, the two groups `Loach, Chicken, Human, Cow` and `Mouse, Whale, Seal, Carp` will be monophyletic, while `Rat` and `Frog` can end up anywhere in the tree.

`-h`

If you call `raxmlHPC -h` this will print a summary of the program options to your terminal.

`-i initialRearrangementSetting`

This allows you to specify an initial rearrangement setting for the initial phase of the search algorithm. If you specify e.g. `-i 10` the pruned subtrees will be inserted up to a distance of 10 nodes away from their original pruning point. If you don't specify `-i`, a "good" initial rearrangement setting will automatically be determined by RAxML (see Section 5.2.1 for further details).

`-j`

Specifies that RAxML shall write intermediate trees found during the search to a separate file after each iteration of the search algorithm. The default setting, i.e. if you do not specify `-j` is that no checkpoints will be written.

`-k`

Specifies that RAxML shall optimize branches and model parameters on bootstrapped trees as well as print out the optimized likelihood. Note, that this option only makes sense when used with the `GTRMIX` or `GTRGAMMA` models (or the respective AA models)!

`-l`

Specify a threshold for sequence similarity clustering. RAxML will then print out an alignment to a file called `sequenceFileName.reducedBy.threshold` that only contains representative sequences for the inferred clusters. The specified threshold must be between 0.0 and 1.0. RAxML uses the QT-clustering algorithm [23] to perform this task. In addition, a file called `RAxML_reducedList.outputFileName` will be written that contains clustering information. This option is turned off by default.
Example: `raxmlHPC -s alg -m GTRCAT -l 0.95 -n TEST`.

`-L`

Same functionality as `-l` above, but uses a less exhaustive and thus faster clustering algorithm. This is intended for very large datasets with more than 20,000-30,000 sequences, and also turned off by default.
Example: `raxmlHPC -s alg -m GTRCAT -L 0.95 -n TEST`.

`-m modelOfEvolution`

Selection of the model of nucleotide substitution or amino acid substitution to be used.
NUCLEOTIDE MODELS

-m GTRCAT: GTR **approximation** with optimization of individual per–site substitution rates and classification of those individual rates into the number of rate categories specified by `-c`. This is only a work-around for `GTRGAMMA` so make sure not to compare alternative topologies based on their `GTRCAT` likelihood values. Therefore, you can not use `GTRCAT` in combination with `-f e` (tree evaluation) and not in combination with multiple analyses on the original alignment (`-#|-N`) option. This is due to the fact that the author assumes that you want to compare trees based on likelihoods if you do a multiple run on the original alignment. If you specify e.g. `-m GTRCAT` and `-# 10` the program will automatically use `GTRMIX` (see below).

-m GTRMIX: This option will make RAxML perform a tree inference (search for a good topology) under `GTRCAT`. When the analysis is finished RAxML will switch its model to `GTRGAMMA` and evaluate the final tree topology under `GTRGAMMA` such that it yields stable likelihood values.

-m GTRGAMMA: GTR (General Time Reversible) model of nucleotide substitution [15] with the Γ model of rate heterogeneity [17]. All model parameters are estimated by RAxML. The `GTRGAMMA` implementation uses **4 discrete rate categories** which represents an acceptable trade-off between speed and accuracy. Note that, this has been hard-coded for performance reasons, i.e. the number of discrete rate categories can not be changed by the user.

-m GTRCAT_GAMMA: Inference of the tree with site-specific evolutionary rates. However, here rates are categorized using the 4 discrete GAMMA rates, following a formula proposed by Yang [17]. Evaluation of the final tree topology is done under `GTRGAMMA`. This option is more for experimental purposes than for everyday use.

-m GTRGAMMAI: Same as `GTRGAMMA`, but with estimate of proportion of invariable sites [24], though I still don't like the idea (see discussion in Section 6).

-m GTRMIXI: Same as GTRMIX, but with estimate of proportion of invariable sites.

-m GTRCAT_GAMMAI: Same as `GTRCAT_GAMMA`, but with estimate of proportion of invariable sites.

AMINO ACID MODELS

Available AA models: Values for `matrixName` (see below): `DAYHOFF` [25], `DCMUT` [26], `JTT` [27], `MTREV` [28], `WAG` [29], `RTREV` [30], `CPREV` [31], `VT` [32], `BLOSUM62` [33], `MTMAM` [34]. With the optional `F` appendix you can specify if you want to use empirical base frequencies. Please note, that for mixed models you **must** in addition specify the per-gene AA model in the mixed model file (see `-q` option below).

-m PROTCATmatrixName[F]: AA matrix specified by `matrixName` (see above for a list) with optimization of individual per–site substitution rates and classification of those individual rates into the number of rate categories specified by `-c`. This is only a work-around for the `GAMMA` model of rate heterogeneity, so make sure **not** to compare alternative topologies based on their `PROTCAT`-based likelihood values. Therefore, you can not use `PROTCAT` in combination with `-f e` (tree evaluation) and not in combination with multiple analyses on the original alignment (`-#|-N`) option. This is due to the fact that the author assumes that you want to compare trees based on likelihoods if you do a multiple run on the original alignment. If you specify e.g. one of the `-m PROTCAT...` models and `-# 10` the program will automatically use the respective `PROTMIX...` model (see below).

-m PROTMIXmatrixName[F]: This option will make RAxML perform a tree inference (search for a good topology) under `PROTCAT....` When the analysis is finished RAxML will switch its model to the respective `PROTGAMMA...` model and evaluate the final tree topology under `PROTGAMMA...` such that it yields stable likelihood values.

-m PROTGAMMAmatrixName[F]: AA matrix specified by `matrixName` with the Γ model of rate heterogeneity. All free model parameters are estimated by RAxML. The `GAMMA` implementation uses **4 discrete rate categories** which represents an acceptable trade-off between speed and accuracy. Note that, this has been hard-coded for performance reasons, i.e. the number of discrete rate categories can not be changed by the user.

-m PROTCAT_GAMMAmatrixName[F]: Inference of the tree under specified AA matrix and site-specific evolutionary rates. However, here rates are categorized using the 4 discrete GAMMA rates that are assigned to sites following a formula by Yang. Evaluation of the final tree topology will be conducted under specified AA matrix + GAMMA. This is mostly for experimental purposes.

-m PROTGAMMAImatrixName[F]: Same as `PROTGAMMAmatrixName[F]`, but with estimate of proportion of invariable sites.

-m PROTMIXImatrixName[F]: Same as `PROTMIXmatrixName[F]`, but with estimate of proportion of invariable sites.

-m PROTCAT_GAMMAImatrixName[F]: Same as `PROTCAT_GAMMAmatrixName[F]`, but with estimate of proportion of invariable sites.

`-M`

Switch on estimation of individual per-partition branch lengths. Only has effect when used in combination with `-q` and an alignment partition file. Branch lengths for individual partitions will be printed to separate files. A weighted average of the branch lengths is also computed by using the respective partition lengths (number of columns per partition). Note that, this does not take into account the "gappyness" of partitions, but I am currently not sure how to solve this problem. By default the `-M` option is turned off for partitioned analyses, i.e., RAxML will compute a joined branch length estimate.
Example: `raxmlHPC -s alg -m GTRGAMMA -q part -M -n TEST`.

`-n outputFileName`

Specify the name of this run, according to which the various output files will be named.

`-o outgroupName(s)`

Specify the name/names of the outgroup taxa, e.g.,`-o Mouse` or `-o Mouse,Rat`. Don't leave spaces between the taxon names in the list! If there is more than one outgroup a check for monophyly will be performed. If the outgroups are not monophyletic the tree will be rooted at the first outgroup in the list and a respective warning will be printed.
Example: `raxmlHPC -s alg -m GTRGAMMA -o Rat,Mouse -n TEST`.

`-p`

Specify a random number seed for the parsimony inferences. This allows you and others to reproduce your results (reproducible/verifiable experiments) and will help me debug the program. This option **HAS NO EFFECT in the parallel MPI version**.
Example: `raxmlHPC -s alg -m GTRGAMMA -p 12345 -n TEST`.

`-P proteinModel`

Specify the file name of an external AA substitution model. The file `proteinModel` must contain a total of 420 floating point number entries in plain ASCII text which can be separated by any kind of whitespaces (tabs, spaces, linebreaks, etc.). The first 400 entries are the substitution rates of the 20 by 20 AA matrix (stored and interpreted in row first order, i.e., the first 20 entries correspond to the first row of the matrix) and the last 20 entries (entries 401-420) are the base frequencies. It is important that the base frequencies sum to 1.0+/-ϵ, since even relatively small deviations might cause numerical instability of AA models. The 400 entries of the 20 by 20 matrix must be symmetric, the program will check if this is the case. The entries on the diagonal matrix will be disregarded, since they can be computed from the non-diagonal entries. You still have to specify an AA substitution model via `-m` to tell the program that it has to read and analyze an AA alignment. It will just extract this information from the respective string, however by specifying, e.g., `-m PROTGAMMAWAGF` it will use empirical base frequencies instead of the frequencies specified in file `proteinModel`.
Example: `raxmlHPC -s alg -m PROTGAMMAWAG -p proteinModel -n TEST`

`-q multipleModelFileName`

This allows you to specify the regions of your alignment for which an individual model of nucleotide substitution should be estimated. This will typically be useful to infer trees for long (in terms of base–pairs) multi-gene alignments. If, e.g.,`-m GTRGAMMA` is used, individual α-shape parameters, GTR-rates, and empirical base frequencies will be estimated and optimized for each partition. IMPORTANT CHANGE w.r.t. previous versions: Since RAxML can now handle mixed DNA and AA alignments you MUST specify the type of data in the partition file, before the partition name. For DNA data this just means that you have to add DNA to each line in the partition file, for AA data this is done by specifying the respective AA substitution matrix you want to use for a partition. If you want to do a mixed/partitioned analysis of a concatenated AA and DNA alignment you can either specify -m GTRGAMMA or, e.g., -m PROTGAMMAWAG, the only thing that will be extracted from the string passed via -m is the model of rate heterogeneity you want to use.

If you have a pure DNA alignment with 1,000bp from two genes `gene1` (positions 1–500) and `gene2` (positions 501–1,000) the information in the multiple model file should look as follows:

```
DNA, gene1 = 1-500
DNA, gene2 = 501-1000
```

If `gene1` is scattered through the alignment, e.g. positions 1–200, and 800–1,000 you specify this with:

```
DNA, gene1 = 1-200, 800-1,000
DNA, gene2 = 201-799
```

You can also assign distinct models to the codon positions, i.e. if you want a distinct model to be estimated for each codon position in gene1 you can specify:

```
DNA, gene1codon1 = 1-500\3
DNA, gene1codon2 = 2-500\3
DNA, gene1codon3 = 3-500\3
DNA, gene2 = 501-1000
```

If you only need a distinct model for the 3rd codon position you can write:

```
DNA, gene1codon1andcodon2 = 1-500\3, 2-500\3
DNA, gene1codon3 = 3-500\3
DNA, gene2 = 501-1000
```

As already mentioned, for AA data you must specify the transition matrices for each partition:

```
JTT,  gene1 = 1-500
WAGF, gene2 = 501-800
WAG,  gene3 = 801-1000
```

The AA substitution model must be the first entry in each line and must be separated by a comma from the gene name, just like the `DNA` token above. You can not assign different models of rate heterogeneity to different partitions, i.e., it will be either `CAT`, `GAMMA`, `GAMMAI` etc. for all partitions, as specified with `-m`.

Finally, if you have a concatenated DNA and AA alignment, with DNA data at positions 1–500 and AA data at 501-1,000 with the WAG model the partition file should look as follows:

```
DNA, gene1 = 1-500
WAG, gene2 = 501-1000
```

Example: `raxmlHPC -s alg -m GTRGAMMA -q part -n TEST.`

`-r constraintFileName`

This option allows you to pass a **binary/bifurcating** constraint/backbone tree in NEWICK format to RAxML. Note, that using this option only makes sense if this tree contains less taxa than the input alignment. The remaining taxa will initially be added by using the MP criterion. Once a comprehensive tree with all taxa has been obtained it will be optimized under ML respecting the restrictions of the constraint tree.
Example: `raxmlHPC -s alg -m GTRGAMMA -r constr -n TEST.`

`-s sequenceFileName`

Specify the name of the alignment data file which must be in relaxed PHYLIP format. Relaxed means that you don't have to worry if the sequence file is interleaved or sequential and that the taxon names are too long.

`-t userStartingTree`

Specifies a user starting tree file name which must be in Newick format. Branch lengths of that tree will be ignored. Note, that you can also specify a non-comprehensive (not containing all taxa in the alignment) starting tree now. This might be useful if newly aligned/sequenced taxa have been added to your alignment. Initially, taxa will be added to the tree using the MP criterion. The comprehensive tree will then be optimized under ML.
Example: `raxmlHPC -s alg -m GTRGAMMA -t tree -n TEST.`

`-T`

PTHREADS VERSION ONLY: Specify the number of threads you want to run. Make sure to set -T to at most the number of CPUs you have on your machine, otherwise, there will be a huge performance decrease! This option is set to 0 by default, the Pthreads version will produce an error if you do not set `-T` to at least 2.
Example: `raxmlHPC-PTHREADS -T 4 -s alg -m GTRGAMMA -n TEST.`

`-u`

Specify the number of multiple BS searches per replicate to obtain better ML trees for each replicate. By default only one ML search per BS replicate is conducted. Personal opinion: I believe that rather than analyzing 100 replicates exhaustively via -u one should better invest this time in just analyzing more replicates, i.e., instead of executing raxmlHPC -b 12345 -# 100 -u 10 -s alg -m GTRGAMMA -n TEST, which will do 10 searches for every one of the 100 replicates, it would be better to do 1,000 replicates with raxmlHPC -b 12345 -# 1000 -s alg -m GTRGAMMA -n TEST. This option only works with standard bootstrapping via `-b`, not the fast one via `-x`.

`-v`

Displays version information.

`-w workingDirectory`

Name of the working directory where RAxML shall write its output files to.

`-x`

Specify an integer number (random seed) and turn on rapid bootstrapping. This will invoke the novel rapid bootstrapping algorithm.
Example: `raxmlHPC -x 12345 -# 100 -m GTRCAT -s alg -n TEST.`

`-y`

If you want to only compute a randomized parsimony starting tree with RAxML and not execute an ML analysis of the tree specify `-y`. The program will exit after computation of the starting tree. This option can be useful if you want to assess the impact of randomized MP and Neighbor Joining starting trees on your search algorithm. They can also be used e.g. as starting trees for Derrick Zwickl's GARLI program for ML inferences, which needs comparatively "good" starting trees to work well above approximately 500 taxa.

`-z multipleTreesFile`

Only effective in combination with the `-f b,-f h,-f m,-f n` options. This file should contain a number of trees in NEWICK format. The file should contain one tree per line without blank lines between trees. For example you can directly read in a RAxML bootstrap result file with `-z`.

`-#|-N numberOfRuns`

Specifies the number of alternative runs on distinct starting trees, e.g., if `-# 10` or `-N 10` is specified RAxML will compute 10 distinct ML trees starting from 10 distinct randomized maximum parsimony starting trees. In combination with the `-b` option, this will invoke a multiple bootstrap analysis. In combination with `-x` this will invoke a rapid BS analysis and combined with `-f a -x` a rapid BS search and thereafter a thorough ML search on the original alignment. We introduced `-N` as an alternative to `-#` since the special character `#` seems to sometimes cause problems with certain batch job submission systems. In combination with `-f j` this will generate `numberOfRuns` bootstrapped alignment files.
Example: `raxmlHPC -s alg -n TEST -m GTRGAMMA -# 20`.

4.4 Output Files

Depending on the search parameter settings RAxML will write a number of output files. The files, a run named `-n exampleRun` will write, are listed below:

RAxML_log.exampleRun: A file that prints out the time, likelihood value of the current tree and number of the checkpoint file (if the use of checkpoints has been specified) after each iteration of the search algorithm. In the last line it also contains the final likelihood value of the final tree topology after thorough model optimization, but *only* if `-m GTRMIX` or `-m GTRGAMMA` have been used. This file is **not written** if multiple bootstraps are executed, i.e. `-#` *and* `-b` have been specified. In case of a multiple inference on the original alignment (`-#` option) the Log-Files are numbered accordingly.

RAxML_result.exampleRun: Contains the final tree topology of the current run. This file is also written after each iteration of the search algorithm, such that you can restart your run with `-t` in case your computer crashed. This file is **not written** if multiple bootstraps are executed, i.e. `-#` *and* `-b` have been specified.

RAxML_info.exampleRun: contains information about the model and algorithm used and how RAxML was called. The final `GTRGAMMA` likelihood(s) (only if `-m GTRGAMMA` or `-m GTRMIX` have been used) as well as the alpha shape parameter(s) are printed to this file. In addition, if the rearrangement setting was determined automatically (`-i` has not been used) the rearrangement setting found by the program will be indicated.

RAxML_parsimonyTree.exampleRun: contains the randomized parsimony starting tree if the program has not been provided a starting tree by `-t`. However, this file will **not be written** if a multiple bootstrap is executed using the `-#` *and* `-b` options.

RAxML_randomTree.exampleRun: contains the completely random starting tree if the program was executed with `-d`.

RAxML_checkpoint.exampleRun.checkpointNumber: Printed if you specified by `-j` that checkpoints shall be written. Checkpoints are numbered from 0 to n where n is the number of iterations of the search algorithm. Moreover, the checkpoint files are additionally numbered if a multiple inference on the original alignment has been specified using `-#`. Writing of checkpoint files is disabled when a multiple bootstrap is executed.

RAxML_bootstrap.exampleRun: If a multiple bootstrap is executed by `-#` *and* `-b` or `-x` **all** final bootstrapped trees will be written to this one, single file.

RAxML_bipartitions.exampleRun: If you used the `-f b` option, this file will contain the input tree with confidence values from 0 to 100 drawn on it. It is also printed when `-f a -x` have been specified, at the end of the analysis the program will draw the BS support values on the best tree found during the ML search.

RAxML_reducedList.exampleRun: If you used `-l` or `-L` this file will contain clustering information in the following format:

```
tax1:tax2,tax3,tax4
tax10:tax9,tax11
..
```

where the first entry in each line is the taxon-name of the respective representative sequence of a cluster, while the remaining ones after `:` are the taxa that have been removed via clustering.

RAxML_bipartitionFrequencies.exampleRun: Contains the pair-wise bipartition frequencies of all trees contained in files passed via `-t` and `-z` when the `-f m` option has been used.

RAxML_perSiteLLs.exampleRun: Contains the per–site log likelihood scores in Treepuzzle format for usage with CONSEL [20]. This file is only printed when `-f g` is specified.

RAxML_bestTree.exampleRun: Contains the best-scoring ML tree of a thorough ML analysis in conjunction with a rapid BS analysis, i.e., when options `-x 12345 -f a` are used.

RAxML_distances.exampleRun: Contains the pair-wise ML-based distances between all taxon-pairs in the alignment. This file is only printed when the `-f x` option is used.

5 How to set up and run a typical Analysis

This is a HOW-TO, which describes how RAxML should best be used for a real-world biological analysis, given an example alignment named `ex_al`. Section 5.1 covers the easy (fully automatic) fast way to run it, using the novel rapid BS algorithm, while Section 5.2 describes the hard, more compute–intensive and more thorough way.

5.1 The Easy & Fast Way

The easy and fast way to infer trees with RAxML and to analyze really large datasets (several genes or more than 1,000 taxa) or to conduct a large number of BS replicates is to use the novel rapid BS algorithm and combine it with an ML search. RAxML will then conduct a full ML analysis, i.e., a certain number of BS replicates and a search for a best–scoring ML tree on the original alignment.

To just do a BS search you would type:

```
raxmlHPC -x 12345 -p 12345 -# 100 -m GTRGAMMA -s ex_al -n TEST
```

Note, that the rapid BS algorithm will override the choice of `GTRGAMMA` and **always use the GTR+CAT approximation** for efficiency! Thus, whether you specify `-m GTRGAMMA`,`-m GTRCAT`, `-m GTRGAMMAI` the result will always be the same. Note that, here I added the `-p` option to pass a random number seed for MP starting tree computations, such that the results of the analysis will always be the same. I would like to encourage users to do so as well, because this will allow me to reconstruct potential bugs more easily.

Now, if you want to run a full analysis, i.e., BS and ML search type:

```
raxmlHPC -f a -x 12345 -p 12345 -# 100 -m GTRGAMMA -s ex_al -n TEST
```

This will first conduct a BS search and once that is done a search for the best–scoring ML tree. Such a program run will return the bootstrapped trees (`RAxML_bootstrap.TEST`), the best scoring ML tree (`RAxML_bestTree.TEST`) and the BS support values drawn on the best-scoring tree (`RAxML_bipartitions.TEST`). Here, the model choice via `-m` plays a role for the ML search. If you specify, e.g., `-m GTRCAT` the ML search will still be conducted under `GTRGAMMA`. So, in general the only thing that matters, is whether you want to use `GTRGAMMA` or `GTRGAMMAI` to include an estimate of the proportion of invariable sites.

Finally, note that, by increasing the number of BS replicates via `-#` you will also make the ML search more thorough, since for ML optimization every 5th BS tree is used as a starting point to search for ML trees. From what I have observed so far, this new ML search algorithm yielded better trees than what is obtained via 20 standard ML searches on distinct starting trees for all datasets with $\leq$ 1,000 sequences. For larger datasets it might be worthwhile to conduct an additional ML search as described in Section 5.2.3, just to be sure.

WARNING note that the rapid BS search will currently ignore commands associated to user tree files passed via -t,-z. However, the constraint and backbone tree options (-g and -r) do work with rapid BS now.

5.2 The Hard & Slow Way

Despite the observation that the default parameters and the rapid BS and ML algorithm described above work well in most practical cases, a good thing to do is to adapt the program parameters to your alignment. This refers to a "good" setting for the rate categories of `-m GTRCAT` and the initial rearrangement setting. If you use mixed models you should add `-q modelFileName` to all of the following commands.

5.2.1 Getting the Initial Rearrangement Setting right

If you don't specify an initial rearrangement setting with the `-i` option the program will automatically determine a good setting based upon the randomized MP starting tree. It will take the starting tree and apply lazy subtree rearrangements with a rearrangement setting of 5, 10, 15, 20, 25. The minimum setting that yields the best likelihood improvement on the starting trees will be used as initial rearrangement setting. This procedure can have two disadvantages: *Firstly*, the initial setting might be very high (e.g. 20 or 25) and the program will slow down considerably. *Secondly*, a rearrangement setting that yields a high improvement of likelihood scores on the starting tree might let the program get stuck earlier in some local maximum (this behavior could already be observed on a real dataset with about 1,900 taxa).

Therefore, you should run RAxML a couple of times (the more the better) with the automatic determination of the rearrangement setting and with a pre-defined value of 10 which proved to be sufficiently large and efficient in many practical cases. In the example below we will do this based on 5 fixed starting trees.

So let's first generate a couple of randomized MP starting trees. Note that in RAxML-VI-HPC 2.2.3 you also **always** have to **specify a substitution model**, regardless of whether you only want to compute an MP starting tree with the `-y` option.

```
raxmlHPC -y -s ex_al -m GTRCAT -n ST0
...
raxmlHPC -y -s ex_al -m GTRCAT -n ST4
```

Then, infer the ML trees for those starting trees using a fixed setting `-i 10` ...

```
raxmlHPC -f d -i 10 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST0 -n FI0
...
raxmlHPC -f d -i 10 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST4 -n FI4
```

and then using the automatically determined setting on the *same starting trees*:

```
raxmlHPC -f d -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST0 -n AI0
...
raxmlHPC -f d -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST4 -n AI4
```

Here, we use the GTRMIX model, i.e. inference under GTRCAT and evaluation of the final tree under GTRGAMMA such that we can compare the final likelihoods for the fixed setting FI0-FI4 and the automatically determined setting AI0-AI4.

The setting that yields the best likelihood scores should be used in the further analyses.

5.2.2 Getting the Number of Categories right

Another issue is to get the number of rate categories right. Due to the reduced memory footprint and significantly reduced inference times the recommended model to use with RAxML on large dataset is GTRMIX if you are doing runs to find the best-known ML tree on the original alignment and GTRCAT for bootstrapping.

Thus, you should experiment with a couple of -c settings and then look which gives you the best Γ likelihood value.

Suppose that in the previous Section 5.2.1 you found that automatically determining the rearrangement setting works best for your alignment.

You should then re-run the analyses with distinct -c settings by increments of e.g. 15 rate categories e.g.:

```
raxmlHPC -f d -c 10 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST0 -n C10_0
...
raxmlHPC -f d -c 10 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST4 -n C10_4
```

You don't need to run it with the default setting of -c 25 since you already have that data, such that you can continue with ...

```
raxmlHPC -f d -c 40 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST0 -n C40_0
...
raxmlHPC -f d -c 40 -m GTRMIX -s ex_al -t RAxML_parsimonyTree.ST4 -n C40_4
```

and so on and so forth.

Since the GTRCAT approximation is still a new concept little is known about the appropriate setting for -c 25. However, empirically -c 25 worked best on 19 real-world alignments. So testing up to -c 55 should usually be sufficient, except if you notice a tendency for final GTRGAMMA likelihood values to further improve with increasing rate category number.

Thus, the assessment of the "good" -c setting should once again be based on the final GTRGAMMA likelihood values.

If you don't have the time or computational power to determine both "good" -c and -i settings you should rather stick to determining -i since it has shown to have a greater impact on the final results.

Also note, that increasing the number of distinct rate categories has a negative impact on execution times.

Finally, if the runs with the automatic determination of the rearrangement settings from Section 5.2.1 have yielded the best results you should then use exactly the *same* rearrangement settings for each series of experiments to determine a good -c setting. The automatically determined rearrangement settings can be retrieved from file RAxML_info.AI_0 ... RAxML_info.AI_4.

5.2.3 Finding the Best-Known Likelihood tree (BKL)

As already mentioned RAxML uses randomized MP starting trees in which it initiates an ML-based optimization. Those trees are obtained by using a randomized stepwise addition sequence to insert one taxon after the other into the tree. When all sequences have been inserted a couple of subtree rearrangements (also called subtree pruning re-grafting) with a *fixed* rearrangement distance of 20 are executed to further improve the MP score.

The concept to use randomized MP starting trees in contrast to the NJ (Neighbor Joining) starting trees many other ML programs use is regarded as an advantage of RAxML. This allows the program to start ML optimizations of the topology from a distinct starting point in the immense topological search space each time. Therefore, RAxML is more likely to find good ML trees if executed several times.

This also allows you to build a consensus tree out of the final tree topologies obtained from each individual run on the original alignment. By this and by comparing the final likelihoods you can get a feeling on how stable (prone to get caught in local maxima) the search algorithm is on the original alignment.

Thus, if you have sufficient computing resources available, in addition to bootstrapping, you should do multiple inferences (I executed 200 inferences in some recent real-world analyses with Biologists) with RAxML on the original alignment. On smaller datasets it will also be worthwhile to use the `-d` option for a couple of runs to see how the program behaves on completely random starting trees.

This is where the `-#` option as well as the parallel MPI version `raxmlHPC-MPI` come into play.

So, to execute a multiple inference on the original alignment on a single processor just specify:

```
raxmlHPC -f d -m GTRMIX -s ex_al -# 10 -n MultipleOriginal
```

and RAxML will do the rest for you. Note that specifying `-m GTRCAT` in combination with `-#` is not a good idea, because you will probably want to compare the trees inferred under `GTRCAT` based on their likelihood values and will have to compute the likelihood of the final trees under `GTRGAMMA` anyway. Thus you should better use `-m GTRMIX` for those analyses.

If you have a PC cluster available you would specify,

```
raxmlHPC-MPI -f d -m GTRMIX -s ex_al -# 100 -n MultipleOriginal
```

preceded by the respective MPI run-time commands, e.g. `mpiexec` or `mpirun` depending on your local installation (please check with your local computer scientist).

It is **important** to note that you should specify the execution of one more process than CPUs available (e.g. you have 8 CPUs → start 9 MPI processes), since one of those is just the master process which collects data and issues jobs to the worker processes and does not produce significant computational load.

5.2.4 Bootstrapping with RAxML

To carry out a multiple non-parametric bootstrap with the sequential version of RAxML just type:

```
raxmlHPC -f d -m GTRCAT -s ex_al -# 100 -b 12345 -n MultipleBootstrap
```

You have to specify a random number seed after `-b` for the random number generator. This will allow you to generate reproducible results. Note that we can use `GTRCAT` here, if we do not want to compare final trees based on ML scores or need bootstrapped trees with branch lengths.

To do a parallel bootstrap type:

```
raxmlHPC-MPI -f d -m GTRCAT -s ex_al -# 100 -b 12345 -n MultipleBootstrap
```

once again preceded by the appropriate MPI execution command. Note that despite the fact that you specified a random number seed the results of a parallel bootstrap are not reproducible.

5.2.5 Obtaining Confidence Values

Suppose that you have executed 200 inferences on the original alignment and 1,000 bootstrap runs. You can now use the RAxML `-f b` option to draw the information from the 1,000 bootstrapped topologies onto some tree and obtain a topology with support values. From my point of view the most reasonable thing to do is to draw them on the best-scoring ML tree from those 200 runs. Suppose, that the best-scoring tree was found in run number 99 and the respective tree-file is called `RAxML_result.MultipleOriginal.RUN.99`.

If you have executed more than one bootstrap runs with the sequential version of RAxML on distinct computers, i.e. 10 runs with 100 bootstraps on 10 machines you will first have to concatenate the bootstrap files. If your bootstrap result files are called e.g. `RAxML_bootstrap.MultipleBootstrap.0, ..., RAxML_bootstrap.MultipleBootstrap.9` you can easily concatenate them by using the LINUX/UNIX `cat` command, e.g.

```
cat RAxML_bootstrap.MultipleBootstrap.* > RAxML_bootstrap.All
```

In order to get a tree with bootstrap values on it just execute RAxML as indicated below:

```
raxmlHPC -f b -m GTRCAT -s ex_al -z RAxML_bootstrap.All
         -t RAxML_result.MultipleOriginal.RUN.99 -n BS_TREE
```

The new output tree format now shows the support values as inner node labels and also displays branch lengths, it can look e.g. like this:

```
((((Human:0.555,((Frog:0.207,(Carp:0.129,Loach:0.192)
100:0.159)70:0.001,Chicken:0.561)100:0.259)65:0.091,
(Whale:0.108,(Cow:0.116,Seal:0.186)55:0.030)65:0.046)
95:0.144,Rat:0.068):0.045,Mouse:0.045);
```

6 Frequently Asked Questions

Q: When performing a bootstrap search using a partitioned model, does RAxML perform a conserved-bootstrap resampling, i.e., does it resample within genes so that partitions are sustained?

That is the case. When performing Bootstraps on partitioned data sets, bootstrapped alignments will be sampled from within partitions, i.e., bootstrapped partitions are sustained and contain exactly the same number of alignment columns as the original partition.

Q: Can I use NEXUS-style input files for analyses with RAxML?

Not directly, but my colleague Frank Kauff (`fkauff@rhrk.uni-kl.de`) at the University of Kaiserslautern has written a cool biopython wrapper called PYRAXML2. This is a script that reads nexus data files and prepares the necessary input files and command-line options for RAxML. You can download the Beta-version of PYRAXML2 at `http://www.lutzonilab.net/downloads/`.

Q: Why don't you like the proportion of Invariable (P-Invar) Sites estimate, despite the fact that you implemented it?

I only implemented P-Invar in RAxML to make some users happy, but I still strongly disagree with its usage.

PERSONAL OPINION: It is unquestionable that one needs to incorporate rate heterogeneity in order to obtain "publishable" results. Put aside the "publish-or-perish" argument, there is also strong biological evidence for rate heterogeneity among sites. The rationale for being sceptical about P-Invar in RAxML is that all three alternatives, `GTRGAMMA`, `GTRCAT`, and P-Invar represent distinct approaches to incorporate rate heterogeneity. Thus, in principle they account for the same phenomenon by different mathematical means. Also some unpublished concerns have been raised that the usage of P-Invar in combination with Γ can lead to a "ping-pong" effect since a change of P-Invar leads to a change in Γ and vice versa. This essentially means that those two parameters, i.e., α and P-Invar can not be optimized independently from each other, and might cause significant trouble and problems during the model parameter (everything except tree topology) optimization process. In fact, I already observed this when I was implementing P-Invar in RAxML on a very small AA dataset.

Although this has never been properly documented, several well-known researchers in phylogenetics share this opinion (Arndt v. Haeseler, Ziheng Yang; quote from an recent email in 2008 regarding this part of the RAxML manual: *I entirely agree with your criticism of the Pinv+Gamma model, even though as you said, it is very commonly used.*, Korbinian Strimmer, personal communications). The following paper [35] touches this problem of dependency between α and P-Invar.

Ziheng Yang kindly provided some additional references that refer to this problem [36, 37, 38, 39, 24].

He also addresses the issue in his recently published book on *Computational Molecular Evolution (Oxford University Press, 2006)*; quote from pages 113–114: *The model is known as I+G and has been widely used. This model is somewhat pathological as the gamma distribution with alpha ¡ 1 already allows for sites with very low rates; as a result, adding a proportion of invariable sites creates a strong correlation between p0 and alpha, making it impossible to estimate both parameters reliably [38, 39, 24]. Another drawback of the model is that the estimate of p0 is very sensitive to the number and divergences of the sequences included in the data. The proportion p0 is never larger than the observed proportion of constant sites; with*

the addition of more and divergent sequences, the proportion of constant sites drops, and the estimate of p0 tends to go down as well.

In any case, I have so far not encountered any difficulties with reviews for the few real phylogenetic analyses [40, 41] I have published with collegues from Biology, when we used GTR+Γ instead of the more widely spread GTR+Γ+I.

Q: Why does RAxML only implement GTR-based models of nucleotide substitution?

For each distinct model of nucleotide substitution RAxML uses a separate, highly optimized set of likelihood functions. The idea behind this is that GTR is the most common and general model for real-world DNA analysis. Thus, it is better to efficiently implement and optimize this model instead of offering a plethora of distinct models which are only special cases of GTR but are programmed in a generic and thus inefficient way.

PERSONAL OPINION: My personal view is that using a simpler model than GTR only makes sense with respect to the computational cost, i.e. it is less expensive to compute. Programs such as Modeltest [42] propose the usage of a simpler model for a specific alignment if the likelihood of a fixed topology under that simpler model is not significantly worse than that obtained by GTR based on a likelihood ratio test. My experience is that GTR always yields a slightly better likelihood than alternative simpler models. In addition, since RAxML has been designed for the inference of large datasets the danger of over-parameterizing such an analysis is comparatively low. Provided these arguments the design decision was taken to rather implement the most general model efficiently than to provide many inefficient generic implementations of models that are just special cases of GTR. Finally, the design philosophy of RAxML is based upon the observation that a more thorough topological search has a greater impact on final tree quality than modeling details. Thus, the efficient implementation of a rapid search mechanisms is considered to be more important than model details. Note that, Derrick Zwickl has independently adapted the same strategy in his very good GARLI code (`http://www.zo.utexas.edu/faculty/antisense/Garli.html`), based on similar considerations (personal communication).

Q: How does RAxML perform compared to other programs?

RAxML has been compared to other phylogeny programs mainly based on real-world biological datasets and best-known likelihood values. Those analyses can be found in [4, 43, 44, 45]. On almost all real datasets RAxML outperforms other current programs with respect to inference times as well as final likelihood values. An exception is Derrick Zwickl's GARLI code which represents a "good" alternative to RAxML for trees containing less than approximately 1,000–1,500 taxa. The main advantages of RAxML with respect to all other programs are the highly optimized and efficient likelihood functions and the very low memory consumption. In particular the implementation of the `GTRCAT` feature allows RAxML to compute huge trees under a realistic approximation of nucleotide substitution which is currently impossible with competing programs due to excessive memory requirements. An initial analysis of the large multi-gene mammalian dataset under `GTRCAT` showed promising results.

Q: Why has the performance of RAxML mainly been assessed using real-world data?

PERSONAL OPINION: Despite the unquestionable need for simulated data and trees to verify and test the performance of current ML algorithms the current methods available for generation of simulated alignments are not very realistic. For example, only few methods exist that incorporate the generation of gaps in simulated alignments. Since the model according to which the sequences are generated on the true tree is pre-defined we are actually assuming that ML exactly models the true evolutionary process, while in reality we simply don't know how sequences evolved. The above simplifications lead to "perfect" alignment data without gaps, that evolved exactly according to a pre-defined model and thus exhibits a very strong phylogenetic signal in contrast to real data. In addition, the given true tree, must not necessarily be *the* Maximum Likelihood tree. This difference manifests itself in substantially different behaviors of search algorithms on real and simulated data. Typically, search algorithms execute significantly less (factor 5–10) topological moves on simulated data until convergence as opposed to real data, i.e. the number of successful Nearest Neighbor Interchanges (NNIs) or subtree rearrangements is lower. Moreover, in several cases the likelihood of trees found by RAxML on simulated data was better than that of the true tree. Another important observation is that program performance can be inverted by simulated data. Thus, a program that

yields “good” Robinson–Foulds distances [46, 47] on simulated data can in fact perform much worse on real data than a program that does not perform well on simulated data. If one is willing to really accept ML as inference criterion on real data one must also be willing to assume that the tree with the best likelihood score is the tree that is closest to the true tree.

My personal conclusion is that there is a strong need to improve simulated data generation and methodology. In addition, the perhaps best way to assess the validity of our tree inference methods consists in an empirical evaluation of new results and insights obtained by real phylogenetic analysis. This should be based on the prior knowledge of Biologists about the data and the medical and scientific benefits attained by the computation of phylogenies.

Q: Why am I getting weird error messages from the MPI version?

You probably forgot to specify the `-#` or `-N` option in the command-line which **must** be used for the MPI version to work properly.

Q: When using mixed models, can I link the model parameters of distinct partitions to be estimated jointly, in a similar as way MrBayes does it?

Currently not, but the implementation of such an option is planned.

7 Things in Preparation

A couple of things are in preparation (to be hopefully released within the next 6 months) which will further expand the capabilities of RAxML. Please be patient with feature requests, since I do not have anybody to help me with program development.

- Built-in bootstopping/convergence criterion
- Linking parameter estimation across mixed models
- ML-based estimate of base frequencies (I have been promising that for a long time now, I know)
- ML-model for morphological/binary data
- ML-based rapid sequence addition option
- More efficient ML function implementation for very gappy multi-gene alignments.

For any further requests or suggestions please send an email to `stamatakis@bio.ifi.lmu.de` or contact me via skype internet telephony, login: stamatak.

Acknowledgments

Many people have contributed to improve RAxML either via personal discussions, email, or skype or by providing real-world alignments and answering all sorts of CS- and biology-related questions. In the hope not to have forgotten anybody I would like to thank the following colleagues (names are in no particular order): Ziheng Yang, Olivier Gascuel, Stephane Guindon, Wim Hordijk, Michael Ott, Olaf Bininda-Emonds, Maria Charalambous, Pedro Trancoso, Tobias Klug, Derrick Zwickl, Jarno Tuimila, Charles Robertson, Daniele Catanzaro, Daniel Dalevi, Mark Miller, Usman Roshan, Zhihua Du, Markus Göker, Bret Larget, Josh Wilcox, Marty J. Wolf, Aggelos Bilas, Alkiviadis Simeonidis, Martin Reczko, Gangolf Jobb, Frank Kauff, James Munro, Peter Cordes, Tandy Warnow, Bernard Moret, Paul Hoover, Jacques Rougemont, Joe Felsenstein, Daniel Lundin.

References

[1] Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution **17** (1981) 368–376

[2] Felsenstein, J.: Phylip (phylogeny inference package) version 3.6 (2004) Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

[3] Stamatakis, A., Blagojevic, F., Nikolopoulos, D., Antonopoulos, C.: Exploring New Search Algorithms and Hardware for Phylogenetics: RAxML Meets the IBM Cell. The Journal of VLSI Signal Processing **48** (2007) 271–286

[4] Stamatakis, A.: RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics **22** (2006) btl446

[5] Felsenstein, J.: Confidence Limits on Phylogenies: An Approach Using the Bootstrap. Evolution **39** (1985) 783–791

[6] Zwickl, D.: Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion. PhD thesis, University of Texas at Austin (2006)

[7] Guindon, S., Gascuel, O.: A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst. Biol. **52** (2003) 696–704

[8] Ott, M., Zola, J., Aluru, S., Stamatakis, A.: Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L. In: ACM/IEEE Supercomputing conference 2007. (2007)

[9] Morrison, D.A.: Increasing the Efficiency of Searches for the Maximum Likelihood Tree in a Phylogenetic Analysis of up to 150 Nucleotide Sequences. Systematic Biology **56** (2007) 988–1010

[10] Stamatakis, A., Auch, A., Meier-Kolthoff, J., Goeker, M.: AxPcoords & parallel AxParafit: statistical co-phylogenetic analyses on thousands of taxa. BMC Bioinformatics **8** (2007) 405

[11] Stamatakis, A., Ott, M., Ludwig, T.: Raxml-omp: An efficient program for phylogenetic inference on smps. In: Proc. of PaCT05. (2005) 288–302

[12] Stamatakis, A.: Phylogenetic models of rate heterogeneity: A high performance computing perspective. In: Proc. of IPDPS2006, Rhodos, Greece (2006)

[13] Minh, B., Vinh, L., Haeseler, A., Schmidt, H.: piqpnni - parallel reconstruction of large maximum likelihood phylogenies. Bioinformatics (2005)

[14] Ripplinger, J., Sullivan, J.: Does Choice in Model Selection Affect Maximum Likelihood Analysis? Systematic Biology **57** (2008) 76–85

[15] Tavar, S.: Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. Some Mathematical Questions in Biology: DNA Sequence Analysis **17** (1986)

[16] Yang, Z.: Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites. J. Mol. Evol. **39** (1994) 306–314

[17] Yang, Z.: Among-site rate variation and its impact on phylogenetic analyses. Trends Ecol. Evol. **11** (1996) 367–372

[18] Dunn, C.W., Hejnol, A., Matus, D.Q., Pang, K., Browne, W.E., Smith, S.A., Seaver, E., Rouse, G.W., Obst, M., Edgecombe, G.D., Sorensen, M.V., Haddock, S.H.D., Schmidt-Rhaesa, A., Okusu, A., Kristensen, R.M., Wheeler, W.C., Martindale, M.Q., Giribet, G.: Broad phylogenomic sampling improves resolution of the animal tree of life. Nature (2008) advance on-line publication.

[19] Schmidt, H., Strimmer, K., Vingron, M., Haeseler, A.: Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing. Bioinformatics **18** (2002) 502–504

[20] Shimodaira, H., Hasegawa, M.: CONSEL: for assessing the confidence of phylogenetic tree selection (2001)

[21] SHIMODAIRA, H., HASEGAWA, M.: MULTIPLE COMPARISONS OF LOG-LIKELIHOODS WITH APPLICATIONS TO PHYLOGENETIC INFERENC. Molecular biology and evolution **16** (1999) 1114–1116

[22] Strimmer, K., Rambaut, A.: Inferring confidence sets of possibly misspecified gene trees Proc. R. Soc. Lond. B **269** (2002) 137–142

[23] Heyer, L., Kruglyak, S., Yooseph, S.: Exploring Expression Data: Identification and Analysis of Coexpressed Genes. Genome Research **9** (1999) 1106–1115

[24] Yang, Z.: Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites (1993)

[25] Dayhoff, M., Schwartz, R., Orcutt, B.: A model of evolutionary change in proteins. Atlas of Protein Sequence and Structure **5** (1978) 345–352

[26] Kosiol, C., Goldman, N.: Different Versions of the Dayhoff Rate Matrix. Molecular Biology and Evolution **22** (2005) 193–199

[27] Jones, D., Taylort, W., Thornton, J.: A new approach to protein fold recognition. Nature **358** (1992) 86–89

[28] Adachi, J.: Model of Amino Acid Substitution in Proteins Encoded by Mitochondrial DNA. Journal of Molecular Evolution **42** (1996) 459–468

[29] Whelan, S., Goldman, N.: A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach. Molecular Biology and Evolution **18** (2001) 691–699

[30] Dimmic, M., Rest, J., Mindell, D., Goldstein, R.: rtREV: An Amino Acid Substitution Matrix for Inference of Retrovirus and Reverse Transcriptase Phylogeny. Journal of Molecular Evolution **55** (2002) 65–73

[31] Adachi, J., Waddell, P., Martin, W., Hasegawa, M.: Plastid Genome Phylogeny and a Model of Amino Acid Substitution for Proteins Encoded by Chloroplast DNA. Journal of Molecular Evolution **50** (2000) 348–358

[32] Mueller, T., Vingron, M.: Modeling Amino Acid Replacement. Journal of Computational Biology **7** (2000) 761–776

[33] Henikoff, S., Henikoff, J.: Amino Acid Substitution Matrices from Protein Blocks. Proceedings of the National Academy of Sciences of the United States of America **89** (1992) 10915–10919

[34] Yang, Z.: Synonymous and Nonsynonymous Rate Variation in Nuclear Genes of Mammals. Journal of Molecular Evolution **46** (1998) 409–418

[35] Gu, X.: Maximum likelihood estimation of the heterogeneity of substitution rate among nucleotide sites. Molecular Biology and Evolution **12** (1995) 546–557

[36] Ren, F., Tanaka, H., Yang, Z.: An Empirical Examination of the Utility of Codon-Substitution Models in Phylogeny Reconstruction. Systematic Biology **54** (2005) 808–818

[37] Minin, V., Abdo, Z., Joyce, P., Sullivan, J.: Performance-Based Selection of Likelihood Models for Phylogeny Estimation. Systematic Biology **52** (2003) 674–683

[38] Mayrose, I., Friedman, N., Pupko, T.: A Gamma mixture model better accounts for among site rate heterogeneity. Bioinformatics **21** (2005)

[39] Sullivan, J., Swofford, D., Naylor, G.: The Effect of Taxon Sampling on Estimating Rate Heterogeneity Parameters of Maximum-Likelihood Models. Molecular Biology and Evolution **16** (1999) 1347–1356

[40] Grimm, G.W., Renner, S.S., Stamatakis, A., Hemleben, V.: A nuclear ribosomal DNA phylogeny of acer inferred with maximum likelihood, splits graphs, and motif analyses of 606 sequences. Evolutionary Bioinformatics Online **2** (2006) 279–294

[41] Gottschling, M., Stamatakis, A., Nindl, I., Stockfleth, E., Alonso, A., Gissmann, L., Bravo, I.G.: Multiple evolutionary mechanisms drive papillomavirus diversification. Molecular Biology and Evolution **24** (2007) 1242–1258

[42] Posada, D., Crandall, K.: Modeltest: testing the model of dna substitution. Bioinformatics **14** (1998) 817–818

[43] Stamatakis, A.: An efficient program for phylogenetic inference using simulated annealing. In: Proc. of IPDPS2005, Denver, Colorado, USA (2005)

[44] Stamatakis, A., Ludwig, T., Meier, H.: New fast and accurate heuristics for inference of large phylogenetic trees. In: Proc. of IPDPS2004. (2004)

[45] Stamatakis, A., Ludwig, T., Meier, H.: Raxml-iii: A fast program for maximum likelihood-based inference of large phylogenetic trees. Bioinformatics **21** (2005) 456–463

[46] Robinson, D.F., Foulds, L.R.: Comparison of weighted labelled trees. Lecture Notes in Mathematics **748** (1979) 119–126

[47] Robinson, D.F., Foulds, L.R.: Comparison of Phylogenetic Trees. Mathematical Biosciences **53** (1981) 131–147