The RAxML-VI-HPC Version 2.0 Manual

Alexandros Stamatakis

Institute of Computer Science, Foundation for Research and Technology–Hellas
stamatak@ics.forth.gr

1 About RAxML

RAxML (Randomized Axelerated Maximum Likelihood) is a program for Maximum Likelihood-based inference of large phylogenetic trees. It has originally been derived from fastDNAml.

1.1 A Brief Version History

This Section provides a brief version history of RAxML, mainly to avoid potential confusion.

- RAxML-II: Initial Implementation of the hill climbing algorithm with the lazy subtree rearrangement technique. Implementation of a parallel MPI-based and distributed version of the program.
- RAxML-III: Implementation of additional models of nucleotide substitution and ML-based optimization of model parameters.
- RAxML-IV: Never existed, don't ask me why.
- RAxML-V: Introduction of the simulated annealing search algorithm, implementation of protein models, OpenMP-parallelization.
- RAxML-VI: Introduction of the novel rapid hill climbing algorithm and automatic determination of a "good" rearrangement setting
- RAxML-VI-HPC: Improvement of the OpenMP parallelization, technical tuning of the GTR-based likelihood models, re-implementation of the MP (Maximum Parsimony) starting tree computations, MPI-based parallel version for multiple bootstrapping.
- RAxML-VI-HPC-2.0: Implementation of various forms of bifurcating and multifurcating constraint trees, implementation of multiple model estimates for multi-gene alignments.

1.2 RAxML-VI-HPC

HPC stands for High Performance Computing and this specific version of RAxML is a dedicated and highly optimized version, which handles **only DNA alignments** under the **GTR model** of nucleotide substitution including rate heterogeneity.

In addition, it only implements the novel, fast, and currently unpublished rapid hill climbing algorithm, which yields significant performance improvements on huge alignments compared to the previous search algorithms. A run-time improvement of factor 67 has been measured for a 25,000-taxon dataset.

The program has explicitly been developed to handle extremely large datasets, such as a 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 length of over 50,000 base pairs (run time: 1 week with the OpenMP version of RAxML on 4 CPUs, memory consumption: 2.9GB).

List of differences with respect to the standard RAxML-VI version:

- Highly optimized likelihood functions, 15%–30% faster than in the standard version.
- Re-implementation of the parsimony starting tree calculations from scratch (RAxML does not use PHYLIP components for this any more) which are also significantly faster (up to factor 10 on 25,000 taxa) than the originally used dnapars implementation.
- An MPI-based (Message Passing Interface) implementation to perform multiple bootstraps or multiple analyses on the original alignment in parallel on a cluster. Note that, this is not a parallel implementation of the search algorithm, which is nonetheless under preparation.
- Re-Implementation of the branch-length and model optimization function for the `-m GTRGAMMA` model which is now numerically more stable.
- Implementation of the `-m GTRMIX` model which allows you to infer trees under the faster `-m GTRCAT` **approximation/work-around** (it is explicitly not called model any more) but then evaluates the likelihood of final topologies under the `-m GTRGAMMA` model.
- new command-line option `-# n` which allows you to specify that RAxML should execute `n` subsequent analyses on the initial alignment or `n` subsequent non-parametric bootstraps when used in combination with the `-b` option.
- Implementation of multiple model estimate for multi-gene alignments (`-q` option).
- New options to specify bifurcating and multifurcating constraint trees (`-g` and `-r` options).
- Ability to read incomplete (not containing all taxa) starting trees to add e.g. newly sequenced/aligned taxa to your previously computed tree.
- Option to start ML search from complete random starting tree (`-d` option).

1.3 Citing RAxML

If you use sequential RAxML-VI-HPC please cite [8]. In case you are using the OpenMP version of RAxML please also cite [9]. The novel rapid hill-climbing

algorithm will be described in a future paper. Until this gets published please also cite my web-page www.ics.forth.gr/~stamatak when using RAxML-VI-HPC.

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. Also note, that likelihood values obtained by RAxML-VI-HPC can not be directly compared to those of RAxML-VI (standard version) either. This is due to changes in the likelihood function implementation and model parameter optimization procedure!

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 [1]. I mostly optimize final topologies with PHYML to avoid potential criticism that my experiments are biased in favor of RAxML.

PERSONAL OPINION: Differences in Likelihood scores:

In theory all ML programs implement the same numerical 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 get 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. In my experiments I have observed differences among final likelihood values between GARLI, IQPNNI, PHYML, RAxML (every program showed a different value). RAxML likelihood values typically differ by a greater amount from those obtained by other programs. The rationale for this is that the general strategy adopted in RAxML is to trade exactness of the likelihood score for speed with respect to the calculations (re-ordering of instructions, low-level optimization etc.) and the scaling strategy for very small likelihood values which is only an approximate scaling. 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. Note that, if you perform a bootstrap analysis you don't need to worry too much about likelihood values anyway.

2.2 The GTRCAT Mystery

There is a paper available now [6] which describes what `GTRCAT` is and why I don't like `GTRGAMMA`. The main idea behind `GTRCAT` is to allow for inte-

gration of rate heterogeneity into phylogenetic analyses at a significantly lower computational cost (about 4 times faster) and memory consumption (4 times lower). However, due to the way individual rates are optimized and assigned to rate categories in `GTRCAT` (for details on this please read the paper), this approximation is numerically instable. This means: **DO NOT COMPARE ALTERNATIVE TREE TOPOLOGIES BASED ON THEIR GTRCAT LIKELIHOOD VALUES!**

There is a large possibility for a biased assessment of trees. This is the reason why `GTRCAT` is called approximation instead of model.

3 Installation, Compilers, Platforms

RAxML-VI-HPC can be download at www.ics.forth.gr/~stamatak as open source code. To install RAxML-VI download the `RAxML-VI-HPC.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`.
2. `raxmlHPC-OMP` the OpenMP-parallelized version of RAxML which runs on 2-way (or dual processor), 4-way, and 8-way CPUs. It is best compiled with the `pgcc` (PGI) compiler by typing `make -f Makefile.pgi`.
3. `raxmlHPC-MPI` the MPI-parallelized version for all types of clusters to perform parallel bootstraps or multiple inferences on the original alignment, compile with the `mpicc` (MPI) and `gcc` compilers by typing `make -f Makefile.mpi`.

3.1 When to use which version?

The use of the sequential version is for small datasets and for initial experiments to determine appropriate search parameters.

The OpenMP version will work well for very long alignments (rules of thump: $\geq 3,000$ base pairs under `GTRGAMMA` and $\geq 5,000$ base pairs under `GTRCAT`). If your alignments are not that long and you have e.g. a dual-processor available it is better to run independent parallel booststraps or multiple analyses on them using the sequential version. **Do not forget** to set the number of threads `OMP_NUM_THREADS` that will be executed per node to the number of CPUs. If you have a `bash` shell and a 4-way Opteron make sure to set `export OMP_NUM_THREADS=4`.

The MPI-version is for executing your 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. WARNING: The current MPI-version will only work properly if you specify the "-#" option in the command line, since it has been designed to do multiple inferences in parallel!

The best hardware to run RAxML on is currently the AMD Opteron [9] architecture.

3.2 Processor Affinity with the OpenMP Version

An important aspect if you want to use the OpenMP version of the program is to find out how your operating system/platform handles processor affinity of threads. Within the shared-memory context processor affinity means that if you run e.g. 4 threads on a 4-way CPU the threads 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, 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.

Note, that methods to enforce processor affinity vary among operating systems and installations. Thus, the best thing to do is to contact your system administrator or supercomputing center.

4 The RAxML Options

```
raxmlHPC[-MPI|-OMP] [-a weightFileName]
                    [-b bootstrapRandomNumberSeed]
                    [-c numberOfCategories]
NEW                 [-d]
NEW                 [-e likelihoodEpsilon]
                    [-f d|e]
NEW                 [-g groupingFileName]
                    [-h]
                    [-i initialRearrangementSetting]
                    [-j]
                    [-m GTRCAT|GTRMIX|GTRGAMMA]
NEW                 [-q multipleModelFileName]
NEW                 [-r constraintFileName]
                    [-t userStartingTree]
                    [-w workingDirectory]
                    [-v]
                    [-y]
                    [-# numberOfRuns]
                    -s sequenceFileName
                    -n outputFileName
```

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 bootstraps, i.e. in combination with the `-#` *or* `-#` *and* `-b` options. Note, that if you do not specify `-#` the parallel code will not work properly!
3. `raxmlHPC-OMP` only makes sense if you have a really long alignment (in terms of base pairs) and you have access to 2-way or 4-way CPUs.

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`.

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
```

```
-b bootstrapRandomNumberSeed
```

This option allows you to turn on non-parametric bootstrapping. 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 reproducable despite the fact that you specify a random number seed.

```
-c numberOfCategories
```

This option allows you to specify the number of distinct rate categories used into which the individually optimized rates for each individual site are "thrown" under `-m GTRCAT`. The results in [6] indicate that the default of `-c 25` works fine in most practical cases.

`-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.

`-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 more quickly. 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.

`-f algorithm`

This option allows you to select the type of algorithm you want RAxML to execute. When you specify `-f d` which is also the default, RAxML will execute the new rapid hill-climbing algorithm which is intended for huge data. When `-f e` is specified RAxML will optimize the model parameters and branch lengths of a topology provided via the `-t` option under `GTRGAMMA`.

`-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.

`-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 5.1 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.

`-h`

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

`-m modelOfEvolution`

Selection of the model of nucleotide substitution to be used. The default setting is `-m GTRCAT`.

- `-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 (`-#`) 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 model of nucleotide substitution with the Γ model of rate heterogeneity. 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.

`-n outputFileName`

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

`-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

base frequencies will be estimated and optimized for each partition. If you have an 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:

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

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

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

Finally, 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:

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

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

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

```
-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.

```
-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.

`-v`

Displays version information.

`-w workingDirectory`

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

`-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.

`-# numberOfRuns`

Specifies the number of alternative runs on distinct starting trees. E.g. if `-# 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

4.1 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 rearrangment setting found by the program will be indicated as well.
– `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 completly random starting tree if the program was executed with `-d`.
– `RAxML_checkpoint.exampleRun.checkpointNumber` if it has been 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` **all** final bootstrapped trees will be written to this one, single file.

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`.

Despite the observation that the default parameters work well in most practical cases, the first 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.

5.1 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:

```
raxmlHPC -y -s ex_al -n ST0
...
raxmlHPC -y -s ex_al -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 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 5.1 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 5.1 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 rearrangment settings can be retrieved from file `RAxML_info.AI_0 ... RAxML_info.AI_4`.

5.3 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 (10 or better 100) 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
```

preceeded 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.

5.4 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 reproducable results. Note that we can use `GTRCAT` here, because we do not want to compare final trees based on ML scores.

To do a parallel bootstrap type:

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

once again preceeded 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 reproducable.

6 Frequently Asked Questions

Q: Why does RAxML not implement a proportion of Invariable (P-Invar) Sites estimate?

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 not implementing 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. Gangolf Jobb has not implemented P-Invar in his Treefinder [2] (www.treefinder.de) program based on similar concerns.

Q: Why does RAxML-HPC only implement GTRCAT and GTRGAMMA models?

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 [3] 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 (www.bio.utexas.edu/grad/zwickl/web), based on similar considerations (personal communication).

Q: Why does RAxML focus on DNA-based tree inference?

The whole RAxML project started from a high performance computing perspective, i.e. with the goal to compute huge trees on parallel computers. Since DNA data has only 4 states in contrast to 20 for protein data the computational cost (execution times and memory consumption) to compute large trees with many taxa with DNA data is lower. Thus, we are currently able to obtain larger trees based on DNA data.

PERSONAL OPINION: My personal view, although this might not necessarily correspond to the current opinion in the community is that the models of nucleotide substitution for DNA data are currently better understood. In addition DNA data provides more signal than protein data due to the redundancies in some protein encodings. So, from my point of view, if you have the corresponding DNA data which encodes proteins, better use the DNA data because it will probably exhibit a better phylogenetic signal and execute much faster.

Tips & Tricks: My colleague Olaf Bininda-Emonds has developed a nice little script [4] (available at www.personal.uni-jena.de/~b6biol2/) for converting DNA sequence data to protein data, align the protein data with ClustalW[1] and then re–convert it into a DNA alignment. If you have the DNA data available, but want to align with amino acids, and then use RAxML to infer trees, Olaf's script will be very helpful.

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 [5] [7] [8]. 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

[1] The ClustalW and most other alignment algorithms typically perform much better on protein data.

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 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 `-#` option in the command-line which **must** be used for the MPI version to work properly.

7 Things in Preparation

A couple of things are in preparation (to be released within the next 6 months) which will further expand the capabilities of RAxML:

– MPI-based parallelization of the actual search algorithm for parallel inference of a single huge tree on a cluster

For any further requests or proposals that you might have please send an email to stamatak@ics.forth.gr 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): 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.

References

1. S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52(5):696–704, 2003.
2. G. Jobb, A. Haeseler, and K. Strimmer. Treefinder: A powerful graphical analysis environment for molecular phylogenetics. *BMC Evolutionary Biology*, 4, 2004.
3. D. Posada and K. Crandall. Modeltest: testing the model of dna substitution. *Bioinformatics*, 14(9):817–818, 1998.
4. O. R.P.Bininda-Emonds. transalign: using amino acids to facilitate the multiple alignment of protein-coding dna sequences. *BMC Bioinformatics*, 6(156), June 2005.
5. A. Stamatakis. An efficient program for phylogenetic inference using simulated annealing. In *Proc. of IPDPS2005*, Denver, Colorado, USA, 2005.
6. A. Stamatakis. Phylogenetic models of rate heterogeneity: A high performance computing perspective. In *Proc. of IPDPS2006*, Rhodos, Greece, 2006.
7. A. Stamatakis, T. Ludwig, and H. Meier. New fast and accurate heuristics for inference of large phylogenetic trees. In *Proc. of IPDPS2004*, 2004.
8. A. Stamatakis, T. Ludwig, and H. Meier. Raxml-iii: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 2005.
9. A. Stamatakis, O. M., and L. T. Raxml-omp: An efficient program for phylogenetic inference on smps. In *Proc. of PaCT05*, pages 288–302, 2005.