Using RAxML-NG in Practice

Alexey M. Kozlov[1]
Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies
[Schhloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany]
Alexey.Kozlov@h-its.org
http://orcid.org/0000-0001-7394-2718

Alexandros Stamatakis[2]
Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies,
Karlsruhe Institute of Technology, Institute for Theoretical Informatics
[Schhloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany]
Alexandros.Stamatakis@h-its.org
https://orcid.org/0000-0003-0353-0691

Abstract

RAxML-NG is a new phylogentic inference tool that replaces the widely-used RAxML and ExaML tree inference codes. Compared to its predecessors, RAxML-NG offers improvements in accuracy, flexibility, speed, scalability, and user-friendliness. In this chapter, we provide practical recommendations for the most common use cases of RAxML-NG: tree inference, branch support estimation via non-parametric bootstrapping, and parameter optimization on a fixed tree topology. We also describe best practices for achieving optimal performance with RAxML-NG, in particular, with respect to parallel tree inferences on computer clusters and supercomputers. As RAxML-NG is continuously updated, the most up-to-date version of the tutorial described in this chapter is available online at: `https://cme.h-its.org/exelixis/raxml-ng/tutorial`.

2012 ACM Subject Classification Applied computing → Life and medical sciences → Computational biology → Molecular evolution

Keywords and phrases phylogenetic inference, maximum likelihood, parallel processing, HPC

Supplement Material https://github.com/amkozlov/ng-tutorial

Funding This work was financially supported by the Klaus Tschira Foundation

1 Introduction

RAxML [21, 22] is a widely-used tool for maximum likelihood (ML) based phylogenetic inference. It has been cited by more than 25,000 publications over the last 15 years. More recently, we introduced ExaML [20, 8], a variant of RAxML with several novel features including checkpointing, improved load balancing, and an efficient fine-grained MPI parallelization. These improvements were particularly important for being able to analyze large-scale phylogenomic datasets on compute clusters and supercomputer systems [12, 5]. However, ExaML only offered a core subset of RAxML functionality. It lacks several important functions such as bootstrapping and comprehensive starting tree generation. These limitations, and its dependency on MPI, made ExaML more difficult to install and use, and therefore presumably limited its adoption.

[1] [This work was financially supported by the Klaus Tschira Foundation]
[2] [This work was financially supported by the Klaus Tschira Foundation]

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

With RAxML-NG, we introduced one single code that scales from the laptop to the supercomputer. It combines the parallel efficiency of ExaML with functional completeness of RAxML. Furthermore, RAxML-NG is more user-friendly than RAxML/ExaML because of a simplified installation process, the re-engineered command line interface, and new default settings that cover the most common usage scenarios (see Appendix A).

RAxML-NG can be downloaded at `https://github.com/amkozlov/raxml-ng`. The corresponding documentation is available via a `GitHub` wiki at `https://github.com/amkozlov/raxml-ng/wiki`. Technical implementation details and benchmarking results can be found in [7] and Chapter 4 of [9]. We also offer extensive user support via the RAxML google group: `https://groups.google.com/forum/#!forum/raxml`.

All datasets used in this chapter can be downloaded from `https://github.com/amkozlov/ng-tutorial`.

IMPORTANT NOTE: You will need RAxML-NG 0.8.0b or later for this tutorial, so please make sure you have the right version:

```
$ raxml-ng -v

RAxML-NG v. 0.8.0 BETA released on 11.01.2019 by The Exelixis Lab.
```

2 Pre-processing the alignment

2.1 Sanity check

Before starting the actual analysis, it is strongly recommended to perform a multiple sequence alignment (MSA) sanity check by calling RAxML-NG with the `'--check'` option.

```
$ raxml-ng --check --msa bad.fa --model GTR+G
```

This command will check the MSA for several common format issues as well as data inconsistencies including:

- duplicate taxon names
- invalid characters in taxon names
- duplicate sequences
- fully undetermined ('gap-only') sequences and columns
- incorrect or incompatible evolutionary models, partitioning scheme and starting trees (if provided)

Performing this check before starting the analysis is very important, since based on our experience, a large proportion of failed RAxML runs are due to tree or MSA format errors!

Let us take a closer look at the output of our sanity check invocation:

```
WARNING: Fully undetermined columns found: 2

WARNING: Fully undetermined sequences found: 2

WARNING: Sequences t3 1200bp and t8 are exactly identical!
WARNING: Duplicate sequences found: 1

ERROR: Following taxon name contains invalid characters: t9'
ERROR: Following taxon name contains invalid characters: t6)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
ERROR: Following taxon name contains invalid characters: t3 1200bp

ERROR: Alignment check failed (see details above)!
```

It seems that this MSA file has almost every conceivable problem. RAxML-NG will attempt to automatically fix the most common issues, for instance, by removing fully undetermined columns and sequences, replacing invalid characters in taxon names etc. To achieve this, it will write an analogously updated/fixed MSA file back to disk:

```
NOTE: Reduced alignment (with duplicates and gap-only sites/taxa removed)
NOTE: was saved to: /home/alexey/ng-tutorial/bad.fa.raxml.reduced.phy
```

Let us now repeat the sanity check with the fixed file:

```
$ raxml-ng --check --msa bad.fa.raxml.reduced.phy --model GTR+G

[..]
Alignment can be successfully read by RAxML-NG.
```

2.2 Compression and conversion to binary format

For large alignments, we recommend using the `'--parse'` command after, or, instead of `'--check'`:

```
$ raxml-ng --parse --msa prim.phy --model GTR+G
```

In addition to the MSA sanity check, this command will compress alignment patterns and store the MSA in a binary format (RAxML Binary Alignment, RBA):

```
NOTE: Binary MSA file created: prim.phy.raxml.rba
```

In the process of pattern compression, RAxML-NG identifies identical MSA sites and converts them into a single site ('pattern') with a weight corresponding to their number of occurrence. Since this compression step can potentially require quite some time for broad supermatrix MSAs, directly loading a RBA file is (substantially) faster compared to parsing and loading a plain FASTA or PHYLIP file. This parsing speed is important for large-scale parallel tree inferences with say, 500 cores or more, as virtually no time is lost in the beginning for parsing the file and the cores can almost immediately start with the likelihood calculations (see [8, Supplement Section 3] for more details).

In addition, `'--parse'` will estimate the memory requirements and optimal number of CPUs/threads for the particular MSA:

```
* Estimated memory requirements                : 2 MB
* Recommended number of threads / MPI processes: 2
```

Even though these estimates are approximate, they provide a 'good' starting point for experimentation (see Section 7 for details) to determine the optimal number of cores that will yield maximum parallel efficiency.

3 Inferring ML trees

Let us now infer a tree under the `GTR+GAMMA` (general time reversible model of nucleotide substitution with a Γ model of rate heterogeneity) model with default parameters. We

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

will use 2 threads as suggested above, and provide a fixed random number seed to ensure reproducibility. By using a fixed random number seed RAxML-NG will always produce the same sequence of random numbers and therefore a failed run can be easily reproduced for debugging. Note that, we will also always use a new name via the `'--prefix'` output file name option for each RAxML-NG example run to avoid overwriting preceding output files.

```
$ raxml-ng --msa prim.phy --model GTR+G --prefix T3 --threads 2 --seed 2
```

The above command will perform 20 tree searches using 10 random and 10 parsimony-based starting trees. In the end it will pick the best-scoring topology:

```
Analysis options:
  run mode: ML tree search
  start tree(s): random (10) + parsimony (10)
```

This default setting represents a reasonable choice for most practical cases. However, computational resources permitting, we might want to increase the number of starting trees to explore the tree space more thoroughly:

```
$ raxml-ng --msa prim.phy --model GTR+G --prefix T4 --threads 2 --seed 2
    --tree pars{25},rand{25}
```

Conversely, we can also just perform a quick-and-dirty search from a single random starting tree using the `--search1` command:

```
$ raxml-ng --search1 --msa prim.phy --model GTR+G --prefix T5 --threads 2
    --seed 2
```

Let us now compare the results of all three alternative tree inference runs:

```
$ grep "Final LogLikelihood:" T{3,4,5}.raxml.log

T3.raxml.log:Final LogLikelihood: -5708.923977
T4.raxml.log:Final LogLikelihood: -5708.923977
T5.raxml.log:Final LogLikelihood: -5708.979717
```

This looks quite good: the likelihood surface appears to have a clear peak, which RAxML-NG finds regardless of the search parameters.

We use the term *likelihood surface* in a colloquial/subjective way to describe the space of all possible tree topologies and their respective likelihood scores. If the likelihood surface is smooth there seems to be one clear peak that is identified by several independent searches. If the surface is rough, we typically observe a plethora of substantially different tree topologies but with statistically indistinguishable likelihood scores. Rough likelihood surfaces are frequently observed for large single gene MSAs with $1,000$ or more sequences.

Let us get back to our example. We observe that the tree 'T5' has a slightly worse likelihood. The question arises if it also has a distinct topology. We can check this by using the `--rfdist` command to compute the topological Robinson-Foulds (RF) distance [17] between all trees we have inferred:

```
$ cat T{3,4}.raxml.mlTrees T5.raxml.bestTree > mltrees
$ raxml-ng --rfdist --tree mltrees --prefix RF

[...]
```

 doi:10.20944/preprints201905.0056.v1


```
Loaded 71 trees with 12 taxa.

Average absolute RF distance in this tree set: 0.000000
Average relative RF distance in this tree set: 0.000000
Number of unique topologies in this tree set: 1
```

This tells us that, in fact, all 71 resulting topologies (one per starting tree) are identical, so we can be optimistic that we found *the* globally optimal ML tree. The slight numerical deviations we observe for the likelihood scores are due to numerical round-off error propagation. To conduct calculations computers rely on so-called floating-point numbers that are just an imperfect representation of the real numbers on the machine.

Unfortunately, not all datasets are as well-behaved as our initial test dataset:

```
$ raxml-ng --msa fusob.phy --model GTR+G --prefix T6 --seed 2 --threads 2
$ grep "ML tree search #" T6.raxml.log

[00:00:03] ML tree search #1, logLikelihood: -9974.668088
[00:00:07] ML tree search #2, logLikelihood: -9974.666644
[00:00:11] ML tree search #3, logLikelihood: -9974.669417
[00:00:15] ML tree search #4, logLikelihood: -9974.664855
[00:00:19] ML tree search #5, logLikelihood: -9974.663779
[00:00:22] ML tree search #6, logLikelihood: -9974.666906
[00:00:26] ML tree search #7, logLikelihood: -9974.668155
[00:00:30] ML tree search #8, logLikelihood: -9974.664340
[00:00:33] ML tree search #9, logLikelihood: -9974.666937
[00:00:37] ML tree search #10, logLikelihood: -9974.666388
[00:00:40] ML tree search #11, logLikelihood: -9980.601114
[00:00:43] ML tree search #12, logLikelihood: -9974.675123
[00:00:46] ML tree search #13, logLikelihood: -9980.602470
[00:00:49] ML tree search #14, logLikelihood: -9974.671637
[00:00:52] ML tree search #15, logLikelihood: -9980.602668
[00:00:54] ML tree search #16, logLikelihood: -9980.601182
[00:00:57] ML tree search #17, logLikelihood: -9974.672801
[00:01:00] ML tree search #18, logLikelihood: -9974.668668
[00:01:03] ML tree search #19, logLikelihood: -9974.669997
[00:01:06] ML tree search #20, logLikelihood: -9980.607281
```

This example illustrates why it is so important to use multiple starting trees: we can see that some searches converged to a local optimum with a substantially lower likelihood (-9980.607281 vs. -9974.669997). Once again, let's check if the resulting trees differ topologically:

```
$ raxml-ng --rfdist --tree T6.raxml.mlTrees --prefix RF6
[...]
Loaded 20 trees with 38 taxa.

Average absolute RF distance in this tree set: 3.157895
Average relative RF distance in this tree set: 0.045113
Number of unique topologies in this tree set: 2
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
Pairwise RF distances saved to: <...>/RF6.raxml.rfDistances
```

So we have 2 distinct topologies in our set of 20 inferred trees, which correspond to two distinct likelihood values we observed in the tree search output. Let's look at the individual pairwise RF distances which are printed to the `RF6.raxml.rfDistances` file:

```
$ cat RF6.raxml.rfDistances

0       1       0       0.000000
0       2       0       0.000000
0       3       0       0.000000
0       4       0       0.000000
0       5       0       0.000000
0       6       0       0.000000
0       7       0       0.000000
0       8       0       0.000000
0       9       0       0.000000
0       10      8       0.114286
0       11      0       0.000000
0       12      8       0.114286
0       13      0       0.000000
0       14      8       0.114286
0       15      8       0.114286
0       16      0       0.000000
0       17      0       0.000000
0       18      0       0.000000
0       19      8       0.114286
[...]
```

As we can see, all 10 searches from the random starting trees (trees 0 to 9) found the best-scoring topology (RF=0, logL=−9974), whereas 5 out of 10 searches from a parsimony starting tree converged to a local optimum (RF = 8, logL = −9980). Ideally, one should also check whether the likelihood difference between both topologies is statistically significant. This could be done by e.g. CONSEL tool [18] that implements a large number of statistical significance tests.

4 Bootstrapping and branch support

NOTE: As of v.0.8.0*b*, RAxML-NG only supports the *standard* bootstrap algorithm (corresponding to the `-b` option in standard RAxML). It is subsantially slower than *rapid* bootstrapping implemented in standard RAxML (`-x` or `-f a` options), but returns more accurate support values.

4.1 Inferring bootstrap trees

RAxML-NG can perform the standard non-parametric bootstrap by re-sampling alignment columns and re-inferring a tree for each bootstrap (BS) replicate MSA:

```
raxml-ng --bootstrap --msa prim.phy --model GTR+G --prefix T7 --seed 2 --
    threads 2
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

By default, RAxML-NG employs so-called MRE-based bootstopping test [15] to automatically determine the sufficient number of BS replicates. The diagnostic statistics is evaluated after every 50 BS tree inferences, and once its value drops below the cutoff, the analysis stops. The key motivation for the bootstopping criterion is to ensure that neither too few (unstable/inaccurate support values) nor too many (waste of CPU time) replicates are computed. To assess stability of support values, the bootstopping criterion repeatedly splits the current set of BS replicate trees at random into two tree sets of equal size and subsequently compares the support values induced by these sets. If the induced support values are not substantially different it suggests that bootstrapping should stop.

Let us now infer some BS replicates:

```
  bootstrap replicates: max: 1000 + bootstopping (autoMRE, cutoff:
    0.030000)
[...]
[00:00:15] Bootstrap tree #50, logLikelihood: -5762.777409
[00:00:15] Bootstrapping converged after 50 replicates.
```

This converged quickly! Let us now manually increase the number of BS replicates to be on the safe side:

```
raxml-ng --bootstrap --msa prim.phy --model GTR+G --prefix T8 --seed 2 --
    threads 2 --bs-trees 200
```

Bootstrap convergence can also be assessed *after* the BS inference by using the '–bsconverge' command. Note that, we can also change the bootstopping cutoff value to make the test more or less stringent:

```
$ raxml-ng --bsconverge --bs-trees T7.raxml.bootstraps --prefix T9 --seed
    2 --threads 2 --bs-cutoff 0.01

# trees   avg WRF  avg WRF in %   # perms: wrf <= 1.00 %  converged?
     50    7.400          1.644                        0     NO
Bootstopping test did not converge after 50 trees
```

The cutoff here represents the weighted RF (WRF) distance between extended majority rule consensus trees calculated on the respective randomly split BS tree set. By default we calculate 1000 such random splits of the tree set and average the WRF distances over them.

As we can see, with a 1% WRF cutoff 50 replicates are not enough. What about 200?

```
$ raxml-ng --bsconverge --bs-trees T8.raxml.bootstraps --prefix T10 --seed
     2 --threads 2  --bs-cutoff 0.01

# trees   avg WRF  avg WRF in %   # perms: wrf <= 1.00 %  converged?
     50    7.400          1.644                        0     NO
    100   11.702          1.300                      245     NO
    150   13.960          1.034                      457     NO
    200   16.484          0.916                      648     NO
Bootstopping test did not converge after 200 trees
```

Still no convergence, but the WRF distance (`avg WRF in \%`) is steadily decreasing as we add more replicates, and now lies below the 1% cutoff for 648 out of the 1000 random splits of the BS tree set (convergence requirement: > 990). This looks promising, and we

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

can expect convergence after few hundred replicates. Luckily, bootstraps are independent, an we can thus reuse the 200 BS trees we have already inferred. So let's add 400 additional BS replicate trees. **It is extremely important to specify a distinct random seed for the second run, otherwise first 200 trees of the second run will be identical to the first run!**

```
raxml-ng --bootstrap --msa prim.phy --model GTR+G --prefix T11 --seed 333
    --threads 2 --bs-trees 400
```

Now, we can simply concatenate the BS replicate trees from both runs, and re-assess the convergence:

```
$ cat T8.raxml.bootstraps T11.raxml.bootstraps > allbootstraps
$ raxml-ng --bsconverge --bs-trees allbootstraps --prefix T12 --seed 2 --
    threads 1 --bs-cutoff 0.01

# trees   avg WRF  avg WRF in %   # perms: wrf <= 1.00 %  converged?
     50    7.400          1.644                       0     NO
     100  11.702          1.300                     245     NO
     150  13.960          1.034                     457     NO
     200  16.484          0.916                     648     NO
     250  17.410          0.774                     841     NO
     300  18.900          0.700                     927     NO
     350  20.060          0.637                     942     NO
     400  22.076          0.613                     969     NO
     450  23.856          0.589                     973     NO
     500  26.164          0.581                     985     NO
     550  27.844          0.563                     985     NO
     600  28.462          0.527                     991     YES
Bootstopping test converged after 600 trees
```

Now we have convergence, even with a more stringent bootstopping cutoff. However, we had to conduct 600 BS replicate searches instead of just 50. On large datasets, this quickly becomes computationally expensive. Hence in practice, the default bootstopping cutoff value of an average WRF of 3% should be sufficient in most cases [15].

4.2 Computing branch support

Now, what can we do with the BS trees? We can either summarize them via some sort of consensus tree (strict, majority, majority rule extended, e.g., using standard RAxML or some other tool) or we can map them onto the best-scoring ML tree that we inferred on the original MSA. It is debatable what might the best way of summarizing BS trees might be, but there seems to be a trend toward mapping the BS support values onto the best-scoring/best-known ML tree (remember: finding *the* globally optimal ML tree is computationally hard), so let us do that.

We will use the ML tree obtained in run `T3` (see Section 3):

```
raxml-ng --support --tree T3.raxml.bestTree --bs-trees allbootstraps --
    prefix T13 --threads 2
```

Now, we can actually look at this best-known ML tree including supports, `T13.raxml.support` using some tree viewer (e.g., Dendroscope or FigTree). **Beware**: due to confusion between

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

node and *branch* attributes in the NEWICK format, some viewers have or had issues concerning correct branch support visualization [1]. If possible (e.g., in recent versions of Dendroscope), you should specify that support values must be interpreted as *edge* labels.

Alternatively, we can also compute the so-called *Transfer Bootstrap Expectation* (*TBE*) support metric recently suggested by Lemoine *et al.* [11] as follows:

```
$ raxml-ng --support --tree T3.raxml.bestTree --bs-trees allbootstraps --
    prefix T14 --threads 2 --bs-metric tbe
```

While the standard bootstrap support metric (*Felsenstein's bootstrap*, *FBP*) relies on binary presence/absence of bipartitions from replicate trees in the best-known ML tree, TBE is based on a gradual 'transfer' distance. Transfer distance between two branches equals to the minimum number of taxa that have to be transfered (or removed) to make those branches identical (that is, both branches split the set of taxa in identical subsets). TBE support for a branch in the ML tree is computed based on the *minimum* transfer distance between this branch and *any* branch in the BS replicate tree; in other words, we compare each ML tree branch to its respective closest branch in the BS replicate tree (please see [11] for details). For this reason, TBE can better recover support in very large trees with thousands of taxa. This is because, bipartitions that exactly match those in the best-known ML tree are rarely present in replicates, and thus FBP usually yields low support, especially for deep branches.

As shown above, TBE can be computed from the same set of bootstrap replicate trees, so there in no need to repeat the compute-intensive tree inference step. However, the TBE computation itself is more expensive than FBP. This can be noted when computing the TBE on large trees: e.g., on a laptop, RAxML-NG v0.8.0 needs ≈ 20 seconds per BS replicate tree on the $9,000$ taxon dataset from [11]. However, this time is still negligible compared to the time required for BS replicate tree inference.

Finally, RAxML-NG offers a convenient "all-in-one" analysis mode for really lazy users (analogous to `-f a` in standard RAxML):

```
$ raxml-ng --all --msa prim.phy --model GTR+G --prefix T15 --seed 2 --
    threads 2 --bs-metric fbp,tbe
```

This will do all of the above steps (20 ML inferences on the original MSA, inferring bootstrap replicate trees, and drawing support values using both FBP and TBE on the best-scoring tree) with just a single command:

```
$ ls T15.*
T15.raxml.bestModel  T15.raxml.bestTree
T15.raxml.bootstraps T15.raxml.log
T15.raxml.mlTrees    T15.raxml.rba
T15.raxml.startTree  T15.raxml.supportFBP
T15.raxml.supportTBE
```

Please note, that for taxa-rich alignments running such a complete analysis with the `--all` command can take extremely long. It is therefore recommended to estimate the runtime required for a single tree search first, for instance, by using the `--search1` command. Based on the results, one might consider allocating more CPU cores and/or using the coarse-grained parallelization (see Section 7.7).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

5 Tree likelihood evaluation

5.1 Basics

Another standard task is to evaluate trees, that is, to compute the likelihood of a given fixed tree topology by just optimizing model and/or branch length parameters on that fixed tree. This operation is frequently needed in model and hypothesis testing.

The basic option is `--evaluate`. It will re-optimize all branch lengths and all free model parameters. This default behavior can be altered with `--opt-branches on/off` and `--opt-model on/off`. There is also the `--loglh` command which is a short alias for

```
--evaluate --opt-branches off --opt-model off --nofiles
```

that is, it will compute and print the likelihood of the tree(s) without optimizing anything and without creating any output files. For instance, we can re-compute the likelihood of `T3` with *default* model parameters as follows:

```
$ raxml-ng --loglh --msa prim.phy --model GTR+G --tree T3.raxml.bestTree
    --threads 2

   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 1.000000 (ML),
    weights&rates: (0.250000,0.136954) (0.250000,0.476752)
    (0.250000,1.000000) (0.250000,2.386294)
   Base frequencies (ML): 0.250000 0.250000 0.250000 0.250000
   Substitution rates (ML): 1.000000 1.000000 1.000000 1.000000 1.000000
    1.000000

Final LogLikelihood: -6420.095053
```

In contrast, after re-optimizing all model parameters we obtain:

```
$ raxml-ng --evaluate --msa prim.phy --model GTR+G --tree T3.raxml.
    bestTree --threads 2 --nofiles

   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 0.377068 (ML),
    weights&rates: (0.250000,0.013550) (0.250000,0.164429)
    (0.250000,0.705224) (0.250000,3.116797)
   Base frequencies (ML): 0.354236 0.321458 0.080986 0.243320
   Substitution rates (ML): 3.989744 45.320369 3.326172 2.533579 36.939966
     1.000000

Final LogLikelihood: -5709.002997
```

Finally, we can fix some parameters to certain values and optimize others:

```
$ raxml-ng --evaluate --msa prim.phy --model GTR+G{2.0}+F{0.2/0.3/0.4/0.1}
     --tree T3.raxml.bestTree --threads 2 --nofiles

   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 2.000000 (user),
    weights&rates: (0.250000,0.293275) (0.250000,0.655014)
    (0.250000,1.069990) (0.250000,1.981722)
   Base frequencies (user): 0.200000 0.300000 0.400000 0.100000
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
   Substitution rates (ML): 44.454379 47.979464 65.161744 2.413970
    252.745302 1.000000

Final LogLikelihood: -6158.335994
```

5.2 Comparing different models

Let us now conduct some small tests that show how the likelihood improves as we add more and more free parameters to our model. For this, we will use the best-scoring ML tree from Section 3 again.

Let us first evaluate the tree under the most simple model, Jukes-Cantor (`JC`):

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model JC --tree T3.
    raxml.bestTree --prefix E1
```

Now, let us add the Γ model of rate heterogeneity:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model JC+G --tree T3.
    raxml.bestTree --prefix E2
```

Now let us use a simple `GTR` model (without rate heterogeneity):

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model GTR --tree T3.
    raxml.bestTree --prefix E3
```

`GTR` with the `GAMMA` model of rate heterogeneity, but using empirical base frequencies:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model GTR+G+FC --tree
    T3.raxml.bestTree --prefix E4
```

And now also conducting a ML estimate of the base frequencies:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model GTR+G+FO --tree
    T3.raxml.bestTree --prefix E5
```

Finally, using 4 free rates [23] instead of `GAMMA`-distributed rates:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model GTR+R4+FO --tree
    T3.raxml.bestTree --prefix E6
```

Let us check the results:

```
$ grep logLikelihood E*.raxml.log

E1.raxml.log:[00:00:00] Tree #1, final logLikelihood: -6424.203056 <- JC
E2.raxml.log:[00:00:00] Tree #1, final logLikelihood: -6272.469063 <- JC+
    GAMMA
E3.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5934.158958 <- GTR
E4.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5719.408956 <- GTR
    + GAMMA + empirical base freqs
E5.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5709.002997 <- GTR
    + GAMMA + estimated base freqs
E6.raxml.log:[00:00:01] Tree #1, final logLikelihood: -5706.008654 <- GTR
    + FreeRate + estimated base freqs
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

Unsurprisingly, models with more free parameters yield better likelihood scores. However, this does not mean that we should always use the most parameter-rich model. Instead, it is common to use information theoretical criteria such as `AIC` (Akaike Information Criterion), `AICc` (corrected AIC; AIC with a correction for small sample sizes) or `BIC` (Bayesian Information Criterion) to penalize parameter-rich models and thereby avoid overfitting the data. The three aforementioned criteria are implemented in RAxML-NG:

```
$ grep "AIC score" E*.raxml.log

E1.raxml.log:AIC score: 12890.406112 / AICc score: 12891.460907 / BIC
    score: 12991.209684  <- JC
E2.raxml.log:AIC score: 12588.938126 / AICc score: 12590.094698 / BIC
    score: 12694.541868  <- JC+G
E3.raxml.log:AIC score: 11926.317917 / AICc score: 11928.322525 / BIC
    score: 12065.522849  <- GTR
E4.raxml.log:AIC score: 11498.817912 / AICc score: 11500.963241 / BIC
    score: 11642.823014  <- GTR+G+FC
E5.raxml.log:AIC score: 11478.005995 / AICc score: 11480.151323 / BIC
    score: 11622.011097  <- GTR+G+FO
E6.raxml.log:AIC score: 11482.017308 / AICc score: 11484.940742 / BIC
    score: 11650.023260  <- GTR+R4+FO
```

For all criteria, model with the lowest score should be preferred. As we can see, the `GTR+G+FO` model scores best according to all three information theoretical criteria evaluated, even though it yields a lower likelihood than `GTR+R4+FO`. This example illustrates the importance of formal model selection. In practice, one should use specialized tools such as ModelTest-NG [2], IQTree/ModelFinder [6], or PartitionFinder [10] for this task.

6 Partitioned analyses

6.1 Partitioned model definition

So far, we always used a single evolutionary model for all MSA sites. This is biologically rather unrealistic, since different genes and/or codon positions typically exhibit distinct substitution patterns. Therefore, it is common to divide MSA sites into subsets or *partitions*, to which we can assign individual evolutionary models. In the most simple case, we can assign identical *models* to all partitions, but allow for independent *model parameter* estimates:

```
$ cat prim.part

GTR+G+FO, NADH4=1-504
GTR+G+FO, tRNA=505-656
GTR+G+FO, NADH5=657-898
```

The RAxML-NG partition file format is similar to that of standard RAxML and ExaML. Each line defines a partition, and contains the evolutionary model specification, the partition name, and the MSA site range(s). **Note that, the evolutionary model specification is not compatible with that in RAxML/ExaML!** In particular, rate heterogeneity has to be defined for each partition individually, that is, we specify `GTR+G` for every partition with the Γ model instead of using a global `-m GTRGAMMA` switch on the command line as

in standard RAxML/ExaML. Therefore, special care has to be taken when using legacy partition files.

Below, we show a more sophisticated example, where we use different per-partition substitution matrices and rate heterogeneity models, and also split the first gene by codon position:

```
$ cat prim2.part

GTR+G+FO, NADH4=1-504/3,2-504/3
JC+I, tRNA=505-656
GTR+R4+FC, NADH5=657-898
HKY, NADH4p3=3-504/3
```

Here, we use the *stride* notation to separate codon positions. For instance, `1-504/3` means *"every 3rd position in the range between 1 and 504"*.

6.2 Likelihood evaluation with partitioned models

Now, let us try to evaluate the likelihood on a fixed tree topology as in Section 5, but using a partitioned model (we will also increase the log output verbosity to be able to insepct the estimated parameter values):

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model prim.part --tree
    T3.raxml.bestTree --prefix P1 -log verbose

Optimized model parameters:

   Partition 0: NADH4
   Speed (ML): 1.045481
   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 0.320532 (ML),
    weights&rates: (0.250000,0.007108) (0.250000,0.120533)
    (0.250000,0.628725) (0.250000,3.243634)
   Base frequencies (ML): 0.347608 0.343620 0.074289 0.234483
   Substitution rates (ML): 1.110014 16.895228 0.903118 0.001000 11.861976
     1.000000

   Partition 1: tRNA
   Speed (ML): 0.505287
   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 0.300774 (ML),
    weights&rates: (0.250000,0.005358) (0.250000,0.105097)
    (0.250000,0.597100) (0.250000,3.292444)
   Base frequencies (ML): 0.362527 0.230093 0.151307 0.256073
   Substitution rates (ML): 66.393654 308.024274 43.477166 37.411363
    671.608883 1.000000

   Partition 2: NADH5
   Speed (ML): 1.216009
   Rate heterogeneity: GAMMA (4 cats, mean),  alpha: 0.614255 (ML),
    weights&rates: (0.250000,0.056061) (0.250000,0.320104)
    (0.250000,0.888421) (0.250000,2.735414)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
   Base frequencies (ML): 0.360963 0.322304 0.061324 0.255409
   Substitution rates (ML): 67.157660 1000.000000 56.903929 148.358484
    530.324413 1.000000
```

As we can see from the output above, even though we assigned the `GTR+G+FO` model to all three partitions, each of them has independent estimates of the parameter values (α shape parameter of the `GAMMA` distribution, base frequencies, and `GTR` substitution rates).

Let us repeat this evaluation using the second, more complex partition scheme:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model prim2.part --tree
    T3.raxml.bestTree --prefix P2 -log verbose
```

and compare the likelihoods for `P2` vs. `P1` vs. single `GTR+G+FO` model:

```
$ grep logLikelihood {E5,P1,P2}.raxml.log

E5.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5709.002997
P1.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5673.027260
P2.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5673.868809
```

So `P1` has the best likelihood score, closely followed by `P2`. But both `P1` and `P2` also introduce more free parameters compared to `GTR+G+FO`:

```
$ grep "Free parameters" {E5,P1,P2}.raxml.log

E5.raxml.log:Free parameters (model + branch lengths): 30
P1.raxml.log:Free parameters (model + branch lengths): 50
P2.raxml.log:Free parameters (model + branch lengths): 52
```

Hence, we will once again use `AIC/BIC` criteria to assess the model complexity versus likelihood score trade-off (lower=better):

```
grep "AIC score" {E5,P1,P2}.raxml.log

E5.raxml.log:AIC score: 11478.005995 / AICc score: 11480.151323 / BIC
    score: 11622.011097
P1.raxml.log:AIC score: 11446.054521 / AICc score: 11452.075772 / BIC
    score: 11686.063024
P2.raxml.log:AIC score: 11451.737617 / AICc score: 11458.260694 / BIC
    score: 11701.346461
```

The situation is less clear now: `AIC` and `AICc` favor the `P1` model, whereas `GTR+G+FO` has the best `BIC` score. Unfortunately, there seems to be no general consensus with respect to which information criterion is superior. Therefore, unfortunately the decision whether to use `AIC`, `AICc` or `BIC` is left to the user. Furthermore, the computation of `AICc` and `BIC` scores requires the knowledge of *sample size*. In the context of phylogenetics, both the number of alignment sites (columns) and the total number of alignment characters ($sites \times taxa$) have been proposed as sample size definitions (see e.g. [16] and references therein). In RAxML-NG, we define sample size as number of alignment sites, which is a more conservative option, also used by e.g., ModelTest-NG [2] and IQTree [14].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

6.3 Branch length linkage

In the output shown in Section 6.2, there is an extra parameter called `Speed` estimated for each partition:

```
Optimized model parameters:

   Partition 0: NADH4
   Speed (ML): 1.045481
[..]
   Partition 1: tRNA
   Speed (ML): 0.505287
[..]
   Partition 2: NADH5
   Speed (ML): 1.216009
```

What does it mean? In partitioned analyses, there are three common ways to estimate branch lengths (sometimes called *branch linkage* models):

- **linked**: all partitions share a common set of (global) branch lengths. This is the most simple model with the lowest number of parameters ($\#branches$). However, it is often considered too unrealistic, as it is known that genes (or genome regions) evolve at different speeds.
- **unlinked**: each partition has its own, independent set of branch lengths. This model allows for the highest flexibility, but it also introduces a huge number of free parameters ($\#branches * \#partitions$), which makes it prone to overfitting.
- **scaled (proportional)**: a global set of branch lengths is estimated as in 'linked' mode, but each partition has an individual scaling factor; per-partition branch lengths are obtained by multiplying the global branch lengths with these individual scalers. This approach represents a compromise that allows to model distinct evolutionary rates across partitions while, at the same time, only introducing a moderate number of free parameters ($\#branches + \#partitions$).

RAxML-NG supports all three branch linkage models described above; they can be selected using the `--brlen` option. A recent simulation study by Duchêne *et al.* [3] showed that the `scaled` branch linkage model offers the best fit for a large number of typical representative datasets. This confirms the intuition about its 'good' flexibility versus complexity trade-off. Hence, RAxML-NG uses the `scaled` branch linkage model for partitioned analyses by default. **Please note, that standard RAxML and ExaML use the linked branch length model by default. This should be kept in mind when comparing likelihoods and resulting topologies with those obtained via RAxML-NG!**

So let us now explore how the `linked` and `unlinked` models behave on our toy dataset:

```
$ raxml-ng --evaluate --msa prim.phy --threads 2 --model prim.part --tree
    T3.raxml.bestTree --prefix P3 --brlen linked

$ raxml-ng --evaluate --msa prim.phy --threads 2 --model prim.part --tree
    T3.raxml.bestTree --prefix P4 --brlen unlinked
```

As could be expected, more complex models yield better likelihood scores (`unlinked` > `scaled` > `linked`):

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
$ grep logLikelihood {P1,P3,P4}.raxml.log

P1.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5673.027260 <-
    scaled
P3.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5678.429054 <-
    linked
P4.raxml.log:[00:00:00] Tree #1, final logLikelihood: -5648.348677 <-
    unlinked
```

However, the induced likelihood score difference is not always large enough to justify using additional model parameters:

```
grep "AIC score" {P1,P3,P4}.raxml.log

P1.raxml.log:AIC score: 11446.054521 / AICc score: 11452.075772 / BIC
    score: 11686.063024  <- scaled
P3.raxml.log:AIC score: 11452.858107 / AICc score: 11458.398743 / BIC
    score: 11683.266270  <- linked
P4.raxml.log:AIC score: 11476.697354 / AICc score: 11496.994752 / BIC
    score: 11908.712661  <- unlinked
```

Once again, we observe a disagreement between the `AIC/AICc` and `BIC` criteria, which choose `scaled` and `linked` branch length models, respectively. However, all three criteria exclude the extremely parameter-rich `unlinked` model.

6.4 Tree searches with partitioned models

In the previous subsection, we used partitioned models to re-evaluate the likelihood of the ML tree obtained under the `GTR+G` model. But what if we re-run tree search from scratch under a partitioned model? Will this alter the resulting likelihoods and/or topologies?

```
$ raxml-ng --msa prim.phy --model prim.part --prefix P5 --threads 2 --seed
    2 --brlen scaled

$ raxml-ng --msa prim.phy --model prim.part --prefix P6 --threads 2 --seed
    2 --brlen linked

$ raxml-ng --msa prim.phy --model prim.part --prefix P7 --threads 2 --seed
    2 --brlen unlinked
```

Checking the new likelihood scores

```
$ grep "Final LogLikelihood" {P5,P6,P7}.raxml.log

P5.raxml.log:Final LogLikelihood: -5672.951995
P6.raxml.log:Final LogLikelihood: -5678.301081
P7.raxml.log:Final LogLikelihood: -5648.204296
```

shows that they are almost identical to the values obtained on the `T3` topology (see Section 6.2). Moreover, all three partitioned runs converged to the same ML tree topology as the unpartitioned `T3` run (see Section 3).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

Of course, this observation will not hold for all datasets. However, there is some evidence that the choice of the DNA substitution models has a rather limited influence on the resulting tree topology [4]. Of course, this result does not mean that model selection should not be conducted. However, it suggests that subtle details such as the conflicts between `AIC(c)` versus `BIC` or sample size definition are of minor concern in practice.

7 Parallelization and performance

7.1 Introduction

RAxML-NG supports three levels of parallelism: CPU instruction level (vectorization), intra-node (multithreading),and inter-node (MPI) parallelism. Unlike standard RAxML/ExaML, a single RAxML-NG executable offers *all* parallelism levels. The desired parallelism level can be configured at run-time (MPI support is optional and should be enabled at compile-time).

As of v.0.8.0*b*, RAxML-NG only supports *fine-grained* parallelization across MSA sites. This is the same parallelization approach that has been used in the PThreads version of standard RAxML and ExaML. It is conceptually different from the *coarse-grained* parallelizations across independent tree searches or tree moves as implemented in RAxML-MPI or IQTree-MPI [14], respectively. With fine-grained parallelization, the number of CPU cores that can be efficiently utilized is limited by the MSA "width" (=number of site patterns). For instance, using 20 cores on a single-gene protein alignment with 300 sites would be suboptimal, and using 100 cores would most probably result in a huge slowdown. In order to prevent wasting CPU time and energy, RAxML-NG will warn you – or, in extreme cases, even refuse to run – if you try to assign too few MSA sites to a core.

Coarse-grained parallelization, although not directly implemented in RAxML-NG, can be easily emulated as shown in Section 7.7.

7.2 Multithreading (pthreads)

By default, RAxML-NG will start as many threads as there are CPU cores available on your system. Most modern CPUs employ so-called *hyperthreading* technology, which makes each *physical* core appear as two *logical* cores to software. Hyperthreading can be beneficial for some programs, but **RAxML-NG achieves the best performance when run with one thread per physical core**. Therefore, RAxML-NG will try to detect if CPU supports hyperthreading, and will reduce the number of threads accordingly.

For instance, on a laptop with an Intel i7-8550U processor, RAxML-NG will detect 4 (physical) cores and use 4 threads by default:

```
  parallelization: PTHREADS (4 threads), thread pinning: OFF
```

even though this CPU has 8 logical cores:

```
$ lscpu -e

CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE MAXMHZ    MINMHZ
0   0    0      0    0:0:0:0       yes    4000,0000 400,0000
1   0    0      1    1:1:1:0       yes    4000,0000 400,0000
2   0    0      2    2:2:2:0       yes    4000,0000 400,0000
3   0    0      3    3:3:3:0       yes    4000,0000 400,0000
4   0    0      0    0:0:0:0       yes    4000,0000 400,0000
5   0    0      1    1:1:1:0       yes    4000,0000 400,0000
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
6   0    0      2    2:2:2:0       yes    4000,0000 400,0000
7   0    0      3    3:3:3:0       yes    4000,0000 400,0000
```

Unfortunately, it is very hard to reliably detect situations when hyperthreading is *supported* by the CPU, but *disabled* in BIOS. For instance, this setup can be found on Amazon AWS as well as on some clusters. In this situation, RAxML-NG can underestimate the number of available physical cores. Thus, it is recommended to use the `--threads` option and manually set the number of threads, to be on the safe side.

7.3 MPI and hybrid MPI/pthreads

If compiled with MPI support, RAxML-NG can leverage multiple compute nodes for a single analysis. Please check your cluster documentation for system-specific instructions on running MPI programs as this is different for every cluster. In MPI-only mode, you should start 1 MPI process *per physical CPU core* (the number of threads will be set to 1 by default).

However, in most cases, a hybrid MPI/pthreads setup will be more efficient in terms of both, runtime, and memory consumption. Typically, you would start 1 MPI rank *per compute node*, and 1 thread per physical core (e.g. `--threads 16` for nodes equipped with dual-slot octa-core CPUs). Here is a sample job submission script for the cluster at our research institute using 4 nodes × 16 cores:

```
#!/bin/bash
#SBATCH -N 4
#SBATCH -B 2:8:1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --hint=compute_bound
#SBATCH -t 08:00:00

raxml-ng-mpi --msa rbcl.phy --model GTR+G --prefix H1 --threads 16
```

Once again, please consult your cluster documentation to find out how to properly configure a hybrid MPI/pthreads run. **Please note that incorrect configuration can result in extreme slowdowns and hence waste time and resources!**

7.4 Thread pinning

For attaining optimal performance, it is crucial to ensure that only one RAxML-NG thread is running on each physical CPU core. Usually, the operating system can handle the thread-to-core assignment automatically. However, some (misconfigured) MPI runtimes tend to pack all threads onto a single CPU core, resulting in abysmal performance. To avoid this situation, each thread can be "pinned" (explicitely assigned to) to a particular CPU core.

In RAxML-NG, thread pinning is enabled by default only in the hybrid MPI/pthreads mode when 1 MPI rank per node is used. You can explicitly enable or disable thread pinning with `--extra thread-pin` and `--extra thread-nopin`, respectively.

7.5 Vector instructions

RAxML-NG will automatically detect the best (fastest) set of vector instructions available on your CPU, and use the respective computational kernels to achieve optimal performance. On modern Intel CPUs, this autodetection mechanism appears to work pretty well, so most

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

probably you will not need to worry about this. However, you can force RAxML-NG to use a specific set of vector instructions with the `--simd` option, for instance,

```
$ raxml-ng --msa prim.phy --model GTR+G --prefix V1 --threads 2 --seed 2
    --simd sse
```

to use SSE3 vector instructions, or

```
$ raxml-ng --msa prim.phy --model GTR+G --prefix V2 --threads 2 --seed 2
    --simd none
```

to use non-vectorized (scalar) instructions. This option might be useful for debugging, but otherwise using non-optimal vectorization should be avoided as it incurs a substantial performance penalty:

```
$ grep "Elapsed time:" {T3,V1,V2}.raxml.log

T3.raxml.log:Elapsed time: 7.802 seconds  <- AVX (autodetect)
V1.raxml.log:Elapsed time: 15.394 seconds <- SSE
V2.raxml.log:Elapsed time: 21.663 seconds <- scalar
```

7.6 Determining the optimal number of threads

One of the most frequent question we get from RAxML users is: *How many threads should I use?*. As always, simple questions are the toughest ones. You might as well ask: *How fast should I drive?*. In both cases, the answer would be: *It depends*. It depends on where you drive (dataset), your vehicle (system), and your priorities (time versus money/energy). In RAxML-NG, we have implemented some warning signs and radar speed guns, for your own safety. As in real life, you are free to ignore them (with the `--force` option), which can result in two things: (1) earlier arrival, or (2) lost time and money. Fortunately, unlike on the road, you can experiment with RAxML-NG safely, and we encourage you to do so.

A reasonable workflow for analyzing a large dataset would be as follows. First, run RAxML-NG in parser mode, that is,

```
$ raxml-ng --parse --msa rbcl.phy --model GTR+G+F --prefix rbcl
```

This command will generate a binary MSA file (`rbcl.raxml.rba`), which can subsequently be loaded by RAxML-NG much faster than the original FASTA alignment. Furthermore, it will print the estimated memory requirements and the recommended number of threads for this dataset:

```
[00:00:00] Reading alignment from file: rbcl.phy
[00:00:00] Loaded alignment with 436 taxa and 1371 sites

Alignment comprises 1 partitions and 1001 patterns

NOTE: Binary MSA file created: rbcl.raxml.rba

* Estimated memory requirements                : 54 MB
* Recommended number of threads / MPI processes: 4
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1

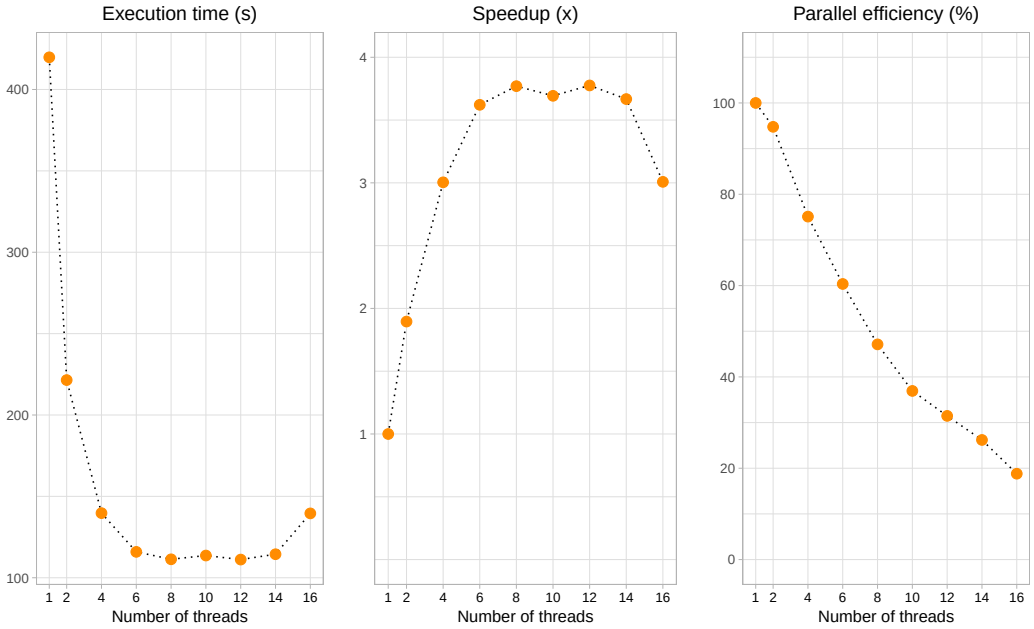

Figure 1 Typical scaling of RAxML-NG on a small alignment (here: 436 taxa and 1,371 DNA sites).

The recommended number of threads is computed using a simple heuristic and should yield a decent runtime/resource utilization trade-off in most cases. It also constitutes a good starting point for your experiments: you can now run tree searches with a varying number of threads, for instance,

```
$ raxml-ng --search1 --msa rbcl.raxml.rba --seed 1 --threads 2
$ raxml-ng --search1 --msa rbcl.raxml.rba --seed 1 --threads 4
$ raxml-ng --search1 --msa rbcl.raxml.rba --seed 1 --threads 8
```

and so on. For a small example dataset as in our example, the execution time will decrease initially as we add more threads, but will then quickly level off (leftmost plot below). Although the maximum speedup of $\approx 3.75\times$ can be attained with $8-14$ threads (middle plot), it induces a rather poor parallel efficiency of 60%-30% (right plot). The recommended number of threads (4) yields a reasonable speedup ($3\times$) without compromising the parallel efficiency too much (75%). Finally, if we use an excessively large number of cores (≥ 16 in this example), execution times will start to *increase* again. Although the actual speedups will vary across datasets and systems, the general trend will stay the same. Therefore, it is up to the user to decide how many resources (=higher CPU time) can be sacrificed to obtain the results faster (=lower execution time).

7.7 Coarse-grained parallelization for short alignments

If you want to utilize a large number of CPU cores for analyzing a small ("single-gene") alignment, please have a look at our ParGenes pipeline [13] which implements coarse-grained parallelization and dynamic load balancing. ParGenes is freely available at `https://github.com/BenoitMorel/ParGenes`.

Alternatively, coarse-grain parallelization can easily be emulated by executing multiple

RAxML-NG instances, but with distinct random seeds. For instance, let us assume that we want to run an "all-in-one" analysis on the dataset described in Section 7.6, and we want to use a server with 16 CPU cores. As Figure 1 shows, the fine-grained parallelization across 16 cores is very inefficient for this dataset. We will therefore use fine-grained parallelization with 2 cores per tree search, which means we can run $16/2 = 8$ RAxML-NG instances in parallel. First, we will infer 24 ML trees, using 12 random and 12 parsimony-based starting trees. Hence, each RAxM-NG instance will run searches from $24/8 = 3$ starting trees. Below is a sample `SLURM` script for doing this:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 8
#SBATCH -B 2:8:1
#SBATCH --threads-per-core=1
#SBATCH --cpus-per-task=2
#SBATCH -t 02:00:00

for i in `seq 1 4`;
do
  srun -N 1 -n 1 --exclusive raxml-ng --search --msa rbcl.raxml.rba --tree
     pars{3}    --prefix CT$i --seed $RANDOM --threads 2 &
done

for i in `seq 5 8`;
do
  srun -N 1 -n 1 --exclusive raxml-ng --search --msa rbcl.raxml.rba --tree
     rand{3}   --prefix CT$i --seed $RANDOM --threads 2 &
done

wait
```

Of course, this script has to be adapted for your specific cluster configuration and/or job submission system. You can also use `GNU parallel`, or directly start multiple RAxML-NG instances from the command line. Please pay attention to the ampersand symbol (`&`) at the end of each RAxML-NG command line: it is extremely important here, since if you forget the ampersand all RAxML-NG instances will run one after another and *not* in parallel! Furthermore, we add `--exclusive` flag to tell ensure that 'raxml-ng' instances will be assigned to *distinct* CPU cores (this is default behavior with some SLURM configurations, but not always).

Once the job has finished, we can inspect the likelihoods:

```
$ grep "Final LogLikelihood" CT*.raxml.log | sort -k 3

CT7.raxml.log:Final LogLikelihood: -30621.004116
CT6.raxml.log:Final LogLikelihood: -30621.537107
CT2.raxml.log:Final LogLikelihood: -30621.699234
CT3.raxml.log:Final LogLikelihood: -30622.534482
CT1.raxml.log:Final LogLikelihood: -30622.783250
CT8.raxml.log:Final LogLikelihood: -30623.963471
CT5.raxml.log:Final LogLikelihood: -30623.020351
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0056.v1


```
CT4.raxml.log:Final LogLikelihood: -30623.378857
```

and select the best-scoring tree (`CT7.raxml.bestTree` in our case):

```
$ ln -s CT7.raxml.bestTree best.tre
```

The same trick can be applied to bootstrapping. For the sake of simplicity, let us infer $8 \times 15 = 120$ replicate trees:

```
for i in ‘seq 1 8‘;
do
  raxml-ng --bootstrap --msa rbcl.raxml.rba --bs-trees 15 --prefix CB$i --
    seed $RANDOM --threads 2 &
done

wait
```

Now, we can simply concatenate all replicate tree files (`*.raxml.bootstraps`) and then proceed with the bootstrap convergence check as well as branch support calculation as usual (see Section 4):

```
$ cat CB*.raxml.bootstraps > allbootstraps

$ raxml-ng --bsconverge --bs-trees allbootstraps --prefix CS --seed 2 --
    threads 1

$ raxml-ng --support --tree best.tre --bs-trees allbootstraps --prefix CS
    --threads 1
```

There are two things to keep in mind when conducting this type of coarse-grained parallelization. First, memory consumption will grow proportionally to the number of RAxML-NG instances running in parallel. That is, in our case, an estimate given by the `--parse` command should be multiplied by 8. Second, correct thread allocation (1 thread per CPU core) is crucial for achieving the optimal performance. Hence, we recommend to check thread allocation, for instance, by running `htop` after your initial script submission.

A Summary of changes compared to RAxML 8.x

RAxML-NG offers multiple improvements and extensions compared to standard RAxML 8.x. On the other hand, not all features of standard RAxML are implemented as of RAxML-NG v0.8.0b (most notably, rapid bootstrapping and CAT/PSR model [19]). Furthermore, several important defaults have been changed to be in line with best practices: for instance, RAxML-NG is using multiple starting trees and scaled branch lengths by default. To give a better overview, we provide a side-by-side comparison of standard RAxML 8.x and RAxML-NG 0.8.0b in Table 1.

Option/Feature	RAxML 8.x	RAxML-NG 0.8.0b
Features		
Bootstrapping	standard, rapid	standard
Parallelization scheme	fine-grained (PTHREADS) coarse-grained (MPI)	fine-grained (PTHREADS and MPI)
Checkpointing	NO	YES
Binary MSA	NO	YES
Evolutionary models		
Rate heterogeneity across sites (RHAS)	GAMMA, p-inv, CAT	GAMMA, p-inv, FreeRate
RHAS linkage	global	per-partition
Branch length linkage	linked, unlinked	linked, unlinked, scaled
LG4X with linked branches	YES	NO
User-specified parameter values	NO	YES
Defaults		
Starting tree(s)	parsimony (1)	parsimony(10)+random(10)
Stationary state frequencies	empirical	ML estimate
Branch lengths linkage	linked	scaled
Number of boostrap replicates	100	AUTO (bootstopping)

Table 1 Differences in features and default settings between standard RAxML and RAxML-NG v0.8.0b

Acknowledgements

This work was funded by the Klaus Tschira Foundation. The authors wish to thank the following former students of our 2018 summer school on computational molecular evolution for useful comments on the initial draft of this book chapter: Sunitha Manjari and Loïc Meunier.

References

1 Lucas Czech, Jaime Huerta-Cepas, and Alexandros Stamatakis. A critical review on the use of support values in tree viewers and bioinformatics toolkits. *Molecular Biology and Evolution*, 34(6):1535–1542, 2017. URL: http://dx.doi.org/10.1093/molbev/msx055, doi:10.1093/molbev/msx055.

2 Diego Darriba. ModelTest-NG: Best-fit evolutionary model selection. https://github.com/ddarriba/modeltest, 2018. Website. Accessed December 03, 2018.

3 David A Duchene, K. Jun Tong, Charles S.P. Foster, Sebastian Duchene, Robert Lanfear, and Simon Y.W. Ho. Linking branch lengths across loci provides the best fit for phylogenetic inference. *bioRxiv*, 2018. URL: https://www.biorxiv.org/content/early/2018/11/09/467449, arXiv:https://www.biorxiv.org/content/early/2018/11/09/467449.full.pdf, doi:10.1101/467449.

4 Michael Hoff, Stefan Orf, Benedikt Riehm, Diego Darriba, and Alexandros Stamatakis. Does the choice of nucleotide substitution models matter topologically? *BMC Bioinformatics*, 17(1):143, Mar 2016. URL: https://doi.org/10.1186/s12859-016-0985-x, doi:10.1186/s12859-016-0985-x.

5 Erich D Jarvis, Siavash Mirarab, Andre J Aberer, Bo Li, Peter Houde, Cai Li, Simon YW Ho, Brant C Faircloth, Benoit Nabholz, Jason T Howard, et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346(6215):1320–1331, 2014.

6 Subha Kalyaanamoorthy, Bui Quang Minh, Thomas KF Wong, Arndt von Haeseler, and Lars S Jermiin. Modelfinder: fast model selection for accurate phylogenetic estimates. *Nature methods*, 14(6):587, 2017.

7 Alexey Kozlov, Diego Darriba, Tomas Flouri, Benoit Morel, and Alexandros Stamatakis. Raxml-ng: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference. *bioRxiv*, 2018. URL: https://www.biorxiv.org/content/early/2018/10/18/447110, doi:10.1101/447110.

8 Alexey M. Kozlov, Andre J. Aberer, and Alexandros Stamatakis. ExaML version 3: a tool for phylogenomic analyses on supercomputers. *Bioinformatics*, 31(15):2577–2579, 2015. URL: http://dx.doi.org/10.1093/bioinformatics/btv184, doi:10.1093/bioinformatics/btv184.

9 Oleksii Kozlov. *Models, Optimizations, and Tools for Large-Scale Phylogenetic Inference, Handling Sequence Uncertainty, and Taxonomic Validation*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2018. doi:10.5445/IR/1000081661.

10 Robert Lanfear, Paul B Frandsen, April M Wright, Tereza Senfeld, and Brett Calcott. Partitionfinder 2: new methods for selecting partitioned models of evolution for molecular and morphological phylogenetic analyses. *Molecular Biology and Evolution*, 34(3):772–773, 2016.

11 F. Lemoine, J. B. Domelevo Entfellner, E. Wilkinson, D. Correia, M. Dávila Felipe, T. De Oliveira, and O. Gascuel. Renewing Felsenstein's phylogenetic bootstrap in the era of big data. *Nature*, 556(7702):452–456, April 2018. URL: http://dx.doi.org/10.1038/s41586-018-0043-0, doi:10.1038/s41586-018-0043-0.

12 Bernhard Misof, Shanlin Liu, Karen Meusemann, Ralph S Peters, Alexander Donath, Christoph Mayer, Paul B Frandsen, Jessica Ware, Tomáš Flouri, Rolf G Beutel, et al. Phylogenomics resolves the timing and pattern of insect evolution. *Science*, 346(6210):763–767, 2014.

13 Benoit Morel, Alexey M Kozlov, and Alexandros Stamatakis. Pargenes: a tool for massively parallel model selection and phylogenetic tree inference on thousands of genes. *Bioinformatics*, page bty839, 2018. URL: http://dx.doi.org/10.1093/bioinformatics/bty839, doi:10.1093/bioinformatics/bty839.

14 Lam-Tung Nguyen et al. IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, 32(1):268–274, 2015. URL: http://mbe.oxfordjournals.org/content/32/1/268.abstract, doi:10.1093/molbev/msu300.

15 Nicholas D. Pattengale, Masoud Alipour, Olaf R.P. Bininda-Emonds, Bernard M.E. Moret, and Alexandros Stamatakis. How many bootstrap replicates are necessary? *Journal of Computational Biology*, 17(3):337–354, 2010. PMID: 20377449. URL: https://doi.org/10.1089/cmb.2009.0179, arXiv:https://doi.org/10.1089/cmb.2009.0179, doi:10.1089/cmb.2009.0179.

16 David Posada and Thomas R. Buckley. Model selection and model averaging in phylogenetics: Advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. *Systematic Biology*, 53(5):793–808, 2004. URL: http://dx.doi.org/10.1080/10635150490522304, doi:10.1080/10635150490522304.

17 D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131 – 147, 1981. URL: http://www.sciencedirect.com/science/article/pii/0025556481900432, doi:https://doi.org/10.1016/0025-5564(81)90043-2.

18 Hidetoshi Shimodaira and Masami Hasegawa. Consel: for assessing the confidence of phylogenetic tree selection. *Bioinformatics*, 17(12):1246–1247, 2001.

19 A. Stamatakis. Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective. In *Proc. of IPDPS2006*, HICOMB Workshop, Proceedings on CD, Rhodos, Greece, April 2006.

20 A. Stamatakis and A.J. Aberer. Novel parallelization schemes for large-scale likelihood-based phylogenetic inference. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1195–1204, 2013. doi:10.1109/IPDPS.2013.70.

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

22 Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

23 Z Yang. A space-time process model for the evolution of DNA sequences. *Genetics*, 139(2):993–1005, 1995. URL: http://www.genetics.org/content/139/2/993.