BIOINFORMATICS

Vol. 00 no. 00 2005
Pages 1–7

ON-LINE SUPPLEMENT: RAxML-VI-HPC: Maximum Likelihood-based Phylogenetic Analyses with Thousands of Taxa and Mixed Models

Alexandros Stamatakis [a*]

[a] Swiss Federal Institute of Technology Lausanne, School of Computer & Communication Sciences, Lab Prof. Moret, STATION 14, CH-1015 Lausanne, Switzerland

ABSTRACT

Summary: This on-line supplement provides a detailed description of the technical optimizations, the experimental setup and the results.

Availability: diwww.epfl.ch/~stamatak

Contact: Alexandros.Stamatakis@epfl.ch

1 OVERVIEW OF RAXML

RAxML is a program that incorporates several efficient tree search algorithms. The Lazy Subtree Rearrangement (LSR) technique for fast hill-climbing searches has been introduced in RAxML-III (Stamatakis *et al.*, 2005*a*) and the simulated annealing algorithm has been introduced with RAxML-V (Stamatakis, 2005).

RAxML-V has been parallelized with OpenMP (www.openmp.org) to exploit the intrinsic loop-level parallelism of the ML function (Stamatakis *et al.*, 2005*b*). The OpenMP version, which has been further improved in RAxML-VI-HPC scales well on long multi-gene alignments, e.g. an 2,182 taxon mammalian alignment with 51,089 base pairs. Figure 1 indicates the parallel performance improvement on 1 versus 8 CPUs on one node of the CIPRES cluster (see Section 4) during the first three iterations of the search algorithm (speedup: 6.74). Please note that—apart from this Figure—in the current paper only the strictly sequential version of RAxML has been used. Based on the exploitation of the same type of parallelism, the program has recently also been ported to a Graphics Processing Unit (Charalambous *et al.*, 2005).

RAxML-VI-HPC is the new version of RAxML which is available at diwww.epfl.ch/~stamatak including a comprehensive manual. RAxML-VI-HPC has been explicitly designed to conduct large-scale real-world biological analyses under the GTR (General Time Reversible (Rodriguez *et al.*, 1990)) model of nucleotide substitution with rate heterogeneity. An important feature of RAxML is the implementation of the GTR+CAT (GTR with per-site rate categories) approximation (Stamatakis, 2006) as a work-around for GTR+Γ (Γ model of rate heterogeneity, (Yang, 1994, 1996)). The implementation of GTR+CAT in RAxML-VI-HPC is important within the scope of this paper because it leads to a significantly lower memory footprint in comparison to other programs.

The GTR+Γ model is typically implemented using 4 discrete rate categories (hard-coded in GARLI, default in PHYML/IQPNNI), which represents a reasonable trade-off between accuracy and efficiency. The huge disadvantage of such a typical implementation of Γ in comparison to CAT is that approximately the quadruple amount of floating point operations *and* memory is required to compute the likelihood function. Despite the obvious computational disadvantages, Γ is usually preferred over CAT due to its better statistical properties.

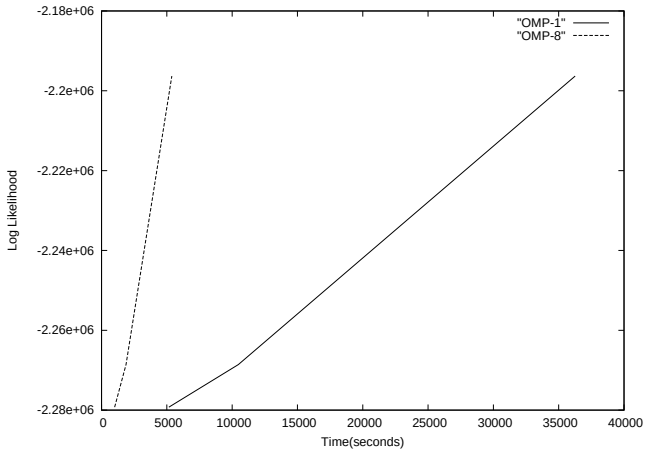

Fig. 1. Run time improvement for the first three iterations of the search algorithm of the OpenMP version of RAxML-VI-HPC on 1 and 8 CPUs on a 51,089 bp long multi-gene alignment of 2,182 mammals

A recent empirical study (Stamatakis, 2006) (preprint available at diwww.epfl.ch/~stamatak) on 19 real-world biological datasets comprising 73 up to 1,663 DNA sequences indicates that CAT can be used as a vehicle to circumvent the complexity of the Γ model under *exactly the same search algorithm*. This study also shows that RAxML under CAT outperforms IQPNNI and PHYML in nearly all cases under Γ. Especially, on large datasets, where memory consumption becomes a limiting factor, RAxML with GTR+CAT performs better in all cases. The results in the current paper lead to the same conclusion. To the best of the author's knowledge, there is currently no other program which implements CAT in a similar way as RAxML to avoid the computational complexity of Γ.

Finally, it is important to note that the final RAxML likelihood values can not be directly compared to those of other programs. This is due to the fact that ML programs differ in their numerical implementations. Note e.g. that IQPNNI and PHYML also yield slightly different likelihood values for the same tree. For example, very small partial likelihood values might be scaled in different ways to avoid numerical underflow. Thus, in order to compare

*to whom correspondence should be addressed

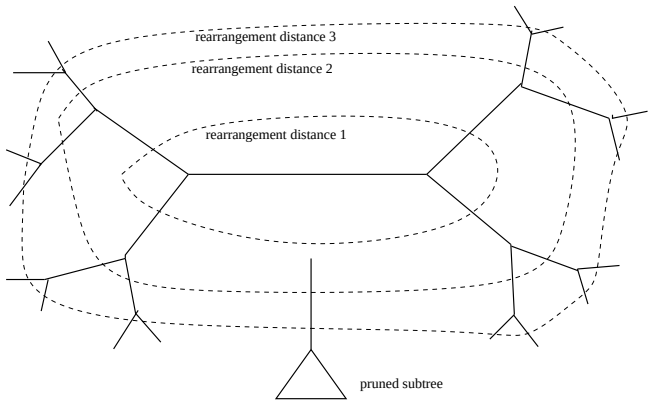

Fig. 2. Rearrangement distances of 1, 2, and 3 for a pruned subtree at the central branch

final trees obtained by different ML programs one has to optimize the branch lengths and the model parameters of final tree topologies with one and the same program.

1.1 Basic Hill-Climbing Algorithm

A high amount of programming effort has been invested to improve performance of RAxML. However, the most important observation from the optimization of RAxML, which is also applicable to other ML programs, is that frequently storing topologies and particularly branch lengths for several thousands of taxa can be become the operation that *dominates* run-times.

Initially, the basic steps of the hill-climbing algorithm will be outlined since they are important for the optimizations introduced here.

As already mentioned the basic technique used in RAxML-VI-HPC is called Lazy Subtree Rearrangement (LSR). One LSR cycle consists in removing (pruning) a subtree `s` from the currently best tree t_{best} and re-inserting it into all neighboring branches up to a distance of `m` nodes (rearrangement distance) from the pruning position. Figure 2 outlines the rearrangement distances of 1, 2, and 3 nodes for a pruned subtree at the central branch.

The LSR mechanism is called lazy because only the three branch-lengths adjacent to the insertion point are optimized (for details see Stamatakis *et al.* (2005*a*)), such that the new tree topology is only being pre-scored. However, if one of these pre-scored topologies has a better likelihood score than t_{best} it is stored in an appropriate topology list `bestLSR` with one entry and reconstructed after the LSR for the specific subtree has been completed. In addition, the 20 best pre-scored tree topologies encountered during one iteration of the hill climbing algorithm are stored in a separate list (`bestList20`).

One iteration of the hill-climbing algorithm consists in applying a sequence of $2n$ LSR cycles, where n is the number of taxa, to the currently best topology t_{best}. If the likelihood of t_{best} is improved by the i-th LSR cycle LSR$_i$ $i = 1, ..., 2n$ the changed topology is kept, i.e. $t_{best} := t_i$. Thus, one iteration of the sequential algorithm (one iteration of $2n$ LSR cycles) generates a sequence of $k \leq 2n$ distinct topologies with increasing likelihood values $t_{i_1} \rightarrow t_{i_2} \rightarrow ... \rightarrow t_{i_k}$. After completion of $2n$ LSR cycles the topologies contained in `bestList20` are restored and all their branches are thoroughly optimized.

In the hill-climbing version of RAxML the process of executing $2n$ LSRs and exhaustive optimization of the 20 trees in `bestList20` is continued until no better tree is found.

The steps performed during one LSR cycle, given the subtree `s`, the rearrangement distance `m`, the likelihood of the currently best tree `bestLikelihood`, and the two tree lists `bestLSR` and `bestList20` are outlined in the pseudo-code below.

```
1. Remove subtree s from the tree
2. Optimize the branch where s has been pruned
3. for all branches b within a distance of
       m nodes from the pruning point do
   3.1 Save all branch lengths
   3.2 Insert s into b to construct newTree
   3.3 Optimize the three branches adjacent to
       the insertion point
   3.4 Calculate likelihood value
       newLikelihood := likelihood(newTree)
   3.5 If newLikelihood > bestLikelihood
           store newTree in bestLSR
           bestLikelihood := newLikelihood
   3.6 Store newTree in bestList20 if applicable
   3.7 Restore all branch lengths
```

2 OPTIMIZATIONS OF RAXML

Sections 2.1 and 2.2 describe how a large amount of expensive branch length and topology storage operations can be removed, by taking into account that LSRs only change the topology and branch lengths locally. Section 2.3 describes how the number of re-computed likelihood vectors can be reduced and Section 2.4 describes the determination of "good" rearrangement settings. Finally, Sections 2.5, 2.6, and 2.7 cover the improvement of the starting tree component, the parallelization, and other minor technical improvements.

2.1 Storing Branch Lengths

As indicated in the pseudocode provided in the previous Section, the original implementation of RAxML stores topologies in both data structures `treeList20` and `bestLSR` after *each* insertion of a subtree into an alternative branch, i.e. after each single LSR during an LSR cycle.

If one considers the worst case (assuming that all moves of one LSR cycle improve the tree topology and no LSR encounters a leaf) for a rearrangement distance of 5, this would result in storing every new tree topology twice: once in `bestList20` and once in `bestLSR`. The total number of topologies that have to be saved during one LSR cycle would thus be $2 * (2^2 + 2^3 + 2^4 + 2^5 + 2^6) = 248$.

Storing a topology can become a very expensive operation, since every time $2n - 3$ branches and $2n - 2$ nodes have to be copied. In the worst case, 495,256 floating point values will have to be copied during one LSR cycle, for a 1,000 taxon alignment. Moreover, the same amount of storage operations (495,256) has to be performed to store and restore *all branch lengths*, such that in total 990,512 values have to be stored during an LSR cycle.

The fact that only 3 branches are altered per subtree insertion (single LSR) and 1 when the subtree is pruned at the beginning of each LSR cycle has not been consequently exploited in the old RAxML

implementation. Before and after each insertion of a specific subtree into an alternative branch the lengths of *all* branches were saved and restored.

As already mentioned restoring branch lengths can become very expensive with growing tree size, given that an array of $2n - 3$ elements has to be copied each time a subtree is inserted into an alternative branch of the tree. Since the maximum rearrangement distance is by default ≥ 5 in RAxML (Stamatakis *et al.*, 2005*a*), a large number of inserts is performed per subtree.

However, for a single LSR it suffices to store the original three branches before the subtree insertion and restore them when the subtree is removed again. In addition, the branch at the original pruning point is optimized at the beginning of each LSR cycle (Step 2 in the pseudocode), such that a total of 4 branches has to be stored. Therefore, Steps 3.1/3.7 can be replaced by `save/restore four branches` in the pseudocode of Section 1.1.

2.2 Storing Topologies

A closely-related optimization concerns the way in which improved tree topologies are stored and re-stored during the LSR process.

As outlined above, in the old version this involved frequently storing and restoring all branches and a tree structure with $2n - 2$ nodes. In the original version an improved tree was immediately stored after insertion of the subtree into an alternative branch (Step 3.5). However, in order to restore an improved tree immediately after completion of an LSR cycle, it is sufficient to store the three branch lengths at the insertion point, the pruning branch length, the position of the insertion branch `b`, and the root node of the subtree `s` in an appropriate data structure. The respective data structure will henceforth be called rearrangement descriptor.

This rearrangement descriptor has a constant number of only six parameters, instead of $\approx 4n$ parameters (branches and nodes). Thus, restoring a pre-scored topology after the completion of an LSR cycle can be rapidly performed by using the rearrangement descriptor. In addition, a large number of likelihood vectors (also called partial likelihood arrays) can be re-used and does not need to be re-computed. In contrast to this, in the original version of RAxML *all* likelihood vectors were updated bottom-up when a tree was restored after completion of an LSR cycle. Thus, the operation of Step 3.5 `store newTree in bestLSR` can be replaced by `store rearrangement descriptor`.

However, in order to maintain `bestList20`, some topologies still have to be saved and restored including their branch lengths and entire tree structure. To further accelerate the program, trees are now stored only *after completion and not during* an LSR cycle. The best topology obtained during an entire LSR cycle (irrespective of `bestLikelihood`) is also restored using a rearrangement descriptor and stored in `bestList20` such that Step 3.6 can be removed. This means that instead of potentially storing a tree at each single LSR in `bestList20` (worst case) it is now stored only after completion of the LSR cycle for one subtree.

Note that this change leads to a slight alteration of the hill-climbing path since in the previous version it would be possible that more than one topology in `bestList20` originates from the same LSR cycle whereas in RAxML-VI-HPC this is not the case.

2.3 Restoring Likelihood Vectors

Another relatively time-consuming and unnecessary operation in the previous release, concerns the initial traversal of the tree after local changes in the topology (not shown in the pseudocode).

RAxML uses a rooted view of the tree with respect to likelihood vector computations, i.e. only one likelihood vector is allocated to each inner node. As a consequence, if the tree is initially rooted at a branch p and then re-rooted at q, only those likelihood vectors along the path $p \rightarrow q$ have to be recomputed.

In the previous version a complete initial traversal was performed after completion of each LSR cycle. However, an LSR cycle only changes likelihood vectors located at nodes within the rearrangement distance `m` from the pruning point (see Figure 2). The branches and likelihood vectors outside this region remain unaffected by the LSR cycle. Thus, the global initial traversal can be replaced by a regional initial traversal around the pruning point, which significantly reduces the number of recomputed likelihood vectors.

This is a well-known technique, which is also used e.g. by Hordijk and Gascuel (2005) and had not been implemented for that specific part of the computations in RAxML.

2.4 Good Rearrangement Distance

An issue which becomes important for huge trees is to determine a "good" rearrangement distance or adequate dataset-dependent search parameters. This is also pointed out by Hordijk and Gascuel (2005).

In RAxML-VI-HPC the algorithm initially determines the best rearrangement distance by applying distances of 5, 10,..., 25 for one iteration of LSR cycles, to the original MP starting tree. The lowest rearrangement distance which yields the best likelihood improvement is then selected for the inference process.

Despite the additional computations which are performed and the simplicity of the current method, a "good" rearrangement distance leads to significant likelihood and performance improvements on alignments with a large evolutionary diameter (e.g. the 6,722 and 7,769 taxa alignments in Table 2).

However, in one case (an alignment of 1,924 closely-related taxa), the determination of the rearrangement setting drove the program into a local optimum, and the trees obtained were by 200 Log Likelihood units inferior to those obtained via the default search parameters.

A better criterion to determine an appropriate rearrangement setting based on the input data will be subject of future work. Nonetheless, the possibility to experiment with the search parameter settings is now provided by the significant speedup obtained via the technical optimizations.

2.5 Improved Starting Trees

Another significant improvement consists in the complete re-implementation of the random stepwise addition sequence starting tree computations which use the Maximum Parsimony (MP) criterion (Starting trees obtained by this procedure are henceforth called randomized MP starting trees).

The previous implementation was based on the dnapars component from Felsenstein's PHYLIP package. In RAxML-VI-HPC the MP computations have been implemented directly in RAxML in a significantly more efficient manner.

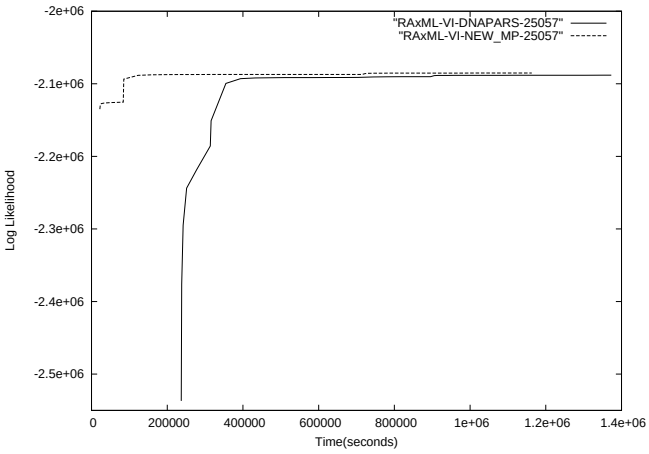

Fig. 3. RAxML GTR+CAT Likelihood development over Time with the dnapars and proprietary MP starting tree implementation

This re-implementation yields accelerations of starting tree computations which scale up with alignment size. Speedups range from 2.1 on 1,000 taxa and up to 10.8 on 25,000 taxa.

In addition, dnapars yielded "bad" MP starting trees on some datasets $\geq$ 2,000 taxa, due to an unidentified bug. The effect of this error in dnapars was particularly severe on the 25,057 taxon alignment of protobacteria. The performance improvement and qualitative improvement due to the re-implementation is outlined in Figure 3. This graph indicates the GTR+CAT Likelihood development over time (including the time required to compute the starting tree) for this large dataset on a dnapars starting tree and a proprietary RAxML starting tree.

Note however, that RAxML is still able to reach "good" likelihood values starting from the significantly worse starting tree. This ability to reach a good final likelihood value from a significantly worse starting tree is mainly due to the mechanism for detecting "good" rearrangement settings described in Section 2.4.

2.6 Parallelization

There already exists the coarse-grained parallel implementations of the RAxML search algorithm (Stamatakis *et al.*, 2004).

However, this parallelization focused on computing one single large tree in parallel. Nonetheless, for a real-world Biological analysis, a large number of multiple bootstraps and inferences on the original alignment using distinct starting trees is required. Therefore, the most reasonable and most useful approach for Biologists to exploit a supercomputer or even Grid of supercomputers consists in exploiting this type of embarrassing parallelism with MPI. If required, RAxML can also be compiled to take advantage of hybrid parallelism with MPI/OpenMP on hybrid parallel machines.

In contrast to previous versions of RAxML, the optimization of per-site evolutionary rates for the GTR+CAT approximation was also parallelized with OpenMP.

2.7 Minor Technical Improvements

The minor technical improvements cover a slight reduction of the memory footprint by approximately 5% with respect to RAxML-V by appropriate changes in the data structures.

Finally, the speed of the likelihood functions for GTR+CAT and GTR+Γ could be improved by approximately 20-30% in comparison to RAxML-V via low-level optimizations of the source code and increased re-use of already computed values.

3 EXPERIMENTAL SETUP

All tests were executed on the Infiniband-cluster at the Technische Universität München, equipped with 36 2.4GHz Quad-Opteron nodes and 8GB of main memory per node and on the CIPRES (Cyberinfrastructure for Phylogenetic Research, www.phylo.org) cluster at the San Diego Supercomputing Center equipped with 16 8-way 2.4GHz Opteron nodes with 16GB of main memory per node.

The most recent versions of IQPNNI (v3.0), MrBayes (v3.1), GARLI (v0.94) and PHYML (v2.4.4) have been used. The source code of all programs was compiled using `gcc 3.3.3`.

Note that PHYML with SPR (Subtree Pruning & Re-Grafting) has not been included into this performance study because the respective software represents a proof-of-concept implementation of the algorithmic ideas presented by Hordijk and Gascuel (2005). In the currently available implementation the scaling of very small likelihood values, which is required to avoid numerical underflow in particular on datasets $\geq$ 500 taxa has been removed, which causes the program to crash due to numerical errors.

In order to study program performance, 10 previously assembled datasets comprising between 1,000 and 8,780 taxa have been used. For the sake of completeness the alignment lengths (# bp) and the number of distinct patterns/columns in each alignment are indicated in Table 1.

The datasets are described and available for public download (if permission has been granted by the authors) at diwww.epfl.ch/~stamatak (material frame). For each dataset 5 randomized MP starting trees have been computed with RAxML-VI-HPC.

Table 1. Alignment lengths in bp and number of distinct patterns/columns

# Taxa	# bp	# patterns	# Taxa	# bp	# patterns
1000	5547	3364	1497	1241	1241
1663	1577	1576	1728	1276	1275
2000	1251	1251	2560	1232	1232
4114	1263	1263	6722	1122	1122
7769	851	851	8780	1217	1217

3.1 Program Settings

In order to underline the fact that the ability to obtain a set of "good" distinct starting trees with MP is a feature of RAxML, IQPNNI and PHYML have been executed only once per dataset starting from their respective neighbor joining starting trees.

MrBayes, GARLI, and RAxML were executed once per dataset/starting tree combination (5 runs per alignment). In addition, the analyses of MrBayes, IQPNNI, and PHYML have been restricted to the 5 smaller datasets, since they showed to be less efficient than

RAxML and GARLI on large datasets. Moreover PHYML and IQPNNI encountered technical problems, such as numerical underflow and memory shortage on the larger datasets.

It might appear, that this setup does not fully account for the stochastic nature of GARLI (genetic algorithm) and MrBayes (MC^3-based Bayesian inference). However, GARLI and MrBayes were invoked with 5 distinct randomized MP starting trees per dataset such that the "randomness" in their searches can be accommodated. In addition, the "good" starting trees from RAxML provided GARLI and MrBayes a substantial head-start over complete random starting trees. The manuals of GARLI and MrBayes mention that "good" starting trees should be used (GARLI) or could help to improve performance (MrBayes) on large datasets.

The GTR+Γ model of evolution was used on all datasets with all programs except RAxML. As already mentioned, RAxML was executed using the GTR+CAT (`-m GTRCAT`) model of nucleotide substitution with 25 individual rate categories instead of GTR+Γ. The rationale for selecting GTR+CAT lies in the increased speed and the reduced memory footprint. In addition, the results of the current study confirm the results of the larger study on Γ and CAT (Stamatakis, 2006). The number of 25 rate categories is the default value of RAxML which empirically works well in most cases.

All model parameters were estimated by the respective programs. The execution time was limited to a maximum of 60 hours (a weekend) for the studies on the smaller datasets (1,000-2,000 taxa) since this represents an acceptable and reasonable amount of time to run large-scale analyses.

Finally, the branch lengths and model parameters of all final tree topologies obtained by the programs were optimized under GTR+Γ using the respective program option in RAxML, because this function is significantly faster than the respective option in PHYML and a large amount of trees had to be evaluated. In addition, only RAxML was able to evaluate the larger datasets inferred with GARLI and RAxML.

Thus, all likelihood values reported in this paper are RAxML-based Likelihood values! **This step is very important since the likelihood values obtained by different phylogeny programs can not directly be compared due to numerical differences in the implementations of the likelihood functions among almost all programs. All Log Likelihood values presented in all Tables and Figures are RAxML Log Likelihood values!**

The randomized MP starting trees and final RAxML result trees are available at wwwbode.in.tum.de/~stamatak/Study.tar.gz.

The following paragraphs describe *exactly* how the programs were executed.

RAxML-VI-HPC Settings: RAxML-VI-HPC was executed with the rapid hill climbing algorithm under the GTR+CAT approximation (-m GTRCAT) using 25 distinct (-c 25) rate categories.

MrBayes Settings: MrBayes was executed once on each starting tree/dataset combination.

In order to reduce the memory consumption and accelerate the program MrBayes was executed using only one independent analysis (instead of the default of two) with three heated and one cold chain under the GTR+Γ model of nucleotide substitution. Note, that this might have a negative impact on the final likelihood value obtained by MrBayes. On the other hand, this reduced the memory consumption by a factor of 2 and accelerated the inference process at the same time.

For each run two likelihood values were determined: One based on the last single sampled tree and one based on the majority-rule consensus tree containing the last 5% of all sampled trees which better accounts for the distinct nature of the Bayesian approach.

IQPNNI Settings: IQPNNI was run with the GTR+Γ model of substitution with 4 discrete rate categories, the stopping rule switched off (-s off) and a a high number of iterations (-n 100000000) to ensure that the program would execute for 60 hours. In fact, some improved topologies were still encountered during the late stages of the search. The default neighbor joining starting tree was used for the searches.

Note that, the CAT model as implemented in IQPNNI was not used. This is due to the fact that it differs in various ways from the implementation in RAxML. When the CAT model in IQPNNI is specified the actual search is conducted using the corresponding plain model without rate heterogeneity and only the intermediate results between iterations are evaluated using the CAT model (L.S. Vinh, personal communication). In addition, IQPNNI does not perform a rate categorization step (Stamatakis, 2006) but determines individual rates for each site.

PHYML Settings: PHYML was run under GTR+Γ with 4 discrete rate categories using the standard neighbor joining starting tree.

GARLI Settings: GARLI was executed once on each starting tree/dataset combination under GTR+Γ. For all runs the memory available to store likelihood vectors of the tree population was set to `availablememory=2048` and `megscalememory=2000`. The number of generations was set to `stopgen=50000000` and the stopping criteria were set to `enforcetermconditions=1` and `genthreshfortopoterm=10000`. The rearrangement distance was set to the default of `limsprrange=6`.

Finally, an additional set of tests was conducted to determine the minimum `megscalememory` memory setting for each alignment (indicated in column GARLI-MIN of Table 3). Note however, that using the minimum setting for `megscalememory` might decrease the performance of GARLI, since likelihood vectors of tree populations will have to be re-computed instead of being re-loaded from memory. For details on program settings please refer to the GARLI manual which is available at www.zo.utexas.edu/faculty/antisense/Garli.html.

4 RESULTS

The current Section covers the performance of RAxML-VI-HPC versus RAxML-V (Section 4.1) as well that of RAxML-VI-HPC versus MrBayes, GARLI, IQPNNI, and PHYML (Section 4.2).

4.1 RAxML-VI-HPC versus RAxML-V under GTR+CAT

In order to measure the speedup of RAxML-VI-HPC over RAxML-V and to assess the impact of the slight algorithmic and significant technical changes RAxML-V and RAxML-VI-HPC were executed only on *one single* starting tree per dataset (the starting trees indexed by `_0` in wwwbode.in.tum.de/~stamatak/Study.tar.gz) using the GTR+CAT approximation. The rationale for using only one starting tree per dataset is that the performance comparison with the author's own program does not need to be as thorough as with competing programs.

Table 2. Performance of RAxML-VI-HPC versus RAxML-V

Taxa	L-VI	L-V	T-VI	T-V	VI>V	Speedup
1000	-349668	-349655	07:28	27:16	void	3.65
1497	-197794	-197851	04:57	14:18	void	2.89
1663	-273941	-273915	06:33	17:40	void	2.70
1728	-168025	-168043	04:17	19:16	void	4.49
2000	-365502	-365472	13:28	113:36	void	8.44
2560	-318722	-318719	08:01	66:08	void	8.26
4114	-326815	-326892	19:15	191:16	12:24	15.42
6722	-482126	-482407	34:21	496:58	15:02	33.05
7769	-499861	-500302	34:44	494:42	09:26	52.43
8780	-846355	-846715	63:35	493:18	17:57	27.47

Final Log Likelihood values were then determined by re-evaluating the final trees under GTR+Γ using the respective RAxML program option. The runs of RAxML-V on the 6,722, 7,769 and 8,780-taxon datasets were interrupted due to the very long execution times, i.e. they did not run to completion.

Table 2 indicates the GTR+Γ Log Likelihood (columns L-VI and L-V) values and overall execution times for RAxML-V and RAxML-VI-HPC (columns T-VI and T-V). In addition, the time at which RAxML-VI-HPC found a tree that had a better likelihood than the tree obtained by RAxML-V (column VI $>$ V) is indicated for the larger datasets (4,114-8,780). Here the effect of the determination of a "good" rearrangement setting (see Section 2.4) on the improvement of the likelihood value is notable. Nonetheless, the effect of the selected rearrangement setting is dataset-dependent, which explains the large variations of speedup values among the 4 larger alignments (column Speedup).

The general deviations in final likelihood values of the smaller datasets in Table 2 are mainly due to the slightly altered hill-climbing search path (see Section 2.2).

A previous comparison of RAxML-VI versus RAxML-V under the HKY85 model on 5 starting trees per dataset indicates that the programs reach very similar final Log Likelihood values on average (respective plots are available on-line at diwww.epfl.ch/~stamatak, material frame).

In order to illustrate the efficiency of the GTR+CAT approximation over GTR+Γ Figure 4 depicts the GTR+Γ Log Likelihood development over time (seconds) on the same starting tree. The alignment contains 8,864 aligned 16S sequences of Bacteria which have been obtained from the Pace Laboratory at the University of Colorado at Boulder within the framework of a common project to infer a backbone tree for Bacteria.

4.2 RAxML-VI-HPC versus MrBayes, PHYML, IQPNNI, GARLI under GTR+Γ

Table 3 indicates the main memory consumption of each program per dataset. Column VI-CAT provides the memory consumption of RAxML-VI-HPC under the GTR+CAT approximation and column VI-Γ indicates the memory consumption of RAxML-VI-HPC under GTR+Γ. Column GARLI-MIN indicates the minimum memory requirements of GARLI. Note that even under GTR+Γ RAxML needs on average 1.8 times less memory than GARLI with minimum settings.

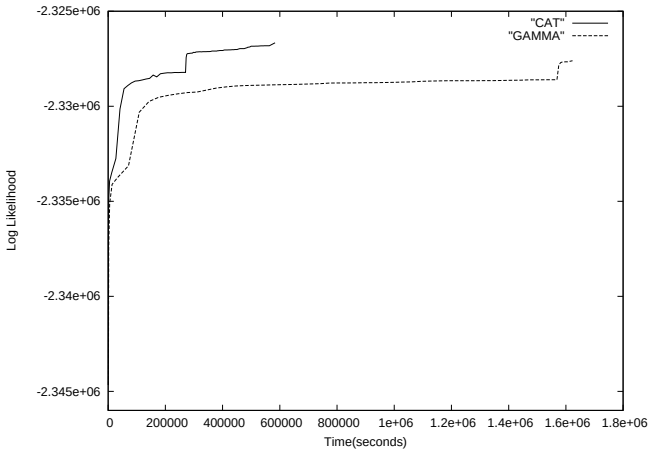

Fig. 4. RAxML Γ Likelihood development over Time for inferences with GTR+CAT and GTR+Γ on an alignment of 8,864 Bacteria

Table 3. Memory Consumption of RAxML-VI-HPC under GTR+CAT, and RAxML-VI-HPC, GARLI-Minimum, GARLI, IQPNNI, PHYML, MrBayes under GTR+Γ

Taxa Unit	VI-CAT MB	VI-Γ GB	GARLI-MIN GB	GARLI GB	IQPNNI GB	PHYML GB	MrBayes GB
1000	137	0.43	0.78	1.6	4.8	3.2	2.3
1497	74	0.24	0.49	1.3	void	1.8	1.2
1663	104	0.34	0.59	1.5	3.8	2.5	1.7
1728	87	0.29	0.49	1.3	void	2.1	1.4
2000	100	0.33	0.59	1.3	3.7	2.4	1.6
2560	125	0.40	0.78	1.6	void	void	void
4114	206	0.67	1.27	1.7	void	void	void
6722	299	1.0	1.76	1.8	void	void	void
7769	262	0.85	void	void	void	void	void
8780	420	1.4	void	void	void	void	void

The remaining columns indicate the memory consumption of GARLI, IQPNNI, PHYML, and MrBayes. In comparison to IQPNNI and PHYML, RAxML with GTR+Γ requires 8–10 times less memory than IQPNNI and PHYML.

Table 4 provides the final Log Likelihood values which have *all* been evaluated with RAxML-VI-HPC under GTR+Γ. Note, that for RAxML, GARLI, and MrBayes those are average values over 5 runs on 5 distinct randomized MP starting trees per dataset. In addition, for MrBayes only the likelihood of the majority rule consensus trees is indicated (MrBayes-CONS), since consensus trees were 4–27 Log Likelihood units better than the single last tree.

PHYML and IQPNNI were executed only once on each dataset using their proprietary neighbor joining starting trees. On several replicas (2 on 1,000 taxa, all 5 on 1,497, 3 on 1,728, and 3 on 2,000) the MP starting tree of RAxML showed a better Log Likelihood than the final tree obtained by PHYML *after* ML optimization. Finally, RAxML also found the best-scoring topology on *every* starting tree/dataset combination and not only on average.

Table 5 indicates the execution times of the programs in hours and minutes. Note that, IQPNNI and MrBayes on the 5 smaller datasets were interrupted after 60 hours. Also note that IQPNNI

Table 4. Log Likelihood Values: RAxML-VI-HPC versus GARLI, IQPNNI, PHYML, MrBayes

Taxa	RAxML-VI	GARLI	IQPNNI	PHYML	MrBayes-CONS
1000	-349651	-349754	-349914	-350738	-349855
1497	-197811	-198036	void	-199479	-198304
1663	-273912	-274081	-274535	-274846	-274468
1728	-168048	-168177	void	-168935	-168267
2000	-365462	-365758	-366747	-367640	-366193
2560	-318757	-318901	void	void	void
4114	-326698	-327429	void	void	void
6722	-482195	-483407	void	void	void
7769	-499866	void	void	void	void
8780	-846387	void	void	void	void

Table 5. Execution Times: RAxML-VI-HPC versus GARLI, IQPNNI, PHYML, MrBayes

Taxa	RAxML-VI	GARLI	IQPNNI	PHYML	MrBayes
1000	09:24	07:13	60:00	10:29	60:00
1497	05:33	09:12	void	10:09	60:00
1663	07:40	10:44	60:00	06:04	60:00
1728	06:20	05:44	void	04:35	60:00
2000	12:47	11:52	60:00	02:44	60:00
2560	08:04	14:25	void	void	void
4114	24:37	50:32	void	void	void
6722	43:10	115:33	void	void	void
7769	35:41	void	void	void	void
8780	65:11	void	void	void	void

crashed on the 1,497 and 1,728 taxon datasets due to numerical underflow. Finally, GARLI could not be executed on the 7,769 and 8,780 taxon datasets due to memory limitations (Derrick Zwickl, personal communication).

The run times for RAxML and GARLI provided in Table 5 are very similar on the 6 smaller datasets up to 2,560 taxa with GARLI being faster than RAxML-VI in 3 cases and vice-versa.

However, this comparison does not take into account the fact that RAxML yields better final likelihood values. To this end, for each distinct run all intermediate checkpoint files written by RAxML have also been evaluated under GTR+Γ in order to determine the first point in time at which RAxML encountered a tree with a better Log Likelihood than the final GARLI tree. Table 6 provides this additional comparison between execution times of RAxML and GARLI.

The average time over the 5 distinct starting trees per dataset at which RAxML reached GARLI Likelihood values is indicated in column R-VI $>$ GARLI. The third column (GARLI) once again indicates the GARLI execution times.

Note however, that this is only an additional indication which is useful for algorithm design, since in practice users will run both programs to completion. An interesting observation within the algorithmic context is that RAxML-VI reached the final GARLI likelihood values in nearly all cases after completion of the first iteration with fast analytical local branch length optimization turned off (see Stamatakis *et al.* (2005*b*), pp. 458–459).

It must be emphasized that the GARLI run-times indicated in Table 6 already include the final asymptotic convergence phase and the time required for final model optimization, whereas RAxML-VI is still in the topological search phase at this point.

Table 6. Additional Comparison: RAxML-VI-HPC versus GARLI

Taxa	R-VI $>$ GARLI	GARLI	Taxa	R-VI $>$ GARLI	GARLI
1000	03:02	07:13	1497	02:00	09:12
1663	02:38	10:44	1728	01:35	05:44
2000	02:52	11:52	2560	03:04	14:25
4114	04:53	50:32	6722	05:02	115:33

REFERENCES

Charalambous,M., Trancoso,P. and Stamatakis,A. (2005) Initial experiences porting a bioinformatics application to a graphics processor. In *In Proceedings of the 10th Panhellenic Conference on Informatics (PCI 2005)* pp. 415–425.

Hordijk,W. and Gascuel,O. (2005) Improving the effi ciency of SPR moves in phylogenetic tree search methods based on maximum likelihood. *Bioinformatics*, **21** (24), 4338–4347.

Rodriguez,F., Oliver,J., Marin,A. and Medina,J. (1990) The general stochastic model of nucleotide substitution. *J. Theor. Biol.*, **142**, 485–501.

Stamatakis,A. (2005) An effi cient program for phylogenetic inference using simulated annealing. In *Proc. of IPDPS2005*, Denver, Colorado, USA.

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

Stamatakis,A., Ludwig,T. and Meier,H. (2004) Parallel inference of a 10.000-taxon phylogeny with maximum likelihood. In *Proc. of Euro-Par2004* pp. 997–1004.

Stamatakis,A., Ludwig,T. and Meier,H. (2005*a*) Raxml-iii: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, **21** (4), 456–463.

Stamatakis,A., Ott,M. and Ludwig,T. (2005*b*) Raxml-omp: an effi cient program for phylogenetic inference on smps. In *Proc. of PaCT05* pp. 288–302.

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

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