Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees based on the Maximum Likelihood Method

Alexandros Stamatakis

Lehrstuhl für Rechnertechnik und Rechnerorganisation

Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees based on the Maximum Likelihood Method

Alexandros Stamatakis

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Hans Michael Gerndt

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Arndt Bode
2. Univ.-Prof. Dr. Christoph Zenger
3. Univ.-Prof. Dr. Thomas Ludwig
   Ruprecht-Karls-Universität Heidelberg

Die Dissertation wurde am 23.06.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.10.2004 angenommen.

Abstract

The computation of large phylogenetic (evolutionary) trees from DNA sequence data based on the maximum likelihood criterion is most probably NP-complete. Furthermore, the computation of the likelihood value for one single potential tree topology is computationally intensive.

This thesis introduces a number of algorithmic and technical solutions which for the first time enable parallel inference of large phylogenetic trees comprising up to 10.000 organisms with maximum likelihood.

The algorithmic part includes a technique to accelerate the computation of likelihood values, as well as novel search-space heuristics which significantly accelerate the tree inference process and yield better final trees at the same time.

The technical part covers technical solutions for the acquisition of the enormous amount of required computational resources such as parallel MPI-based and distributed seti@home-like implementations of the basic sequential algorithm.

Finally, the program has been used to compute a biologically significant initial small "tree of life" containing 10.000 representative organisms from the three domains: Bacteria, Eukarya, and Archaea based on data from the ARB database.

Acknowledgements

Many people have contributed to this thesis.

First of all I would like to thank Prof. Bode for the excellent working atmosphere at the Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR) and the trust and freedom he granted me for my research. Furthermore, I am grateful to Prof. Zenger who agreed to evaluate this thesis as 2nd reviewer. I am particularly thankful to Prof. Ludwig who has been accompanying and supporting me for many years now during my studies and doctoral research. Dr. Harald Meier, the leader of the high performance bioinformatics group at the LRR, deserves special gratitude for providing me fundamental biological knowledge.

From my colleagues at the Technische Universität München (TUM) I would like to thank Markus Lindermeier, Martin Mairandres, Jürgen Jeitner, Edmond Kereku, and Markus Pögl for their kind support on various issues and their good company over the last years.

I would also like to mention several colleagues from outside the TUM who have greatly helped me to accomplish this work: Ralf Ebner from the Leibniz RechenZentrum (LRZ), Gerd Lanferman from the Max-Planck Institut (MPI) Potsdam, and the HPC team from the Regionales Rechenzentrum Erlangen (RRZE).

I am especially grateful to my student Michael Ott who contributed to this thesis by implementing the distributed versions of RAxML.

Finally, I would like to thank my parents for their ever-lasting support of my work and ideas.

Contents

List of Figures

List of Tables

1

Introduction

> Die Verzerrung der Wahrheit im Bericht ist der wahrheitsgetreue Bericht über die Realität.
>
> *Karl Kraus*

This initial Chapter provides the motivation for conducting research in high performance computational biology and phylogenetics, summarizes the scientific contribution of the work, and describes the structure of this thesis.

1.1 Motivation

The immense accumulation of DNA and other relevant biological raw data through DNA sequencing techniques during recent years has lead to the emergence of a new interdisciplinary filed in computer science and biology: Bioinformatics. One main issue in Bioinformatics is to organize and represent this huge mass of data appropriately and to keep pace with its constant growth. As an example the increase of available DNA sequence data in the GenBank [35] database is outlined in Figure 1.1 (GenBank growth data is available at WWW.NCBI.NLM.NIH.GOV/GENBANK/GENBANKSTATS.HTML).

Another key objective of Bioinformatics is to extract useful information from the enormous amount of available data and thereby enable new insights into the system of life. However, there also exist areas of research in computational biology which are not based on DNA sequence data, such as simulation of metabolic pathways or genetic networks.

Unfortunately, many interesting problems and algorithms in Bioinformatics, such as inference of perfect phylogenies or optimal multiple sequence alignment are NP-complete and computationally extremely intensive. Therefore, High Per-

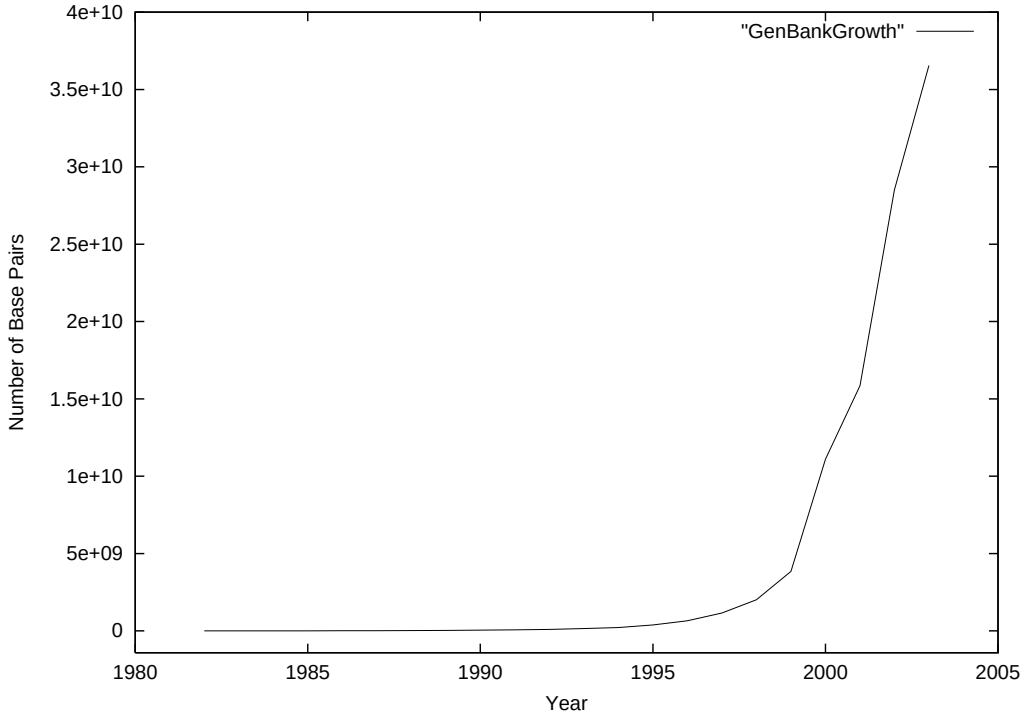

Figure 1.1: Growth of sequence data in GenBank

formance Computing Bioinformatics (HPC Bioinformatics) represents a particular difficult challenge due to its strong interdisciplinarity, since concepts from Biology, Theoretical Computer Science, and High Performance Computing have to be integrated into a single computer program. Therefore, progress in this field can only be achieved by a combination of algorithmic, technical, and biological advances.

The evolutionary history of mankind and all other living and extinct species on earth is a question which has been preoccupying mankind for centuries. The theory of evolution including the survival of the fittest, initially postulated by C. Darwin [24] lead to rather controversial discussions in the beginning (see Figure 1.2, taken from WWW.JULIANTRUBIN.COM/BIOLOGYJOKES.HTML) but is now broadly accepted.

Typically, evolutionary relationships among organisms are represented by an evolutionary tree. Therefore, the construction of a "tree of life" comprising all living and extinct organisms on earth ranging from simple *Bacteria* up to the *Homo Sapiens*, or vice versa if one prefers, has been a fascinating and challenging idea since the emergence of evolutionary theory.

Figure 1.2: Charles Darwin as seen by a contemporary cartoonist

"Classic" phylogenetic (evolutionary) trees for a set of organisms are constructed by comparing the presence/absence of certain distinguishing characteristics of those species, such as the number of legs, type of bones, etc. In contrast to trees obtained by computational methods which assume hypothetic ancestors those phylogenies also include known common ancestors. However, the question arises how to compare organisms which can not be classified by obvious phenomenological properties such as *Bacteria* or *Archaea* and above all how to compare those simple organisms with animals or plants. The basic structure of the tree of life which contains organisms from the three domains Archaea, Bacteria and Eukarya is provided in Figure 1.3. Members of these three domains are mainly distinguished by chemical properties as well as by the structure of their cell walls and cell membranes.

Archaea and Bacteria differ in that the Archaea usually live in extreme environments and are less common in *normal* environments because they were probably out-competed by Bacteria. Bacteria are widespread: There exist more Bacteria in a person's mouth than there are people in the world. Most human diseases are caused by the Bacteria, rather than by the Archaea. In fact, no pathogenic Archaea are currently known. Finally, organisms such as plants, animals, fungi, and protozoa [1] are all members of the Eukarya.

The first computational approaches to phylogenetics date back to the early 60's [16, 28]. The full potential of molecular phylogeny was revealed in a paper by Zuckerkandl and Pauling [154]. During this period of time the hypothesis emerged that certain regions of molecular sequences might contain evolutionary information. The idea of using a special, highly conserved region of the DNA, the

[1] Protozoa are single-celled creatures with nuclei that show some characteristics usually associated with animals, most notably motility and heterotrophy.

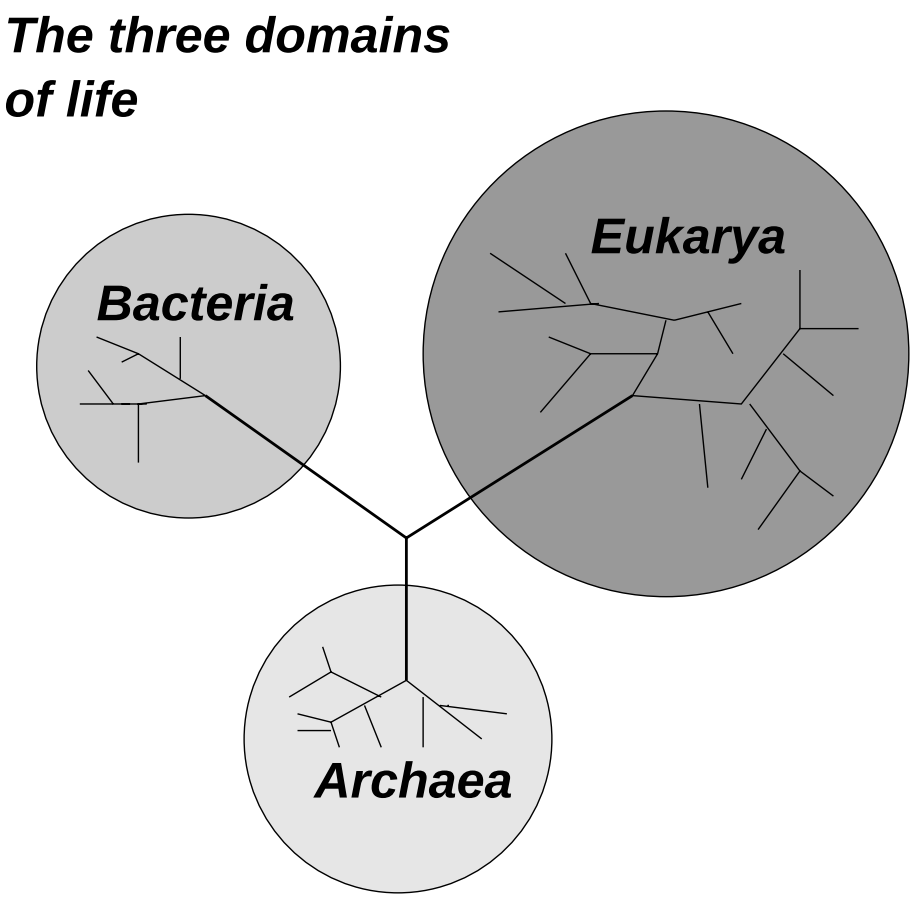

Figure 1.3: Domains of life: Eukarya, Bacteria, and Archaea

16S small subunit ribosomal Ribonucleic Acid (ssu rRNA), to conduct evolutionary analysis comprising all living species is firstly mentioned in a seminal paper by G.E. Fox et al. [33].

Thus, since the required biological raw data has now become available and due to the high algorithmic and computational complexity of underlying algorithms and models the inference of a “tree of life” containing representative species of all living organisms on earth is one of the “grand challenges” of Bioinformatics in our days. Important applications of large phylogenetic trees in medical and biological research are discussed in Section 2.2 (pp. 8).

In the spirit of this “grand challenge” this thesis covers the development of novel concepts for inference of large phylogenies based on the maximum likelihood method, which along with bayesian phylogenetic inference has proved to be the most adequate as well as accurate model for inference of huge and complex

trees. Thus, the overall goal of this work can be stated as: HOW TO INEXPENSIVELY COMPUTE MORE ACCURATE TREES IN LESS TIME?

1.2 Scientific Contribution

The problem of maximum likelihood phylogenetic tree reconstruction is unfortunately believed to be NP-complete, which induces the necessity to introduce appropriate heuristics. Furthermore, the computation of the likelihood score for one single potential tree topology is computationally extremely expensive. Thus, progress in this field can not be achieved by brute-force allocation of all available computational resources or simple parallelization of existing sequential algorithms. Instead, progress is driven by algorithmic innovation. Parallelization of phylogeny programs can only provide the gain of one or two additional orders of magnitude (in terms of computable tree size) due to the complexity of underlying algorithms.

Thus, the main contribution of this work consists in two basic algorithmic innovations (outlined in Chapter 4): The implementation of a novel algorithmic optimization of the likelihood function and the implementation of very fast *and* accurate new search space heuristics. The implementation of those two ideas in AxML (Axelerated Maximum Likelihood) and RAxML (Randomized AxML) respectively, lead to significant run-time and qualitative improvements. For example, the inference time for a 1000-species tree could be reduced from 18.000 (state of the art 2001) over 9.000 (state of the art 2002) to 17 CPU hours (2003) yielding a tree with a significantly better likelihood score at the same time.

The parallel and distributed implementations of those basic algorithmic ideas are considered as useful spin-offs and described in Chapter 5. However, the technical part of this thesis also generated three interesting results:

Firstly, the non-deterministic parallel implementation of the new search space heuristics rendered partially superlinear results.

Secondly, in order to provide inexpensive solutions for acquiring the large amount of required computational resources without using dedicated supercomputers a distributed meta-computing version of the program (similar to seti@home) has been implemented and successfully tested.

Thirdly, PC clusters and CPU architectures have shown to be the most adequate processor architecture for this type of code, and thus also contribute to inexpensive tree computations.

Finally, this thesis also contains an interesting biological result which is outlined in Section 6.6 (pp. 109). Based on sequence data which has been carefully selected from the ARB [76] database a first small “tree of life” containing 10.000 important representative species from all three domains: Eukarya, Bacteria, and

Archaea, could be computed using the methods and computer programs developed in this thesis. To the best of the author's knowledge this is the largest phylogenetic analysis by maximum likelihood to date.

The scientific results of this thesis have been incrementally published in nine reviewed conference papers [122, 123, 124, 125, 126, 128, 130, 131, 133], four journal articles [118, 119, 120, 121], two non-reviewed reports [129, 132] and as conference poster [127]. All papers are available in PDF format at WWWBODE.CS.TUM.EDU/~STAMATAK/PUBLICATIONS.HTML. Finally, the experiences gathered during 3 years of research in high performance computational biology will be presented within the framework of a joint half-day tutorial with Prof. T. Ludwig on "High Performance Computing in Bioinformatics" at the 4th International Conference on Bioinformatics and Genome Regulation and Structure (BGRS2004, Novosibirsk, Russia, July 2004).

1.3 Structure of the Thesis

The remainder of this thesis is built around the core Chapters 4 and 5. Chapter 2 provides a general introduction to phylogenetic tree inference. The subsequent Chapter 3 includes the most important models of evolution within the context of this thesis and gives an extensive description of the maximum likelihood method. Furthermore, it lists the most important and efficient state of the art sequential and parallel phylogeny programs based on statistical models of evolution. Following the core Chapters 4 and 5, Chapter 6 describes experimental results and performance of the respective implementations, including program accelerations, speedup values and improvements in tree quality. Chapter 6 also covers the biologically relevant inference of the 10.000-species tree based on the methods developed in the preceding chapters. Finally, Chapter 7 contains the conclusion and addresses important aspects of future work which will enable inference of even larger trees.

2

Phylogenetic Tree Inference

Manchmal sieht man vor lauter Bäumen den Wald nicht mehr.
German proverb

This Chapter introduces the relevant biological background, the prerequisites for computing a phylogeny, outlines the benefits of evolutionary trees for medical and biological research, and addresses problem complexity.

2.1 What is a Phylogenetic Tree?

First of all, it has to be stated that evolution must not necessarily be represented by a tree, i.e. using a tree to depict a phylogeny is already an initial assumption. There are some convincing arguments, such as lateral gene transfer between species which justify other forms of representation. The recent introduction of phylogenetic networks [40, 117] and respective methods provides an alternative to the tree model.

However, phylogenetic tree inference and evolution are still not properly understood and phylogenetic networks further augment the complexity of the problem. Thus, those alternative models are not well-suited for representation of large evolutionary relationships, at least at the current state of research. Therefore, this issue will not be further discussed within this context, although one should always be aware of the fact that the tree is not *the* model.

The tree model is further constrained by defining a phylogenetic tree to be a complete unrooted binary tree, i.e. all nodes have either degree 1 or 3. This is the standard definition of phylogenies used in almost every computational context. However, incomplete or n-ary binary trees as obtained by supertree or consensus

tree methods are addressed later on in this work (Sections 3.6 and 3.7, pp. 34 and 38). Methods to determine the root of unrooted binary trees are also outside the scope of this work due to the complexity of the problem. However, one common method to root a tree consists in using so-called outgroup species. This means that the DNA sequence of a species, which is not closely related to any of the organisms under consideration, is added to the alignment. After the completion of the phylogenetic analysis the outgroup species is used to root the tree.

An unrooted complete binary evolutionary tree represents the evolutionary history of a set of n species, which in the specific case are represented by their DNA sequences. Those n species are located at the tips of the tree topology, whereas the $n - 2$ inner nodes represent hypothetic extinct ancestors of those organisms. The branch-lengths between nodes usually stand for the time it took one organism to evolve into another new—not necessarily better—one.

A classic example for a phylogenetic tree is given in Figure 2.1. This subtree or clade represents the evolutionary relationship between humans and monkeys projected on a vertical time axis.

2.2 Obtaining new Insights from Phylogenetic Trees

At this point the question: WHAT DO WE NEED PHYLOGENETIC TREES FOR? has to be answered.

It has already been mentioned that for microorganisms such as Bacteria, without evident phenomenological characteristics, computing a phylogeny is the only practical approach to determine their evolutionary history. Furthermore, it is the only feasible method to summarize the relationships among all living organisms in one single tree of life.

Such large trees can contribute to medical and biological research in several ways. For example if a new, unknown, and dangerous bacterium x appears which threatens humanity, it might be inserted into an existing tree using computational methods. After insertion of bacterium x the biologist can identify close relatives of x, for which there exist appropriate treatments. Thus, in a time-critical situation one can rapidly derive appropriate therapies by consulting phylogenies.

A result published by Korber et al. [64] in *Science* that times the evolution of the HIV-1 virus demonstrates that maximum likelihood techniques can be effective in solving biological problems. Phylogenetic trees have already witnessed applications in numerous practical domains, such as in conservation biology [6, 29] (illegal wale hunting), epidemiology [14] (predictive evolution), forensics [88] (dental practice HIV transmission), gene function prediction [20], and drug development [41]. A paper by D. Bader et al. [5] addresses interesting industrial applications of phylogenetic trees, e.g. in the area of commercial drug discovery.

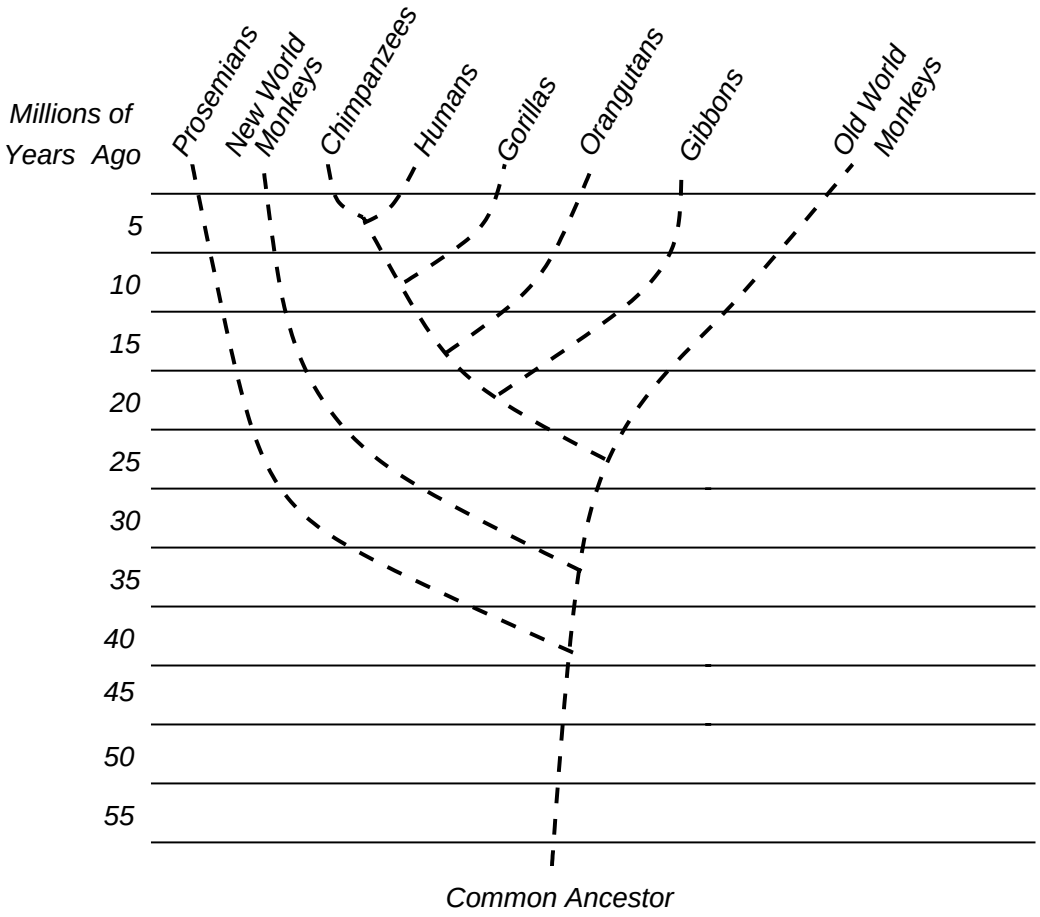

Figure 2.1: Phylogenetic subtree representing the evolutionary relationship between monkeys and the homo sapiens

In a recent review [106] Sanderson and Driskell provide a nice overview over the challenge of constructing large phylogenies and respective current and future problems as well as directions of research. Potential problems and solutions are also discussed in Sections 6.6 and 7.2 (pp. 109 and pp. 114) of this thesis as well as in [123].

Finally, the computation of a tree of life is generally considered to be a “grand challenge” in the field of HPC Bioinformatics. Not surprisingly, in 2003 the National Science Foundation (NSF) in the United States announced a 11.600.000$ tree of life initiative which is co-located at 13 leading research institutions across the U.S. (project web site: WWW.PHYLO.ORG).

Thus, the computation of phylogenetic trees is not a "l'art pour l'art" invention by a bunch of theoretical computer scientists but a practical scientific issue of great importance.

2.3 Prerequisites for Phylogenetic Tree Inference

As already mentioned the inference of phylogenetic trees is usually based on a multiple alignment of DNA or protein sequence data which serves as input for phylogeny programs. Therefore, irrespective of the algorithm used, the quality of the final result can only be as good as the quality of the alignment. Thus, a "good" multiple alignment of sequences is *the* most important prerequisite for conducting a phylogenetic analysis. This section provides a brief introduction to the computation of multiple alignments and a notion of underlying complexity.

Note however, that—apart from DNA sequence data—higher-level genetic information such as gene order data can also be used as input for phylogenetic analyses. Phylogenetic inference based on gene order data is however computationally harder than alignment-based inference and only comparatively small trees comprising less then 50 organisms could be computed so far [83]. Some progress has recently been achieved though, by application of the Markov Chain Monte Carlo (MCMC) technique [82]. Since the approach is currently not apt for inference of significantly larger trees it will not be further discussed at this point.

2.3.1 Computation of Multiple Alignments

Sequence alignment is one of the most basic operations in Bioinformatics. The sequences obtained from the laboratory are simple strings consisting of the 4 bases `A` ,`C`, `G`, `T` (`U` is equivalent to `T` for rRNA sequences). An excellent general introduction to sequence alignment can be found in Chapter 3 of [110].

A priori, those sequences have different lengths due to insertions, deletions, and substitutions of base-characters as well as sequencing errors. The alignment process corrects those errors and intends to identify corresponding homologous regions and construct sequences of equal length by insertion of gaps (`-`) into the sequence, according to a specified optimality criterion.

Initially, the alignment of two sequences $S1, S2$ is considered. In this case the optimality criterion is a scoring function $sf(S1_i, S2_i)$ which penalizes mismatches at position i between bases (e.g. -1) or bases and gaps (e.g. -2), and assigns high scores to matches (e.g. $+1$). Thus, the optimal alignment of two sequences is the one with maximum score according to the selected scoring-scheme.

For example one can consider the two sequences below:

```
S1: GACGGATTAG
S2: GATCGGAATAG
```

One optimal alignment of those two sequences, since there might exist several optimal alignments, would be the one indicated below:

```
S1: GA-CGGATTAG
S2: GATCGGAATAG
```

Note, that sequence comparison comprises two fundamental alignment types: local alignments of specific substrings and global alignments of the entire sequences.

Optimal global and local alignments of two sequences can be computed by dynamic programming approaches which store the alignment scores of all possible substrings in a matrix and then perform backtracking to construct the optimal alignment. The basic algorithms are known as Needelman-Wunsch [84] for global alignments and Smith-Waterman [113] for local alignments. The improved versions of those fundamental algorithms run in quadratic time and require linear or quadratic space as well.

For the computation of multiple alignments of n sequences initially the definition of the scoring function has to be extended. For example, one function which is often used is the so-called sum-of-pairs score $sp(S1_i, ..., Sn_i)$, which takes as arguments the characters at sequence position i of all sequences $S1, ..., Sn$ and is defined as:

$$sp(S1_i, ..., Sn_i) = sf(S1_i, S2_i) + ... + sf(S1_i, Sn_i) + ... + sf(Sn - 1_i, Sn_i)$$

Thus, the complexity of such a scoring function is already quadratic in n. For multiple alignments the same dynamic programming schemes as for pairwise sequence alignment can be applied. Unfortunately, it has been shown that computing optimal multiple alignments is NP-complete under most reasonable scoring functions [148]. Furthermore, both time *and* space requirements grow exponentially with the number of sequences. Therefore, multiple sequence alignment is still one of the most important research issues in HPC Bioinformatics and a plethora of heuristics, optimizations, parallel algorithms as well as alternative methods and scoring functions has been proposed for this problem. For example ClustalW [18] is one of the most widely-used multiple sequence alignment programs. A paper by Thompson et al. [143] provides a nice overview and in depth performance analysis of common multiple alignment methods.

Since the computation of multiple alignments is an extremely broad field it can not be covered in detail within the context of this thesis. This section intends

only to provide a notion of the complexity of the alignment problem and a certain influence of "religious" beliefs concerning e.g. the choice of the appropriate scoring function which has significant impact on final results.

2.3.2 Adequate DNA Portions

A somehow different problem, which has caused controversial discussions among Biologists consists in the selection of the appropriate DNA-portions for inference of phylogenetic trees. One has to select a region which appears in the DNA of all organisms of interest and which is highly conserved. In contrast, highly variable regions do not generate a reliable phylogenetic signal. Furthermore, this region must have some evolutionary significance, since there might very well exist regions which are highly conserved but do not contain any phylogenetic information. The 16S ribosomal rRNA is believed to be one of those adequate regions since it is universally distributed among organisms, exhibits constancy of function and changes relatively slowly compared e.g. to most proteins. The importance of the 16S rRNA for phylogenetic analysis of Prokaryotes has been outlined e.g. in a paper by Fox et al. [33].

Despite the fact that the selection of an appropriate region is not so important for the abstract computational problem of phylogenies, it requires serious consideration as soon as one desires to compute biologically significant results.

2.3.3 The ARB Database

The ARB [2, 76] database (arbor, Latin: tree) provides both, a large amount (currently more than 30.000 organisms) of curated small subunit Ribonucleic Acid (ssu rRNA) data and an excellent alignment quality. As outlined in the previous Section those two properties are the essential prerequisites for the computation of phylogenetic trees.

The ARB software environment has been developed over the last ten years in a joint initiative by the Lehrstuhl für Mikrobiologie and the Lehrstuhl für Rechnertechnik und Rechnerorganisation of the Technische Universität München. About ten years ago an increasing amount of small subunit rRNA raw data was becoming available from *primary databases* such as GenBank [8] or EMBL from the European Bioinformatics Institute [116] (EBI).

ARB represents a so-called *secondary database*, i.e. a database system which includes and integrates a large variety of individual tools to maintain, update, correct, represent, and extract useful information from the primary data.

The two key objectives of the ARB project are to provide:

1. The maintenance of a structured integrative secondary database containing processed primary structures and any type of additional information assigned to the individual sequence entries by the user.

2. A comprehensive selection of directly interacting software tools, along with a central database which are controlled via a common Graphical User Interface (GUI).

Thus, the in-house availability of the ARB database and the accumulated experience of over 10 years of ARB development and maintenance in combination with the biological expertise provided by the involved biologists, provides a solid basis to select and extract alignments of high quality for inference of large and biologically significant phylogenetic trees.

2.4 Problem Complexity

The main computational problem of phylogenetic inference consists in the large number of potential alternative tree topologies, which unfortunately grows exponentially with the number of species. Given n organisms the amount of possible unrooted binary trees is [28]:

$$\prod_{i=3}^{n}(2i-5)$$

Some exemplary figures for this formula are outlined in Table 2.1. Note, that for 50 organisms there exist almost as many alternative tree topologies as there are atoms in the universe ($\approx 10^{80}$).

Number of Organisms	Number of alternative Trees
3	1
4	3
5	15
6	105
7	945
10	$2.027.025$
15	$7.905.853.580.625$
20	$2.21 * 10^{20}$
50	$2.84 * 10^{76}$

Table 2.1: Number of possible trees for phylogenies with 3–50 organisms

Due to this combinatorial explosion one can suspect that phylogenetic tree inference is NP-hard. The fact that the hardness of phylogenetic reconstruction has been demonstrated for less elaborate discrete phylogeny models (see below) reinforces this suspicion. The NP-hardness of maximum likelihood is hard to formalize and prove since the method yields floating point scores and incorporates branch length optimization (see Chapter 3), i.e. the problem is not discrete.

Some earlier theoretical work in this area of genome analysis focused on finding *perfect phylogenies*. Perfect phylogenies require that for each character in each column, the taxa containing that character in that column of the alignment form a subtree of the phylogeny. Kannan and Warnow have a polynomial time algorithm for finding *perfect phylogenies* [61] under certain reasonable restrictions. However, like many problems associated with genome analysis, the general version of the *perfect phylogeny* problem is NP-complete [9].

While most elaborate tree-scoring functions do not strive to meet this *perfect phylogeny* criterion it is still widely believed that computing phylogenies that meet any sort of effective or reasonable criteria is NP-hard. This has been demonstrated e.g. for the parsimony criterion (see Section 3.3, pp. 17) in [25]. Note, that for maximum likelihood this has not been demonstrated so far, due to the significantly superior mathematical complexity of the model. However maximum likelihood is widely believed to be NP-hard among involved researchers.

The question which arises at this point is: “How to score those alternative topologies and how to design appropriate heuristics, in order to find the best possible (mostly suboptimal) tree according to the selected criterion?”.

In general, a fast scoring function enables the analysis of a greater part of the search space, whereas slower and more elaborate functions usually return better trees. This has repeatedly been demonstrated in recent comparative surveys [39, 147]. Thus, there is a “classic” tradeoff between execution speed and expected final tree quality.

Summary

The current Chapter introduced the biological background and the prerequisites for computing evolutionary trees. Furthermore, important applications of phylogenetic trees in medical and biological research have been mentioned. In addition, the ARB database was described which represents the main data source for experiments conducted within the framework of this thesis. Finally, the problem complexity of phylogenetic analyses was addressed. All fundamental aspects of phylogenetic tree inference, such as basic tree reconstruction models and search algorithms, statistical inference methods, as well as current state-of-the-art implementations are addressed in the following Chapter.

3

Phylogeny Models and Programs

> Ich habe nie eine einzige Bemerkung allein gemacht, sondern es fiel mir allzeit noch eine zweite ein.
>
> *Jean Paul*

This Chapter covers the basic models and algorithms for phylogenetic tree inference. It describes basic mechanisms to augment confidence into the final result and discusses methods for comparing phylogeny programs. Furthermore, it includes a survey of current state of the art sequential and parallel phylogeny programs which implement statistical models of evolution.

3.1 Basic Model Classification

There exist two basic classes of phylogeny models which can be distinguished by their usage of the information contained in the input alignment of n species.

The first class, makes only indirect use of the data by computing a corresponding symmetric $n \times n$ distance matrix Δ containing all pairwise distances between sequences, according to some function $\delta(Sequence_i, Sequence_j)$ $i, j = 1, ..., n$. The definition of δ and of the respective optimal distance-based tree have substantial impact on problem complexity. Function δ needs to be meaningful in a biological context.

The second class, the so-called character-based methods make direct use of the alignment data, by computing tree-scores on a column by column basis. This means that at each inner node of a topology a vector has to be computed containing integer or floating point values to score the tree. Those vectors are typically computed in a bottom-up tree-traversal towards a virtual root vr. The information

of the updated vector at vr is then combined by mathematical operations into one single tree score value.

In general, distance-based methods are faster and less accurate than character-based methods. Those simple methods can be deployed to rapidly obtain initial estimates of phylogenies which can also be used as starting trees for character-based methods (see Section 3.9.1). Furthermore, they represent the only computationally feasible method for computation of extremely large trees.

Finally, numerous computer studies [51, 52, 66, 103] based on synthetic data (see Section 3.8) have shown that character-based algorithms recover the true tree or a tree which is topologically closer related to the true tree more frequently than distance-based methods.

3.2 Distance-based Methods

The two most common distance-based methods are the Unweighted Pair-Group Method with Arithmetic Mean [114] (UPGMA) and Neighbor Joining [105] (NJ).

In those models the distances in matrix Δ represent the fraction of dissimilarities, i.e. amount of different nucleotide sites, between sequences. It is reasonable to assume that a pair of sequences is closer related if it differs in only 5% of sites instead of e.g. 40%. However, the assumption that the more time has passed from the divergence of two organisms from a common ancestor, the more diverse they will be, is problematic. This problem arises since different lineages (paths to tips from a common parent node i) may evolve at different speeds and/or subsequent substitutions at the same site (alignment column) are likely to occur. Thus, especially in the case of subsequent (multiple) substitutions at sites, an organism j far down the lineage might appear to be closer related to the parent in terms of $\delta(Sequence_i, Sequence_j)$ compared to intermediate organisms of the lineage. Although some corrective measures have been proposed this remains the basic problem of distance-based analyses.

Note, that both basic algorithms presented here implicitly assume a model of minimum evolution, i.e. suppose that nature selected the shortest path in terms of base substitutions to evolve one organism into another.

3.2.1 UPGMA

The UPGMA algorithm starts building the tree by selecting the most closely-related pair of sequences from Δ_1. Those two sequences are connected by a branch and a node which is placed in the center of the branch (the branch lengths correspond to the distances between organisms). In the subsequent step those two initial sequences are regarded as one, i.e. as cluster. At this step a matrix Δ_2 of

size $n-1$ is computed in respect to the clustered pair of sequences. This process is repeated until Δ_n has reached a size of 1. Thereafter, the matrix set $\Delta_n, ..., \Delta_1$ is used to construct the respective tree starting at the root.

The UPGMA algorithm contains two intrinsic assumptions: *Firstly*, that the tree is additive, and *secondly* that it is ultrametric. Additivity means that the length of the path between any pair of leaves i, j in the tree must be equal to $\delta(Sequence_i, Sequence_j)$, while ultrametricity means that all organisms are equally distant from the root. Those two properties represent rather utopic assumptions and oversimplify the problem.

Due to these implicit assumptions and limitations UPGMA is not frequently used to establish phylogenies nowadays.

3.2.2 Neighbor Joining

Neighbor joining works in a similar way as UPGMA in that it uses a set of distances matrices for tree reconstruction as well. However, NJ does not cluster nodes but computes distances to internal nodes as well, thereby resolving restrictions imposed by ultrametricity and additivity (see above) of the UPGMA method. Initially, NJ computes the divergence of an organism from *all* other organisms by summing up the individual distances. Thereafter, NJ calculates a corrected distance matrix Δ_1, selects the pair of sequences with the lowest corrected distance and connects them via an inner node. At this point the distances between each sequence and the inner node are calculated which do *not* need to be identical. This inner node is then used to replace the initial pair of organisms in Δ_2 of size $n-1$ and the process is repeated. Finally, the tree is reconstructed in the same way as for UPGMA.

An in-depth discussion of the most common distance measures for Neighbor Joining is provided in Chapter 11 of [140]. Note, that this algorithm does not automatically yield the tree with the minimum overall distance [45]. A good overview which covers most common distance methods can be found in a survey conducted by Swofford et al. [140]. Finally, the program BIONJ [34] by O. Gascuel represents a recent and very popular open source code implementation of Neighbor Joining.

3.3 Parsimony Criterion

In the same spirit as distance-based methods maximum parsimony also assumes a model of minimum evolution. It differs however from distance-based models since it defines minimum evolution on a site-per-site (column-by-column) basis of the alignment. Thus, parsimony assumes that the most credible tree is the

one which requires the smallest amount of changes, i.e. number of nucleotide substitutions in the tree.

Therefore, parsimony algorithms intend to find the trees, since there can exist many equally parsimonious topologies, that minimize the number of required evolutionary steps.

After those initial considerations the computation of the parsimony score is outlined by example of a 5-taxon tree which is depicted in Figure 3.1. Only the computation of the number of changes for one site, i.e. one nucleotide, is considered since the overall score for an alignment can be obtained by summing up the parsimony scores of all individual *informative* sites. This simple addition of individual column scores also induces the implicit assumption that sites evolve independently from each other.

Homogeneous alignment columns, i.e. columns that consist of the same base in all organisms (see also Section 4.1, pp. 54) are called *uninformative*, since they do not provide information for the parsimony score.

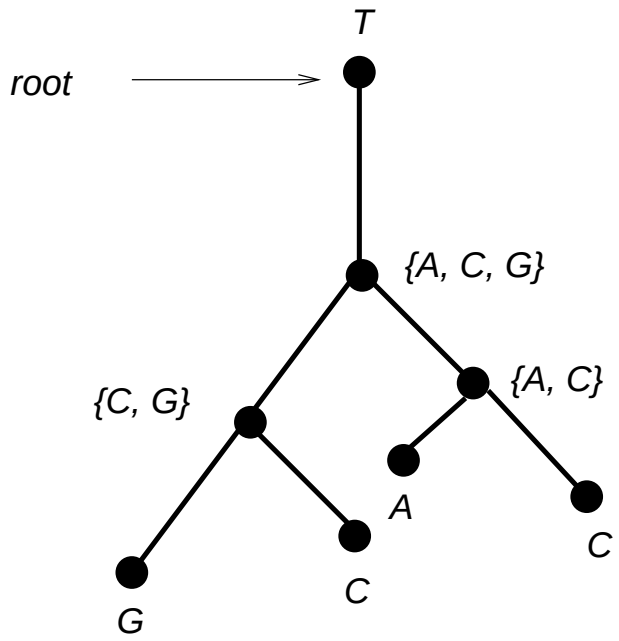

Figure 3.1: Parsimony score computation by example

Initially, the tree which consists of the 5 nucleotides `T,G,C,A,C` is rooted at tip `T`. Thereafter, starting bottom-up at the tips the inner nodes of the rooted tree are assigned sets of possible inner states, i.e. `{G,C}`, `{A,C}`, and `{A,G,C}` respectively. In a subsequent top-down step the required number of changes in the tree is computed. Note, that at each inner node a state contained

in the parent state set has to be selected. If the respective state sets do not overlap an arbitrary state is chosen. This choice has however no effect on the overall score. The computation of the number of changes for two different choices of inner states in the example tree is outlined in Figure 3.2 and 3.3. Branches where changes occur are indicated by dotted lines, the parsimony score is 4 for all possible assignments with the specific root in the example.

After this step the score is computed for all remaining possible rootings of the tree and the minimum parsimony score obtained during this process corresponds to the parsimony score of the tree. For example, if the same tree is rooted at nucleotide `G` the score will be 3.

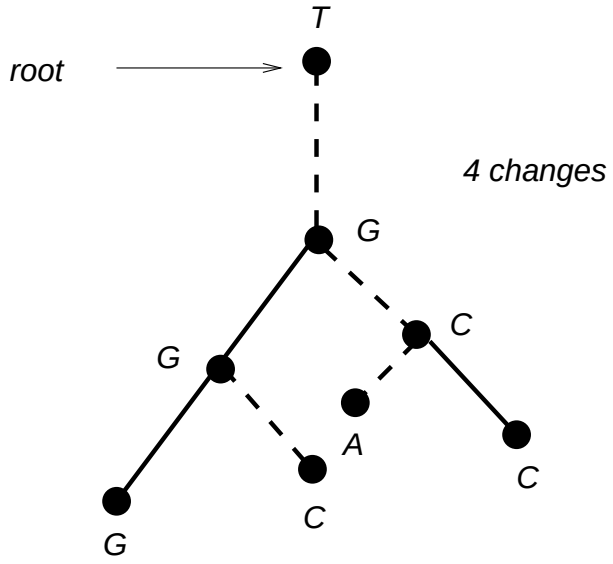

Figure 3.2: Parsimony score computation by example: one possible assignment

The tree building algorithms for maximum parsimony searches face similar problems and deploy similar techniques as heuristic (maximum) likelihood searches. Those heuristics are outlined in Section 3.9.1.

As NJ and UPGMA the parsimony scoring-scheme implicitly assumes a concrete model of evolution. Furthermore, parsimony faces similar problems as distance-based methods with long lineages, since it does not account for potentially unobserved nucleotide substitutions, e.g. transitions of type `A`$\rightarrow$`A` . Felsenstein [30] found an example for which parsimony fails for trees with strongly divergent rates of evolution among lineages, whereas Hendy et al. [44] later suggested the term “long branch attraction” for a more general failure scenario of the parsimony criterion with equal rates of change throughout the tree.

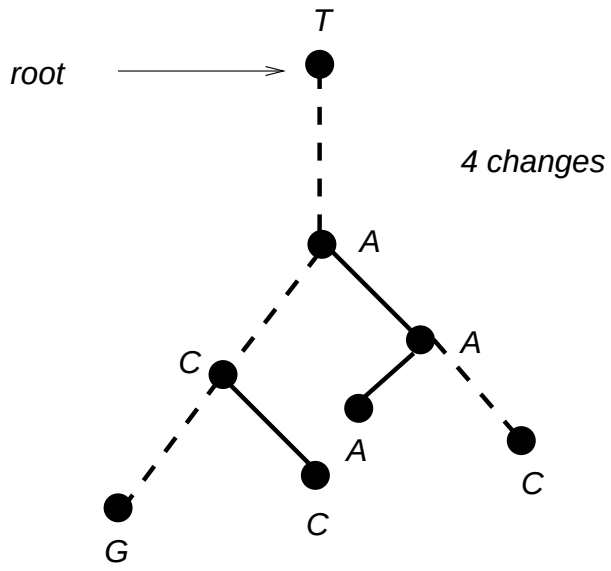

Figure 3.3: Parsimony score computation by example: another possible assignment

Among the most-widely used implementations are the commercial PAUP [92] package and dnapars from Felsenstein's PHYLIP package [93] which is available as open source code. NONA [37] by P.A. Goloboff is also claimed to be very fast but unfortunately not freely available. A nice, brief discussion of parsimony analyses can be found in [73] and a more detailed one in [140].

3.4 Maximum Likelihood Criterion

The problem of most topology score functions so far is that the algorithms implicitly assume a model of evolution, mostly minimum evolution or a molecular clock. The molecular clock assumes that evolutionary events, i.e. nucleotide substitutions occur regularly at certain time intervals (clock ticks). Thus, the molecular clock does not take into account and does not provide a model for variation in evolutionary speed at different points in time. Furthermore, distance-based methods do not fully exploit the information contained in the alignment. In 1981 J. Felsenstein [31] proposed a statistical method which allows for explicit specification of evolutionary models and which represents a computationally feasible approach at the same time. In this seminal paper Felsenstein describes how to infer phylogenetic trees under a simple probabilistic model of DNA evolution based on maximum likelihood.

Maximum likelihood in this specific case means that one intends to find the topology which yields the highest probability of producing (evolving) the observed data (alignment). Note, that the likelihood of a tree is not the probability, that the tree is the correct one.

The problems which arise within this context are how to compute the likelihood of a set of sequences placed in a given tree, how to optimize branch-lengths in order to obtain the maximum score for that particular tree, and how to devise a probabilistic model of nucleotide substitution.

Having resolved those problems the most important difficulty still remains to be solved: HOW TO SEARCH FOR THE TREE WHICH MAXIMIZES THE LIKELIHOOD OVER ALL POTENTIAL TREE TOPOLOGIES?

As already mentioned this problem is widely believed to be NP-complete, mainly due to the exponential explosion in the number of possible topologies (see Section 2.4, pp. 13). The most common search space heuristics are discussed separately in Section 3.9.1, since the development of fast and precise heuristics represents the most outstanding algorithmic challenge for the design of maximum likelihood programs.

3.4.1 Calculating the Likelihood of a Tree

The likelihood of a tree can be computed if a model providing the probability that a sequence S_1 evolves into sequence S_2 among a branch t (time-segment) is available. Furthermore, it is assumed that individual sites (nucleotides) of the sequence evolve independently. As in the parsimony model this is a very restrictive and critical assumption from a biologist's point of view, which however has to be made to reduce the complexity of computing the likelihood score. Under this assumption the score of a tree can be computed site by site and finally be obtained by taking the product of the individual sites. Thus, it is sufficient to concentrate on the computation of the probability for a single site. The function, $P_{ij}(t), i, j = 1, ..., 4$ where values of i and j represent the four bases `A,C,G,T` , gives the probability that a base in state i evolves into state j after time t.

For those base transition probabilities a Markov-process is assumed, i.e. the probability of $i \rightarrow j$ is independent from the history of i regarding prior evolutionary events.

The only property such a model of base substitution should have is reversibility: if a base evolves into another, it is replaced by `A,C,G,T` with probabilities $\pi_A, \pi_C, \pi_G, \pi_T$. Reversibility requires $\forall i, j, t$:

$$\pi_i P_{ij}(t) = P_{ji}(t)\pi_j$$

The reversibility property means that the evolutionary process is identic if followed forward *or* backward in time. The reasons for which this property is required will be explained later on in the current Section.

This very general definition of the evolutionary model used at this point is known as General Time Reversible model of nucleotide substitution (GTR). It is however sufficient to explain the mechanism of likelihood computation at a high level of abstraction. The plethora of different models which have been proposed deserve a separate discussion in Section 3.4.3.

In the following the computation of the likelihood, given the evolutionary model, is explained by the simple example tree of Figure 3.4.

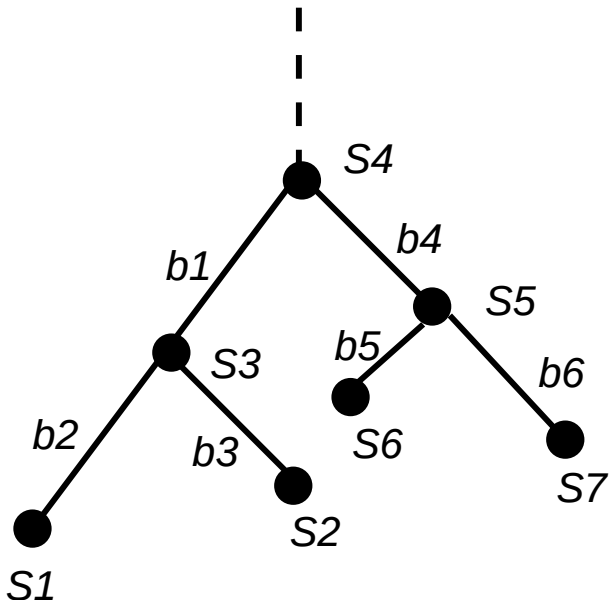

Figure 3.4: Rooted example tree with root at node S_4

The branch lengths of the tree are given by b_i and the sequences by $S_1, ..., S_7$. However the known sequences drawn from the alignment are only S_1, S_2, S_6, S_7 which are located at the tips and S_3, S_4, S_5 are unknown common ancestors. Initially, a rooted tree is considered which has its root at S_4. If S_3, S_4, S_5 were known the likelihood L could be computed as the product of probabilities of change along each branch times the prior probability π_{S_4} at S_4:

$$L = \pi_{S_4} P_{S_4 S_3}(b_1) P_{S_4 S_5}(b_4) P_{S_3 S_1}(b_2) P_{S_3 S_2}(b_3) P_{S_5 S_6}(b_5) P_{S_5 S_7}(b_6)$$

Unfortunately, S_3, S_4, S_5 are unknown such that the likelihood is the sum over all possible nucleotide states `A,C,G,T` at the inner nodes S_3, S_4 and S_5 of the tree:

$$L = \sum_{S_4=A}^{T} \sum_{S_3=A}^{T} \sum_{S_5=A}^{T} \pi_{S_4} P_{S_4 S_3}(b_1) P_{S_4 S_5}(b_4) P_{S_3 S_1}(b_2) P_{S_3 S_2}(b_3) P_{S_5 S_6}(b_5) P_{S_5 S_7}(b_6)$$

This expression contains $4^3 = 64$ terms. However for n species it has 4^{n-1} terms which rapidly becomes a large number. A reduction in the required number of arithmetic operations can be achieved by shifting some summations to the right:

$$L = \sum_{S_4=A}^{T} \pi_{S_4} \left(\sum_{S_3=A}^{T} P_{S_4 S_3}(b_1) \left[P_{S_3 S_1}(b_2) \right] \left[P_{S_3 S_2}(b_3) \right] \right) \left(\sum_{S_5=A}^{T} P_{S_4 S_5}(b_4) \left[P_{S_5 S_6}(b_5) \right] \left[P_{S_5 S_7}(b_6) \right] \right)$$

The pattern of the parentheses $\big([][]\big)\big([][]\big)$ in the transformed expression corresponds exactly to the structure of the tree topology in this example. Thus, the expression for the likelihood can be evaluated in a bottom-up scheme starting at the tips of the tree by application of a postorder tree traversal. One can therefore define a recursive procedure for the computation of the overall likelihood value by using conditional likelihoods for subtrees at a node k of the tree. Let $L_{S_k}^{(k)}$ be the likelihood of the data in the subtree rooted at k, given that the nucleotide state s at k is fixed. If k is a tip and consists e.g. of nucleotide `A` $L_A^{(k)} = 1$ and $L_C^{(k)} = L_G^{(k)} = L_T^{(k)} = 0$.

Otherwise, if nodes i and j are immediate descendants of k all four entries can be computed by applying:

$$L_{S_k}^{(k)} = \Big(\sum_{S_i=A}^{T} P_{S_k S_i}(b_i) L_{S_i}^{(i)} \Big) \Big(\sum_{S_j=A}^{T} P_{S_k S_j}(b_j) L_{S_j}^{(j)} \Big)$$

If this procedure is executed recursively until node S_4 of the example is reached the four conditional likelihoods become available at ${L_{S_4}}^{(4)}$, and the overall likelihood of the tree for this specific alignment site is:

$$L = \sum_{S_4=A}^{T} \pi_{S_4} {L_{S_4}}^{(4)}$$

The probabilities π_A through π_T have to be the prior probabilities , often also called base frequencies, of detecting each of the four bases `A,C,G,T` at point

S_4 of the tree. Those probabilities are usually drawn empirically from the alignment data. Since maximum likelihood postulates an evolutionary steady state in base composition, those probabilities correspond to the overall base composition of the input alignment. Thus, $P_{ij}(t)$ needs to be specified such as to guarantee that the probabilistic process maintains this base composition. Usually, it is assumed that the specific base composition for an alignment is obtained by external evidence, i.e. it does not directly form part of the maximum likelihood process. A more detailed discussion of base frequencies is postponed to Section 3.4.3 on evolutionary models, as well.

At this point it has to be explained for which reason the Markov process needs to be reversible. Reversibility is required to establish a useful property for the computation of maximum likelihood-based phylogenetic trees which Felsenstein calls the "pulley principle". Therefore, one can consider the last two steps of the likelihood computation process of the example in Figure 3.4 for nodes S_3, S_4, S_5 once again:

$$L = \sum_{S_4=A}^{T} \pi_{S_4} \left(P_{S_4S_3}(b_1)L_{S_3}^{(3)}\right) \left(P_{S_4S_5}(b_4)L_{S_5}^{(5)}\right)$$

A brief derivation can be deployed to show that the value of L remains unchanged if the same length x is added to b_1 and subtracted from b_4 i.e.

$$L = L' = \sum_{S_4=A}^{T} \pi_{S_4} \left(P_{S_4S_3}(b_1 + x)L_{S_3}^{(3)}\right) \left(P_{S_4S_5}(b_4 - x)L_{S_5}^{(5)}\right)$$

Thus, L exclusively depends on b_1 and b_4 via their sum. This means that the *virtual root* (node S_4 in the example) which is required to compute the likelihood value bottom-up can be placed anywhere between S_3 and S_5. Therefore, the *virtual root* of the tree can be regarded as pulley, i.e. if all components of the tree are moved down on one side and moved up on the other side by the same x the likelihood remains *exactly* identical. In addition, the above consideration can be applied recursively to the tree, such that irrespective of the point at which the *virtual root* is placed to compute the likelihood score of a thereby *rooted* unrooted tree the obtained likelihood value will remain unchanged. The unrooted example tree showing all *virtual root* (node S_4 in the rooted example) placement possibilities and the way how the *virtual root* can be moved along a branch is outlined in Figure 3.5.

In comparison to parsimony this is a great advantage of the maximum likelihood model. Furthermore, it reduces the computational complexity which however still remains high compared to other methods.

Therefore, the Markov process must be reversible in order to allow for application of the important *pulley principle*. In addition, the *pulley principle* is very

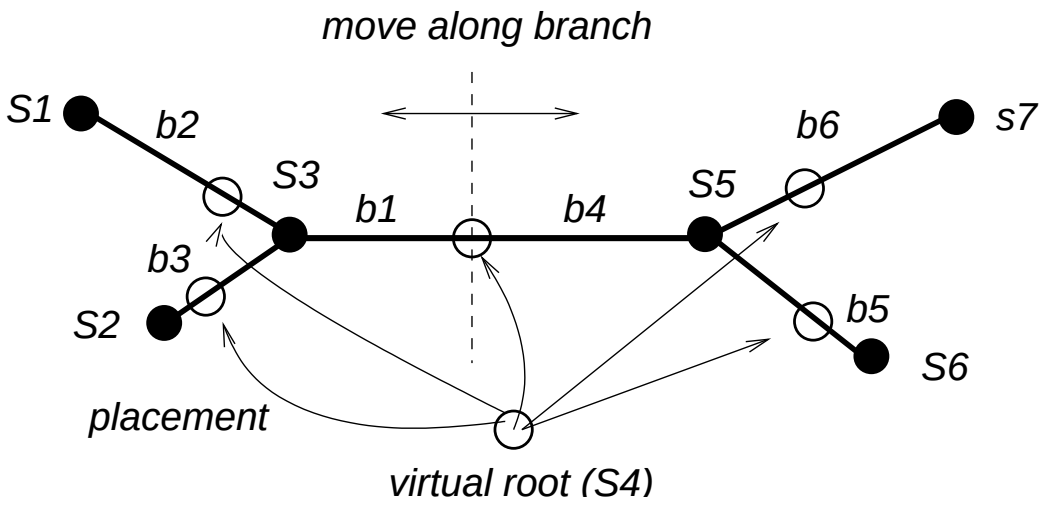

Figure 3.5: Unrooted example tree with virtual root placement possibilities, likelihood remains unaffected

important for branch length optimization, which is outlined in the subsequent Section.

Finally, note that in most computer programs the log likelihood values are computed due to numerical reasons, e.g. a tree with a log likelihood of -10000 is better than a tree with a log likelihood of -12000. The likelihood values provided for real data experiments in Chapter 6 (pp. 87) of this thesis are *always* the log likelihood values.

3.4.2 Optimizing the Branch Lengths of a Tree

Up to this point it has been demonstrated how one can compute the likelihood value of an individual unrooted tree.

However, the branch lengths of this individual tree need to be optimized in order to obtain the maximum likelihood value for the specific topology.

As already mentioned the *pulley principle* allows the *virtual root* to be placed in any branch b_i of the tree. Since the value of each b_i needs to be optimized such as to maximize the likelihood of the specific topology the *pulley principle* can be deployed to individually optimize each b_i in turn with respect to the current lengths of the other branches. This iterative process can be repeatedly applied to all of the b_i until no further alteration of any b_i yields an improved likelihood. Due to the *pulley principle* it is guaranteed that during this process the likelihood of the overall tree will constantly increase until convergence.

In order to optimize an individual branch b connecting node S_i with S_j the *virtual root* is placed immediately besides S_i, i.e. with a distance of 0 to S_i. Thereafter, iterative numerical methods can be deployed to progressively improve the likelihood of the tree by alterations of b.

In [31] Felsenstein proposes a specific case of the general Expectation Maximization (EM) algorithm by Dempster et al. [27]. In fastDNAml [86] the faster converging Newton-Raphson method is implemented, whereas Gascuel and Guidon deploy Brent's [12] simple method for optimization of one-parameter functions in PHYML [39] which does not require function derivatives. It is important to note though, that a single phylogenetic tree topology might possess multiple local optima for distinct branch length and model parameter configurations [22].

3.4.3 Models of Base Substitution

One of the main advantages of maximum likelihood over other methods consists in that it explicitly allows for specification of a model of nucleotide substitution. Since sequences evolve from a common ancestor via base mutations one has to specify appropriate probabilities of nucleotide change which in the end will enable computation of the missing part in the previous Sections: the P_{ij}'s.

The Markov model assumed here means that the substitution probability of a base does not depend upon its history but only on the immediate predecessor. Furthermore, it is assumed that those probabilities are identic in the entire tree (homogeneous Markov process).

The concrete model is represented by a 4×4 matrix, which is usually named Q. This matrix provides the rate of change for all possible nucleotide mutations from bases

```
A|C|G|T -> A|C|G|T
```

during infinitesimal time dt. The current presentation of evolutionary models in this Section proceeds top-down, i.e. from the most general to the most special case of this matrix. The most general form of matrix Q is given below:

$$Q = \begin{bmatrix} -\mu(a\pi_C + b\pi_G + c\pi_T) & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu g\pi_A & -\mu(g\pi_A + d\pi_G + e\pi_T) & \mu d\pi_G & \mu e\pi_T \\ \mu h\pi_A & \mu j\pi_C & -\mu(h\pi_A + j\pi_C + f\pi_T) & \mu f\pi_T \\ \mu i\pi_A & \mu k\pi_C & \mu l\pi_G & -\mu(i\pi_A + k\pi_C + l\pi_G) \end{bmatrix}$$

Factor μ is the mean instantaneous substitution rate whereas $a, b, ..., l$ are relative base parameters which correspond to each of the possible 12 substitution

types between distinct bases. As already mentioned the variables $\pi_A, ..., \pi_T$ are the base frequencies of the 4 bases. The expression for mutations among equal bases, e.g. `A->A` is defined such that the sum of elements in the respective row equals to 0. However, this general model is not time-reversible, since

$$\pi_i P_{ij}(t) = P_{ji}(t)\pi_j$$

does not hold as different relative rates have been defined for symmetric entries of the matrix. As already mentioned the *pulley principle* is of outstanding importance due to computational reasons and requires time reversibility. Thus, the general model has to be restricted accordingly by defining symmetrical relative rates, i.e. setting $g = a, h = b, i = c, j = d, k = e, l = f$.

$$Q = \begin{bmatrix} -\mu(a\pi_C + b\pi_G + c\pi_T) & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu a\pi_A & -\mu(a\pi_A + d\pi_G + e\pi_T) & \mu d\pi_G & \mu e\pi_T \\ \mu b\pi_A & \mu d\pi_C & -\mu(b\pi_A + d\pi_C + f\pi_T) & \mu f\pi_T \\ \mu c\pi_A & \mu e\pi_C & \mu f\pi_G & -\mu(c\pi_A + e\pi_C + f\pi_G) \end{bmatrix}$$

This matrix represents the most general form of a time reversible nucleotide substitution process. The General Time Reversible (GTR) model has been proposed independently by Lanave et al. [67] and Rodriguez et al. [102] and is also implemented in RAxML.

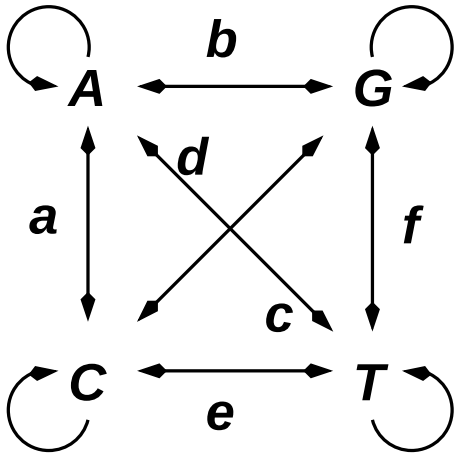

Figure 3.6: Schematic representation of the GTR model parameters

All simpler models can be obtained by further restricting the parameters of this matrix. An abstract and more readable representation of the transition types in the GTR model is provided in Figure 3.6. As among general tree scoring methods like

neighbor joining, parsimony, and maximum likelihood there is also a tradeoff in complexity between simple and elaborate models of nucleotide substitution, since more parameters have to be estimated and more terms have to be evaluated (see below).

A model which can be implemented in a more efficient way, e.g. in RAxML, is the HKY85 [42] model (Hasegawa, Kishino, Yano, 1985), which represents a good tradeoff between accurate modeling and speed. The HKY85 model allows for distinct base frequencies $\pi_A, ..., \pi_T$ but only for two classes of nucleotide substitutions: transitions and transversions. The rationale for this is that transitions occur between bases

A|G<->A|G & C|T<->C|T

which are chemically more closely related and transversion between

A|G<->C|T

Thus, transitions are assumed to occur at a different rate than transversions which is usually expressed by the transition/transversion ratio κ. The HKY85 model as depicted below is derived from GTR by setting $\pi_R = \pi_A + \pi_G$, $\pi_Y = \pi_C + \pi_T$, $a = c = d = f = 1$ and $b = e = \kappa$.

$$Q = \begin{bmatrix} -\mu(\kappa\pi_G + \pi_Y) & \mu\pi_C & \mu\kappa\pi_G & \mu\pi_T \\ \mu\pi_A & -\mu(\kappa\pi_T + \pi_R) & \mu\pi_G & \mu\kappa\pi_T \\ \mu\kappa\pi_A & \mu\pi_C & -\mu(\kappa\pi_A + \pi_Y) & \mu\pi_T \\ \mu\pi_A & \mu\kappa\pi_C & \mu\pi_G & -\mu(\kappa\pi_C + \pi_R) \end{bmatrix}$$

Two simpler models can be derived from HKY85 by either setting $\pi_A = \pi_C = \pi_G = \pi_T = 0.25$ to obtain the Kimura-2-Parameter (K2P [63]) model or allowing only one type of substitution rate, i.e. $a = b = c = d = e = f = 1$ in the Felsenstein 81 (F81 [31]) matrix. Thus, K2P can be represented by the following matrix:

$$Q = \begin{bmatrix} -\mu(\kappa + 2) & \mu\pi_C & \mu\kappa\pi_G & \mu\pi_T \\ \mu\pi_A & -\mu(\kappa + 2) & \mu\pi_G & \mu\kappa\pi_T \\ \mu\kappa\pi_A & \mu\pi_C & -\mu(\kappa + 2) & \mu\pi_T \\ \mu\pi_A & \mu\kappa\pi_C & \mu\pi_G & -\mu(\kappa + 2) \end{bmatrix}$$

The matrix of the F81 model is depicted below:

$$Q = \begin{bmatrix} -\mu(\pi_G + \pi_Y) & \mu\pi_C & \mu\kappa\pi_G & \mu\pi_T \\ \mu\pi_A & -\mu(\pi_T + \pi_R) & \mu\pi_G & \mu\kappa\pi_T \\ \mu\kappa\pi_A & \mu\pi_C & -\mu(\pi_A + \pi_Y) & \mu\pi_T \\ \mu\pi_A & \mu\kappa\pi_C & \mu\pi_G & -\mu(\pi_C + \pi_R) \end{bmatrix}$$

The most simple and ancient model is known as the Jukes-Cantor (JC69 [59]) model which has equal base frequencies, i.e. $\pi_A = \pi_C = \pi_G = \pi_T = 0.25$ and only one type of substitution, i.e. $a = b = c = d = e = f = 1$.

$$Q = \begin{bmatrix} -\mu\frac{3}{4} & \mu\frac{1}{4} & \mu\frac{1}{4} & \mu\frac{1}{4} \\ \mu\frac{1}{4} & -\mu\frac{3}{4} & \mu\frac{1}{4} & \mu\frac{1}{4} \\ \mu\frac{1}{4} & \mu\frac{1}{4} & -\mu\frac{3}{4} & \mu\frac{1}{4} \\ \mu\frac{1}{4} & \mu\frac{1}{4} & \mu\frac{1}{4} & -\mu\frac{3}{4} \end{bmatrix}$$

Generally, the various substitution models can be classified according to the number of different substitution types they allow for (minimum 1, maximum 6) and if they incorporate different or equal base frequencies. An overview over the hierarchy of the most common models is provided in Figure 3.7.

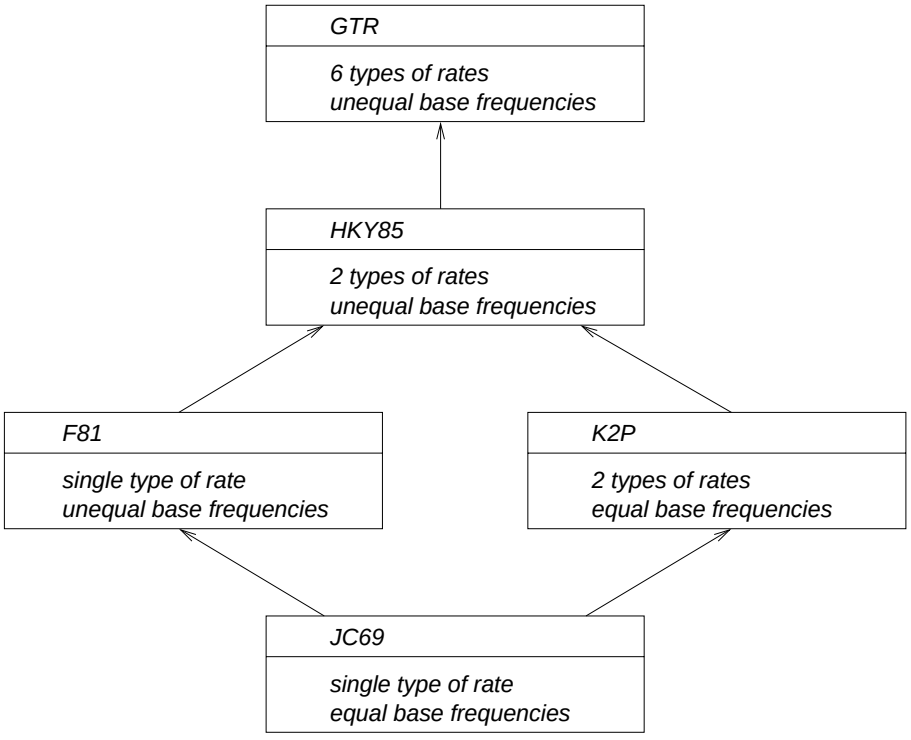

Figure 3.7: Hierarchy of probabilistic models of nucleotide substitution

It has however still not been specified how to obtain the values for $P_{ij}(t)$ since the matrices provide rates of change for dt. The substitution probability matrix can be calculated as:

$$P(t) = e^{Qt}$$

This expression can be evaluated by decomposing Q into eigenvectors and eigenvalues by application of well-established mathematical techniques. For most models there exist simple expressions for eigenvalues, which enable a direct analytical computation of required values [140].

In the formula (indicated below) of the Jukes-Cantor model (JC69) one has only to distinguish between two cases, the probability of observing a substitution or not.

$$P_{ij}(t) = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{-\mu t} & (\mathbf{i} = \mathbf{j}) \\ \\ \frac{1}{4} - \frac{1}{4}e^{-\mu t} & (\mathbf{i} \neq \mathbf{j}) \end{cases}$$

In addition, for the HKY85 model one has to further differentiate if a substitution represents a transition or a transversion. For convenience let $\pi_{sum}(j) = \pi_A + \pi_G$ if nucleotide j forms part of the purines (`A,G`) or $\pi_{sum}(j) = \pi_C + \pi_T$ if j is a pyrimidine (`C,T`). Furthermore let $S = 1 + \pi_{sum}(j)(\kappa - 1)$.

$$P_{ij}(t) = \begin{cases} \pi_j + \pi_j \left(\frac{1}{\pi_{sum}(j)} - 1 \right) e^{-\mu t} + \left(\frac{\pi_{sum}(j) - \pi_j}{\pi_{sum}(j)} \right) e^{-\mu t S} & (\mathbf{i} = \mathbf{j}) \\ \\ \pi_j + \pi_j \left(\frac{1}{\pi_{sum}(j)} - 1 \right) e^{-\mu t} + \left(\frac{\pi_j}{\pi_{sum}(j)} \right) e^{-\mu t S} & (\mathbf{i} \neq \mathbf{j}, \mathbf{Ts}) \\ \\ \pi_j \left(1 - e^{\mu t}\right) & (\mathbf{i} \neq \mathbf{j}, \mathbf{Tv}) \end{cases}$$

Finally, note that models of sequence evolution can be devised in an analogous way for protein sequences. The only difference is that those matrices will be of size 20×20.

Unfortunately, the flexibility which maximum likelihood provides through model choice induces the closely related problem of model choice. Some authors suggest one should select the model which yields the best likelihood for one specific tree. However, depending on the selected model and rate parameters optimal topologies for different models of nucleotide substitutions can differ significantly. Posada et al. [94] have written a computer program called Modeltest which seeks to find the appropriate model and optimize model parameters for a

tree built with Neighbor Joining. Although a neighbor joining tree will not correspond to a "good" maximum likelihood tree in most cases experimental results in [152] suggest that it is a practicable approach to optimize model parameters for a suboptimal tree as long as it is not too wrong, i.e. completely chosen at random. However, Modeltest has not attained great popularity due to its long execution times for large trees. Usually, the choice of the evolutionary model lies within the responsibility of the Biologist performing the phylogenetic analysis. If nothing is known about the model which best fits the data the GTR model represents a good choice if substitution parameters (the 6 rates) are optimized by the respective computer program.

Another issue within this context is that of rate variation among sites in alignments, since mostly not all sites evolve at the same speed. This becomes particularly severe if e.g. alignments from different genes have been concatenated to form one large alignment with potentially stronger phylogenetic signal. In this particular case apart from different substitution rates even different models of nucleotide substitution might be required for distinct regions of the alignment. It has been demonstrated, e.g. in [152], that maximum likelihood inference under the assumption of rate homogeneity can lead to erroneous results if rates vary among sites.

Rate heterogeneity among sites can be simply accommodated by adding an additional per-site (per alignment column) rate component $r_k, k = 1, ..., m$ to the $P_{ij}(t)$, where m is the length of the alignment. For example in the JC69 model the probability of change would be:

$$P_{ij}(t) = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{-\mu r_k t} & (\mathbf{i} = \mathbf{j}) \\ \\ \frac{1}{4} - \frac{1}{4}e^{-\mu r_k t} & (\mathbf{i} \neq \mathbf{j}) \end{cases}$$

Typically, such an assignment of rate categories to sites corresponds to some functional classification of sites. Moreover, this is usually performed based on some *a priori* analysis of the data. G. Olsen has developed a program called DNArates [85] which performs a maximum likelihood estimate of the individual per site substitution rates for a given input tree.

A computationally significantly more complex form of dealing with heterogeneous rates, due to the fact that one additional parameter has to be estimated, consists in using either discrete or continuous stochastic models for the rate distribution at each site. In this case every site has a certain probability of evolving at any rate contained in a given probability distribution. For example a concrete distribution of the likelihood for one site is obtained by summing over all products of likelihoods for the discrete rates times the probability from the distribution.

In the continuous case likelihoods must be integrated over the entire probability distribution.

The most common distribution types are the continuous [151] and discrete [152] Γ distributions.

3.5 Bayesian Phylogenetic Inference

Bayesian phylogenetic inference is relatively new compared to parsimony and maximum likelihood methods, since its application emerged in the mid-90ies [77, 78, 97]. Recently, it has experienced great impact [49], mainly through the release of an efficient program called MrBayes [50] by Huelsenbeck et al.

Holder et al. provide an interesting review of traditional and bayesian approaches in [47] . The fundamental construct of bayesian analysis are posterior probabilities, i.e. estimated probabilities which are based on some model (prior expectation). Those are estimated after acquisition of some knowledge about the data.

Bayesian analysis is closely-related to maximum likelihood from a computational perspective since the method used to calculate values for individual topologies corresponds exactly to maximum likelihood. The main difference however is that while maximum likelihood strives to find the tree which maximizes the probability of observing the data given the tree, bayesian inference searches the tree which maximizes the probability of the tree under the data and the model of evolution. Thus, in the bayesian case likelihoods are scaled to true probabilities, such that the sum over the probabilities of all potential tree topologies is 1.0. In contrast to maximum likelihood searches, bayesian inference searches for a set of best trees, instead of a single optimal tree. Due to the vast amount of tree topologies it is not feasible to compute posterior probabilities for those trees analytically. Instead, a sampling technique which is known as Metropolis Coupled Markov Chain Monte Carlo (MC^3 [80]) algorithm is implemented e.g. in MrBayes to sample individual trees from the distribution of posterior probabilities.

A typical bayesian analysis commences with a random or user-specified starting tree and the specification of the model of evolution along with the respective model parameters. Those input parameters represent the common initial state of all Markov chains (recommended number of chains: 2–4). Note, that in the manual of MrBayes Huelsenbeck suggests that it is preferable to use a random starting tree in order not to make any a priori assumptions about the correct tree. However, in [125] and in Figure 6.8 (p. 103) it is demonstrated that the specification of user-starting trees computed with maximum likelihood can have positive impact on bayesian analysis and chain convergence speed.

The chains generally perform the following operation: Each chain conducts a separate search and carries out state transitions between alternative *model-parameter/topology* combinations $mt_i \rightarrow mt_{i+1}$ by applying *minor* topological changes and/or *minor* alterations of the model parameters. Note, that due to the fact that the mt_i, mt_{i+1} of a chain differ only marginally, the evaluation of the probability is generally significantly faster than for maximum likelihood searches. This is due to the fact that a large amount of values in the tree remains unchanged and can therefore be re-used.

Another important property of this model, which should help to avoid local optima, is that backward moves towards topologies with lower scores are possible since m_{i+1} is only accepted when $Probability(m_{i+1}) > Random(0,1)$ where $Random(0,1)$ is an equally distributed random number between 0 and 1. Otherwise the chain remains in state mt_i and a different alteration is tested in the following step $i+2$. In addition to this property, Metropolis-Coupling Markov Chains by occasionally exchanging mt_i's between distinct chains, should further reduce the risk to get trapped in local optima [80].

This process of state transitions, which are called generations in the related terminology, has to be repeated thousands of times until the chains have reached a stable state (converged). The most important part of an MC^3 implementation of bayesian phylogenetic inference is the design of a sophisticated *tree proposal mechanism* which greatly influences convergence speed and quality of final results.

An abstract representation of the basic MC^3 method is provided in Figure 3.8.

The most critical issue of bayesian phylogenetic inference (the $64.000 question according to Huelsenbeck) is to determine at which point of time the chains have reached stable values. This is usually assessed by plotting the best log likelihood value from the respective chains over generation numbers. To illustrate the dangers and surprises which are intrinsic to MC^3 phylogenetic reconstruction one has to consider Figure 3.9. In the upper part of Figure 3.9 the chain seems to have reached stationarity, whereas in the lower part the likelihood might continue to increase after a long interval of apparent stationarity. The lower diagram of this Figure also outlines how a reference likelihood computed with a standard ML program can support decisions about chain convergence. A discussion of potential pitfalls of bayesian inference which also includes some methodological aspects, can be found in [48]. A real-world example for failure of bayesian analysis is provided in Figure 6.5 (p. 101).

An important advantage of bayesian analysis which has mainly triggered its great popularity is that—in case of convergence—it generates a collection of "good" trees and allows for estimation of their posterior probabilities. In addition, it enables assignment of consensus-based clade probability values (see Section 3.6). Moreover, these posteriors are based upon the integrated likelihood,

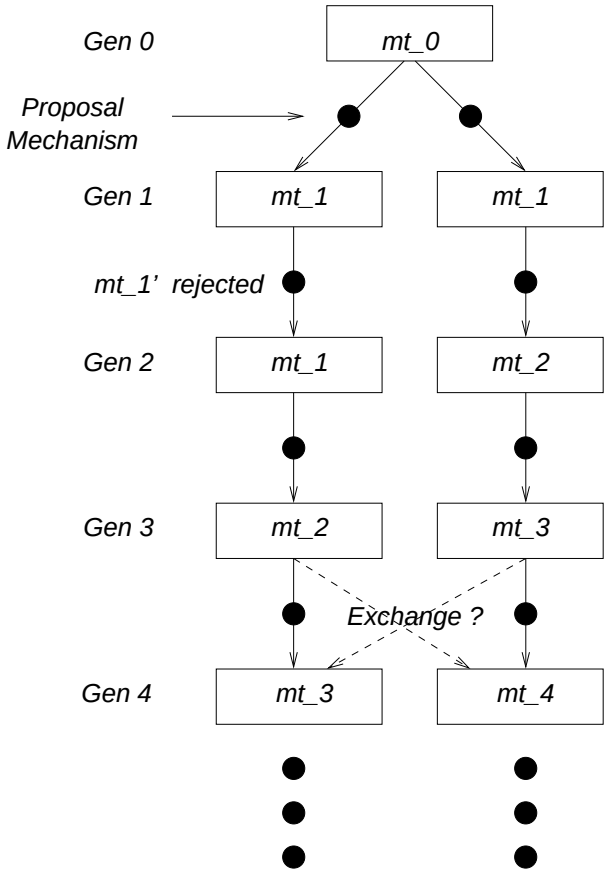

Figure 3.8: Abstract representation of a bayesian MC^3 tree inference process with two Metropolis-Coupled Markov Chains

i.e. the likelihood averaged over model parameters and branch length values. Thus, bayesian programs take into account uncertainties that standard maximum likelihood methods are not able to handle.

3.6 Measures of Confidence

An important issue which arises when building production trees based upon real data with distance-based, parsimony, or maximum likelihood methods is how to obtain confidence into the final result since the true tree is usually not known. Normally, a maximum likelihood analysis will return a suboptimal tree, due to the more or less exhaustive heuristics used.

A broadly accepted method to assign confidence values to a specific, biologically significant tree topology is to compute several distinct trees (typically 100–

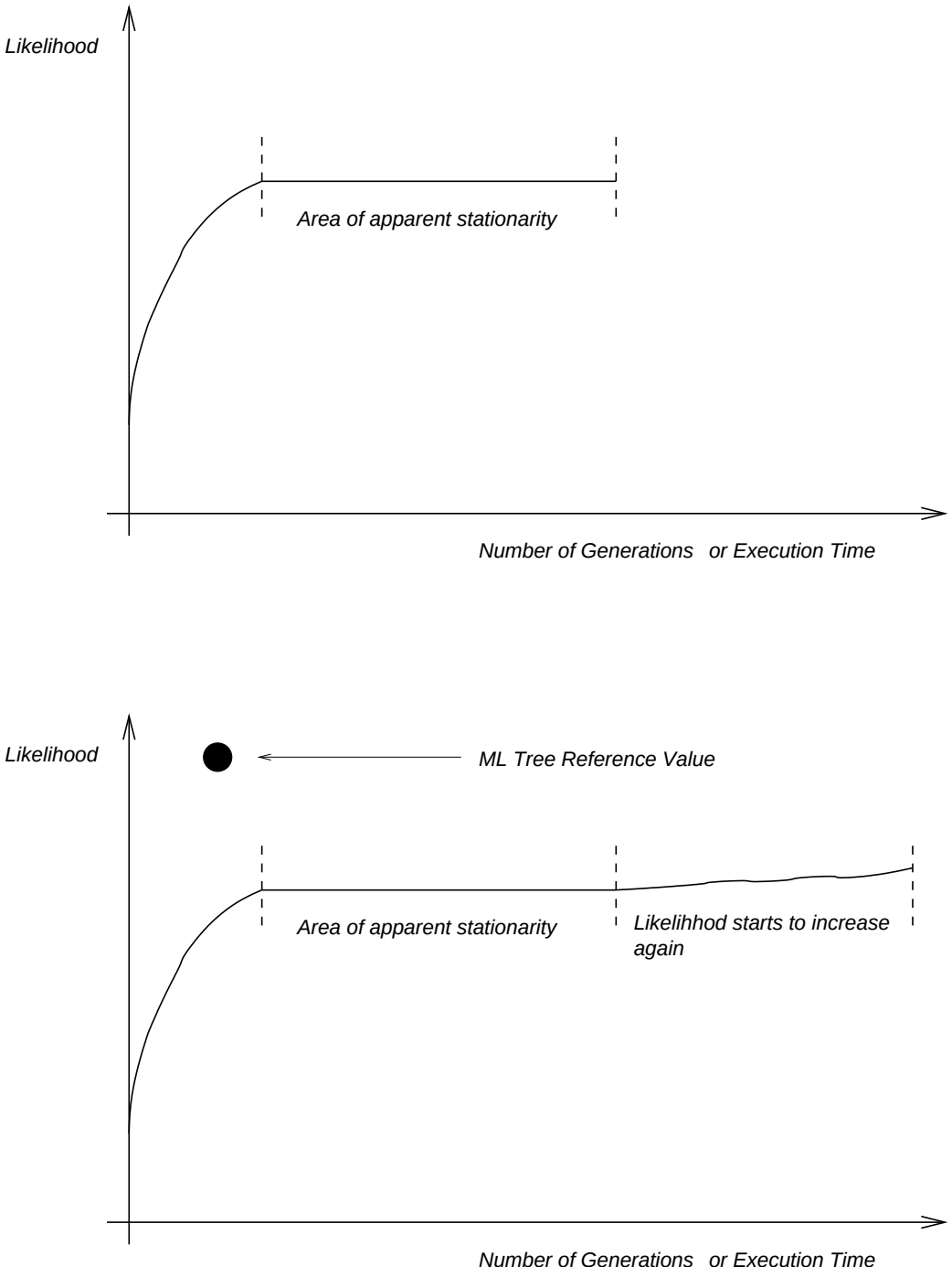

Figure 3.9: Outline of the MCMC convergence problem

1.000 trees) for the same input data and *exactly* the same model of evolution. Within this context confidence into the tree refers to the reliability of the tree topology itself rather than of concrete branch lengths.

After computation of this set of trees, a final consensus tree is built using e.g. the consense [57] program included in Felsenstein's PHYLIP package.

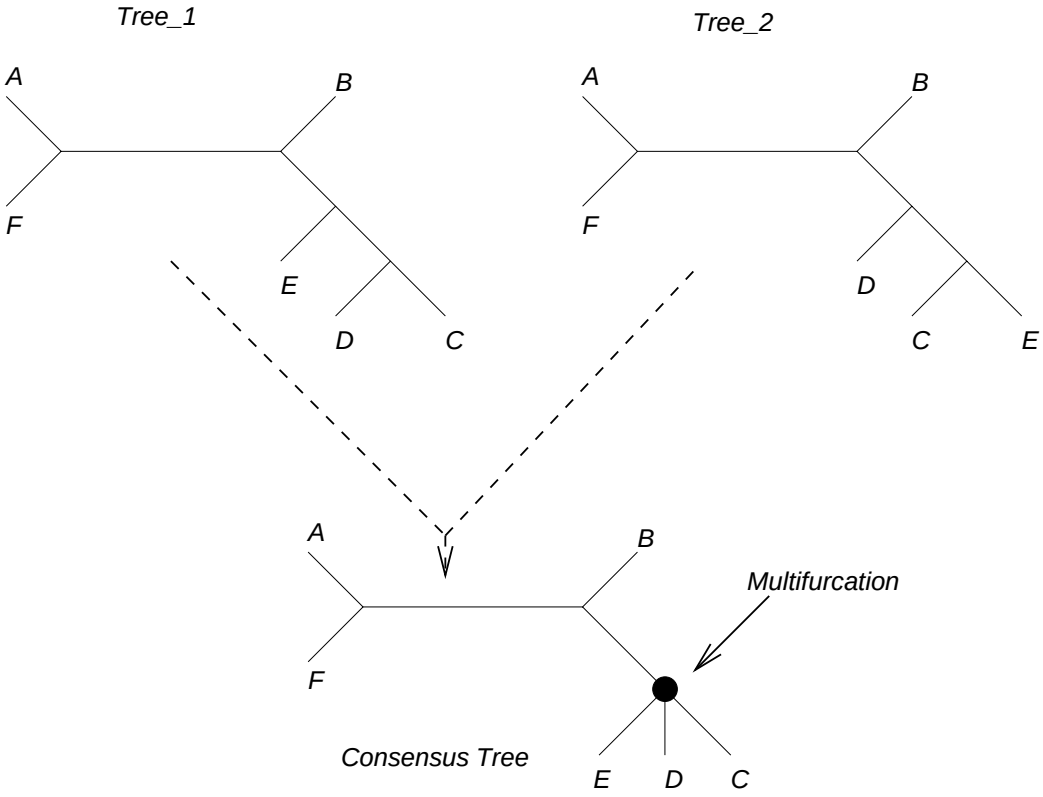

Figure 3.10: Example of an unresolved (multifurcating) consensus tree

Consensus tree building methods assign a simple confidence value to each clade of the output tree. This value indicates in how many of the trees the specific clade appeared. Thus, given a set of e.g. 100 trees an inner node showing a value of 95 means that the clade appeared in 95% of all trees and confidence into that part of the final tree is high. Such a node is called well-resolved. If a node, divided the tree into identical sequence subsets in only 20% of all trees confidence is comparatively low (low/bad resolution).

Most consensus programs can produce strict, majority rule, as well as extended majority rule consensus trees.

Strict consensus trees only include bifurcating nodes if they form part of all trees and construct multifurcations for non-resolvable nodes, i.e. application of

this rule does usually not yield binary trees. The problem with this approach is that those multifurcating (unresolved) trees have to be transformed into binary ones by some additional algorithmic step.

Note, that the problem of optimal taxon addition to an existing tree is also a computationally challenging problem. In fact, it has been demonstrated in [10] that it is NP-complete under the parsimony criterion and thus most probably under maximum likelihood as well.

Majority rule consensus methods accept clades if they appear in the majority of input trees, i.e. majority rule consensus trees generally also include some multifurcations but less than strict consensus trees. Finally, the extended majority rule allows for computation of strictly bifurcating consensus trees. Figure 3.10 provides a simple example of two distinct tree topologies containing the same 7 organisms `A,B,C,D,E,F` that can not be resolved under the strict or majority rule consensus methods. The extended majority rule would either yield Tree_1 or Tree_2 as consensus tree.

Up to this point no method to compute 100 or even 1.000 distinct trees for the same input data set has been described.

If the tree building algorithm incorporates some non-determinism, such as RAxML (see Section 4.2, pp. 62), the program can easily be executed several times and will render distinct output trees. Another good choice consists in using different programs, a practice which should be adopted anyway since there are significant differences in the exhaustiveness of searches.

However, if the program is deterministic, i.e. always yields the same output tree for the same input data, the only way to produce distinct final results is to alter the input data by a technique called bootstrapping. Bootstrapping an alignment is a fairly simple operation which consists in randomly selecting columns of the original alignment and placing them into the new alignment, i.e. some columns can be deleted and others replicated. The only restriction is that the new alignment must have the same length as the original one. The individual runs for generating the required set of trees are then executed with distinct bootstrapped alignments. An example of an initial alignment and a respective bootstrap for that alignment is provided below:

```
initial alignment       bootstrapped alignment

12345678                28245786

AATGGGTT                ATAGGTTG
CC--GGGC                CCC-GGCG
AATGG-CA                AAAGGCA-
AAG-G-CC                ACA-GCC-
```

Note, that bootstrapping is also important in a statistical context to assess robustness of the tree under slightly altered input data.

The main problem of such an analysis with maximum likelihood consists in the immense computational cost, which does not make the approach feasible for large alignments.

However, there exist some models and programs like treepuzzle and MrBayes (see [50, 137] and Section 3.9.2) which include consensus-based methods and statistical approaches for computing confidence values.

3.7 Divide-and-Conquer Approaches

From a computer scientist's point of view an obvious idea to accelerate computations of large trees would be to deploy a divide-and-conquer approach. Given the at least quadratic complexity of most tree building algorithms (see Section 3.9.1) in the number of organisms (e.g. at least $2n^2 - 9n + 8$ different topologies are analyzed in one single rearrangement step on the full tree with a rearrangement setting of 1 [31]), splitting up the alignment into smaller subalignments for which respective subtrees can be computed more rapidly appears to be a reasonable approach.

Such a divide-and-conquer approach consists of four basic computational steps which are outlined below:

1. Divide the original alignment into overlapping subalignments
2. Infer a phylogenetic tree for each subalignment
3. Merge the overlapping subtrees into a single tree, commonly called supertree, using dedicated supertree construction methods
4. Resolve potentially multifurcating nodes of the supertree

Though the approach appears to be fairly simple, it faces various difficult problems: The requirement to calculate overlapping subtrees is imposed by the supertree building operation in step 3. The subalignments need to overlap, i.e. share some common sequences in order to provide a hint where they should be reconnected. Furthermore, those subalignments should be selected intelligently such as to facilitate the subtree reconstruction and supertree construction steps as well as to reduce the number of multifurcating nodes which require resolution. For distance- and parsimony-based models so called disk covering methods DCM [53] and DCM2 [54] have been devised for this purpose.

A quite distinct problem concerning DCM and DCM2 is that the authors are very protective about their codes, such that neither executables nor source code

are available to other researchers. To the best of the author's knowledge there is currently no method available to subdivide large alignments based on likelihood metrics.

Despite the fact, that inference of individual subtrees (step 2) can be carried out easily by using e.g. some standard parsimony or maximum likelihood program, supertree construction is still a relatively new field.

The most common methods to construct supertrees are the Matrix Representation with Parsimony [7, 95] (MRP) and the Strict Consensus Merger (SCM). The SCM-method is a dedicated method for merging subtrees obtained by DCM-based decompositions.

In one of the few comparative surveys on the performance of supertree against integral tree building methods [104] DCM decompositions in combination with SCM (DCM+SCM) have shown to perform better than DCM+MRP as well as random decompositions with either SCM or MRP in terms of tree scores and inference times (differences in parsimony scores ranging typically between factor 1.002-1.004 and inference times between 1.2 and 1.4).

However, the best combination according to this survey, i.e. DCM2+SCM, does not perform significantly better than integral methods in terms of final tree scores and inference times for parsimony analyses. Furthermore, in the final step of the DCM2+SCM analyses conducted within the framework of this survey, the multifurcating nodes were resolved by using a relatively simple and fast procedure from PAUP [104]. As already mentioned in Section 3.6 resolving multifurcating trees is a complex problem which has not received enough attention although it is very important within this context. Thus, this last step requires additional investigation and algorithmic refinement.

Another current issue is that the performance of DCM2+SCM and integral methods on large real and simulated data sets requires further comparative surveys. Furthermore, the resolution of multifurcations means that at the end the final supertree will still require to be globally optimized which justifies the need for integral phylogeny programs that are able to handle large trees especially in regard to memory requirements.

Thus, supertree computation is a direction of research which will gain momentum as available sequence data grows and many important issues remain unresolved, in particular for maximum likelihood.

3.8 Testing & Comparing Phylogeny Programs

When one designs a new algorithm or model the question arises how to assess the performance of the new program.

The basic quantitative performance parameters are execution speed and final tree quality. Furthermore, the memory requirements of maximum likelihood or bayesian programs can become a limiting factor (see Section 6.6, pp. 109 and [126]) for computation of large phylogenies consisting of more than 500 to 1.000 organisms. The qualitative properties of programs include e.g. the ability to handle different types of input data such as DNA and protein sequence alignments, sophisticated models of evolution, or model parameter optimization methods.

The major problem of performance analysis is the assessment of tree quality. It would not be so difficult if true trees or optimal trees according to the criterion were known. Note, that the true tree need not always be the optimal tree. For example in experiments conducted on simulated data (see below) RAxML and PHYML frequently encountered trees with better likelihoods than the likelihood of the synthetic true tree.

Since the problem appears to be NP-complete optimal reference topologies according to the selected criterion can only be computed exhaustively for trees with up to 15 or 20 sequences. Furthermore, an evaluation based on such small trees might not provide a clear image since any heuristics will still explore a relatively large fraction of search space compared to the fraction explored for larger trees ($>$ 100 taxa) and are thus more likely to converge to the true tree.

Usually, for real world data the true tree is not known, except for phylogenies of organisms with evident phenomenological characteristics such as most animals or plants. Such kind of trees can be established by "traditional" non-computational methods which nonetheless still represent only an hypothesis. In case of maximum likelihood or bayesian inference, it has to be assumed that the best-known tree (in terms of its likelihood score) for a real world data set, represents the most plausible result. Note, that small deviations ($<$ 1%) between final likelihood values are significant due to the asymptotic convergence of likelihood values over time. Furthermore, those apparently small differences in likelihood show to be significant when e.g. the Shimodaira-Hasegawa [111] likelihood ratio test is applied to them which provides a measure for the significance of this δ. A particularly extreme example of asymptotic convergence is outlined in Figure 6.3 on page 98 for a 500 taxon alignment.

Thus, one solution to evaluate program performance for codes using the *same scoring function* is to publish a set of real world alignments including the respective best-known trees and computation times with various programs under a fixed set of parameters (e.g. for maximum likelihood: model of evolution, transition/transversion ratio etc.). The distribution version of RAxML includes the first phylogenetic benchmark set of 9 real world alignments comprising 101 up to 1000 sequences (available at: WWW-BODE.CS.TUM.EDU/~STAMATAK/RESEARCH.HTML).

A completely different approach consists in the utilization of synthetic (simulated) data which can also be used to compare programs based on *different scoring functions*. The simulation process starts by building a randomized tree under some biologically reasonable restrictions using e.g. r8s by M.J. Sanderson [107]. Thereafter, a synthetic alignment of a predefined length is generated that fits to the tree under a specified model of evolution. One of the most widely-used programs for generation of synthetic alignments is Seq-Gen [96]. In that way, an alignment becomes available for which the true tree is known. The phylogeny program under consideration is then executed for this synthetic alignment. Finally, the topologies are compared using the Robinson-Foulds rate [101] which provides a relative measure for topological dissimilarity. Most comparative surveys of phylogeny programs use synthetic data. Despite the importance of synthetic data for assessing the quality of different phylogeny methods such as neighbor joining, parsimony, and maximum likelihood, substantial differences between different heuristics for maximum likelihood or bayesian searches might not become apparent.

This is due to the fact that synthetic data creates the illusion of a perfect world: the model of evolution is known a priori and the alignment does not contain gaps or sequencing errors and thus a strong phylogenetic signal. As a consequence the inference process for synthetic data converges faster and more steadily to a near-optimal tree and is less dependent on the heuristics and the phylogeny method deployed. During analyses of simulated data with RAxML it was observed repeatedly that the relative differences in likelihood values between parsimony starting trees and final trees were significantly smaller than for real data sets.

Thus, in order to obtain a complete and more objective image of program performance, the combination of synthetic and real world data experiments is mandatory.

3.9 State of the Art Programs

This survey of related work is limited to programs using statistical models since they have repeatedly shown to be the most accurate methods for phylogenetic analysis. The focus is on maximum likelihood as well as bayesian methods, and currently available parallel implementations.

The site maintained by Felsenstein [93] lists most available programs for phylogenetic inference.

3.9.1 Algorithms for Tree Building & Sequential Codes

In general, heuristic maximum likelihood searches can be implemented in three basic ways:

Firstly, they can start from scratch and insert organisms progressively into the tree, potentially applying some additional optimizations to the intermediate trees (intermediate refinement).

Secondly, they can start with an initial global tree already containing all organisms built by a simpler method such as parsimony, neighbor joining or even with a random tree. The likelihood of such a starting tree is then progressively optimized by application of a standard pattern of topological changes.

Thirdly, in a similar way to supertree methods (see Section 3.7) a program can first construct a set of, or even all, small trees of a fixed size (usually 4-taxon trees which are also known as quartets) and thereafter reconstruct the whole tree by application of consensus tree methods to the set of quartets.

3.9.1.1 Progressive Algorithms

The most widely used progressive algorithm is stepwise addition proposed initially by Felsenstein in [31] and implemented in dnapars [93]. It starts with the only possible three taxon tree t_3 and then progressively inserts the remaining $n-3$ taxa into the tree, in the order they appear in the alignment. A new branch is connected to each new taxon $k+1$ and possible insertions into *all* of the $2k-3$ branches of the currently best tree t_k comprising k sequences are tested. After each insertion the optimal branch lengths and the likelihood value of the such generated new topology t_{k+1} containing organism $k+1$ are computed. The tree with the best likelihood value among the $2k-3$ analyzed topologies of the addition test phase is then used to insert taxon $k+2$. The algorithm terminates when taxon n has been inserted into tree t_{n-1}. This basic process is outlined in Figure 3.11.

In addition, the tree can optionally be further refined by application of rearrangements to the intermediate trees $t_k, k < n$ (local rearrangements) and/or the final tree t_n (global rearrangements).

The various topological alteration mechanisms (including subtree rearrangements) to improve the likelihood of a given topology will be discussed in more detail in the following Section 3.9.1.2.

As already mentioned stepwise addition is implemented in dnapars. A more recent implementation of this algorithm is fastDNAml [86], which uses a faster converging mathematical method for branch length optimization and represents an extremely efficient implementation on a technical level. Furthermore, fastDNAml implements the optional *quickadd option* which optimizes only the three branches adjacent to the insertion point of sequence $k+1$ during the addition test phase. This enables a rapid prescoring of alternative topologies. Furthermore, since all other branches of tree t_k remain unchanged through this procedure, a large number of likelihood vectors can be reused if the tree is traversed in an intelligent manner. This alternative method is depicted in Figure 3.12. Branches

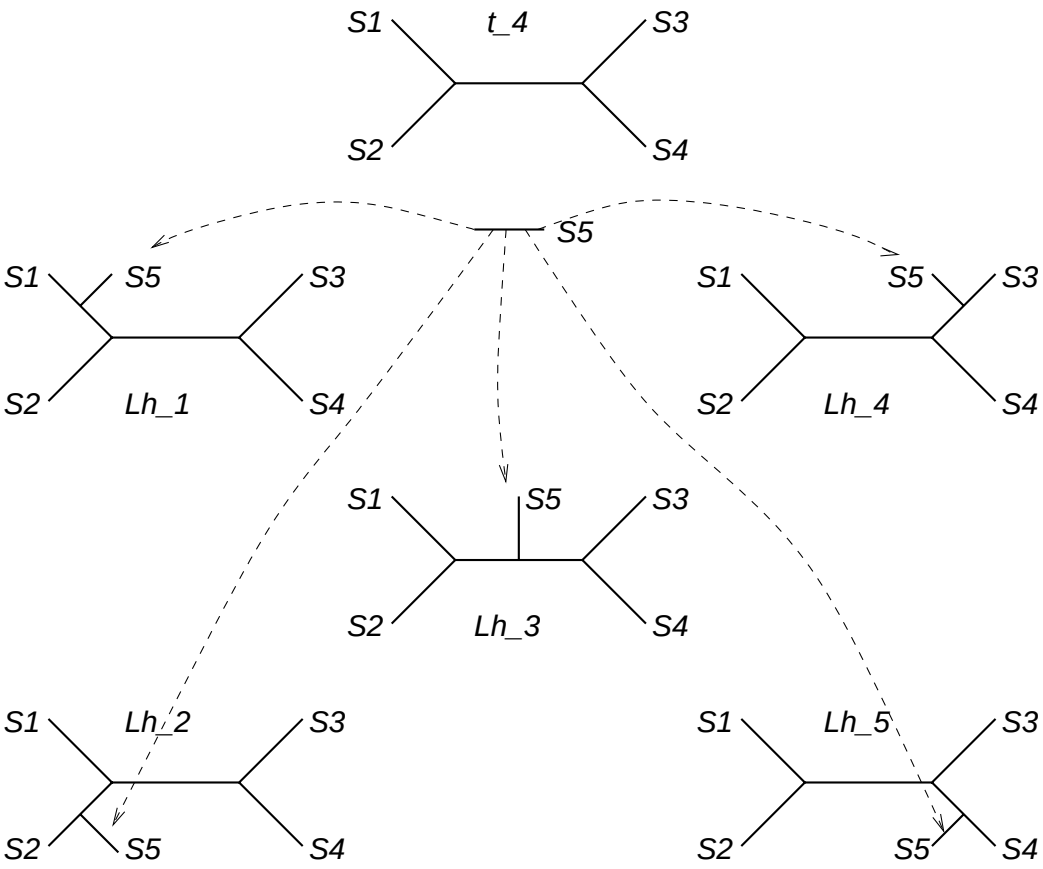

Figure 3.11: Example for stepwise addition

which are optimized are indicated by bold dotted lines, whereas inner nodes at which the likelihood vector is not updated are marked with circles. The direction of the insertion traversal is indicated by thin dotted arrows.

The last representative from this class of algorithms which will be mentioned here is TrExML [149] which has been derived from fastDNAml. TrExML implements a more exhaustive tree search than fastDNAml, since an optimal small tree comprising the first 5–10 sequences (specified by program parameter a for all) is computed exhaustively. Furthermore, instead of maintaining only one currently best tree t_k, TrExML maintains a list of such trees. Sequence $k+1$ is then inserted into all trees of that list.

An important property of the stepwise addition algorithm is that the final result depends on the input order of sequences. In order to generate a set of different final trees one can generate randomized input order permutations, a technique

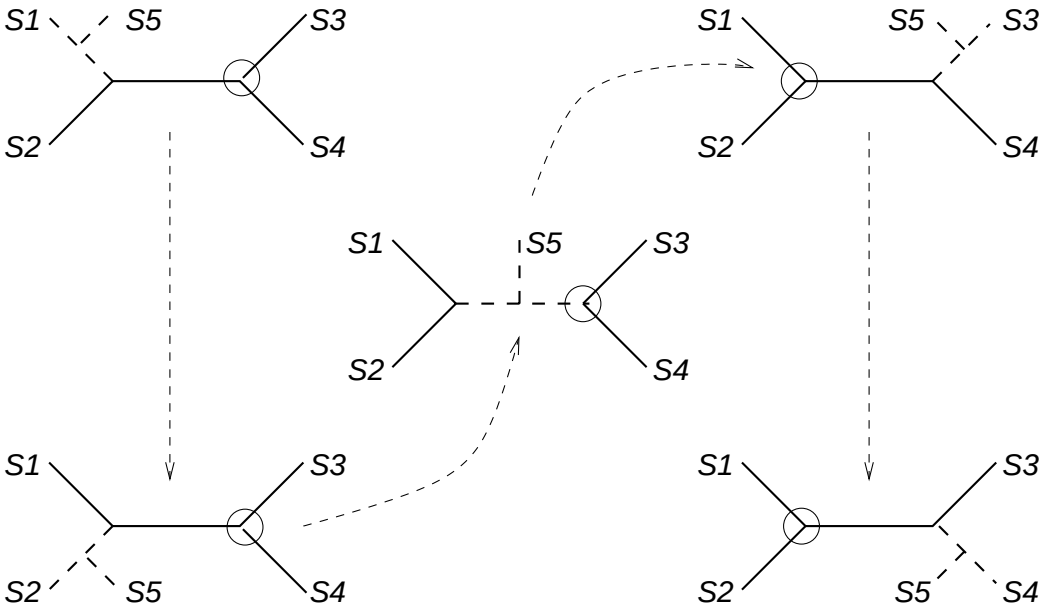

Figure 3.12: Example for stepwise addition with quickadd option

also known as jumbling. The evident idea to compute a *good* input permutation in order to obtain better final trees has largely failed [4, 62].

3.9.1.2 Global Algorithms

As already mentioned these algorithms start with a global initial tree which already contains *all* organisms of the alignment. Those starting trees are typically computed by simpler and faster methods like neighbor joining or parsimony.

A rather special case is star decomposition which is implemented e.g. in PAML [90], where the computation starts with a single inner n-ary center node to which all sequences are directly connected. This tree is then progressively refined to an unrooted binary tree.

The most important part of such algorithms is the topological alteration mechanism, i.e. a standard tree modification pattern which is repeatedly applied to the currently best tree until no improved tree in terms of likelihood value can be found.

The three most common techniques are **subtree rearrangements** (also known as subtree pruning and regrafting), **Nearest Neighbor Interchange** (NNI) and **Tree Bisection & Reconnection** (TBR).

Subtree rearrangements are carried out by removing each subtree of the current tree at a time and re-inserting it into different branches of that tree. Usually, a subtree is re-inserted into the surrounding branches of its deletion point. The depth up to which the subtree will be re-inserted is specified by the rearrangement

setting, i.e. a rearrangement setting or rearrangement stepwidth of 1 means that the subtree is only inserted into the immediate neighbors. Higher rearrangement settings yield significantly better trees but are computationally more expensive at the same time. In a worst-case scenario, for a rearrangement stepwidth of $1 : 2^2$ alternative trees have to be evaluated per subtree, for a stepwidth of $2 : 2^2 + 2^3$ topologies per subtree and for a stepwidth of $n : 2^2 + 2^3 + ... + 2^{n+1}$ respectively. Usually every alternative rearranged topology is entirely branch-length optimized. An example for subtree rearrangements is provided in Figure 3.13.

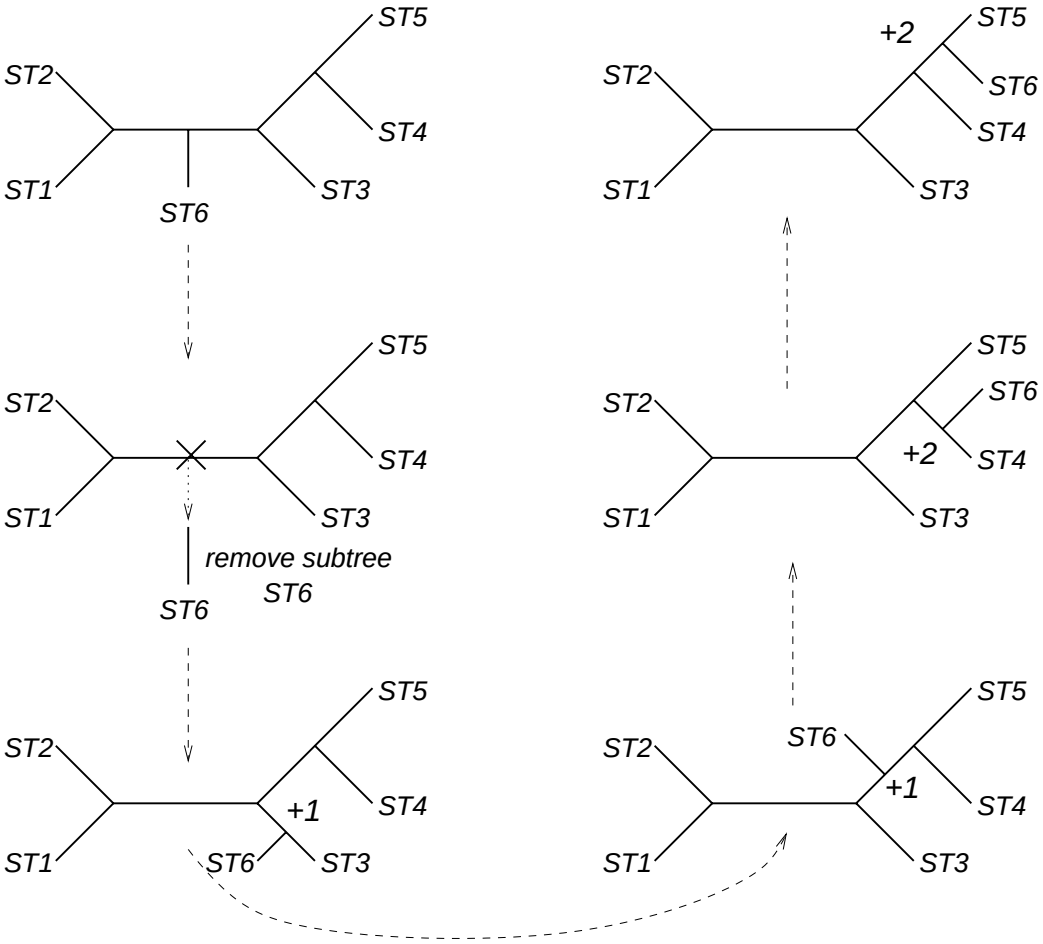

Figure 3.13: Possible rearrangements of subtree ST6

Tree bisection & reconnection initially splits the tree into two subtrees by erasing an inner branch. Thereafter, it reconnects them by placing a branch between all branch pairs of the two subtrees. TBR is outlined in Figure 3.14.

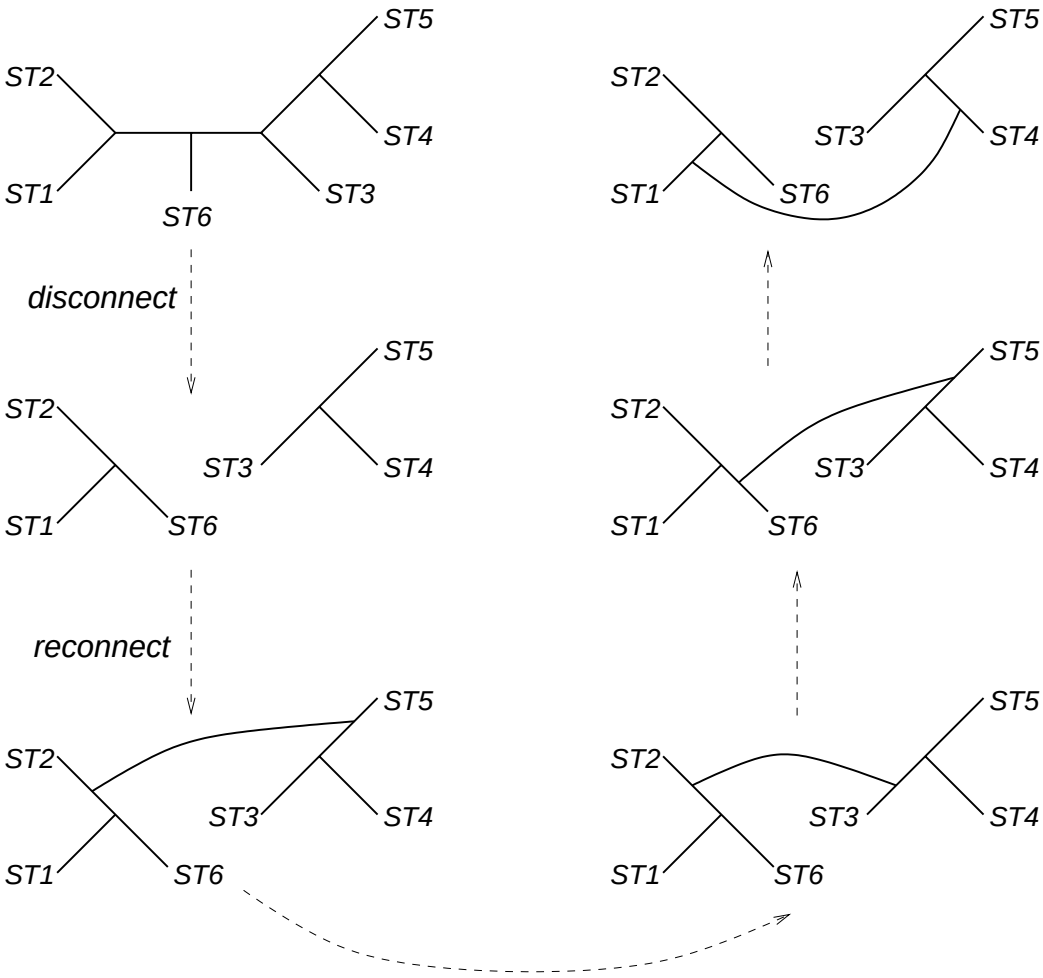

Figure 3.14: A possible bisection and some possible reconnections of a tree

Nearest neighbor interchange exchanges the 4 subtrees located at every inner branch of the tree, i.e. interchanges subtree positions. A modified version of this algorithm, which is depicted in Figure 3.15 is implemented in PHYML [39].

Some genetic algorithms which have recently been proposed [71, 72, 112] traverse tree-space by (sometimes randomly) perturbing a population of trees via modifications of branch lengths and topologies and combining the best trees until an optimum is reached. Moreover, due to the fact that these methods build a number of trees they enable approximation of posterior probabilities of trees or clades. Furthermore, they allow backward steps, since they occasionally accept trees with lower likelihood values, to avoid getting caught in local optima.

However, nearly every algorithm which globally optimizes tree topologies (not necessarily branch lengths), implements a variation of those three basic topology

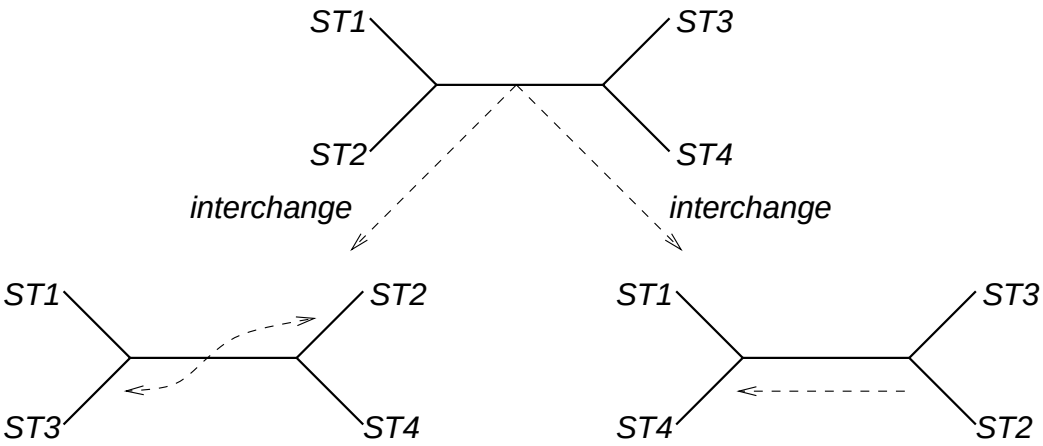

Figure 3.15: All possible nearest neighbor interchanges for one inner branch

alteration (perturbation) procedures: be it the tree proposal/perturbation mechanism of a bayesian or a genetic implementation, or be it hill climbing heuristics.

3.9.1.3 Quartet Algorithms

Quartet algorithms initially calculate the likelihood values of all 3 possible 4-taxon trees, or simply quartets, for all $\binom{n}{4}$ combinations of 4 taxa from the input alignment.

Thereafter, quartets and single sequences are integrated during the *puzzling step* into several potential final trees. Note, that those intermediate trees are highly dependent from the input order of quartets and sequences.

In the final *consensus step* a majority rule consensus tree is built from the intermediate set of trees.

The most popular implementation of quartet puzzling is a program called treepuzzle [137] which is popular among biologists since the final tree includes a consensus-based measure of confidence.

However, more recently quartet puzzling algorithms have lost momentum due to inacceptable inference times and comparatively poor final results [99]. Therefore, quartet-puzzling is not applicable to inference of large phylogenetic trees with more than 100 organisms.

3.9.2 Performance of Sequential Codes

A recent comparative survey of widely-used state of the art phylogeny programs using statistical approaches such as fastDNAml, MrBayes, PAUP [92], and treepuzzle [137] has been conducted by T.L. Williams et al. [147]. The most important

result of this paper is that MrBayes outperforms all other analyzed phylogeny programs in terms of speed and tree quality.

However, this survey is entirely based on synthetic data. As outlined in Section 3.8 and demonstrated through experimental results in this thesis as well as in [124] additional experiments with real data can lead to different conclusions and a more differentiated image. Furthermore, the largest alignments contained only 60 sequences. Thus, the results of this survey do not necessarily apply to inference of large trees and real data sets. In addition, this paper does not cover genetic algorithms which generally converge faster than MrBayes [39].

More recently, Guidon and Gascuel published a paper about their new program PHYML [39], which is very fast and outperforms other recent approaches including bayesian and genetic algorithms. The program MetaPIGA by Lemmon et al. [71] represent the currently most efficient genetic algorithm for phylogenetic analysis.

PHYML is a "traditional" maximum likelihood program which seeks to find the optimal tree in respect to the likelihood value and is also capable of optimizing model parameters. The PHYML publication includes a comparative survey based on both, large real world alignments (218 & 500 taxa), as well as 50 synthetic 100 taxon alignments.

A comparative analysis of MrBayes, RAxML, and PHYML including 9 real world (101–1000 sequences) as well as the same 50 synthetic data sets used in [39] is provided in [124] and Section 6.4 (pp. 94).

Thus, to the best of the author's knowledge MrBayes and PHYML are currently the fastest and most accurate representatives of bayesian and "traditional" approaches to phylogenetic tree inference using statistical models of nucleotide substitution. Therefore, the focus is on those two programs for assessing performance of RAxML in this thesis.

One should however be careful when comparing bayesian with maximum likelihood methods due to subtle differences in the statistical models as outlined in Figure 3.16, Chapter 6 (pp. 87), and [47]. This is due to the fact that bayesian methods optimize topologies by integration of the likelihood over a broader range of model parameters, whereas maximum likelihood methods search for the peak likelihood of all topologies with usually fixed or restricted model parameters. Thus, a bayesian analysis might not yield a tree with a peak likelihood value as obtained from a maximum likelihood search but a topology which is supported by a broader range of model parameter combinations. A schematic representation of this difference between bayesian and likelihood methods is provided in Figure 3.16 for the likelihood of a hypothetic final tree obtained by likelihood and bayesian analysis over some model parameter x, e.g. the transition/transversion ratio.

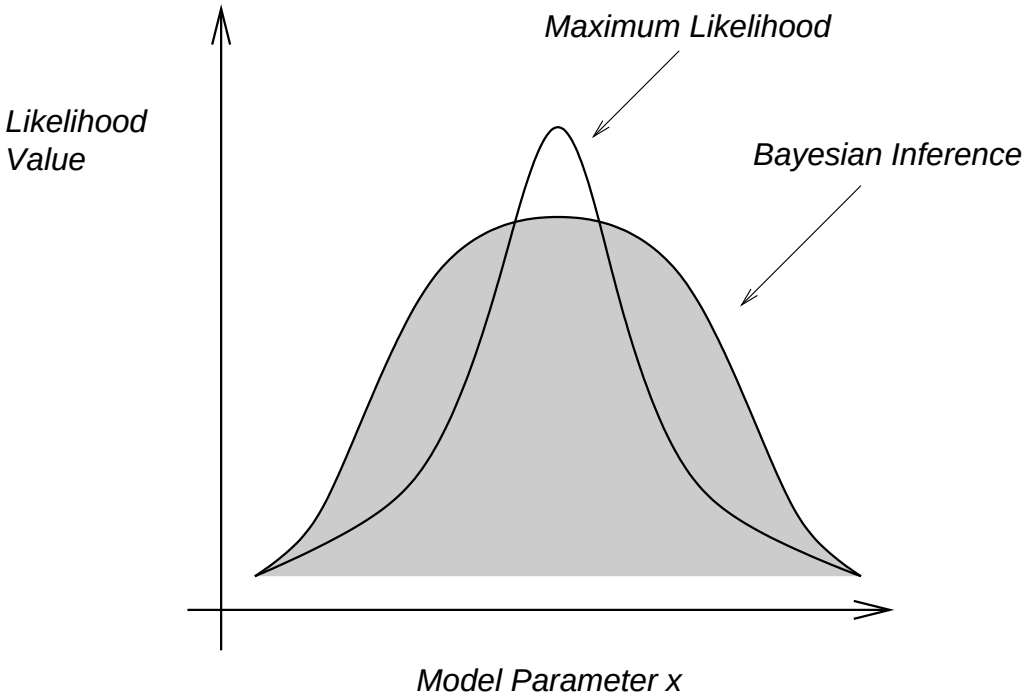

Figure 3.16: Schematic difference in likelihood distribution over some model parameter x for a hypothetic final tree topology obtained by bayesian and maximum likelihood methods

Another important aspect of program quality within the context of computing large trees for more than 1.000 organisms are memory requirements. The respective memory consumption of PHYML, MrBayes and RAxML is provided in Section 6.6 (pp. 109).

3.9.3 Parallel & Distributed Codes

Most popular parallel implementations of phylogeny programs have been designed to run on parallel distributed memory (MIMD, Multiple Instruction Multiple Data) machines. Therefore, they use the widely-spread Message Passing Interface [141] (MPI) for communication.

Parallel MPI-based implementations are available for the following popular sequential programs: DNAml [17], fastDNAml [135], treepuzzle [108], and MrBayes [50]. In addition, Brauer et al. [11] provide a parallel MPI-based implementation of a genetic search algorithm.

There also exists a shared-memory parallelization of fastDNAml called very-fastDNAml [145] which has rarely been used for phylogenetic analyses however. It is parallelized with the TreadMarks [142] distributed shared memory system.

On the one hand—except for bayesian approaches—the parallelization of these codes be it traditional or genetic search algorithms is fairly straight-forward since alternative tree topologies are evaluated independently by a set of processors in a standard master-worker scheme. Furthermore, tree topologies are communicated in a short standard string representation. Thus, communication overhead is not a crucial factor in most parallel implementations, which show "good" relative speedup values of around 90% and good scalability up to 64 processors. The low communication and synchronization costs and infrequent communication events—especially with growing number of sequences—also allow for grid and distributed computing approaches as a means of attaining the required computational resources. The only large-scale distributed implementation and execution of phylogenetic inference the author is aware of has been presented at the "Supercomputing 2000 (SC2000)" conference by Snell et al. [115]. Snell proposed a distributed approach for computation of large parsimony trees using the DOGMA [58] framework.

On the other hand Bayesian approaches represent a more difficult challenge since the MC^3 process is closely coupled and analyzes a significantly larger number of similar topologies. The main problem concerns the parallelization of one single Markov chain, since it is evident that n distinct Metropolis-coupled chains can be easily distributed and executed on n processors and occasionally exchange results. However, the creation of n distinct Markov Chains will result in extremely bad speedup values. Thus, the computations of each individual chain have to be parallelized, a task which requires a supercomputer with high performance communication links or at least a powerful PC cluster. Parallel implementations of bayesian analyses deploy similar techniques and face related problems as parallel fluid dynamics applications [32], such as load balancing and synchronization.

Generally, parallelization of phylogenetic codes results only in the gain of 1 to 2 orders of magnitude in terms of computable tree size, due to the computational complexity of underlying heuristic algorithms. Thus, parallel programs are only as good as the search algorithms of underlying sequential programs. Therefore, according to the author's personal experience in the field, the main focus of investigation should be on supporting and observing new algorithmic developments which sometimes yield advances of several orders of magnitude rather than executing parallel versions of out-dated—with respect to recent algorithmic developments—programs such as DNAml, fastDNAml, and treepuzzle, on large supercomputers. In addition, parallelization of maximum likelihood programs does not represent a scientifically challenging task in most cases and is thus considered just as useful spin-off of algorithmic development.

As an example consider the case of parallel fastDNAml [135] which was presented at the "Supercomputing 2001 (SC2001)" conference. The largest tree computed with parallel fastDNAml on an IBM RS/6000 XP using up to 64 processors contained 150 sequences. In 2003 RAxML was able to compute the best-known tree, i.e. better than all of the 10 trees inferred by 10 parallel executions of fastDNAml, for the same 150 taxon alignment within less than 10 minutes on a single Xeon 2.4GHz CPU.

Another severe example of needless brute-force resource allocation is the HPC challenge which was conducted during the "Supercomputing 2003 (SC2003)" conference [139]. A team around C. Stewart (author of parallel fastDNAml), received the price for the most geographically distributed application with a grid implementation of parallel fastDNAml [136] based on PACX-MPI [89]. They conducted a large scale analysis of arthropod evolution [1] distributed across supercomputer sites on several continents. Despite the fact that the technical achievement represents an unchallenged success the waste of CPU hours by disregarding recent algorithmic advances and insisting on the use of the fastDNAml algorithm which originally dates back to 1994 yields the whole event less impressive.

3.9.3.1 parallel fastDNAml

Despite all criticism concerning parallel fastDNAml, it is freely available as open source code and still widely in use on a large variety of supercomputers. In addition, the algorithmic optimization (yielding $\approx 50\%$ of performance improvement over parallel fastDNAml) described in Section 4.1 (pp. 54) was implemented in parallel fastDNAml and the respective MPI-based program was presented at "Supercomputing 2002 (SC2002)" as PAxML [131]. Therefore, parallel fastDNAml is analyzed in more detail at this point:

The program implements a simple master-worker architecture, which includes an additional intermediate `foreman` component between `master` and `workers` mainly for error handling in the Grid implementations mentioned above. The `master` and `forman` processes are responsible for generating, distributing, and collecting topology evaluation jobs to the `workers` and thus hardly produce load. Apart from some initialization routines the `worker` processes only offer a `treeEvaluate()` service which receives a tree in string representation and optimizes its likelihood. The resulting tree is then packed again into a string representation and sent back to the `foreman`.

For the inference process one can distinguish between two basic phases of the computation: the *stepwise addition phase* during progressive insertion of sequences and the *rearrangement phase*, i.e. the rearrangements on intermediate

[1] The size of the alignment used in this analysis has not yet been published.

topologies and the complete tree (see Section 3.9.1.1). Since distinct topologies can be evaluated independently, the only synchronization points occur between the different stages of stepwise addition, i.e. $t_k \rightarrow t_{k+1}$ for trees of size t_k and t_{k+1} where $k \leq n$ as well as at the end of individual rearrangement iterations.

The speedup of parallel fastDNAml typically ranges between values of ≈ 5 on 8 workers and ≈ 54 on 64 workers.

Summary

This Chapter provided an introduction to basic models and algorithms for phylogenetic tree inference and addressed basic mechanisms to obtain confidence into final results. Moreover, methods for comparing phylogeny programs have been discussed. Finally, current state of the art sequential and parallel phylogeny programs which implement statistical models of evolution have been addressed. The next Chapter describes novel algorithmic optimizations and new heuristics which enable rapid inference of large maximum likelihood-based trees. Those ideas have been implemented in a program called RAxML (Randomized Axelerated Maximum Likelihood) which is able to compute better trees in less time than the programs and algorithms mentioned in Section 3.9 of the current Chapter.

4

Novel Algorithmic Solutions

In Rothenburg ob der Tauber,
Da sitzt ein Akadem;
Und was er fühlt, ist sauber,
Und was er denkt, System.

Erich Weinert

The main goal of algorithmic improvements for maximum likelihood-based phylogenetic tree inference is to design algorithms which require less time and yield equally good or even better trees in terms of final likelihood values than comparable programs.

Firstly, this goal can be achieved by implementing algorithmic optimizations of the likelihood function which consumes the by far greatest amount of overall execution time (usually $\geq$ 90%) in maximum likelihood programs. Algorithmic optimizations usually consist in detecting equal patterns and reusing already computed values.

Secondly, both qualitative improvements as well as run-time reductions can be achieved simultaneously by implementation of novel search space heuristics.

The two Sections of this Chapter cover the design of a novel, purely algorithmic optimization of the likelihood function [130, 131, 133] and introduce novel, fast, and accurate search space heuristics [124] which have been derived from experimental work [125, 127].

4.1 Novel Algorithmic Optimization: AxML

In order to design an algorithmic optimization of the likelihood function one can search for identical patterns in the multiple alignment and utilize them to expedite the process of likelihood computation.

Therefore, the notion of column equalities is introduced. Two columns in an alignment are equal if they consist of exactly identical bases. All equal columns of an alignment form part of a *column class*. Moreover, two types of columns are distinguished: a homogeneous column consists of identical bases, e.g. contains only A's or gaps, whereas a heterogeneous column consists of distinct bases. An example which illustrates this definition is provided in Figure 4.1.

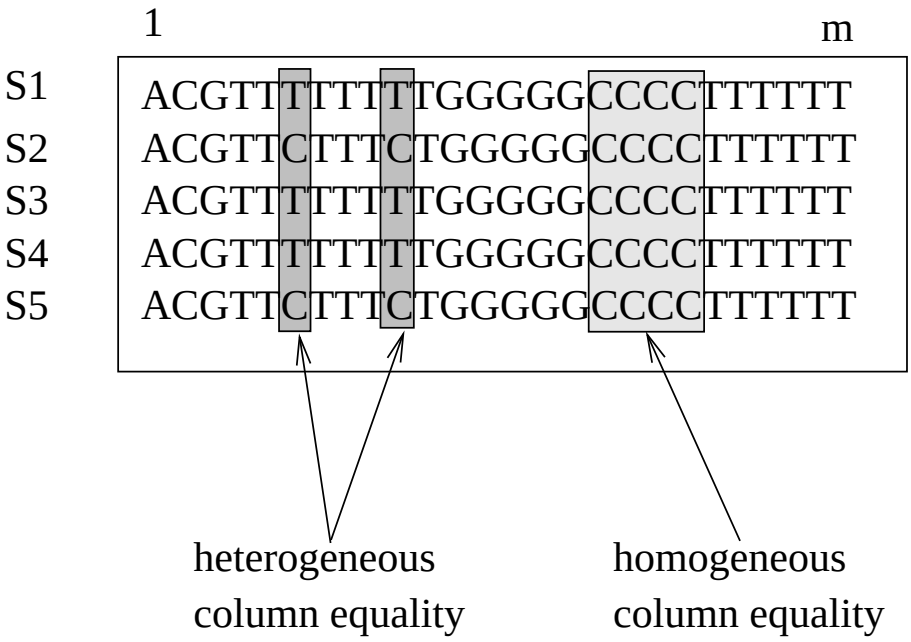

Figure 4.1: Heterogeneous and homogeneous column equalities

More formally, let $s_1, ..., s_n$ be the set of aligned input sequences as depicted in the upper matrix of Figure 4.2.

Let m be the number of sequence positions of the alignment. Two columns of the input data set i and j are equal if $\forall s_k, k = 1, ..., n : s_{ki} = s_{kj}$, where s_{kj} is the j-th position of sequence k. One can now calculate the number of equivalent columns for each *column class* of the input data set.

After calculating column classes, one can compress the input data set by keeping a single representative column for each column class, removing the equivalent

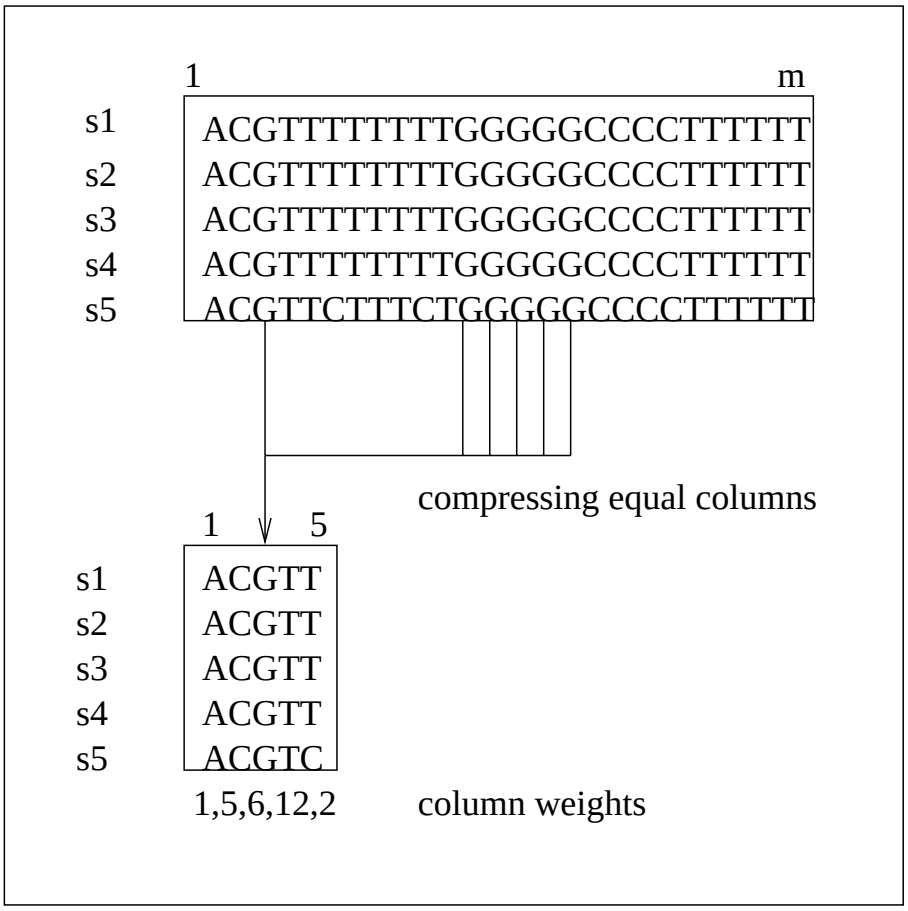

Figure 4.2: Global compression of equal column

columns of the specific class and assigning a count of the number of columns the selected column represents, as depicted in Figure 4.2.

Since a necessary prerequisite for a phylogenetic tree calculation is a high-quality multiple alignment of the input sequences one might expect quite a large number of column equalities on a global level. In fact, this kind of global data compression is already performed by most programs.

The fundamental idea of this algorithmic optimization is to extend this compression mechanism to the subtree level, since a large number of column equalities might be expected on the subtree level. Depending on the size of the subtree, fewer sequences have to be compared for column equality and, thus, the probability of finding equal columns is higher.

None the less, the analysis of subtree column equalities is restrained to homogeneous columns for the following reason:

The calculation of heterogeneous equality vectors at an inner node p is complex and requires the search for c^k different column equality classes, where k is the number of tips (sequences) in the subtree of p and c is the number of distinct values the characters of the sequence alignment are mapped to. For example, fastDNAml uses 15 different values. This overhead would not amortize well over the additional column equalities one would obtain, especially when $c^k > m'$ where m' is the length of the compressed global sequence alignment.

Now, one can derive an efficient and easy way to recursively calculate subtree column equalities using Subtree Equality Vectors (SEVs).

Let s be the virtual root placed in an unrooted tree for the calculation of its likelihood value. Let p be the root of a subtree with children q and r, relative to s. Let ev_p (ev_q, ev_r) be the equality vector of p (q, r, respectively), with size m'. The value of the equality vector for node p at position i, where $i = 1, ..., m'$ can be calculated by the following function (see example in Figure 4.3):

$$ev\_p(i) := \begin{cases} ev\_q(i) & if \quad ev\_q(i) = ev\_r(i) \\ -1 & else \end{cases} \tag{4.1}$$

If p is a leaf, $ev\_p(i)$ is set to $ev\_p(i) := map(sequence\_p(i))$, where, $map()$ is a function that maps the character representation of the aligned input sequence $sequence\_p$, at leaf p to values $0, 1, ..., c$. Thus, the values of an inner SEV ev_p, at position i, range from $-1, 0, ..., c$, i.e. -1 if column i is heterogeneous and from $0, ..., c$ in the case of an homogeneous column.

For SEV values $0, ..., c$ a pointer array $ref\_p(c)$ is maintained, which is initialized with $NULL$ pointers, for storing the references to the first occurrence of the respective column equality class in the likelihood vector of the current node p.

Thus, if the value of the equality vector $ev\_p(j) > -1$ and $ref\_p(ev\_p(j)) \neq NULL$ for an index j of the likelihood vector $lv\_p(j)$ of p, the value for the specific homogeneous column equality class $ev\_p(j)$ has already been calculated for an index $i < j$ and a large block of floating point operations can be replaced by a simple value assignment $lv\_p(j) := lv\_p(i)$. If $ev\_p(j) > -1$ and $ref\_p(ev\_p(j)) = NULL$, $ref\_p(ev\_p(j))$ is assigned to the address of $lv\_p(j)$, i.e. $ref\_p(ev\_p(j)) := adr(lv\_p(j))$.

The additional memory required for equality vectors is $O(n * m')$. The additional time required for calculating the equality vectors is $O(m')$ at every node.

The initial approach renders global run time improvements of 12% to 15% [1]. These result from an acceleration of the likelihood evaluation function between 19% and 22%, which in turn is achieved by a reduction in the number of floating point operations between 23% and 26% in the specific function.

[1]The percentages mentioned in this section were obtained during initial tests and program development on a Sun-Blade-1000.

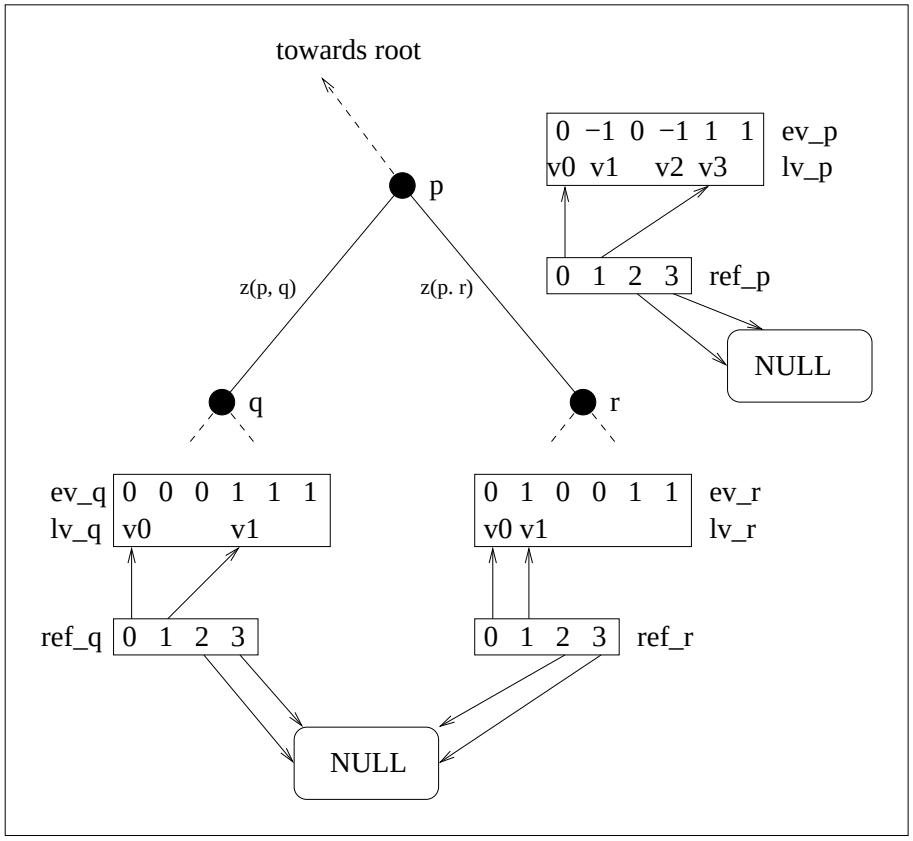

Figure 4.3: Example likelihood-, equality- and reference-vector computation for a subtree rooted at p

It is important to note that the initial optimization is only applicable to the likelihood evaluation function, and *not* to the branch length optimization function. This limitation is due to the fact that the SEV calculated for the *virtual* root placed into the topology under evaluation, at either end of the branch being optimized, is very sparse, i.e. has few entries > -1. Therefore, the additional overhead induced by SEV calculation does generally not amortize well with the relatively small reduction in the number of floating point operations (2% - 7%). Note however, that the SEVs of the *real* nodes at either end of the specific branch do not need to be sparse, since this depends on the number of tips in the respective subtrees.

In the following it is demonstrated how to efficiently exploit the information provided by a SEV, in order to achieve a further significant reduction in the number of floating point operations by extending this mechanism to the branch length optimization function.

To make better use of the information provided by an SEV at an inner node p with children r and q, it is sufficient to analyze at a high level how a single entry i of the likelihood vector at p, $lv\_p(i)$, is calculated:

$$lv\_p(i) := f(g(lv\_q(i), z(p,q)), g(lv\_r(i), z(p,r)), \tag{4.2}$$

where $z(p,q)$ ($z(p,r)$) is the length of the branch from p to q (p to r, respectively).

This exactly corresponds to the formula given in Section 3.4.1 (pp. 23) for recursively computing the likelihood in the tree:

$$L_{S_p}^{(p)} = \Big(\sum_{S_q=A}^{T} P_{S_p S_q}(z(p,q)) L_{S_p}^{(p)} \Big) \Big(\sum_{S_r=A}^{T} P_{S_p S_r}(z(p,r)) L_{S_r}^{(r)} \Big)$$

Recall from Section 3.4.3 (pp. 30) that the expressions for the $P_{S_p S_q}(z(p,q))$ rapidly become complex for more sophisticated models of sequence evolution, e.g. for the HKY85 model:

$$P_{pq}(t) = \begin{cases} \pi_q + \pi_q \left(\dfrac{1}{\pi_{sum}(q)} - 1 \right) e^{-\mu t} + \left(\dfrac{\pi_{sum}(q) - \pi_q}{\pi_{sum}(q)} \right) e^{-\mu t S} & (\mathbf{p} = \mathbf{q}) \\ \pi_q + \pi_q \left(\dfrac{1}{\pi_{sum}(q)} - 1 \right) e^{-\mu t} + \left(\dfrac{\pi_q}{\pi_{sum}(q)} \right) e^{-\mu t S} & (\mathbf{p} \neq \mathbf{q}, \mathbf{Ts}) \\ \pi_q \left(1 - e^{\mu t}\right) & (\mathbf{p} \neq \mathbf{q}, \mathbf{Tv}) \end{cases}$$

Thus, function $g()$ is a computationally expensive function, that calculates the likelihood of the left (right) branch of p, depending on the branch length $z(p,q)$ ($z(p,r)$) and the value of $lv\_q(i)$ ($lv\_r(i)$, respectively). Whereas $f()$ performs some simple arithmetic operations for combining the results of $g(lv\_q(i), z(p,q))$ and $g(lv\_r(i), z(p,r))$ into the value of $lv\_p(i)$. Note, that $z(p,q)$ and $z(p,r)$ do not change with i according to the definition of the tree likelihood score in Section 3.4 (pp. 20).

If $ev\_q(i) > -1$ and $ev\_q(i) = ev\_q(j),\ i < j$, then $lv\_q(i) = lv\_q(j)$ and therefore $g(lv\_q(i), z(p,q)) = g(lv\_q(j), z(p,q))$ (the same equality holds for node r). Thus, for any node q one can avoid the recalculation of $g(lv\_q(i), z(p,q))$ for all $j > i$, where $ev\_q(j) = ev\_q(i) > -1$. Those values are precalculated and stored in arrays $precalc\_q(c)$ and $precalc\_r(c)$ respectively, where c is the number of distinct character-value mappings found in the sequence alignment.

The final optimization consists in the elimination of value assignments of type $lv\_q(i) := lv\_q(j)$, for $ev\_q(i) = ev\_q(j) > -1,\ i < j$ where i is the first

entry for a specific homogeneous equality class $ev\_q(i) = 0, ..., c$ in ev_q. Those values need not be assigned due to the fact that $lv\_q(j)$ will never be accessed. Instead, since $ev\_q(j) = ev\_q(i) > -1$ and the value of $g\_q(j) = g\_q(i)$ has been precalculated and stored in $precalc\_q(ev\_p(i))$, $lv\_q(i)$ is accessed through its reference in $ref\_q(ev\_q(i))$.

During the main for-loop in the calculation of lv_p one has to consider 6 cases, depending on the values of ev_q and ev_r. For simplicity $p\_q(i)$ instead of $precalc\_q(i)$ and $g\_q(i)$ instead of $g(lv\_q(i), z(p,q))$ is used.

$$lv\_p(i) := \begin{cases} f(p\_q(ev\_q(i)), p\_r(ev\_r(i))) & if \quad ev\_q(i) = ev\_r(i) > -1, \\ & \quad ref\_p(ev\_r(i)) = NULL \\ skip & if \quad ev\_q(i) = ev\_r(i) > -1, \\ & \quad ref\_p(ev\_r(i)) \neq NULL \\ f(p\_q(ev\_q(i)), p\_r(ev\_r(i))) & if \quad ev\_q(i) \neq ev\_r(i), \\ & \quad ev\_q(i), ev\_r(i) > -1 \\ f(p\_q(ev\_q(i)), g\_r(i)) & if \quad ev\_q(i) > -1, ev\_r(i) = -1 \\ f(g\_q(i), p\_r(ev\_r(i))) & if \quad ev\_r(i) > -1, ev\_q(i) = -1 \\ f(g\_q(i), g\_r(i)) & if \quad ev\_q(i) = -1, ev\_r(i) = -1 \end{cases} \tag{4.3}$$

A simple example for the optimized likelihood vector calculation and the respective data-types used is given in Figure 4.3.

4.1.1 Additional Algorithmic Optimization

Since the initial implementation was designed for no particular target platform and AxML showed to scale best on PC processor architectures (see Section 6.2.2, pp. 90 and [130]), additional algorithmic optimizations have been investigated which are especially designed for these architectures. An additional acceleration can be achieved by a more thorough exploitation of SEV information in function `makenewz(...)`, which optimizes the length of a *specific* branch b and accounts for approximately one third of total execution time. Function `makenewz(...)` consists of two main parts : Initially, a for-loop over all alignment positions is executed for computing the likelihood vector of the virtual root s placed into branch b connecting nodes p and q. Thereafter, a do-loop is executed which iteratively alters the branch length according to a convergence criterion. For calculating the new likelihood value of the tree for the altered branch length within that do-loop, an inner for-loop over the likelihood vector of the virtual root s which uses the data computed by the initial for-loop is executed. The basic structure of this function is outlined in the following pseudo-code. The likelihood vectors at nodes p, q, s are named `lh_p[], lh_q[], lh_s[]` respectively.

```
void makenewz(...)
{
  for(i = 1; i < m'; i++)
    {
      lh_s[i] = compute_lh(lh_p[i], lh_q[i], b);
    }

  do
    {
      b =  alter(b);
      for(i = 1; i < m'; i++)
        lh_s[i] = compute_lh(lh_s[i], b);
    }
  while(!converged);
}
```

A detailed analysis of `makenewz()` reveals two points for further optimization:

Firstly, the do-loop for optimizing branch lengths is rarely executed more than once (see Table 4.1). Furthermore, the inner for-loop accesses the data computed by the initial for-loop. Therefore, the computations performed by the *first* execution of the inner for-loop have been integrated into the initial for-loop. In addition the conditional statement which terminates the iterative optimization process has been appended to the initial for-loop, such as to avoid the execution of the *first* inner for-loop completely.

data	# makenewz() invocations	# makenewz() invocations with $\#iterations > 1$	average # iterations when $\#iterations > 1$
SC_10	1629	132	7.23
SC_20	8571	661	6.14
SC_30	21171	1584	6.17
SC_40	39654	2909	6.21
SC_50	63112	4637	6.26

Table 4.1: makenewz() analysis

Secondly, when more than one iteration is required for optimizing the branch length in the do-loop one can reduce the length of the inner for-loop by using SEVs. The length of the inner for-loop $f = m'$ can be reduced by $nn - c$ the number of non-negative entries nn of the SEV at the virtual root s minus the number c of distinct column equality classes, since one needs to calculate only

one representative entry for each column equality class. Note, that the weight of the column equality class representative is the accumulated weight of all column equalities of the specific class at s. Thus, the reduced length f' of the inner for-loop is obtained by $f' := m' - nn + c$.

The SEV ev_s of the virtual root s is obtained by applying:

$$ev\_s(i) := \begin{cases} ev\_p(i) & if \quad ev\_p(i) = ev\_q(i) \\ -1 & else \end{cases} \tag{4.4}$$

The pseudo-code for the transformed version of `maknewz(...)` is provided below:

```
void makenewz(...)
{
  b' =  alter(b);
  for(i = 1; i < m'; i++)
    {
       ev_s[i] = compute_SEV(ev_p[i], ev_q[i]);
       lh_s[i] = compute_lh(lh_p[i], lh_q[i], b);
       lh_s[i] = compute_lh(lh_s[i], b');
    }

  compute(nn);
  compute(c);

  while(!converged)
    {
      b' =  alter(b');
      for(i = 1; i < m' - nn + c; i++)
        lh_s[i] = compute_lh(lh_s[i], b');
    }

}
```

Typically, the branch length optimization process requires a relatively large average number of iterations to converge if it does not converge after the *first* iteration. Therefore, the optimization scales well despite the fact that the SEV at the virtual root s is comparatively sparse, i.e. $nn - c$ is relatively small compared to m'. Experimental results for some small data sets comprising 10 up to 50 sequences (10_SC,...,50_SC) in Table 4.1 confirm this observation. A detailed description of the the data sets used is provided in Section 6.1 (pp. 87).

4.2 New Heuristics: RAxML

The heuristics of RAxML belong to the class of algorithms outlined in Section 3.9.1 (pp. 41), that optimize the likelihood of a starting tree which already comprises all sequences. In contrast to other programs RAxML starts by building an initial parsimony tree with dnapars from Felsenstein's PHYLIP package [93] for two reasons:

Firstly, parsimony is related to maximum likelihood under simple evolutionary models [144], such that one can expect to obtain a starting tree with a relatively good likelihood value compared to random or neighbor joining starting trees. For example, the 500_ZILLA parsimony starting tree already showed a better likelihood than the final tree of PHYML (see Table 6.7 on page 97).

Secondly, dnapars uses stepwise addition as outlined in Section 3.9.1.1 (pp. 42) for tree building and is relatively fast. The stepwise addition algorithm enables the construction of distinct starting trees by using a randomized input sequence order. Thus, RAxML can be executed several times with different starting trees and thereby compute a set of distinct final trees. The set of final trees can be used to build a consensus tree and augment confidence into the final result. To speed up computations, subtree rearrangements and evaluation of parsimony scores for all possible rootings have been removed from dnapars.

The tree optimization process represents the second and most important part of the heuristics. RAxML performs standard subtree rearrangements by subsequently removing all possible subtrees from the currently best tree t_{best} and re-inserting them into neighboring branches up to a specified distance of nodes. RAxML inherited this optimization strategy from fastDNAml (see Section 3.9.1, pp. 41). One rearrangement step in fastDNAml consists of moving all subtrees within the currently best tree by the minimum up to the maximum distance of nodes specified (lower/upper rearrangement setting). This process is outlined for a single subtree (ST5) and a distance of 1 in Figure 4.4 and for a distance of 2 in Figure 4.5 (not all possible moves are shown). In fastDNAml the likelihood of each thereby generated topology is evaluated by exhaustive branch length optimizations. If one of those alternative topologies improves the likelihood t_{best} is updated accordingly and once again all possible subtrees are rearranged within t_{best}. This process of rearrangement steps is repeated until no better topology is found.

The rearrangement process of RAxML differs in two major points: In fastDNAml after each insertion of a subtree into an alternative branch the branch lengths of the entire tree are optimized. As depicted in Figures 4.4 and 4.5 with bold lines RAxML only optimizes the three local branches adjacent to the insertion point of the subtree either analytically (fast) or by the Newton-Raphson method (slower, see Section 3.4.2, pp. 25) before computing its likelihood value. Since the likeli-

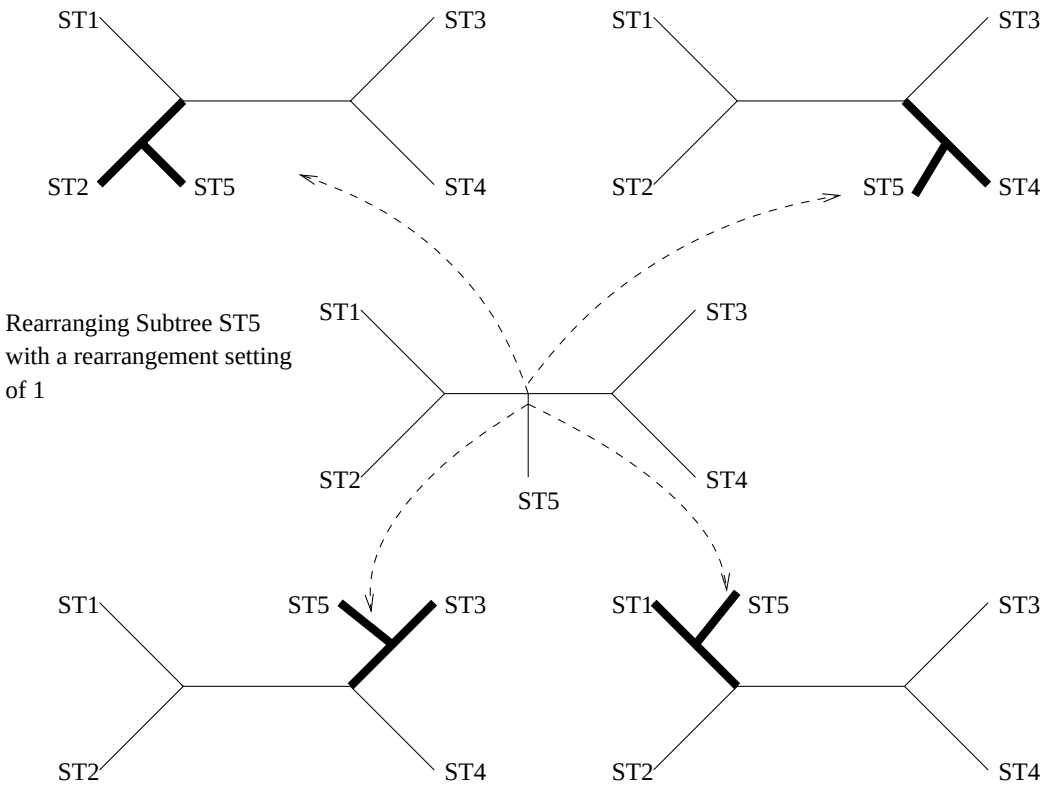

Figure 4.4: Rearrangements traversing one node for subtree ST5, branches which are optimized are indicated by bold lines

hood of the tree strongly depends on the topology per se this fast prescoring can be used to establish a small list of potential alternative trees which are very likely to improve the score of t_{best}. RAxML uses a list of size 20 to store the best 20 trees obtained during one rearrangement step. This list size proves to be a practical value in terms of speed and thoroughness of the search. After completion of one rearrangement step the algorithm performs global branch length optimizations on those 20 best topologies only. The capability to rapidly analyze significantly more alternative and diverse topologies due to a computationally feasible higher rearrangement setting (e.g. 5 or 10) leads to significantly improved final trees.

Another important change especially for the initial optimization phase, i.e. the first 3-4 rearrangement steps, consists in the subsequent application of topological improvements during one rearrangement step. If during the insertion of one specific subtree into an alternative branch a topology with a better likelihood is encountered this tree is kept immediately and all subsequent subtree rearrangements of the current step are performed on the improved topology. The mechanism is outlined in Figure 4.6 for a subsequent application of topological improvements

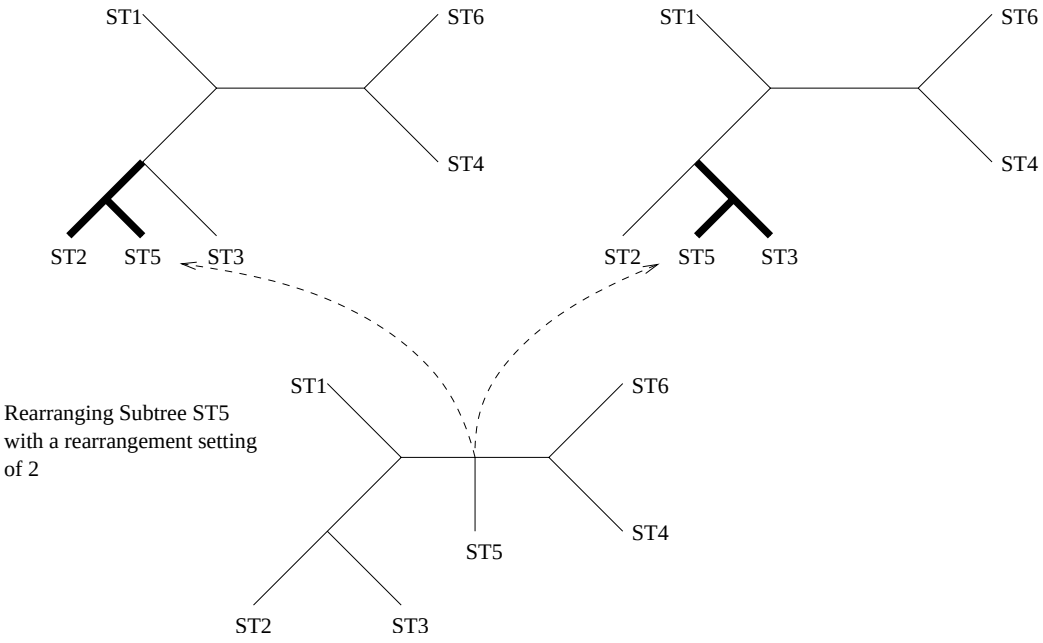

Figure 4.5: Example rearrangements traversing two nodes for subtree ST5, branches which are optimized are indicated by bold lines

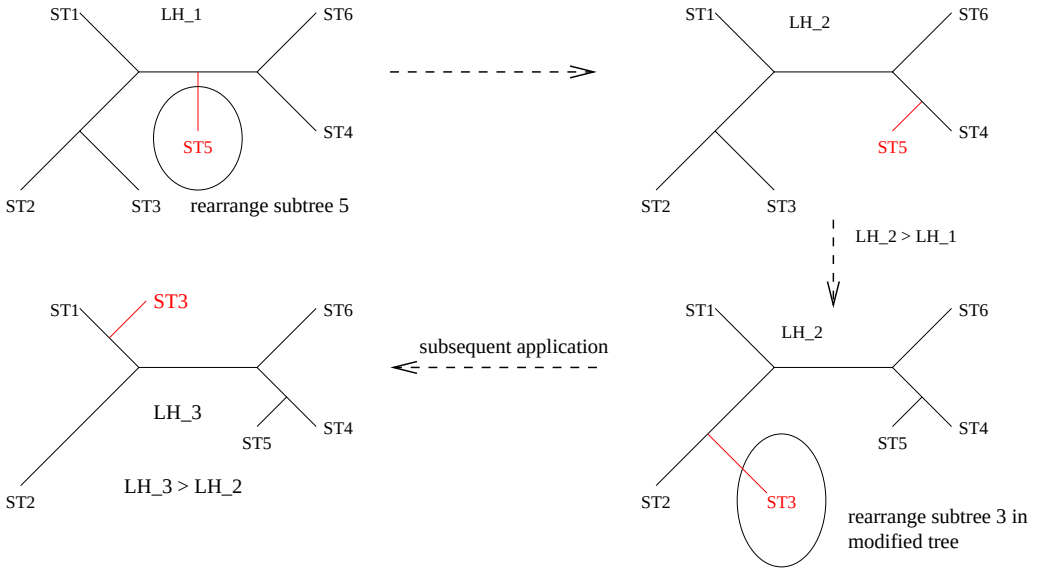

Figure 4.6: Example for subsequent application of topological improvements during one rearrangement step

via subtree rearrangements of ST5 and ST3 on the same initial tree. This enables rapid initial optimization of random starting trees as depicted e.g. for two alignments containing 150 taxa in Figures 6.6 and 6.7 (p. 102 and p. 102).

The exact implementation of the RAxML algorithm is indicated in the C-like pseudocode below. The algorithm is passed the user/parsimony starting tree `t`, the initial rearrangement setting `rStart` (default: 5) and the maximum rearrangement setting `rMax` (default: 21). Initially, the rearrangement stepwidth ranges from `rL = 1` to `rU = rStart`. Fast analytical local branch length optimization `a` is turned off when functions `rearr(...)`, which actually performs the rearrangements, and `optimizeList20()` fail to yield an improved tree for the first time. As long as the tree does not improve the lower and upper rearrangement parameters `rL, rU` are incremented by `rStart`. The program terminates when the upper rearrangement setting is greater or equal to the maximum rearrangement setting, i.e. `rU >= rMax`.

```
RAxML(tree t, int rStart, int rMax)
{
 int rL, rU;
 boolean a = TRUE;
 boolean impr = TRUE;

 while(TRUE)
  {
   if(impr)
    {
     rL = 1;
     rU = rStart;
     rearr(t, rL, rU, a);
    }
   else
    {
     if(!a)
      {
       a = FALSE;
       rL = 1;
       rU = rStart;
      }
     else
      {
       rL += rStart;
       rU += rStart;
      }
     if(rU < rMax)
      rearr(t, rL, rU, a);
     else
      goto end;
    }
   impr = optimizeList20();
  }
 end:
}
```

Summary

The present Chapter covered the design of a novel algorithmic optimization of the likelihood function and introduced novel, fast, and accurate search space heuristics. Those two basic ideas significantly contribute to the resolution of the two main computational problems of maximum likelihood-based analyses: the cost of the tree evaluation function and the efficient traversal of search space. However, the computation of huge trees still requires an enormous amount of computational resources. Thus, parallel, distributed, and Grid-enabled solutions to attain those resources for AxML (SEV-method) and RAxML (SEV-method and new heuristics) are addressed in the next Chapter.

5

Novel Technical Solutions

Dem Ingenieur ist nichts zu schwör.

This Chapter briefly describes a variety of technical solutions which have been devised to obtain the required computational power for inference of large phylogenetic trees with AxML and RAxML respectively.

The CORBA-based distributed implementation of AxML is described in more detail in a paper presented at the "PaCT2003 conference" [128] whereas an overview over all technical solutions which have been devised for AxML is provided in [121]. Finally, the distributed implementation of RAxML is described in [120] and two recent papers [122, 126] provide information on parallel RAxML.

5.1 Parallel and Distributed Solutions for AxML

This Section covers the parallel, distributed, and grid-based technical solutions which have been implemented for AxML. Finally, it addresses a special adaptation of PAxML to the Hitachi SR8000-F1 supercomputer.

5.1.1 Parallel AxML

The implementation of Parallel AxML (PAxML) is entirely based on parallel fastDNAml which is briefly outlined in Section 3.9.3.1 (pp. 51). Since the core component of the worker implementation in parallel fastDNAml consists in the likelihood evaluation function, the SEV-based version of this function had simply to be integrated into the existing code, which otherwise remained unchanged.

Due to the fact, that SEVs only induce a moderate acceleration of the average evaluation time per topology (30% -60%) the expected speedup values remain unaffected compared to those of parallel fastDNAml which typically range between values of ≈ 5 on 8 workers and ≈ 54 on 64 workers.

5.1.2 Distributed Load-managed AxML

DAxML (Distributed AxML), the distributed CORBA-based implementation of AxML has been developed in cooperation with Markus Lindermeier at the Lehrstuhl für Rechnertechnik und Rechnerorganisation who developed the LMC system (see below) within the framework of his Ph.D. thesis. This Section provides a brief introduction to the CORBA-based Load Management System and a description of the respective implementation and adaptation of PAxML. The distributed code has been derived from PAxML and uses a very similar parallelization scheme.

5.1.2.1 The Load Management System

Nowadays applications do not reside on a single host anymore; they are distributed all over the world and interact through well defined protocols. Global interaction is accomplished by so called middleware architectures. The most common middleware architectures for distributed object-oriented applications are the CORBA (Common Object Request Broker Architecture) and the DCOM (Distributed Component Object Model). Environments like CORBA and DCOM cause new problems because of their distribution. A significant problem is load imbalance. As application objects are distributed over multiple hosts, the slowest host determines the overall performance of an application. Load management services intend to compensate load imbalance by distributing workload. This guarantees both, high performance, as well as scalability of distributed applications.

The load management concept uses objects as load distribution entities and hosts as load distribution targets. Workload is distributed by initial placement, migration, and replication.

- *Initial Placement* stands for the creation of an object on a host that has sufficient computing resources in order to efficiently execute the object.

- *Migration* means moving an existing object to another host that promises a more expeditious execution.

- *Replication* is similar to migration but the original object is not removed, such that identical objects called replicas are created. Further requests to

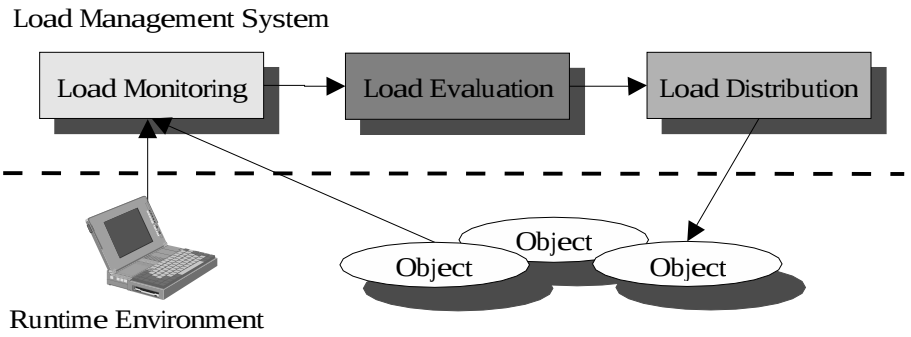

Figure 5.1: The components of the Load Management System LMC

the object are split up among its replicas in order to distribute the workload (requests) among them.

There are two kinds of overload in distributed object-oriented systems: background load and request overload. Background load is caused by applications that are not controlled by the load management system. Request overload means that an object is not capable to efficiently process all requests it receives. Migration is an adequate technique for handling background load but the scalability attained by migration is limited. Replication helps to break this limitations and is an adequate technique for handling request overload.

These concepts have been implemented in the Load Managed CORBA (LMC) system [75]. LMC is a load management system for CORBA. The main components of LMC are shown in Figure 5.1. These components fulfill different tasks and work at different abstraction levels. The load monitoring component offers both, information on available computing resources and their utilization, as well as information on application objects and their resource usage. This data has to be provided dynamically, i.e. at runtime, in order to obtain information about the runtime environment and the respective objects. Load distribution provides the functionality for distributing workload by initial placement, migration, or replication of objects. Finally, the load evaluation component decides about load distribution based on information provided by load monitoring. Those decisions can be attained by a variety of strategies [74].

LMC is completely transparent on the client-side because it uses CORBA's Location Forward mechanism to distribute requests among replicas. On the server-side minor changes to the existing code are necessary for integrating load management functionality into the application. These changes mainly affect the configuration of the Portable Object Adapter (POA). All extensions are seamlessly integrated into the CORBA programming model. Thus, only a minor additional

effort is required by the application programmer for the integration of the services provided by LMC.

For a detailed description of the load management system as well as the initial placement, migration, and replication policies which have been used see [74].

5.1.2.2 Implementation

For designing DAxML, the original parallel code of PAxML was initially simplified by removing the `foreman` component entirely from the system, since error handling can more easily be handled directly by LMC.

Furthermore, the program structure was altered, such as to create all trees of size k: t_k (see Section 3.9.1.1, pp. 42), i.e. all topologies with k leaves, that can be evaluated independently at once, and store them as strings in a work queue. This transformation was performed, in order to provide a means for issuing simultaneous topology evaluation requests (see below).

All alternative tree topologies t_k which are generated either by stepwise addition or tree rearrangements at step k of the search algorithm form part of topology class k.

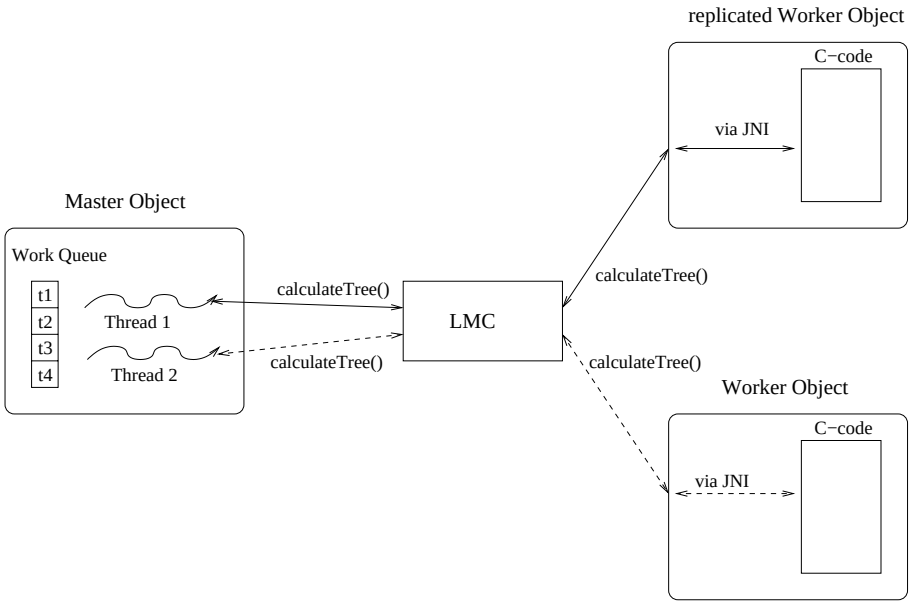

Figure 5.2: System architecture of DAxML

Note, that several sets of trees from topology class k, which have to be evaluated in sequential order, may be generated, depending on the specified rearrange-

ment setting of DAxML. For example one set of trees t_k from the stepwise addition step and typically several sets for each iteration of the rearrangement phase are generated (see Section 3.9.1.1, pp. 42 for details).

Those sets are sufficiently large, such that they do not create a synchronization problem at the respective transition points between distinct sets.

The overhead induced by first creating and storing *all* topologies before invoking the evaluation function is neglectible, since the invocation of the topology evaluation function consumes by far the greatest portion of execution time.

Because LMC is based on a modified JacORB [13] version and only provides services for JAVA/CORBA applications, the simplified code was transformed into a sequential JAVA program using JNI (JAVA Native Interface). Two JAVA classes `master` and `worker` were designed which offer analogous functionalities as their counterparts in PAxML. The basic service provided by the `worker` class is a method called `calculateTree()`, for evaluating a specific tree topology, which in turn invokes the fast native C evaluation function via JNI. The method `calculateTree()` corresponds exactly to the `treeEvaluate()` function in PAxML and parallel fastDNAml (see Section 3.9.3.1, pp. 51).

The `master` component loads and parses the sequence file, passes the input data to the `worker`, generates tree topologies and gathers results.

The transformation of the sequential JAVA code into a LMC-based application was straight-forward, since its class layout already complied with the structure of the distributed application. The `worker` class is encapsulated as CORBA worker object, and provides its topology evaluation function as CORBA service. The state of the CORBA `worker` object consists only of the sequence data, which can be loaded via NFS or directly from the `master` when a `worker` object is created by initial placement, migration, or replication.

Thus, since the sequence data is not modified during tree calculation, replications and migrations of worker objects do not induce any consistency problems.

In the main work-loop of the `master`, a number of threads corresponding to the number of available hosts controlled by LMC is created, in order to perform simultaneous topology evaluation requests. This enables LMC to correctly distribute tree evaluation requests among worker objects on distinct hosts and to ensure optimal distribution granularity. The system architecture of DAxML is outlined in Figure 5.2 for a simple configuration with two worker objects.

5.1.3 AxML on the Grid

Unlike typical supercomputer applications, such as e.g. hydrodynamic simulations, parallel phylogeny programs such as PAxML can easily and quickly be interrupted and restarted, i.e. they are well suited for automatic relocation and execution on available, faster, or more inexpensive resources. Furthermore, they

do not require an excessive amount of memory and final as well as intermediate results (checkpoints) of the tree inference process are stored in a simple and comparatively small in size string format (less than 0.5MB even for 1.000 taxa).

These properties facilitate the implementation of a "phylogenetic grid worm", i.e. an application which is able to migrate through the grid to suitable resources, according to a set of migration criteria, during its execution. Note, that there still exists a plethora of partially unresolved problems in the area of metacomputing such as the co-scheduling problem, fault tolerant communication protocols, or additional communication overhead between distant supercomputers sites [135, 136]. Therefore, the implementation of a "grid worm" represents a realistic approach for performing high throughput phylogenetic tree computations on the grid, given the present state of technology.

GAxML (Grid AxML) has been designed by integrating the migration services of the high-level GMS [68, 69, 70] (Grid Migration Server) into PAxML in cooperation with Gerd Lanfermann from the Max-Planck Institute at Potsdam. Section 5.1.3.1 describes the basic components of the Grid Migration Server. The respective GMS-based implementation of Grid AxML is outlined in the subsequent Section 5.1.3.2.

5.1.3.1 The Grid Migration Server

The Grid Migration Service is developed at the Max-Planck Institute for Gravitational Physics within the framework of the GridLab project. GMS is a tool originally designed to increase the throughput of large-scale relativistic simulations at the institute by means of automatic migration. The GridLab [38] project intends to define and explore Grid functionalities, which will are privided by the Grid Application Toolkit [1].

GMS is a XML-RPC [150] based server system, that provides several RPC migration methods to clients. The most essential services it provides are `ms_migrate`, which issues a migration request and `ms_announce`, which continuously provides migration data to the server, without actually requesting a migration. The latter service and data is useful for automatically restarting the simulation in case of failure.

GMS operates like any other web service via request calls. Along with a migration request, the client needs to provide data, which is required to restart the program on a different host:

- **Data Files:** The client must specify the files which are required to restart the program, e.g. checkpoint files, which describe the current state of the simulation, parameter files, which contain program settings, or other data files.

- **Executable:** The client must inform the server on the executable that will be used to restart the program. If the application is moved to a different hardware architecture, the server has to retrieve or generate an appropriate executable for the new target platform.

- **Startup Command:** Since the server has no knowledge on how the program needs to be started, the client has to inform the server on its startup procedure, e.g. the client has to specify its command-line flags and the order in which input and output files are passed to the application. This information can also contain pre- and postprocessing commands.

- **Resource Requirements:** The client should specify its minimum resource requirements. This information may consist of the necessary number of processors, the minimum memory, or restrictions regarding the supported operating systems and/or supported host systems.

GMS is a high-level service, which is based on low-level services providing file transfer, machine access, and resource evaluation capabilities. A high-level migration request invokes several low-level copy and start request.

The Application Information Server: Like the GMS, the AIS (Application Information Server) is a XML-RPC based information base for storing information on applications, resources and files. It is primarily designed to be accessed by applications through RPCs, which extract information on the state of services, the location of files, etc. The AIS maintains and controls the activity status of any application that registers or de-registers to/from AIS. The AIS can actively track those applications which are able to respond to `ping` requests. If an application is pinged and fails to respond, the AIS declares the respective application as *inoperational* and can either take counter measures to restart it or simply inform the user about the failure.

Like the AIS, the pAIS (personal AIS) is an information database, but aimed at providing simulation-specific data to the scientist. For the pAIS the application provides information on the progress of the computation to the scientist.

In the case of GAxML this information can contain e.g. the current tree, the number of processes, and the estimated runtime to completion. As the application is migrating through the grid, the pAIS serves as contact portal for the scientist to monitor migrating applications. The pAIS abstracts simulation data (like search progress, etc.) from the actual execution platform/host.

5.1.3.2 Implementation of GAxML

Integrating migration functionality into a parallel phylogeny program like PAxML was a challenge, since PAxML was not designed to be executed as migrating application.

In this Section the necessary modifications to PAxML for the integration of migration capabilities, i.e. the design of the GAxML client is outlined. Furthermore, the necessary adaptations and extensions on the server-side of the program, i.e. GMS, AIS are addressed. The overall system architecture is depicted in Figure 5.3.

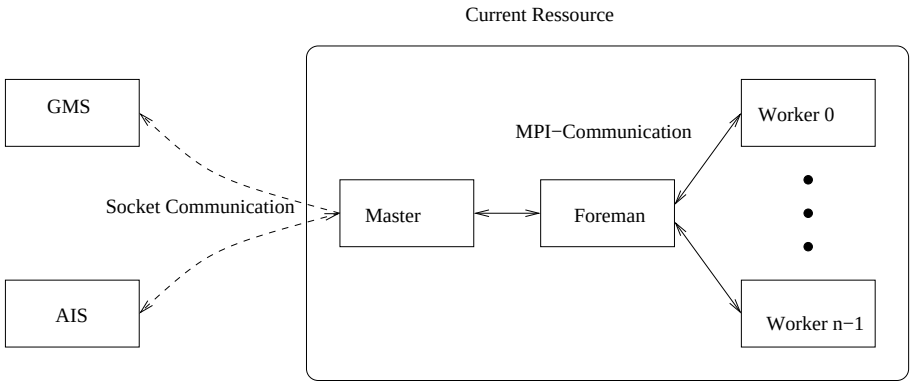

Figure 5.3: System Architecture of GAxML

5.1.3.2.1 Client-side Modifications

The `master` process of PAxML maintains all data required for checkpointing, shutting down and restarting the whole application. Thus, only minor changes to the `master` component were required to integrate migration services into the program, leaving the `foreman` and `worker` components completely unchanged.

All communication with the GMS and AIS is performed via sockets, such that a number of additional communication routines had to be integrated into the `master` process for transmitting requests and status information. Those communication routines transform the respective data structures into serialized XML code and embed it into a `http` header structure which is then written to the respective socket. The server receives and deserializes the data and then executes the respective request.

The Grid Object Description Language (GODsL) Toolkit which provides a uniform description of objects on a grid including file, hardware, resource, and service properties has been deployed for this purpose. It is used for describing the

resource requirements of GAxML and the location of the various files required for restarting the program on a distinct host. The GODsL data structure can be serialized into XML and is therefore compatible with XML-RPC based services, as provided by the GMS. For a detailed description of the Grid Object Description Language Toolkit see [38].

The GAxML `master` process has to provide a *migration infrastructure* (the capability to register and communicate status data to the GMS/AIS as well as to prepare and issue a migration request) on the one hand and *migration triggering* mechanisms (the capability to collect *internal* data for performing migration decisions and to receive *external* migration commands) on the other hand.

Migration Infrastructure: For providing the necessary migration infrastructure the `master` process of GAxML has been modified as follows:

- **Registration:** GAxML announces itself to the Application Information Server (AIS) and sends information about the machine it currently executes on by a socket routine, when the `master` process is started.
- **Runtime Information:** The `master` sends information, containing the currently best tree t_k, when a checkpoint for t_k is written. This information is published by the personal Application Information Server (pAIS).
- **Migration:** If a migration is initiated, the `master` shuts down the `foreman` and `worker` processes, automatically generates the restart command, as well as the restart file, and specifies the resource requirements. This information is sent to the GMS as part of the migration request. Since GAxML always requires the initial sequence file and the current checkpoint file to generate the restart file, two files have to be transferred to the new host: the actual restart file and the original sequence file. The GAxML checkpoint, restart, and sequence files are flat ASCII files and therefore platform-independent.

Migration Triggering: A migration can be initiated when one or more *internal* or *external* migration criteria are met. The following *internal* criteria are monitored and implemented in the `master` process to trigger migrations.

- The main criterion for resource requirements is the number of independent tree evaluation tasks, which constantly increases during the reconstruction process. Those tasks are generated by the `master`, such that if a certain number of tasks/number of workers threshold ratio is exceeded, the `master` initiates a migration request containing a higher number of worker processes.

- A new command-line option `-z` which specifies the time assigned by the respective batch-queuing system has been added to GAxML. The `master` regularly checks if the time is about to expire and initiates a migration if necessary.

Furthermore, the required infrastructure for issuing *external* migrations commands to GAxML is provided. *External* migrations can be triggered either via a manual migration interface from the user or when "better" resources in terms of e.g. faster, more inexpensive, or more suitable architectures such as PC clusters (see Section 6.2.2, pp. 90) become available.

The latter case requires an external service that can gather and evaluate information about other resources in the grid. Such tools are a current research topic in the area of grid computing.

To issue such an automatic or manual *external* migration one has to provide information about the new target platform. When an *external* migration is triggered GAxML will automatically start the checkpointing, shutdown, and migration procedure. Thus, the `master` had to be modified in order to be able to receive *external* request.

Since the `master` process already implements a loop that regularly checks the expiration of the assigned computing time a socket polling mechanism has been integrated into that loop to receive incoming *external* migration requests and `pings` (see below) from the AIS.

5.1.3.2.2 Server-side Adaptations

Application Tracking: The AIS *actively* tracks applications via a `ping` request. The ability to respond to such requests has been integrated into GAxML as described above. The AIS will treat missing `ping` responses as program failures.

GAxML Visualization: In a grid migration environment, the execution of the application is abstracted from the application data, which is of major interest to the scientist. In an automated migration environment it is not possible to determine where the application is currently running or where it will execute next (except in the case of manual migrations), because these decisions will generally depend on changing resource availability. However, monitoring the progress of the computation and assessing the quality of intermediate results is of major importance for any scientific application.

GAxML offers two possibilities for monitoring the progress of the tree reconstruction process.

The `master` regularly sends information about the currently best tree t_k to the pAIS via the `ais_info` remote procedure call. Upon receipt the pAIS publishes this information via a web interface.

In a more advanced approach, GAxML makes use of the file advertisement features in Cactus [15]. Although GAxML is *not* a Cactus application, the GMS is Cactus-based and its web services permit GAxML to indirectly use Cactus visualization features. In this case GAxML sends the currently best tree t_k in its string representation and an appropriate `MIME-Type` extension to the pAIS via an `ais_info2file` remote procedure call. Since these strings are comparatively small, they can easily be handled by the `http` protocol. The `MIME-Type` specifies, how a web server advertises this data in a file and how a web browser treats such a file. A typical extension is e.g. `application/postscript` for opening a postscript viewing program. For visualizing tree data the extension `data/phylo` has been introduced.

When the pAIS receives the `ais_info2file` request, it writes the enclosed tree data to a file, provided it does not exceed a maximum size, and associates the transmitted `MIME-Type` extension with that file. The pAIS then assigns a `html` link to the file and publishes the `URL` on its web-page.

A browser can now be appropriately configured to invoke a phylogenetic tree visualization tool such as ATV [3, 153]. When the user clicks on a link with the `data/phylo` `MIME-Type` ATV is automatically invoked and displays the current tree. Furthermore, ATV has been slightly modified to regularly read in the tree file which is overwritten when a new tree is received from the `master`, i.e. ATV always displays the most recent tree. Thus, independently of where GAxML is currently executed, the scientist can continuously monitor the reconstruction process and see the tree grow on his screen as new sequences are inserted. In Figures 5.4 and 5.5 two screenshots of ATV at different stages in the reconstruction of a 150 taxon tree with GAxML are depicted.

5.1.4 PAxML on Supercomputers

This Section briefly addresses the special adaptation of PAxML to the Hitachi SR8000-F1 supercomputer which is also outlined in [132].

As already mentioned the parallel architecture of PAxML consists of a simple master-worker model, with the master distributing the tree topologies to be evaluated in a simple short string representation to the workers, i.e. communication overhead is insignificant.

Thus, initial tests were carried out in intra-node MPI-mode, in order to keep each worker module as compact as possible and to rapidly evaluate the scalability of SEV-based optimizations (see Section 4.1, pp. 54) to the specific processor architecture of the SR8000-F1.

The first tests rendered rather unfavorable results in terms of run time improvement of PAxML over parallel fastDNAml, compared to the results obtained on conventional PC processor architectures (see Chapter 6, pp. 87). The problem

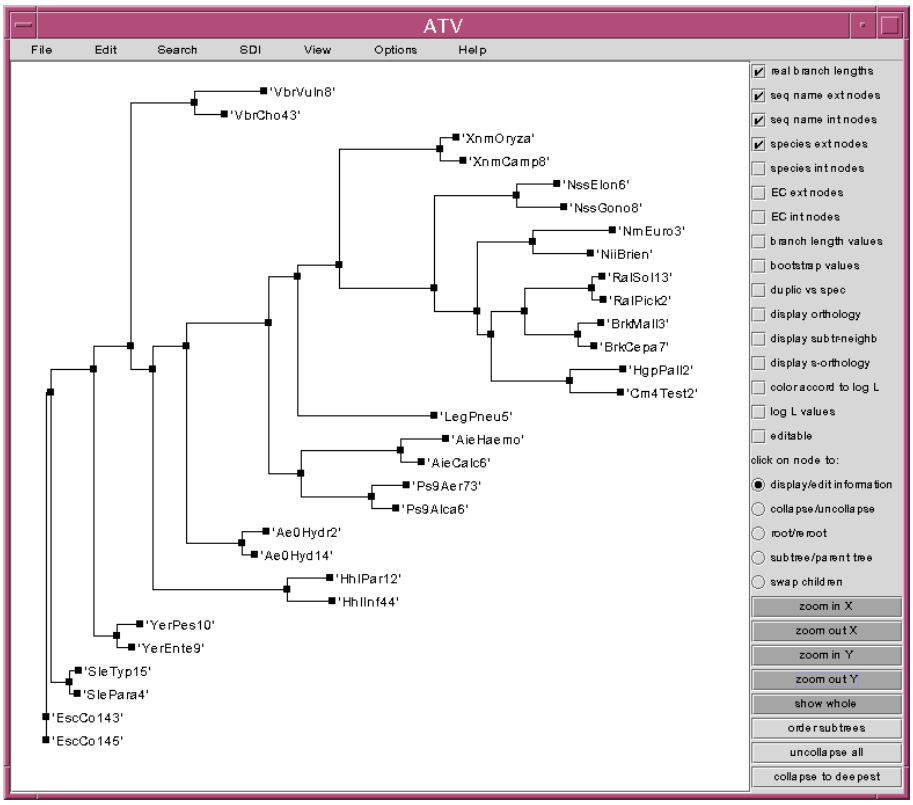

Figure 5.4: GAxML tree visualization with 29 taxa inserted

could however be quickly identified. The case analysis of formula 4.3 (p. 59) in Section 4.1 was originally implemented as nested conditional statement within the computationally expensive for-loops of functions `newview()`, `makenewz()`, and `evaluate()` which calculate the likelihood. This implementation significantly perturbs the pipelining and prefetch mechanisms of Hitachi's hardware architecture.

Therefore, the for-loops are split up within the `newview()`, `makenewz()`, and `evaluate()` functions, and a distinct for-loop is executed for each case. Thus, the evaluation of further conditional statements is not required within the respective loops.

This modification improved program efficiency both in terms of floating point performance and run time reduction, although some additional code had to be inserted for precalculating the loop split.

For example the non-adapted PAxML code rendered 21% of run time improvement compared with 28% for the adapted one over parallel fastDNAml for a 250 taxon phylogenetic analysis executed on 14 workers.

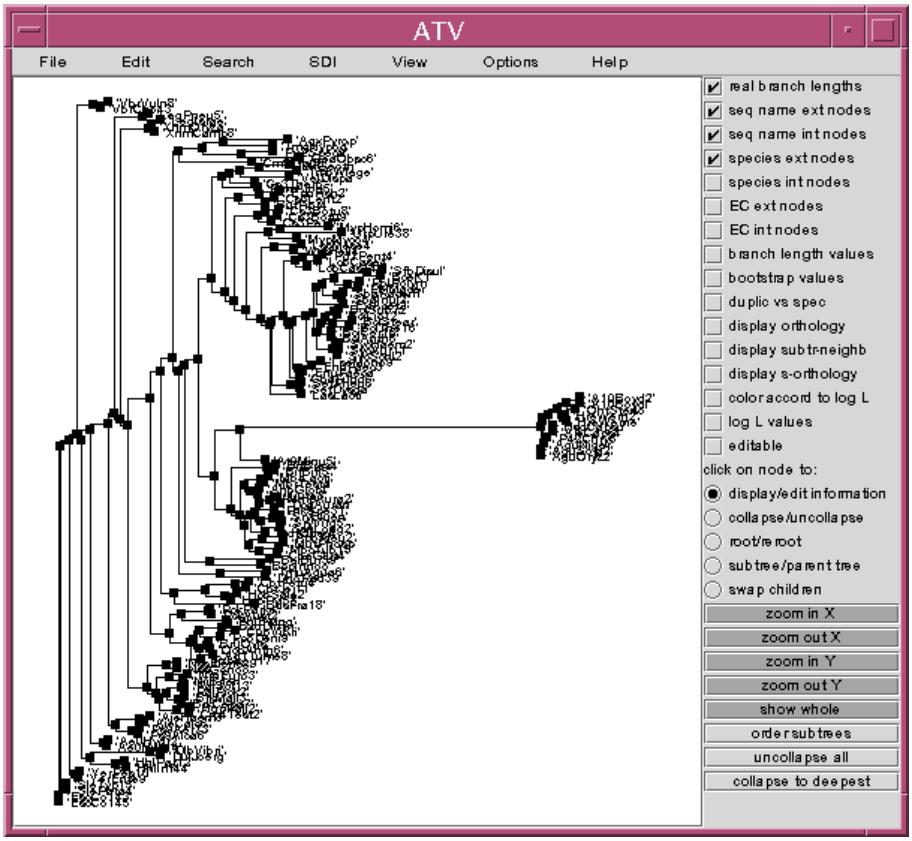

Figure 5.5: GAxML tree visualization with 127 taxa inserted

However, due to the significantly superior efficiency of the SEV technique on PC processors coupled with the substantially lower cost of those platforms, the largest amount of computations was conducted on large PC clusters.

5.2 Parallel and Distributed Solutions for RAxML

This final Section covers the parallel (Section 5.2.1) and distributed (Section 5.2.2) implementations of the significantly faster algorithm of RAxML. As outlined in Section 4.2 (pp. 62) RAxML incorporates novel search space heuristics, which enable inference of 1000-taxon trees in less than 24 hours on a single CPU in contrast to thousands of CPU hours required by PAxML. The RAxML code inherited the SEV implementation from AxML but deploys a significantly different parallelization scheme.

5.2.1 Parallel RAxML

The parallel implementation is based on a simple master-worker architecture and consists of two phases.

In **phase I** the master distributes the alignment file to all worker processes if no common file system is available, otherwise it is read directly from the file. Thereafter, each worker independently computes a randomized parsimony starting tree and sends it to the master process. Alternatively, it is also possible to start the program directly in **phase II** by specifying a tree file name in the command line.

In **phase II** the master initiates the optimization process for the best parsimony or specified starting tree. Due to the high speed of a single topology evaluation of a specific subtree rearrangement by function `rearrangeSubtree()` and the high communication cost, it is not feasible to distribute work by single topologies as e.g. in parallel fastDNAml. Another important argument for a parallelization based upon whole subtrees is that only in this way likelihood vectors at nodes can be reused efficiently within a slightly altered tree (see Section 3.9.1, pp. 41). Therefore, work is distributed by sending the subtree ID (of the subtree to be rearranged) along with the currently best topology `t_best`, to each worker.

The `sequential` and `parallel` implementation of RAxML on the master-side is outlined in the pseudocode of function `rearr()` which actually executes subtree rearrangements. Each worker simply executes function `rearrangeSubtree()`.

```
void rearr(tree t_best, int rL, int rU, boolean a)
{
 boolean impr;
 worker w;
 for(i = 2; i < #species * 2 - 1; i++){
   if(sequential){
     impr = rearrangeSubtree(t_best, i, rL, rU, a);
     if(impr) applySubsequent(t_best, i);
   }
   if(parallel){
     if(w = workerAvailable)
       sendJob(w, t_best, i);
     else putInWorkQueue(i);
   }
 }
 if(parallel){
   while(notAllTreesReceived){
     w = receiveTree(w_tree);
     if(likelihood(w_tree) > likelihood(t_best))
       t_best = w_tree;
     if(notAllTreesSent)
       sendJob(w, t_best, nextInWorkQueue());
  }
 }
}
```

In the sequential case rearrangements are applied to each individual subtree `i`. If the tree improves through this subtree rearrangement `t_best` is updated

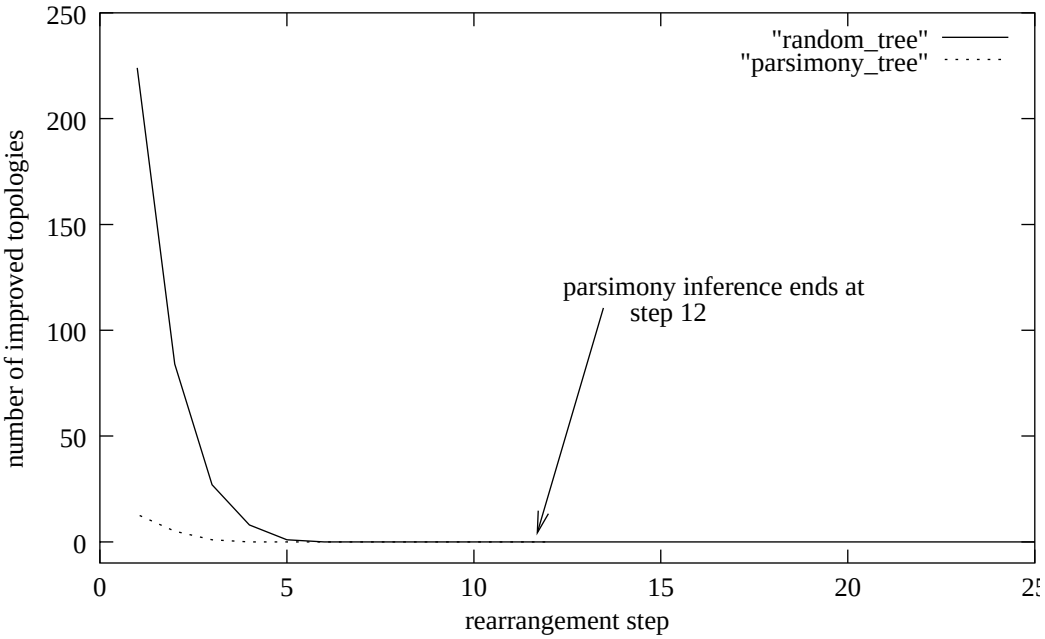

Figure 5.6: Number of improved topologies per rearrangement step for a SC_150 random and parsimony starting tree

accordingly, i.e. subsequent topological improvements are applied. In the parallel case subtree IDs are stored in a work queue. Obviously, the subsequent application of topological improvements during 1 rearrangement step (1 invocation of `rearr()`) is closely coupled. Therefore, the algorithm is slightly modified to break up this dependency according to the following observation: Subsequent improved topologies occur only during the first 3–4 rearrangement steps (initial optimization phase). This behavior is illustrated in Figure 5.6 where the number of subsequently improved topologies per rearrangement step for a phylogenetic reconstruction of a 150 taxon tree with a random and a parsimony starting tree is plotted.

After the initial optimization phase, likelihood improvements are achieved only by function `optimizeList20()`. This phase requires the largest amount of computation time, especially with huge alignments ($\approx$ 80% of execution time).

Thus, during the initial optimization phase only one single subtree ID `i=2,...,#species * 2 - 1` is sent along with the currently best tree `t_best` to each worker for rearrangements. Each worker returns the best tree `w_tree` obtained by rearranging subtree `i` within `t_best` to the master. If `w_tree` has a better likelihood than `t_best` at the master, `t_best = w_tree` is set and the updated best tree is distributed to each worker

along with the following work request. The program assumes that the initial optimization **phase IIa** is terminated if no subsequently improved topology has been encountered during the last three rearrangement steps.

In the final optimization **phase IIb**, communication costs are reduced and granularity is increased by generating only $5 * \#workers$ jobs (subtree ID spans). Finally, irrespective of the current optimization phase the best 20 topologies (or $\#workers$ topologies if $\#workers > 20$) computed by each worker during one rearrangement step are stored in a local worker tree list. When all `#species * 2 - 3` subtree rearrangements of `rearr()` have been completed, each worker sends its tree list to the master. The master process merges the lists and redistributes the 20 ($\#workers$) best tree topologies to the workers for branch length optimization. When all topologies have been globally optimized the master starts the next iteration of function `RAxML()` (see Section 4.2, pp. 62).

Due to the required changes to the algorithm the parallel program is non-deterministic, since final output depends on the number of workers and on the arrival sequence of results for runs with equal numbers of workers, during the initial optimization **phase IIa**. This is due to the altered implementation of the subsequent application of topological improvements during the initial rearrangement steps which leads to a traversal of search space on different paths. However, this solution represents a feasible and efficient approach both in terms of attained speedup values and final tree likelihood values (see Section 6.5.2, pp. 105).

The parallel implementation of RAxML is also described in [122]. The program flow of the parallel algorithm is outlined in Figure 5.7.

5.2.2 Distributed RAxML

The motivation to build a distributed seti@home-like [109] code is driven by the computation time requirements for trees containing more than 1.000 organisms and by the desire to provide inexpensive solutions for this problem which do not require supercomputers.

The main design principle of the distributed code is to reduce communication costs as far as possible and accept potentially worse speedup values than achieved with the parallel implementation. The algorithm of the http-based implementation is similar to the parallel program.

Initially, a compressed (gzipped) alignment file is transfered to all workers which start with the computation of a local parsimony starting tree. The parsimony tree is then returned to the master as in the parallel program.

However, the parallel and distributed algorithms differ in two important aspects which reduce communication costs:

Firstly, RAxML@home does not implement **phase IIa** but only **phase IIb** of the parallel algorithm, to avoid frequent communication and frequent exchange of tree topologies between master and workers.

Secondly, the lists containing the 20 best trees, irrespective of the number of workers, are optimized *locally* at the workers after completion of subtree rearrangements. The branch lengths of the trees in the list are optimized less exhaustively than in the sequential and parallel program. After this initial optimization only the best local tree is thoroughly optimized and returned to the master.

This induces some computational overhead and a slower improvement rate of the likelihood during the initial optimization phase (**phase IIa** of the parallel program) but remains within acceptable limits. The distributed program flow is depicted in Figure 5.8.

5.2.2.1 Technical issues

Some technical issues concerning the implementation of the http-based version of RAxML@home regarding communication, redundancy, and security will briefly be outlined at this point.

The communication infrastructure is provided by a http communication library. The most expensive part in terms of communication costs is the distribution of the alignment file which is compressed using `gzip`. The `gzip` utility shows sufficient compression rates for multiple alignments, e.g. a compression factor of 31 for a 1.000-taxon alignment.

To provide redundancy a queue with timeouts is used to ensure that every subtree rearrangement job is computed. Furthermore, failure procedures have been devised which are able to handle temporary master and worker failures.

An important security scenario is that some workers deliberately return phony trees. If the tree is not in the correct format, this can easily be detected by the routine which reads the respective tree string. The only serious security problem arises when a worker returns a tree that is in the correct format and has a "fake" likelihood, i.e. a likelihood value which is significantly better than the actual likelihood of the topology contained in the message and `t_best` at the master. In this case the likelihood of that topology is "quickly" verified by the master process. This quick verification only performs a superficial and fast likelihood computation of the tree in order to avoid excessive load of the master component. If the difference to the claimed likelihood in the tree string is $< 1\%$ the tree is accepted, otherwise it is rejected. Finally, the MD5 [81] (Message Digest number 5) checksum is used to provide some basic authentification of messages. A detailed technical description of RAxML@home is provided in [87].

Summary

This Chapter provided an overview over technical solutions which have been devised to obtain the required computational power for inference of large phylogenetic trees with AxML and RAxML respectively. For AxML a parallel, distributed, and Grid-enabled version has been presented. In addition, a special adaptation of AxML to the Hitachi supercomputer has been included. Finally, the parallel and metacomputing implementations of RAxML have been described. Table 5.1 summarizes the names and provides brief descriptions of all technical solutions presented in the current Chapter.

RAxML is currently being used to compute a phylogenetic tree containing 2437 mammalian sequences in cooperation with Olaf Bininda-Edmonds from the Lehrstuhl für Tierzucht at TUM.

The next Chapter comprises a performance evaluation of the novel algorithmic and technical solutions which were outlined in Chapter 4 (pp. 53) and the current Chapter respectively.

Program Name	Program Description
PAxML	MPI-based implementation of AxML, parallelization scheme derived from parallel fastDNAml
DAxML	Load Managed CORBA-based implementation, derived from PAxML
GAxML	Migrating Grid-enabled implementation of PAxML
Parallel RAxML	Non-deterministic MPI-based implementation of RAxML
Distributed RAxML	Non-deterministic http-based distributed implementation of RAxML

Table 5.1: Summary of technical solutions for AxML and RAxML

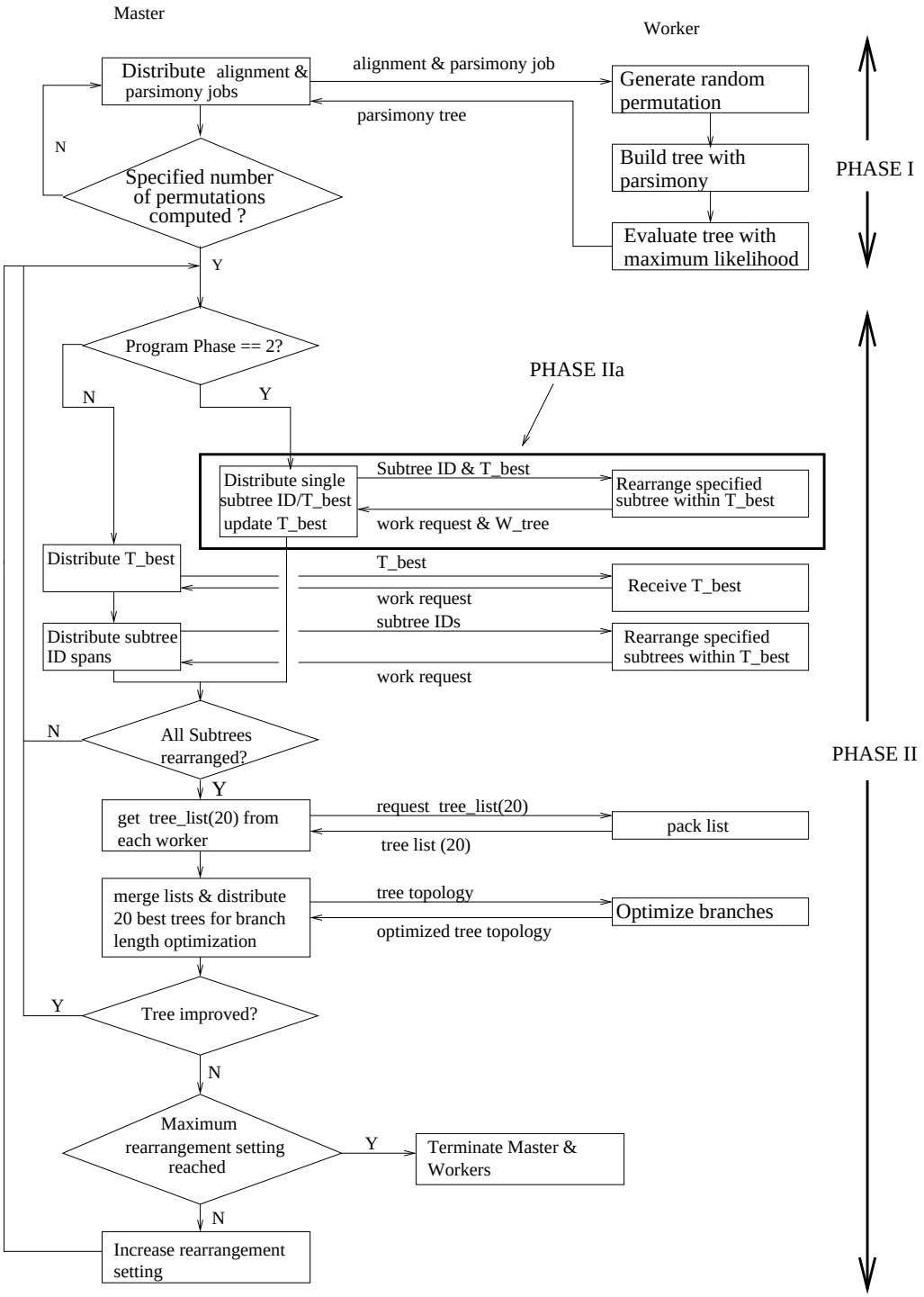

Figure 5.7: Parallel program flow of RAxML

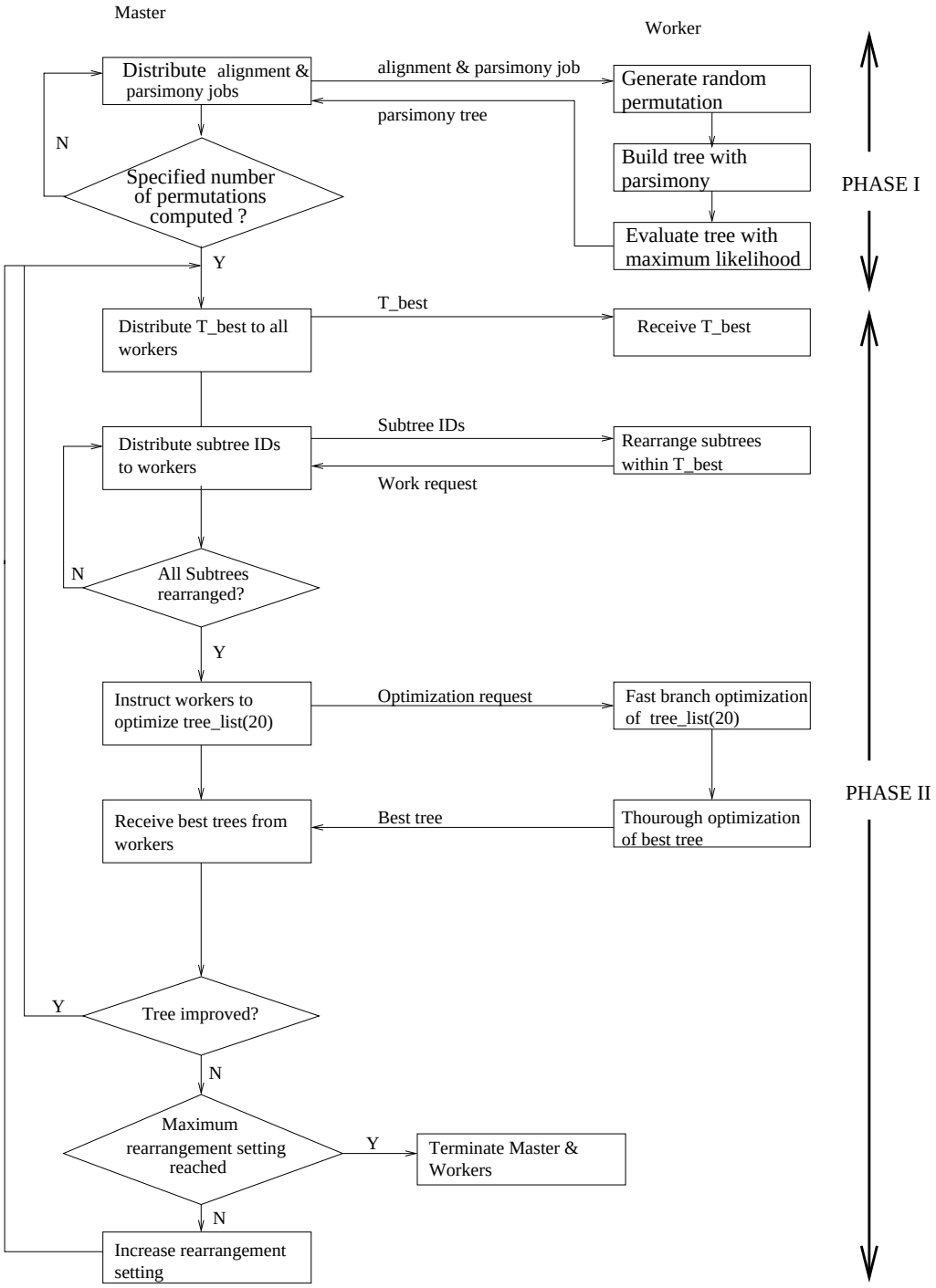

Figure 5.8: Program flow of distributed RAxML

6

Evaluation of Technical and Algorithmic Solutions

Ein jeder ist der Nabel seiner Welt.

Nikolaos Patsiouras

This Chapter summarizes the quantitative and qualitative improvements induced by the implementation of the respective algorithmic as well as technical solutions in AxML and RAxML which are presented in Chapters 4 and 5. Initially, the test data and platforms for the experiments are described. Thereafter, the performance of algorithmic ideas is analyzed and compared to current state of the art programs. The subsequent Section 6.5 covers the performance analysis of technical solutions for AxML and RAxML. The last Section describes the parallel inference of a 10.000 taxon phylogeny with RAxML which has been published in [122].

6.1 Test Data

For conducting experiments alignments comprising 150, 200, 250, 500, 1.000, 2.025 and 10.000 taxa (150_ARB,...,10000_ARB) have been extracted from the ARB small subunit ribosomal ribonucleic acid (ssu rRNA) database [76]. The alignments from ARB contain organisms from the domains Eukarya, Bacteria and Archaea.

In addition, the 56, 101 and 150 sequence data sets (56_SC, 101_SC, 150_SC [135]) were used, which can be downloaded at WWW.INDIANA.EDU/~RAC/HPC/FASTDNAML. Those data sets have been used by C. Stewart et al. to conduct performance analysis of parallel fastDNAml.

The 56_SC data set was used to extract some subalignments containing 10, 20, 30, 40, and 50 sequences (10_SC,...,50_SC) respectively. The larger 101_SC and 150_SC alignments have proved to be very hard to compute, in terms of convergence to best-known likelihood values, especially for MrBayes. According to a personal communication with C. Stewart this is due to the fact that these two data sets contain several hard to classify fungi which randomly scatter throughout the final trees.

Furthermore, two well-known real data sets of 218 [56] and 500 [21] sequences (218_RDPII, 500_ZILLA) were included into the test set. Those two alignments are considered to be ”classic” real data benchmarks. In particular the 500_ZILLA alignment has been extensively studied under the parsimony criterion.

Since Subtree Equality Vectors (SEVs) have also been implemented in TrExML (see Section 3.9.1.1, pp. 42) the respective test data sets with 10 up to 16 sequences (10_T,...,16_T) from the original TrExML publication [149] have also been used.

Finally, 50 synthetic 100-taxon alignments (100_SIM_1,...,100_SIM_50) with a length of 500 base pairs each were used. The respective true reference trees and alignments are available at WWW.LIRMM.FR/W3IFA/MAAS and are used in the comparative survey conducted for PHYML in [39] as well. Details on the generation of those data sets which contain e.g. varying sequence divergence rates can also be found in the cited paper.

For sake of completeness the number of base pairs (# bp) in each alignment is provided in Table 6.1 (ilb means: intentionally left blank).

data	#bp	data	#bp	data	#bp
10_SC	820	10_T	1200	150_ARB	3188
20_SC	820	11_T	1200	200_ARB	3270
30_SC	820	12_T	1200	250_ARB	3638
40_SC	820	13_T	1200	500_ARB	4030
50_SC	820	14_T	1200	1000_ARB	5547
56_SC	820	15_T	1200	2025_ARB	1517
101_SC	1858	16_T	1200	10000_ARB	1217
150_SC	1269	500_ZILLA	759	100_SIM_1–50	500
218_RDPII	4182	ilb	ilb	ilb	ilb

Table 6.1: Alignment lengths

6.2 Test & Production Platforms

A large variety of platforms was used to test and execute large production runs with the sequential, parallel, and distributed versions of AxML and RAxML respectively. For sequential tests various Sun-SPARC, Intel, and AMD processors have been used. The parallel versions of (R)AxML have been compiled and executed on the following platforms:

- Regionales RechenZentrum Erlangen (RRZE [100]): Linux Cluster, equipped with 168 Xeon 2.66GHz processors, interconnected by Gigabit-Ethernet.
- Institut für Wissenschaftliches Rechnen (IWR [43]): HEidelberg LInux Cluster System (HELICS), with 512 AMD Athlon 1.4GHz processors linked by Myrinet.
- Leibniz Rechenzentrum (LRZ [46]): Hitachi SR-8000F1 supercomputer.
- Max-Planck Institut für Strahlenphysik (MPI [79]): SGI Origin 2000.
- Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR [55]): Infiniband Cluster, based on 12 Xeon 2.4GHz and 8 Itanium2 1.3GHz processors, connected via Infiniband.
- Chair for Computer Science in Engineering, Science, and Numerical Programming (TUM [19]): Linux Cluster with 16 Intel Pentium III processors linked by Myrinet.

Finally, up to 50 processors from a cluster of Sun-workstations (Sun-Halle [138]), which is available to CS students at TUM for conducting their homework etc. has been used to carry out initial scalability tests with RAxML@home.

6.2.1 Adequate Processor Architectures

Since neither parallel AxML, nor parallel RAxML have excessive communication costs, compared e.g. to *classic* supercomputer applications like numerical simulations of fluid dynamics, the main criterion regarding the selection of adequate platforms consists in the availability of appropriate processor architectures. Due to the fact that AxML and RAxML use the same likelihood evaluation function including an identical SEV implementation the CPU architecture considerations are analogous for both programs.

Furthermore, the considerations concerning sequential and parallel versions are identical, since the core of both programs which mainly influences execution speed is the likelihood evaluation function.

All experiments conducted so far have clearly shown that standard PC processor architectures, like the AMD Athlon and Opteron series or the Intel Pentium and Xeon series represent the best choice for execution of (R)AxML.

The execution time acceleration of AxML over fastDNAml on these architectures was constantly significantly better ($\approx$ 60%) than on Sun-SPARC architectures ($\approx$ 40%), an SGI Origin 2000 ($\approx$ 30%) and the Hitachi SR-8000F1 supercomputer ($\approx$ 30%) for *exactly identical* input data.

This is mainly due to the specific implementation of SEVs in (R)AxML which require a larger amount of integer operations to compute the subtree equality vector at each node. Furthermore, the elaborate conditional statement of Formula 4.3 (p. 59) in Section 4.1 induces a significant number of conditional jumps within *the* main for-loops of the program in the likelihood evaluation and branch-length optimization functions. In contrast to traditional supercomputer architectures which have mainly been designed for number crunching and regular access schemes to large fields of floating point numbers, PC processors are better suited to handle the increased amount of conditional statements, integer arithmetics, jumps, and the irregular data access in graphs.

As already mentioned in Section 5.1.4 (pp. 77) an effort has been undertaken to adapt PAxML to the Hitachi SR-8000F1 supercomputer via appropriate loop transformations and compiler options [132]. However, PC processor architectures remain the better choice, not only in terms of expected program acceleration, but also in terms of hardware cost.

Due to this circumstance in conjunction with the low communication requirements, PC processors and clusters represent the most adequate and most inexpensive platform for execution of AxML and RAxML. Finally, RAxML@home does not even rely on a cluster infrastructure for inference of large phylogenies.

6.2.2 Performance of PC Processors

RAxML has been used among several other applications to benchmark various recent PC processor architectures at the Lehrstuhl für Rechnertechnik und Rechnerorganisation. For this purpose RAxML was executed with 150_SC for the same starting tree such as to generate exactly equivalent runs, on the CPUs which are listed in Table 6.2. This table also includes the respective compilers which were used for RAxML with an optimization level of `-O3`. In these experiments the native Intel compiler (icc) and the popular GNU compiler (gcc) have been used.

Furthermore, the effect of several advanced compiler options such as e.g. loop unrolling, frame pointer omission, or higher `-O` values was evaluated but no notable improvement of execution times could be observed (a similar behavior has been measured on the Hitachi SR8000-F1 for various compiler switches).

The most interesting result is that RAxML executes best on the AMD processor despite the fact that the program has only been compiled with `gcc`. This might be due to some known problems with branch prediction on Xeon processors.

CPU	compiler	secs
AMD Opteron 244	gcc-3.3.1	335
Intel Xeon 2.4 GHz	gcc-3.3.1	559
Intel Xeon 2.4 GHz	icc-7.1	465
Intel Itanium 1.3 GHz (64bit code)	icc-7.1	512

Table 6.2: RAxML execution times on recent PC processors for a 150 taxon tree

6.3 Run Time Improvement by Algorithmic Optimizations

This Section summarizes the run time improvements attained by the implementation of the SEV (Subtree Equality Vector) technique which is outlined in Section 4.1 (pp. 54). The performance improvements attained for the sequential implementation of SEVs are provided in Section 6.3.1 and the following Section 6.3.2 describes the effect of SEVs on parallel program performance.

6.3.1 Sequential Performance

The execution time improvements of AxML over fastDNAml [86] and ATrExML (SEV-based version of TrExML) over TrExML have been measured for a variety of data sets to demonstrate the efficiency of SEVs.

For those test the most recent release of fastDNAml (v1.2.2) has been used. AxML (v1.7) denotes the initial implementation of SEVs, whereas AxML (v2.5) represents a more sophisticated version of the same program which contains the additional SEV-based algorithmic optimizations presented in Section 4.1.1 (pp. 59).

Note, that all SEV-based implementations yield *exactly* identical results as the non-optimized programs, since the optimization is purely algorithmic.

Table 6.3 lists the sequential execution times of AxML (v1.7), AxML (v2.5) and fastDNAml for 150_ARB, 200_ARB, 250_ARB and 500_ARB and the run time improvement between AxML (v2.5) and fastDNAml (v 1.2.2) on an AMD Athlon 1.4GHz processor. Those tests were conducted without application of intermediate and final subtree rearrangements (see Section 3.9.1.1, pp. 42) in order

data	AxML v1.7	AxML v2.5	fastDNAml v1.2.2	improvement
150_ARB	748 secs	632 secs	1603 secs	60.57%
200_ARB	1443 secs	1227 secs	3186 secs	61.49%
250_ARB	2403 secs	2055 secs	5431 secs	62.16%
500_ARB	12861secs	10476 secs	26270 secs	60.13%

Table 6.3: Performance of AxML (v1.7), AxML (v2.5), and fastDNAml (v1.2.2)

to speed up computations, i.e. the final likelihood values were poor. Moreover, the final likelihood values are irrelevant within this context since the goal is to assess run time improvements for a variety of alignments.

Finally, Figure 6.1 indicates the accumulated evaluation time per topology class t_k during the stepwise addition process for AxML and fastDNAml with the quickadd option (see Section 3.9.1.1, pp. 42) enabled and disabled. This Figure demonstrates the impact of the quickadd option on program performance and that the SEV-technique scales equally well to both program options.

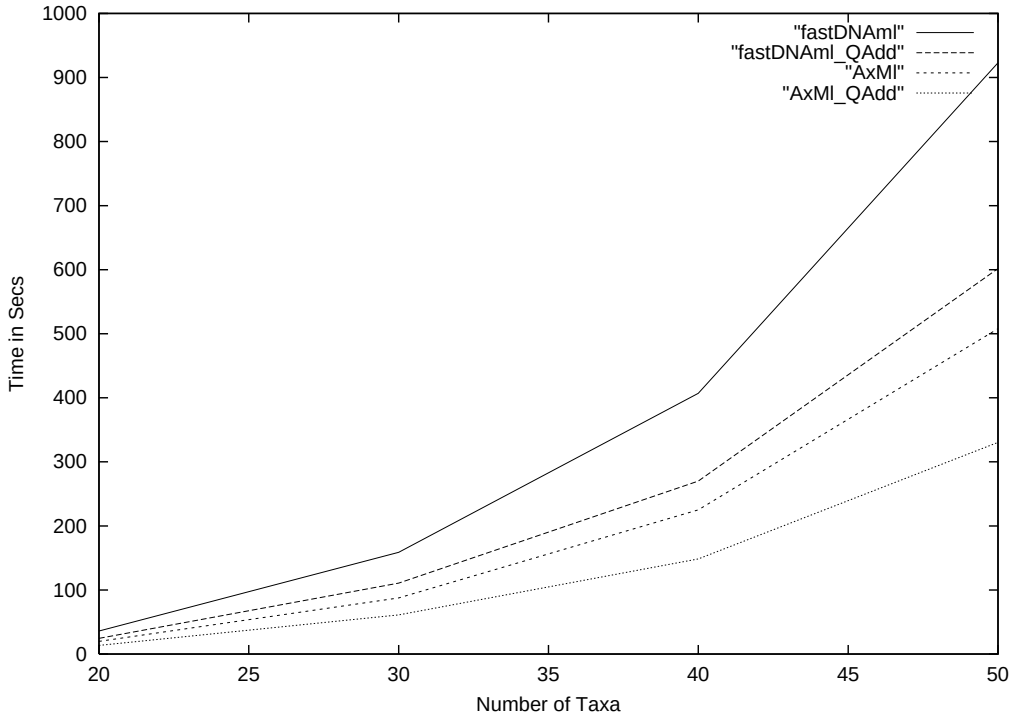

Figure 6.1: AxML and fastDNAml inference times over topology size for quick-add enabled and disabled

In Table 6.4 the run time improvement (in %) of ATrExML over TrExML is listed for the alignment data of the original TrExML publication. The parameter a indicates the size of the starting tree which is optimized exhaustively in TrExML (see Section 3.9.1.1, pp. 42). These experiments demonstrate the general applicability of SEVs to distinct heuristic search algorithms and implementations. The test with TrExML have been conducted on a Sun-Sparc 1000.

data	a	impr.	data	a	impr.	data	a	impr.
10_T	8	38.21%	10_T	9	38.24%	10_T	10	37.23%
11_T	8	38.67%	11_T	9	38.91%	11_T	10	38.39%
12_T	8	39.58%	12_T	9	39.91%	12_T	10	39.94%
13_T	8	40.02%	13_T	9	40.77%	13_T	10	41.08%
14_T	8	39.87%	14_T	9	40.19%	14_T	10	42.26%
15_T	8	40.68%	15_T	9	41.04%	15_T	10	43.00%
16_T	8	40.71%	16_T	9	41.10%	16_T	10	42.26%

Table 6.4: Global run time improvements (impr.) TrExML vs. ATrExML

6.3.2 Parallel Performance

Larger test runs for comparison of PAxML with parallel fastDNAml were conducted using 150_ARB, 200_ARB, and 250_ARB. Intermediate local and final global rearrangements were conducted with a stepwidth of 1 on the Pentium III Linux cluster at the chair for Computer Science in Engineering, Science, and Numerical Programming at TUM.

The overall good scalability of the optimization to the parallel program is due to the fact that the tree evaluation function represents the core of the `worker` components, which perform the actual computation.

Therefore, in the following tables only the number of `worker` processes started is listed since the `foreman` and `master` components of parallel fastDNAml and PAxML hardly produce load. Table 6.5 provides the run time im-

data	#Workers	improvement
150_ARB	8	62.42%
200_ARB	12	63.29%
250_ARB	12	64.60%

Table 6.5: Execution time improvement of PAxML over parallel fastDNAml on a Pentium III Linux cluster

provements in per cent of PAxML over parallel fastDNAml on the Pentium III Linux cluster. Note that, those results are very similar to those obtained for the sequential version of AxML in Table 6.3, i.e. SEVs scale well to the parallel program. This Table demonstrates the actual potential of the SEV technique in terms of floating point operation reduction, especially on inexpensive processors with comparatively weak FPUs. On the other hand, the results obtained on the Hitachi SR-8000F1 confirm the significant impact of hardware architecture on the performance improvement of PAxML over fastDNAml. The values which are listed in Table 6.6 have been obtained with the especially adapted supercomputer program version of PAxML which is described in Section 5.1.4 (pp. 77). The obtained run time improvement of over 26% should not be underestimated however.

data	#Workers	Improvement
150_ARB	14	26.57%
200_ARB	14	28.52%
250_ARB	14	28.40%

Table 6.6: Execution time improvement of PAxML over parallel fastDNAml on the Hitachi SR8000-F1

Since PAxML does not implement heuristics but only a purely algorithmic optimization in all tests and on all platforms PAxML and parallel fastDNAml render *exactly* the same output tree, a fact that can be verified by a simple `diff` on the output files.

6.4 Run Time and Qualitative Improvement by Algorithmic Changes

This Section provides a comparative performance analysis between MrBayes, PHYML, and RAxML on synthetic (simulated) as well as real alignment data. MrBayes and PHYML implement the currently—to the best of the author's knowledge—most efficient and exact phylogenetic searches. Thus, those two programs represent the best state-of-the-art candidates for performance comparison with RAxML. Section 6.4.4 covers failure scenarios of bayesian phylogenetic analysis.

6.4.1 Experimental Setup

To facilitate and accelerate testing the HKY85 [42] model of sequence evolution has been used. Furthermore, the transition/transversion (ts/tv) ratio (see Sec-

tion 3.4.3, pp. 26) has been fixed at 2.0 except for the 150_SC (1.24) and 101_SC (1.45) alignments.

Since the transition/transversion ratio is defined differently in PHYML it has been scaled accordingly for the test runs. The manual of PAML [90] contains a nice description of differences in transition/transversion ratio definitions of different maximum likelihood implementations.

MrBayes does not provide a possibility to set the transition/transversion ratio to a specific value such that the program optimized this parameter in the respective test runs.

However, significant differences in the order of final RAxML, PHYML, and MrBayes likelihood values for different ts/tv settings could not be observed. This is illustrated in Figure 6.2 for the likelihood of the respective final 150_SC topologies over different transition/transversion parameter settings. The significant difference between the likelihood values of the bayesian analysis (MrBayes) and the maximum likelihood analyses (PHYML, RAxML) in this graph indicates that the bayesian inference failed to converge to a biologically reasonable tree for 150_SC. Recall from Section 3.4.1 (pp. 25) that all likelihood values indicated in the current Chapter are log likelihood values and that trees with higher likelihood values are better.

The likelihood values of the final tree topologies of PHYML, RAxML, and MrBayes have been computed with fastDNAml since the likelihood value of a specific topology varies among distinct programs due to numerical differences in their implementations. For example the final 1000_ARB topologies included in Table 6.7 yielded a likelihood of -401118.27 (PHYML tree) and -399775.12 (RAxML tree) respectively when evaluated with PHYML. For 218_RDPII the likelihood values computed with PHYML were -156859.84 (PHYML tree) and -156562.51 (RAxML tree).

For real data MrBayes was executed for 2.000.000 generations using 4 Metropolis-Coupled MCMC (MC^3) chains and the random starting trees (recommended program settings). Furthermore, the sample and print frequency was set to 5000. To enable a fair comparison all 400 output trees have been evaluated with fastDNAml and the value of the topology with the best likelihood and the execution time at that point is reported. For synthetic data MrBayes was executed for 100.000 generations using 4 MC^3 chains and random starting trees. In these experiments sample and print frequencies were set to 500 and a majority-rule consensus tree was built using the last 50 trees. Those significantly faster settings proved to be sufficient since trees for synthetic data converged much faster than trees for real data in the experiments. Exactly identical settings have been used by Guidon et al. [39].

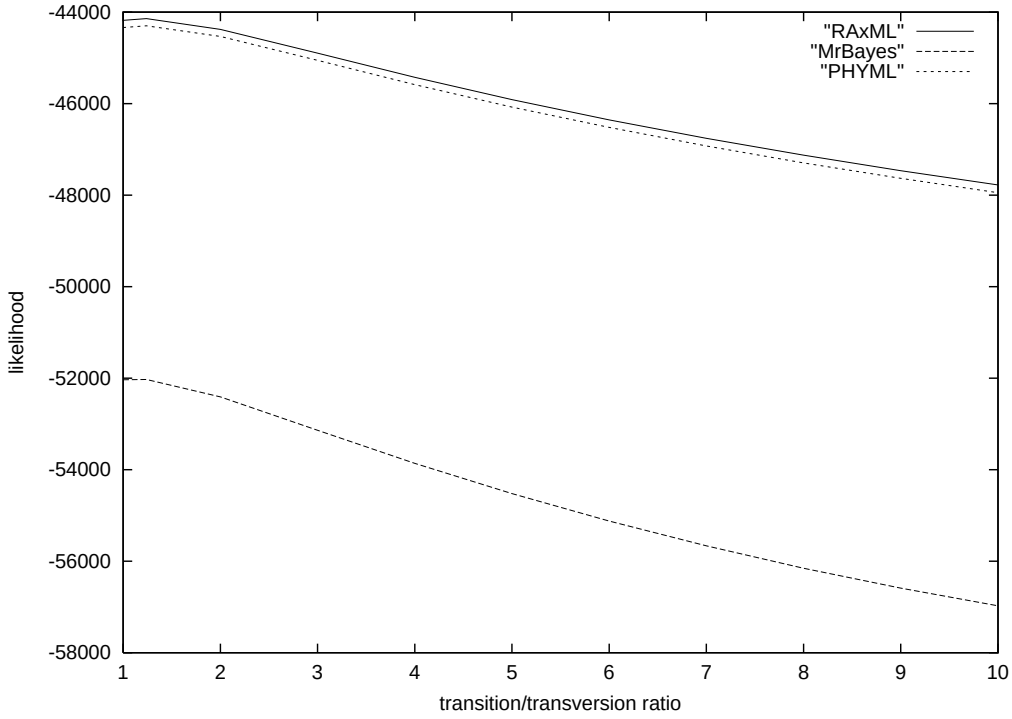

Figure 6.2: RAxML, PHYML, and MrBayes final likelihood values over transition/transversion ratios for 150_SC

Finally, the importance of using several real data alignments becomes evident in these experiments since differences between phylogeny programs can often only be observed with real data.

All sequential tests were performed on an Intel Xeon 2.4 GHz processor. The programs were compiled using `icc -O3` (native Intel compiler).

All alignments including the best final topologies are available along with the RAxML source code at WWWBODE.CS.TUM.EDU/~STAMATAK.

6.4.2 Real Data Experiments

Tables 6.7 and 6.8 (n/a means: data not available) summarize the final likelihood values and execution times in seconds or hours obtained with PHYML, MrBayes, and RAxML. The results listed for RAxML correspond to the best of 10 runs with different randomized parsimony starting trees. For sake of completeness the worst results and worst execution times obtained with RAxML for each data set are listed in a separate Table 6.9 (ilb means: intentionally left blank).

data	PHYML likelihood	secs	RAxML likelihood	secs	R > PHY likelihood	secs
101_SC	-74097.6	153	-73919.3	617	-74046.9	31
150_SC	-44298.1	158	-44142.6	390	-44262.9	33
150_ARB	-77219.7	313	-77189.7	178	-77197.6	67
200_ARB	-104826.5	477	-104742.6	272	-104809.0	99
250_ARB	-131560.3	787	-131468.0	1067	-131549.4	249
500_ARB	-253354.2	2235	-252499.4	26124	-252986.4	493
1000_ARB	-402215.0	16594	-400925.3	50729	-401571.9	1893
218_RDPII	-157923.1	403	-157526.0	6774	-157807.9	244
500_ZILLA	-22186.8	2400	-21033.9	29916	-22036.9	67

Table 6.7: PHYML, RAxML execution times and likelihood values for real data

data	MrBayes likelihood	hrs	PAxML likelihood	hrs
101_SC	-77191.5	11	-73975.9	47
150_SC	-52028.4	14	-44146.9	164
150_ARB	-77196.7	8	-77189.8	300
200_ARB	-104856.4	43	-104743.3	775
250_ARB	-133238.3	44	-131469.0	1947
500_ARB	-263217.8	102	-252588.1	7372
1000_ARB	-459392.4	141	-402282.1	9898
218_RDPII	-158911.6	38	n/a	n/a
500_ZILLA	-22259.0	27	n/a	n/a

Table 6.8: MrBayes, PAxML execution times and likelihood values for real data

In addition, since execution times of RAxML might appear long compared to PHYML column R>PHY indicates the likelihood and the time at which RAxML passed the final likelihood obtained by PHYML for a distinct series of RAxML runs. Finally, the last two columns of Table 6.8 show the final likelihood values and execution times in hours (!) obtained with PAxML. Those results were obtained from parallel executions of PAxML on the HeLiCs [43] cluster and the highest feasible rearrangement setting, in terms of acceptable computation times.

The tremendous reduction of execution times between Tables 6.7 and 6.8 illustrates the algorithmic progress in the field over the last two years, i.e. PAxML can be considered as state-of-the-art program for 2002 but is nowadays easily outperformed by RAxML and PHYML. To date the main contribution of PAxML consists in the introduction of the SEV-technique which was inherited by RAxML.

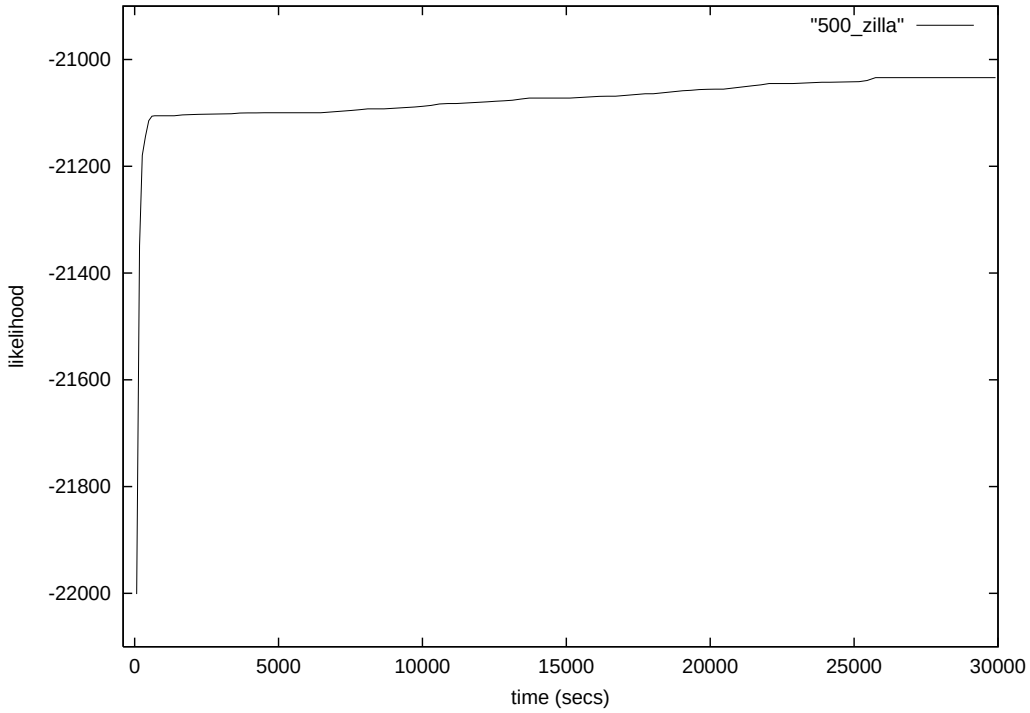

Figure 6.3: RAxML likelihood improvement over time for 500_ZILLA

The long overall execution times of RAxML compared to PHYML are due to the asymptotic convergence of likelihood over time which is typical for the tree optimization process. A particularly extreme example for this type of convergence behavior is illustrated in Figure 6.3 for 500_ZILLA. Therefore, the comparatively small differences in final likelihood values which are usually below 1% should not be underestimated, in terms of the computational effort required to obtain those values. In addition, those apparently small differences prove to be significant when the likelihood-ratio test is applied (see Section 3.8, pp. 39). Despite the fact that 90–95% accuracy is often considered excellent in heuristics for hard optimization problems, heuristics used in phylogenetic reconstruction must be much more accurate. Recent work [146] has revealed that trees computed with maximum parsimony which showed an error rate in respect to the optimal parsimony score of more than 0.01% yielded topologically poor estimates of the real tree. Thus, heuristics for maximum parsimony require at least 99.99% accuracy and probably significantly more on very large data sets to produce topologically accurate (biologically meaningful) trees. At present there exists no analogous survey for maximum likelihood but it is very likely that the required degree of accuracy is similar if not higher. The determination of the required level of accuracy for

data	RAxML	secs	data	RAxML	secs
101_SC	-73982.42	1021	500_ARB	-252631.93	26124
150_SC	-44159.89	467	1000_ARB	-401006.52	66902
150_ARB	-77198.98	305	218_RDPII	-157580.21	7432
200_ARB	-104743.32	1236	500_ZILLA	-21087.46	29916
250_ARB	-131513.04	1758	ilb	ilb	ilb

Table 6.9: Worst execution times and likelihood values for real data from 10 RAxML runs

maximum likelihood-based analyses and the establishment of stopping criteria represents a current issue of research in phylogenetics.

6.4.3 Simulated Data Experiments

Figure 6.4 provides the topological accuracy (relative Robinson-Foulds rate [101], see Section 3.8, pp. 39) of PHYML, RAxML, and MrBayes for 50 distinct 100-taxon alignments which are enumerated on the x-axis.

Recall from Section 3.8 (pp. 39) that the Robinson-Foulds rate provides a measure of relative topological dissimilarity between two trees. A low RF rate indicates that the tree under consideration is topologically closer to the reference tree or true tree in case of synthetic data.

The average Robinson-Foulds rate over the 50 synthetic alignments used in this study is 0.0796 for PHYML, 0.0808 for RAxML, 0.0818 for RAxML with a less exhaustive search and 0.0741 for MrBayes. The average execution time of RAxML was 131.05 seconds and 29.27 seconds for the less exhaustive analysis. PHYML required an average of 35.21 seconds and MrBayes 945.32 seconds.

The experiments illustrate that there appears to be no significant difference between PHYML and RAxML for synthetic data in contrast to the results obtained with real data. Thus, real as well as synthetic data should be used to perform comparative analyses of phylogeny programs. Note, that all 3 programs performed well on synthetic data, since the average topological error rate is below 0.1%.

6.4.4 Pitfalls & Performance of Bayesian Analysis

Two examples which underline potential pitfalls of bayesian analysis with real world alignment data are outlined in Figures 6.5 and 6.6 for the 101_SC and 150_SC alignments respectively. In Figure 6.5 the MrBayes likelihood values are plotted over generation numbers for a MrBayes program execution with a RAxML and a random starting tree. Figure 6.6 plots the likelihood values for 150_SC over

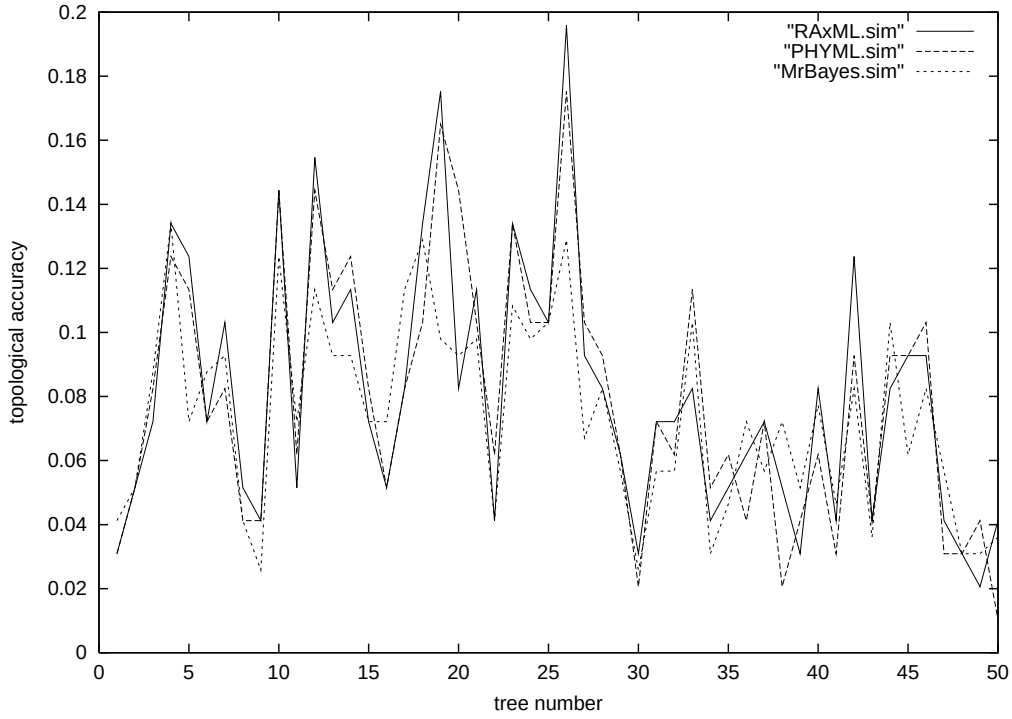

Figure 6.4: Topological accuracy of PHYML, RAxML and MrBayes for 50 100-taxon trees

time for a bayesian and RAxML-based optimization of an identic random starting tree. In addition, Figure 6.6 confirms the rapid tree optimization capabilities of RAxML for random starting trees (the final tree of RAxML showed a likelihood of -44149.18). An additional example of rapid random tree optimization by RAxML in comparison to MrBayes is provided in Figure 6.7 for 150_ARB. The respective final RAxML topology had a likelihood value of -77189.78. However, at least in this example MrBayes does not fail to converge.

The two plots in Figures 6.5 and 6.6 underline the main problem of MCMC analysis which is also pointed out by Huelsenbeck in [48]: When to stop the chain? In both examples the bayesian analysis of random starting trees seems to have reached apparent stationarity. The observed behavior confirms the theoretical concerns about Markov Chain Monte Carlo algorithms described in Section 3.5 (pp. 32) and outlined in Figure 3.9 (p. 35) even though 4 Metropolis-Coupled chains were used in the real world examples presented at this point.

Furthermore, Figure 6.8 and 6.5 demonstrate that “good” user trees obtained by RAxML are useful both as reference and starting trees as well as to significantly accelerate MrBayes.

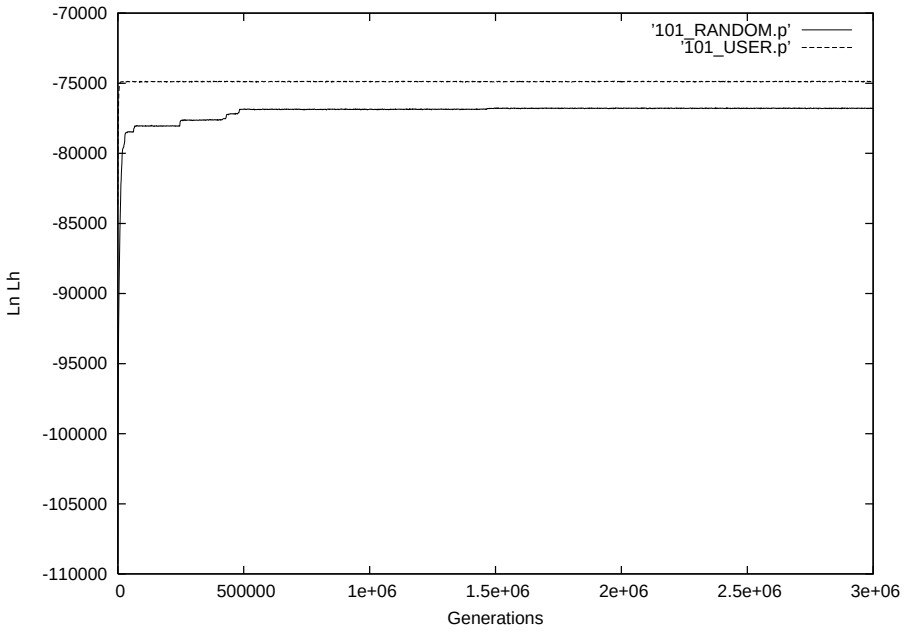

Figure 6.5: Convergence behavior of MrBayes for 101_SC with user and random starting trees over 3.000.000 generations

This justifies the work on fast “traditional” maximum likelihood methods after the emergence and great impact of bayesian methods. Thus, RAxML is not regarded as competitor to MrBayes, but rather as useful tool to improve bayesian inference and vice versa. Therefore, in order to facilitate the analysis process RAxML produces an output file containing the alignment and the final tree in MrBayes input format.

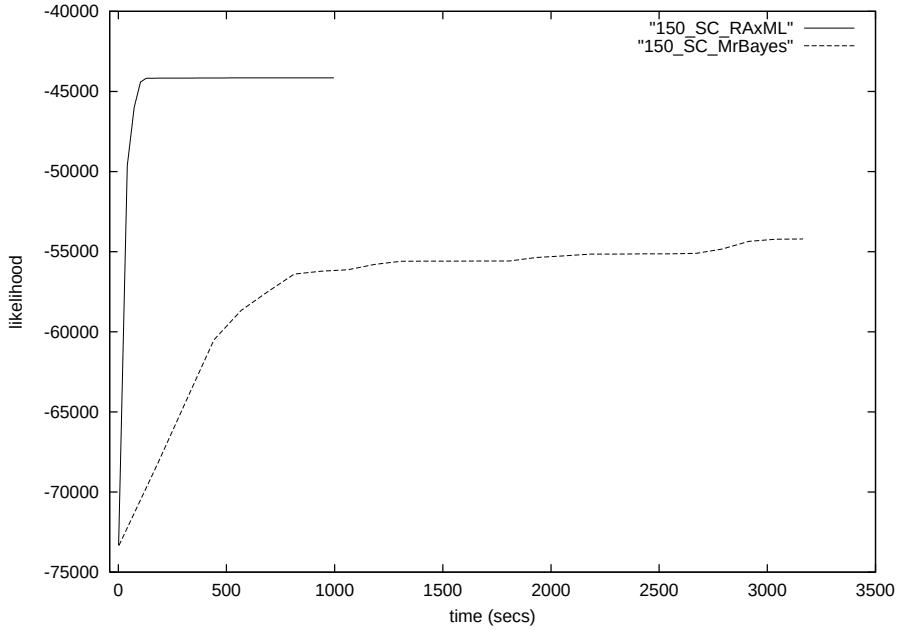

Figure 6.6: 150_SC likelihood improvement over time of RAxML and MrBayes for the same random starting tree

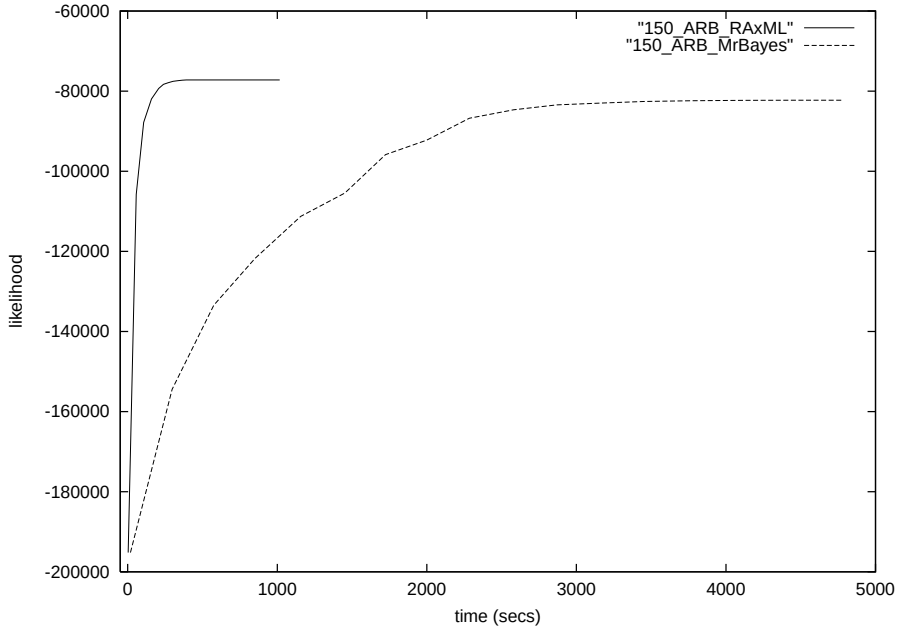

Figure 6.7: 150_ARB likelihood improvement over time of RAxML and MrBayes for the same random starting tree

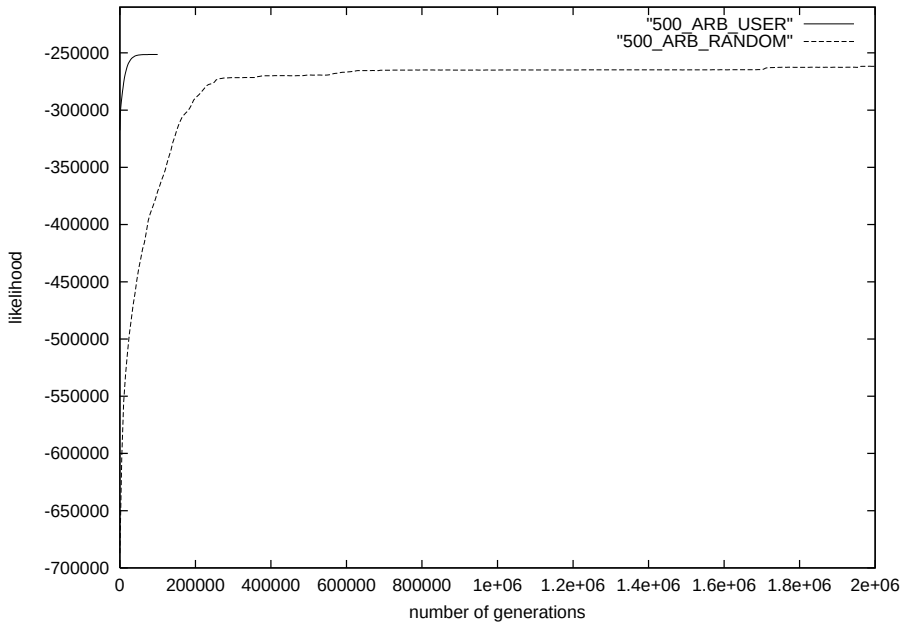

Figure 6.8: Convergence behavior of MrBayes for 500_ARB with user and random starting trees

6.5 Assessment of Technical Solutions

The present Section covers the performance evaluation of the load balanced distributed implementation of AxML and the parallel as well as distributed implementations of RAxML.

6.5.1 Distributed Load-managed AxML

Performance analysis tests with DAxML were conducted on 4 Ethernet connected Sun-Blade-1000 machines of the SUN workstation cluster at the Lehrstuhl für Rechnertechnik und Rechnerorganisation. The 20_SC, 30_SC, 40_SC and 50_SC alignments have been used to evaluate the behavior of DAxML and LMC in terms of CORBA/JNI overhead, impact of the algorithmic optimizations (SEVs), and automatic worker object replication/migration. In Figure 6.9 the impact of the SEV implementation on the speed of the tree evaluation function including JNI and CORBA overhead is plotted.

Two DAxML test runs with a single worker object were conducted, using the standard and optimized tree evaluation function on the 40_SC alignment. The average tree evaluation time per topology class t_k during stepwise addition (see Section 5.1.2.2, pp. 70) was measured. The algorithmic optimizations show anal-

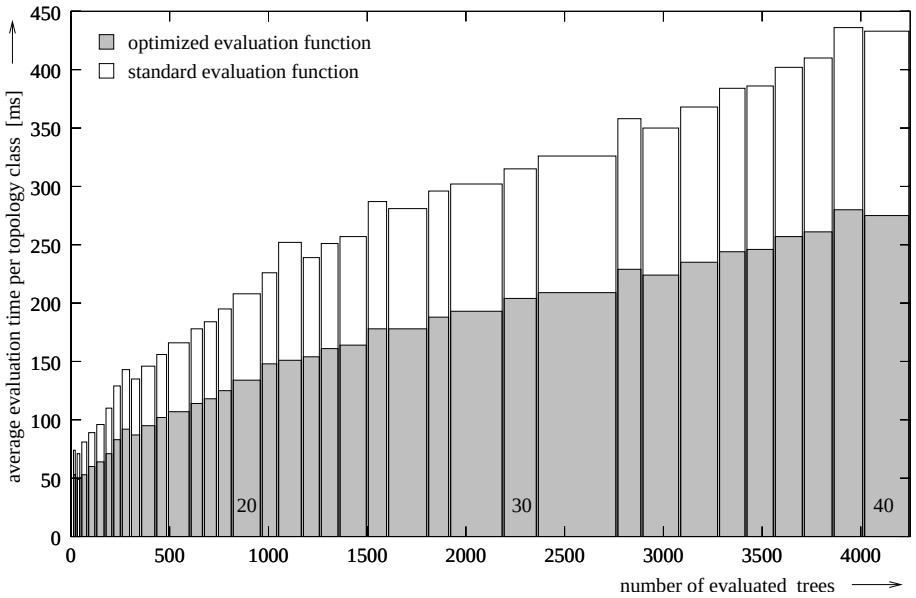

Figure 6.9: Average evaluation time improvement per topology class: optimized (SEV-based) DAxML evaluation function vs. standard fastDNAml evaluation function

ogous performance improvements as observed for the parallel and sequential version of AxML in Section 6.3. All subsequent tests were performed using the optimized evaluation function.

Another important issue is the overhead induced by the integration of CORBA and JNI into DAxML. The communication overhead decreases with increasing tree size (see Figure 6.10), because average evaluation time per tree increases during the computation as depicted in Figures 6.9 and 6.10, whereas the amount of communicated data per topology class remains practically constant. For the same reasons and despite the fact, that some heavy-weight JNI mechanisms such as JAVA callbacks from C have been deployed, the JNI overhead becomes neglectible as the tree grows, since only small amounts of data are passed via JNI.

The average C, JNI, and CORBA tree evaluation times for selected topology classes of size 4, 10, 20, 30, and 40 were measured. As can be seen in Figure 6.10 during the initial phase of the computation, i.e. for size 4 and 10, the CORBA overhead is relatively high but decreases significantly with increasing topology size.

In order to demonstrate the efficiency and soundness of LMC additional test runs using worker object replication and migration have been conducted.

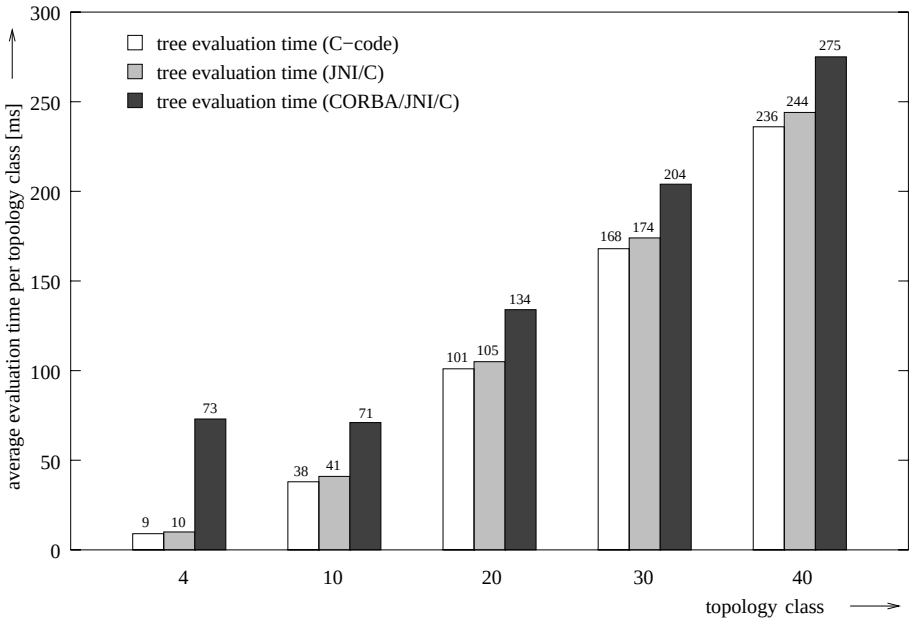

Figure 6.10: JNI and CORBA-communication overhead

Figure 6.11 depicts the correct response of LMC to an increase of background load on a worker object host. Two independent test runs with 40_SC and a single worker object were executed, (i.e. the replication mechanism was switched off) which were located on the same initially unloaded node to measure the evaluation time per topology. Around the evaluation of the 1750th tree topology during the first test run external load was created on the worker object host, which provoked a significant increase in topology evaluation time. The unfavorable situation is correctly resolved by the load balancer via a migration of the worker object to an unloaded host. Finally, Figure 6.12 demonstrates how the average evaluation time per topology class is progressively being improved by 3 subsequent automatic worker object replications performed by LMC, in comparison to a run with automatic replication switched off.

6.5.2 Parallel RAxML

In order to measure the speedup parallel tests with a fixed starting tree for 1000_ARB were conducted. The program was executed on the Hitachi SR8000-F1 [46] at LRZ using 8, 32, and 64 processors (1, 4 and 8 nodes) in intra-node MPI mode, as well as on the 2.66GHz Xeon cluster [100] at RRZE on 1, 4, 8, 16, and 32 processors. For calculating the speedup values only the number of worker

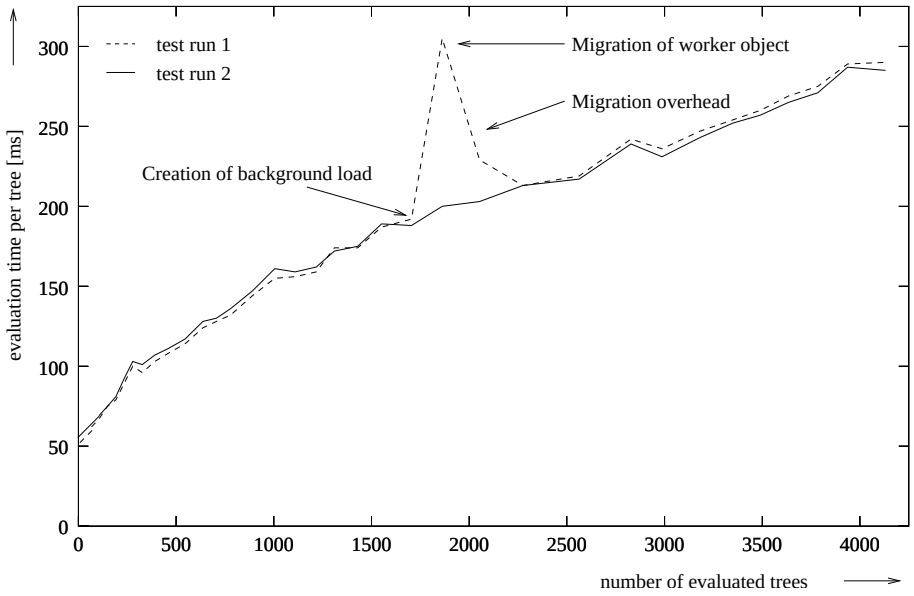

Figure 6.11: Worker object migration after creation of background load on its host

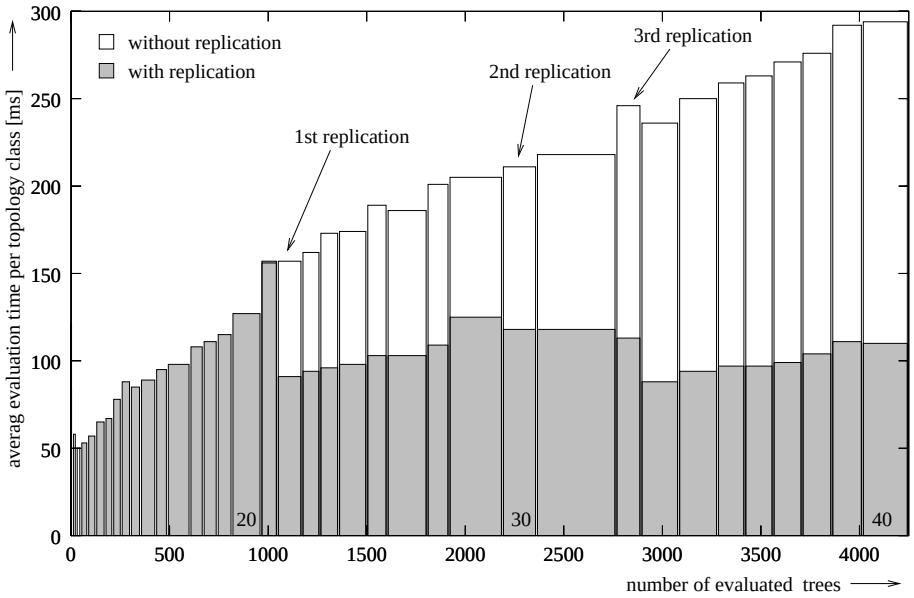

Figure 6.12: Impact of 3 subsequent automatic worker object replications

#workers	Average Likelihood	Average Execution Time (secs)	Platform	P > S
1	-400964.07	67828	Intel	n/a
3	-401025.23	23006	Intel	20117
7	-400917.95	11359	Intel	9233
15	-400951.36	5920	Intel	4779
31	-400942.26	3021	Intel	2199
6	-400911.91	72889	Hitachi	n/a
27	-400953.24	24883	Hitachi	n/a
57	-400912.86	17676	Hitachi	n/a

Table 6.10: RAxML execution times and final likelihood values for 1000_ARB

processes is taken into account, since the master process hardly produces load. In Figure 6.13 “fair” and “normal” speedup values obtained for the experiments with the 1000_ARB data set on the RRZE PC-cluster are plotted.

“Fair” speedup values take into account the first point of time at which the parallel code encounters a tree with a better likelihood than the final tree of the sequential run or vice versa (also indicated in column “P $>$ S” of Table 6.10). These “fair” values better correspond to real program performance. Furthermore, “normal” speedup values which are based on the complete execution time of the parallel program until termination, i.e. the standard speedup definition, irrespective of final likelihood values are also indicated.

Since the effect of non-determinism on program performance had to be evaluated as well, the parallel code was executed 4 times for each job-size and the average “normal” and “fair” execution times as well as likelihood values were calculated. Practically every individual execution of RAxML even on the same number of processors yielded a distinct final tree. Note, that “fair” speedup values need not be superlinear, since the selection of starting trees has a major impact on execution times and final likelihood values.

On the Hitachi SR8000-F1 RAxML was executed once on 8 processors (1 node, 6 workers), 3 times on 32 processors (4 nodes, 27 workers), and twice on 64 processors (8 nodes, 57 workers) in intra-node MPI mode to assess performance.

According to their SPEC [134] data the Intel Xeon processors should roughly be 3-4 times faster than the Hitachi CPUs. A comparison of execution times shows that the acceleration factor is > 6. The poor performance of the Hitachi supercomputer in respect to its SPEC data is due to the arguments listed in Section 6.2.1 of this Chapter. The data from the test runs on the Linux Cluster and the Hitachi supercomputer is also summarized in Table 6.10 (n/a means: data not available).

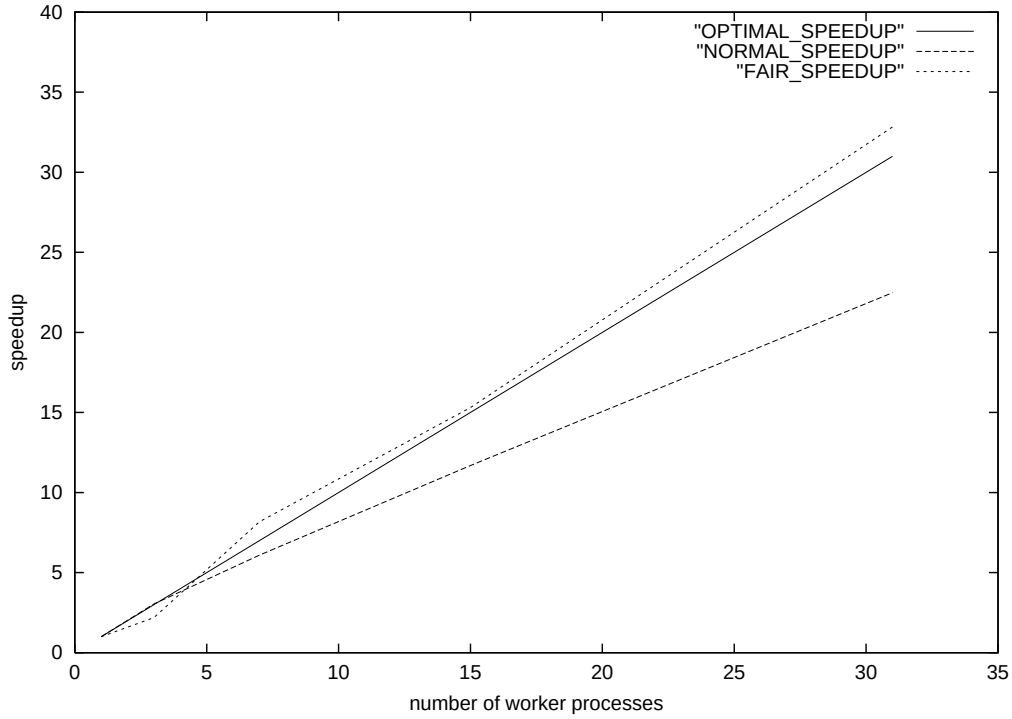

Figure 6.13: Normal, fair, and optimal speedup values for 1000_ARB with 3,7,15, and 31 worker processes on the RRZE PC Cluster

6.5.3 RAxML@home

Initially, the impact of the altered algorithm and the associated computational overhead was measured for the MPI-based prototype on the RRZE Linux cluster. In Table 6.11 the final likelihood values, execution times, and "fair" speedups are indicated.

# workers	Likelihood	Fair Execution Time	Fair Speedup
1	-400970.31	53002	1
3	-400945.43	17871	2.97
7	-400950.58	10693	4.96
15	-400947.24	7542	7.03

Table 6.11: Performance of MPI-based distributed RAxML prototype

In order to assess if the http-based implementation of RAxML@home executes correctly the Xeon 2.4GHz cluster at the LRR was used. In addition, a cluster of 50 relatively slow Ethernet-connected SUN-workstations (SunHalle) was

used to conduct larger scalability tests. RAxML@home returned "good" final trees for 2025_ARB on the Xeon cluster as well as for 1000_ARB on the Sun-Halle and terminated correctly.

Furthermore, the 2025_ARB test returned the best-known tree (compared to a couple of sequential executions) for this alignment within 8 hours on 9 worker processes. It is important to mention that the 2.4GHz Linux cluster was heavily loaded during RAxML@home execution due to computations by other users.

The smaller 1000_ARB alignment required a total running time of 50 hrs on the 50 machines of the SUN cluster, and returned a likelihood of -4001101.32. This value is slightly worse than the likelihood value of the best-known tree due to the non-determinism of the program. The comparatively long execution time in this case is partially due to the slow hardware of the workstation cluster. Furthermore, the execution times of the http-based version of RAxML@home are significantly higher than for the sequential and parallel versions as well as for the MPI-based distributed implementation. In addition, to the more coarse-grained parallelization, this is caused by a partially redundant job distribution, as well as the tree verification mechanism for rejection of potentially biased trees which is outlined in Section 5.2.2 (pp. 82).

6.6 Inference of a 10.000-Taxon Phylogeny with RAxML

The computation of the 10.000-taxon tree was conducted using the sequential, as well as the parallel version of RAxML [122].

As already mentioned, one of the advantages of RAxML consists in the randomized generation of starting trees, which enables inference of distinct trees from different starting points in tree space.

Thus, 5 distinct randomized parsimony starting trees were computed sequentially along with the first 3–4 rearrangement steps on the Infiniband cluster at LRR. This initial phase required an average of 112.31 CPU hours per tree.

Thereafter, several subsequent parallel runs (due to job run-time limitations of 24 hrs) starting with the sequential trees on either 32 or 64 processors at the RRZE 2.66GHz Xeon cluster were executed. The parallel computation required an average of 1689.6 accumulated CPU hours per tree. The best likelihood for 10000_ARB was -949570.16 the worst -950047.78 and the average -949867.27, i.e. the final trees did not differ significantly in terms of final likelihood values.

Note, that PHYML reached a likelihood value of -959514.50 after 117.25 hrs on the Itanium2. Moreover, the parsimony starting trees computed with RAxML had likelihood values ranging between -954579.75 and -955308.00. The average

time required for computing those starting trees on the faster Xeon processor was 10.99 hrs.

Since bootstrapping is not feasible for this large data size and in order to gain some basic information about similarities among the 5 final trees an extended majority-rule consensus tree with consense [57] from PHYLIP (consense constantly exited with a memory error message when passed more than 5 trees) was computed.

The consensus tree has 4777 inner nodes which appear in all 5 trees, 1046 in 4, 1394 in 3, 1323 in 2, and 1153 in only 1 tree (average: 3.72).

The results from this large phylogenetic analysis including all final trees along with the consensus tree are available at: WWWBODE.CS.TUM.EDU/~STAMATAK.

An initial biological analysis of the best final tree clearly showed that Archaea, Bacteria, and Eukarya correctly clustered in individual major clades of the tree.

The screenshot 6.14 from the visualization of the best final tree using the ATV [153] visualization tool demonstrates an outstanding problem which arises with standard tools at huge tree sizes. The visualization is completely confusing since it does not provide any kind of useful information. However, information is only valuable as long as it can be properly displayed; a problem which motivates the need for novel tree visualization concepts.

Finally, it is important to note, that MrBayes and PHYML have particularly high memory requirements compared to RAxML. Therefore, huge trees can not be computed using commodity components such as 32-bit PC clusters. For example RAxML consumed 199MB of main memory for 1000_ARB, PHYML 880MB, and MrBayes 1.195MB respectively. Furthermore, both MrBayes and PHYML exited with error messages due to excessive memory requirements for 10000_ARB on an Intel Xeon 2.4GHz processor equipped with 4GB (!) of main memory.

Therefore, an effort was made to port MrBayes and PHYML to a 64-bit Intel Itanium2 1.3GHz processor with 8GB of main memory. While MrBayes exited for unknown reasons which require further investigation, PHYML finally consumed 8.8GB of main memory. In contrast to PHYML and MrBayes, RAxML used only 800MB for this 10.000 taxon alignment. The new problems which arise with huge trees are discussed in [123].

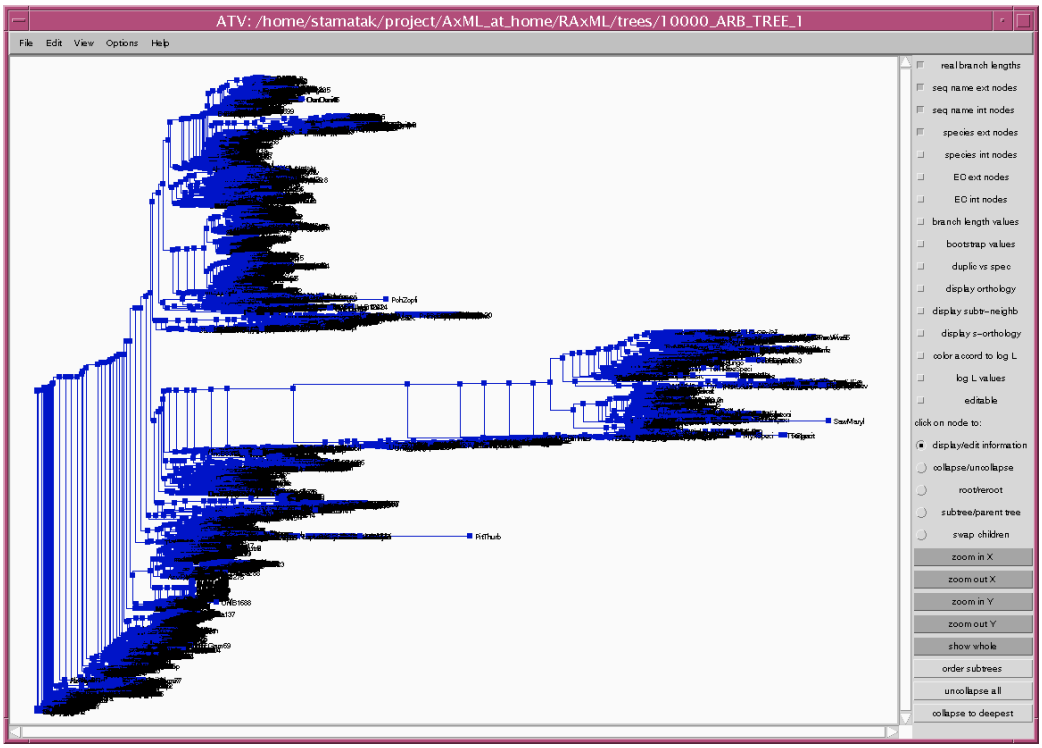

Figure 6.14: Visualization of the 10.000-taxon phylogeny with ATV

Summary

This Chapter addressed the quantitative and qualitative benefits induced by the implementation of the respective algorithmic as well as technical solutions in AxML and RAxML. It shows that the novel algorithmic ideas which have been integrated into RAxML yield substantial improvements both in terms of computable tree size and final results. For real alignment data RAxML currently represents the fastest and most accurate maximum likelihood program. Furthermore, the efficiency of the parallel and distributed implementations has been demonstrated. The final Section of this chapter covered the parallel inference of a 10.000 taxon phylogeny with RAxML, which represents the largest maximum likelihood-based phylogenetic analysis to date. The following Chapter concludes this thesis and addresses algorithmic, technical, and organizational issues which could enable inference of even larger trees in the near future.

7

Conclusion and Future Work

Δεν φοβαμαι τιποτα, δεν ελπιζω τιποτα, ειμαι λευτερος.
I do not fear anything, I do not hope anything, I am free.
Nikos Kazantzakis

This final Chapter provides the conclusion of the work conducted and addresses important aspects of future work in phylogenetics.

7.1 Conclusion

The computation of the "tree of life" containing all living organisms on earth is still one of the "grand challenges" in HPC Bioinformatics.

The currently most accurate methods for phylogenetic tree inference using alignments of DNA sequence data are based on statistical models. The most common models are maximum likelihood and bayesian methods for phylogenetic tree inference.

MrBayes and PHYML (see Section 3.9.2, pp. 47) are currently the fastest and most accurate programs for phylogenetic tree inference.

The work presented in this thesis focussed on the design and development of a sequential program called RAxML which includes novel algorithmic optimizations as well as new search space heuristics. Furthermore, efficient parallel and distributed implementations of RAxML have been presented.

In [124] and Chapter 6 (pp. 87) it has been demonstrated that RAxML performs slightly worse than MrBayes and PHYML for synthetic data but significantly outperforms both programs on 9 real-world data sets containing 101 up to 1.000 organisms both in terms of speed and final likelihood values. Thus, RAxML is able to compute better trees for real data in the same time as PHYML and yields

significantly better final trees according to the likelihood ratio test. As already mentioned the design of stopping criteria is an issue of current research in the field.

Furthermore, the parallel implementation of RAxML shows good speedup values and has been used to compute the—to the best of the author's knowledge—*first* integral, i.e. not using a divide-and-conquer approach, maximum likelihood tree containing 10.000 representative organisms of the domains: Eukarya, Bacteria, and Archaea (see Chapter 6, pp. 87). The computation of the 10.000-taxon tree has also become feasible due to the relatively low memory requirements of RAxML compared to MrBayes and PHYML.

Thus, RAxML is currently one of the fastest and most accurate programs for inference of phylogenetic trees under maximum likelihood, and implements a practicable and inexpensive approach for computation of huge phylogenies on inexpensive PC clusters or clusters of workstations. For example, there exists no parallel implementation of PHYML which limits the size of computable trees. Moreover, the algorithm of PHYML is relatively closely-coupled such that a parallelization appears to be a difficult task.

However, RAxML needs improvements in two key areas:

Firstly, RAxML is currently not able to handle protein data and does not implement the respective models of amino acid substitution.

Secondly, it does currently not offer the Γ model of rate variation (see Section 3.4.3, pp. 26), which is however just a minor implementation issue.

MrBayes and PHYML both provide those advanced features which in general lead to a further increase in run-time and memory requirements.

On the other hand the tree search algorithm of RAxML is straight-forward to parallelize/distribute and the code has significantly inferior memory requirements than MrBayes and PHYML, which can become a serious problem for large trees even on 64-bit architectures. This performance problem has been addressed in Section 6.6 (pp. 109) of this thesis.

At present PHYML and RAxML are still significantly faster by factor 50–200 than MrBayes for large real data trees. The need for maximum likelihood methods after the emergence of bayesian phylogenetic inference and the necessity and advantages of combining both methods are also discussed in [124] and Chapter 6 (pp. 87) of this thesis.

7.2 Future Work

This final Section describes algorithmic, technical, and organizational directions of future work. The main objective is to further improve RAxML regarding the

previously addressed shortfalls and to develop new methods for inference and representation of even larger phylogenetic trees.

The goal is and will remain for quite some years the computation of a large tree of life containing thousands of organisms, as sequence data accumulates and hardware becomes faster. However, the provision of appropriately prepared data is also an important issue, since the computation of trees containing 30.000 organisms—approximately the size of the ARB database—might become feasible in two or three years. Therefore, the collection and preparation of data also forms an integral part of the NFS-funded tree of life project.

7.2.1 Algorithmic Issues

The experience with the development of RAxML has demonstrated that progress in the field is primarily achieved via algorithmic innovation rather than by parallelization and brute-force allocation of all available computational resources. Thus, the most essential part of future work consists in novel algorithmic developments.

Towards Complex Models: As already mentioned RAxML lacks the ability to handle protein data and does not provide for the discrete Γ model of rate heterogeneity. Thus, the first issue would be to implement those features in order to offer a complete and flexible phylogenetic inference tool.

Towards Huge Trees: Despite the fact that RAxML currently enables the computation of comparatively large trees, the size of huge integral trees is limited by memory consumption. Thus, a divide-and-conquer approach is required to intelligently select overlapping sub-alignments for computing smaller subtrees.

In addition, so-called supertree methods to merge overlapping subtrees into one single tree are required. Within this context an urgently required survey comparing the quality of supertree with integral tree methods will be conducted. This survey will be carried out in cooperation with Olaf Bininda-Edmonds (Postdoctoral Research Associate in Bioinformatics at Chair of Animal Breeding, TUM), who is a leading expert in the field of supertree construction.

Towards Better Visualization: An important issue within the context of inferring large trees is the graphical representation of those trees. At present there is no suitable tool available to visualize the 10.000 taxon tree computed with RAxML.

In cooperation with the computer science department of the University of Crete and Prof. I. Tollis (Professor of Computer Science University of Crete, In-

stitute of Computer Science, Foundation for Research and Technology) who is a leading expert in the field of graph visualization, new solutions to appropriately display the information contained in large trees will be exploited.

Towards Improved Tree Proposal Mechanisms? As discussed in Section 3.5 (pp. 32) the most important part of bayesian phylogenetic inference is the tree proposal mechanism of the MCMC analysis. It will be worthwhile to analyze if certain ideas from the RAxML search-space algorithm can be integrated into the tree proposal mechanism of MrBayes in order to accelerate the program.

7.2.2 Technical Issues

Another key objective of future work consists in the search for inexpensive solutions to acquire the large amount of computing power for large trees (e.g. the 10.000-taxon tree still required $\approx$ 1600 accumulated CPU hours on Intel Xeon 2.4GHz and 2.66GHz processors with RAxML).

Towards Distributed Computation of Subtrees: Once methods for selecting appropriate sub-alignments (containing $\approx$ 500-1.000 organisms) have been derived there will still exist enormous computational resource requirements. Subtrees of that size can still be computed sequentially without excessive time/memory requirements using RAxML. Furthermore, the independent inference of a large number of subtrees perfectly suits the distributed programming paradigm. Based on the experience with the preceding distributed implementations of RAxML it is planned to design a simple distributed program for inference of those subtrees.

Towards Utilization of Graphics Processing Units (GPUs): The greatest part of CPU time ($>$ 90%) consumed by phylogenetic inference consists in floating point operations which update the likelihood vectors at each inner node of the tree. Those operations can easily be represented as basic vector operations and are thus easy to parallelize. Krüger and Westermann from the Technische Universität München have demonstrated how GPUs can be used to accelerate programs containing vector-operations [65]. In December 2003 a common project has been initiated to port RAxML to GPUs in order to exploit the intrinsic fine-grained parallelism of RAxML as well. This cooperation shall be continued and eventually extended to clusters of GPUs in 2005.

7.2.3 Organizational Issues

Finally, some important organizational issues are mentioned which might lead to advances in the field on a European and international level.

Phylogeny Competition: The requirement to establish a standard benchmark set including real and simulated data alignments for comparison of maximum likelihood-based programs has lead to the idea of conducting a phylogeny competition at a major Bioinformatics conference. Different teams which are developing phylogeny programs should be invited and compete against each other on a set of alignments which have been selected by an independent committee on identical platforms. First plans for organizing and conducting such a competition have been discussed with Tiffani Williams from the University of New Mexico (UNM) and an agreement has been reached to proceed with the establishment of an organizational core team.

European Tree-of-Life Initiative: In 2003 the National Science Foundation (NSF) announced a 11.600.000$ tree of life initiative which is co-located at 13 leading research institutions across the U.S. Thus, the participation in a planned European counter-initiative with partners in Germany, France, and Greece will be a key objective.

Bibliography

[1] G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, O. Wehrens. The GridLab Grid Application Toolkit. In *Proceedings of HPDC 2002*, 411, IEEE Press, Edinburgh, Scotland, 2002.

[2] ARB project site: WWW.ARB-HOME.DE, visited Mar 2003.

[3] ATV - a phylogenetic tree display tool: WWW.GENETICS.WUSTL.EDU/EDDY/ATV, visited Mar 2004.

[4] C. Babel. Design and Implementation of Distance-based Heuristics in a Program for Genome Sequence Analysis. *System Development Project*, Technische Universität München, 2003.

[5] D.A. Bader, B.M.E. Moret, L. Vawter. Industrial Applications of High-Performance Computing for Phylogeny Reconstruction. In *Proceedings of SPIE ITCom: Commercial Applications for High-Performance Computing*, 4528:159–168, The International Society for Optical Engineering, Denver, USA, 2001.

[6] C.S. Baker, S.R. Palumbi. Which whales are hunted? A molecular genetic approach to whaling. In *Science*, 265:1538–1539, 1994.

[7] B.R. Baum. Combining trees as a way of combining data sets for phylogenetic inference and the desirability of combining gene trees. In *Taxon*, 41:3–10, 1992.

[8] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, D.L. Wheeler. GenBank. In *Nucl. Acid. Res.*, 30:17–20, 2002.

[9] H.L. Bodlaender, M.R. Fellows, M.T. Hallett, T. Wareham, T. Warnow. The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. In *Theor. Comp. Sci.*, 244:167–188, 2000.

[10] M.L. Bonet, M. Steel, T. Warnow, S. Yooseph. Better methods for solving parsimony and compatibility. In *J. Comp. Biol.*, 5:391–408, 1998.

[11] M.J. Brauer, M.T. Holder, L.A. Dries, D.J. Zwickl, P.O. Lewis, D.M. Hillis. Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. In *Molecular Biology and Evolution* 19:1717–1726, 2002.

[12] R. Brent. Algorithms for minimization without derivatives. *Prentice-Hall*, Engelwood Clifts, New Jersey, 1973.

[13] G. Brose. JacORB: Implementation and Design of a Java ORB. In *International Conference on Distributed Applications and Interoperable Systems (DAIS'97)*, International Federation for Information Processing, Cottbus, Germany, 1997.

[14] R.M. Bush, C.A. Bender, K. Subbarao, N.J. Cox, W.M. Fitch. Predicting the evolution of human influenza. In *Science*, 286(5446):1921–1925, 1999.

[15] Cactus Code project site: WWW.CACTUSCODE.ORG, visited April 2003.

[16] J.H. Camin, R.R. Sokal. A method for deducing branching sequences in phylogeny. In *Evolution*, 19:311–326, 1965.

[17] C. Ceron, J. Dopazo, E.L Zapata, M.J. Carazo, O. Trelles. Parallel implemenation of DNAml program on message-passing architectures. In *Parallel Computing*, 24:701–716, 1998.

[18] ClustalW project site: WWW.EBI.AC.UK/CLUSTALW, visited Jun 2004.

[19] Chair for Computer Science in Engineering, Science, and Numerical Programming (TUM): WWWZENGER.INFORMATIK.TU-MUENCHEN.DE, visited Apr 2004.

[20] B.S. Chang, M.J. Donoghue. Recreating ancestral proteins. In *Trends Ecol. Evol.*, 15:109–114, 2000.

[21] M.W. Chase, D.E. Soltis, R.G. Olmstead, D. Morgan, D.H. Les, B.D. Mishler, M.R. Duvall, R.A. Price, H.G. Hills, Y.L. Qiu, K.A. Kron, J.H. Rettig, E. Conti, J.D. Palmer, J.R. Manhart, K.J. Sytsma, H.J. Michaels, W.J. Kress, K.G. Karol, W.D. Clark, M. Hedren, B.S. Gaut, R.K. Jansen, K.J. Kim, C.F. Wimpee, J.F. Smith, G.R. Furnier, S.H. Strauss, Q.Y. Xiang, G.M. Plunkett, P.S. Soltis, S.M. Swensen, S.E. Williams, P.A. Gadek, C.J. Quinn, L.E. Eguiarte, E. Golenberg, G.H. Learn, Jr., S.W. Graham,

S.C.H. Barrett, S. Dayanandan, V.A. Albert. Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcL. In *Annals of the Missouri Botanical Garden*, 80:528–580, 1993.

[22] B. Chor, M. Hendy, B. Holland, D. Penny. Multiple maxima of likelihood in phylogenetic trees: An analytic approach. In *Mol. Biol. Evol.*, 17:1529–1541, 2000.

[23] CONDOR project site: WWW.CS.WISC.EDU/CONDOR, visited Apr 2004.

[24] C. Darwin. On the origin of species by means of natural selection. *John Murray, London*, 1859.

[25] W.E. Day, D.S. Johnson, D. Sankoff. The computational Complexity of inferring rooted phylogenies by parsimony. In *Math. Bios.*, 81:33–42, 1986.

[26] R.W. DeBry, L.G. Abele. The relationship between parsimony and maximum likelihood analyses: tree scores and confidence estimates. In *Mol. Biol. Evol.*, 12:291–297, 1995.

[27] A.P. Dempster, M.N. Laird, D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. In *J. R. Stat. Soc. B.*, 39:1–38, 1977.

[28] A.W.F. Edwards, Cavalli-Sforza. Phenetic and phylogenetic classification. In *Systematics*, 6:67–76, 1964.

[29] D.P. Faith. Genetic diversity and taxonomic priorities for conservation. In *Biol. Conserv.*, 68:69–74, 1992.

[30] J. Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. In *Syst. Zool.*, 27:401–410, 1978.

[31] J. Felsenstein. Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. In *J. Mol. Evol.*, 17:368–376, 1981.

[32] X. Feng, D.A. Buell, J.R. Rose, P.J. Waddell. Parallel algorithms for Bayesian phylogenetic inference. In *Journal of Parallel and Distributed Computing: Special Issue on High-Performance Computational Biology*, 63:707–718, 2003.

[33] G.E, Fox, E. Stackebrandt, R.B. Hespell, J. Gibson, J. Maniloff, T.A. Dyer, R.S. Wolfe, W.E. Balch, R.S. Tanner, L.J. Magrum, L.B. Zablen, R. Blakemore, R. Gupta, L. Bonen, B.J. Lewis, D.A. Stahl, K.R. Luehrsen, K.N. Chen, C.R. Woese. The Phylogeny of Prokaryotes. In *Science*, 209:457–463, 1980.

[34] O. Gascuel. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. In *Mol. Biol. Evol.*, 14:685–695, 1997.

[35] GenBank project site: WWW.NCBI.NIH.GOV/GENBANK, visited Jan 2004.

[36] N. Goldman, P. Anderson, A.G. Rodrigo. Likelihood-based tests of topologies in phylogenetics. In *Syst. Biol*, 49:652–670, 2000.

[37] P.A. Goloboff. Analyzing Large Data Sets in Reasonable Times: Solutions for Composite Optima. In *Cladistics*, 15:415–428, 1999.

[38] GridLab project: WWW.GRIDLAB.ORG, visited Apr 2004.

[39] S. Guindon, O. Gascuel. A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. In *Syst. Biol.*, 52(5):696–704, 2003.

[40] D. Gusfield, S. Eddhu, C. Langley. Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination. In *Proceedings of 2nd IEEE Computer Society Bioinformatics Conference (CSB2003)*, Stanford Univ., Palo Alto, California, IEEE Press, 2003.

[41] P. Halbur, M.A. Lum, X. Meng, I. Morozow, P.S. Paul. New procine reproductive and respiratory syndrome virus DNA and proteins encoded by open-ended frames of an Iowa strain of the virus are used in vaccines against PRRSV in pigs. In *Patent-Filing WO9606619-A1*, 1994.

[42] M. Hasegawa, H. Kishino, T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. In *J. Mol. Evol.*, 22:160–174, 1985.

[43] HeLiCS: HEidelberg Linux Cluster System: HELICS.UNI-HD.DE, visited Jul 2003.

[44] M.D. Hendy, D. Penny. A framework for the quantitative study of evolutionary trees. In *Syst. Zool.*, 38:297–309, 1989.

[45] D.M. Hillis, C. Moritz, B.K. Mable. In *D.M. Hillis, C. Moritz, B.K. Mabel, editors, Molecular Systematics*, Applications of Molecular Systematics:515–543, Sinauer Associates, Sunderland, MA, 1996.

[46] Hitachi SR8000-F1 project site: WWW.LRZ-MUENCHEN.DE/SERVICES/COMPUTE/HLRB, visited Mar 2004.

[47] M.T. Holder, P.O. Lewis. Phylogeny Estimation: Traditional and Bayesian Approaches. In *Nature Reviews Genetics*, 4:275–284, 2003.

[48] J.P. Huelsenbeck, B. Larget, R.E. Miller, F. Ronquist. Potential Applications and Pitfalls of Bayesian Inference of Phylogeny. In *Syst. Biol.*, 51(5):673–688, 2002.

[49] J.P. Huelsenbeck, F. Ronquist, R. Nielsen, J.P. Bollback. Bayesian Inference and its Impact on Evolutionary Biology. In *Science*, 294:2310–2314, 2001.

[50] J.P. Huelsenbeck, F. Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. In *Bioinformatics*, 17(8):754–5, 2001.

[51] J.P. Huelsenbeck, D.M. Hillis. Succes of phylogenetic methods in the four-taxon case. In *Syst. Biol.*, 42:247–264, 1993.

[52] J.P. Huelsenbeck. Performance of phylogenetic methods in simulation. In *Syst. Biol.*, 44:17–48, 1995.

[53] D. Huson, S. Nettles, T. Warnow. Disk-Covering, a fast converging method for phylogenetic tree reconstruction. In *Comp. Biol.*, 6(3):369–386, 1999.

[54] D. Huson, L. Vawter, T. Warnow. Solving large scale phylogenetic problems using DCM2. In *ISMB99*, 118–129, AAAI Press, Heidelberg, Germany, 1999.

[55] INFINIBAND at LRR-TUM:
WWWBODE.CS.TUM.EDU/PAR/ARCH/INFINIBAND, visited Apr 2004.

[56] I. Janse, M. Meima, W. Edwin, A. Kardinaal,G. Zwart. High-Resolution Differentiation of Cyanobacteria by Using rRNA-Internal Transcribed Spacer Denaturing Gradient Gel Electrophoresis. In *Applied and Environmental Microbiology*, 69(11):6634–6643, 2003.

[57] L.S. Jermiin, G.J. Olsen,K.L. Mengersen, S. Easteal. Majority-rule consensus of phylogenetic trees obtained by maximum-likelihood analysis. In *Mol. Biol. Evol.*, 14:1297–1302, 1997.

[58] G. Judd, M. Clement, Q. Snell. The DOGMA approach to high utilization supercomputing. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing HPDC7*, 862–873, IEEE Press, Chicago, USA, 1998.

[59] T. Jukes, C. Cantor. Evolution of protein molecules. In *H. Munro (editor), Mammalian protein metabolism*, III:21–132, Academic Press, New York, 1969.

[60] M. Kallerjso, J.S. Farris, M.W. Chase, B. Bremer, M.F. Fay, C.J. Humphries, G. Petersen, O. Seberg, K. Bremer. Simultaneous parsimony jackknife analysis of 2538 rBCL DNA sequences reveals support for major clades of green plants, land plants, seed plants, and flowering plants. In *Plant Syst. Evol.*, 213:259–287, 1998.

[61] S. Kannan, T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. In *SIAM J. Comput.*, 26(6):1749–1763, 1997.

[62] Y.-H. Kim, S.-K. Lee, B.-R. Moon. Optimizing the order of taxon addition in phylogenetic tree construction using genetic algorithm. In *Proceedings of Pacfic Symposium on Bioinformatics*, 2003.

[63] M. Kimura. A simple method for estimating evolutionary rates of base substitutions by through comparative studies of nucleotide sequences. In *J. Mol. Evol.*, 16:111-120, 1980.

[64] B. Korber, M. Muldoon, J. Theiler, F. Gao, R. Gupta, A. Lapedes, B.W. Hahn, S. Wolinsky, T. Bhattacharya. Timing the Ancestor of the HIV-1 Pandemic Strains. In *Science*, 288:1789–1796, 2000.

[65] J. Krüger, R. Westermann. Linear Algebra Operators for GPU Implementation of Numerical Algorithms. In *Proceedings of SIGGRAPH2003*, 908–916, ACM Press, San Diego, USA, 2003.

[66] M.K. Kuhner, J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. In *Mol. Biol. Evol.*, 11:459-468, 1994.

[67] C. Lanave, G. Preparata, C. Saccone, G. Serio. A new method for calculating evolutionary substitution rates. In *J. Mol. Evol.*, 20:86–93, 1984.

[68] Gerd Lanfermann. Nomadic Migration - A Service Environment for Autonomic Computing on the Grid. Ph.D. thesis, University of Potsdam, 2003.

[69] G. Lanfermann, G. Allen, T. Radke, E. Seidel. Nomadic Migration: Fault Tolerance in a Disruptive Grid Environment. In *Proceedings of CCGRID 2002*, 280–281, ACM/IEEE Press, Brisbane, Australia, 2002.

[70] G. Lanfermann, G. Allen, T. Radke, E. Seidel. Nomadic Migration: A New Tool for Dynamic Grid Computing. In *Proceedings of HPDC 2001*, 429–430, IEEE Press, Redondo Beach, USA, 2001.

[71] A. Lemmon, M. Milinkovitch. The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation. In *Proc. Natl. Acad. Sci. USA*, 99:10516–10521, 2002.

[72] P. Lewis. A genetic algorithm for maximum likelihood phylogeny inference using nucleotide sequence data. In *Mol. Biol. Evol.*, 15:277–283, 1998.

[73] W.H. Li. In *Molecular Evolution, Sinauer Associates, Sunderland, MA*, 112–115, 1997.

[74] alg M. Lindermeier. Ein Konzept zur Lastverwaltung in verteilten objektorientierten Systemen. Ph.D. thesis, Technische Universität München, 2002.

[75] M. Lindermeier. Load management for distributed object-oriented environments. In *Proceedings of 2nd International Symposium on Distributed Objects and Applications (DOA'00)*, OMG, Antwerp, Netherlands, 2000.

[76] W. Ludwig, O. Strunk, R. Westram, L. Richter, H. Meier, Yadhukumar, A. Buchner, T. Lai, S. Steppi, G. Jobb, W. Förster, I. Brettske, S. Gerber, A.W. Ginhart, O. Gross, S. Grumann, S. Hermann, R. Jost, A. König, T. Liss, R. Lüssmann, M. May, B. Nonhoff, B. Reichel, R. Strehlow, **A. Stamatakis**, N. Stuckmann, A. Vilbig, M. Lenke, T. Ludwig, A. Bode, K.-H. Schleifer ARB: A Software Environment for Sequence Data. In *Nucl. Acids Res.*, 32(4):1363–1371, 2004.

[77] B. Mau, M. Newton, B. Larget. Bayesian phylogenetic inference via markov chain monte carlo methods. In *Biometrics*, 55:1–12, 1999.

[78] B. Mau, M. Newton. Phylogenetic Inference for binary data on dendrograms using markov chain monte carlo. In *J. Comp. Graph. Stat.*, 6:122–131, 1997.

[79] Max Planck Institute Potsdam:
WWW.AEI-POTSDAM.MPG.DE/FACILITIES/PUBLIC/COMPUTERS.HTML,
visited Apr 2004.

[80] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller. Equation of state calculations by fast computing machines. In *J. Chem. Phys.*, 21:1087–1092, 1953.

[81] MD5 Homepage (unofficial):
USERPAGES.UMBC.EDU/~MABZUG1/CS/MD5/MD5.HTML, visited Jun 2004.

[82] I. Miklos. MCMC genome rearrangement. In *Bioinformatics*, 19(2):130–137, 2003.

[83] B.M.E. Moret, D.A. Bader, T. Warnow, S.K. Wyman, M. Yan. GRAPPA: a high-performance computational tool for phylogeny reconstruction from gene-order data. In *Proceedings of Botany 2001*, 2001.

[84] S.B. Needelman, C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. In *J. Mol. Biol.*, 48:443–453, 1970.

[85] G. Olsen. DNArates Distribution:
GETA.LIFE.UIUC.EDU/~GARY/PROGRAMS/DNARATES.HTML, visited Apr 2004.

[86] G. Olsen, H. Matsuda, R. Hagstrom, R. Overbeek. fastdnaml: A Tool for Construction of Phylogenetic Trees of DNA Sequences using Maximum Likelihood. In *Comput. Appl. Biosci.*, 10:41–48, 1994.

[87] M. Ott. RAxML@home: Specification and Development of a Globally Distributed Software-Architecture for Inference of Phylogenetic Trees. *Master's thesis*, Technische Universität München, 2004.

[88] C.Y. Ou, C.A. Ciesielski, G. Myers, C.I. Bandea, C.C. Luo, B.T. Korber, J.I. Mullins, G. Schochetman, R.L. Berkelman, A.N. Economou. Molecular epidemiology of HIV transmission in a dental practice. In *Science*, 256(5060):1165–1171, 1992.

[89] PACX-MPI: The Grid-Computing library PACX-MPI, Extending MPI for Computational Grid:
WWW.HLRS.DE/ORGANIZATION/PDS/PROJECTS/PACX-MPI, visited Apr 2004.

[90] PAML Manual (Information on Tr/Tv definitions: page 20):
BCR.MUSC.EDU/MANUALS/PAMLDOC.PDF, visited Nov 2003.

[91] parallel fastDNAml project site:
WWW.INDIANA.EDU/ RAC/HPC/FASTDNAML, visited Feb 2003.

[92] PAUP project site: PAUP.CSIT.FSU.EDU, visited May 2003.

[93] PHYLIP downlaod site and list of phylogeny software:
EVOLUTION.GENETICS.WASHINGTON.EDU, visited Nov 2003.

[94] D. Posada, K.A. Crandall. MODELTEST: testing the model of DNA substitution. In *Bioinformatics*, 14:817–818, 1998.

[95] M.A. Ragan. Phylogenetic inference based on matrix representation of trees. In *Mol. Phyl. Evol.*, 1:53–58, 1992.

[96] A. Rambaut, N.C. Grassly. Seq-Gen: An application for the monte carlo simulation of dna sequence evolution along phylogenetic trees. In *Comp. Appl. Biosc.*, 13:235–238, 1997.

[97] B. Rannala, Z.H. Yang. Probability distribution of molecular evolutionary trees: A new method for phylogenetic inference. In *J. Mol. Evol.*, 43:304–311, 1996.

[98] V. Ranwez, O. Gascuel. Improvement of distance-based phylogenetic methods by a local maximum likelihood approach using triplets. In *Mol. Biol. Evol.*, 19:1952–1963, 2002.

[99] V. Ranwez, O. Gascuel. Quartet-based phylogenetic inference: Improvements and limits. In *Mol. Biol. Evol.*, 18:1103–1116, 2000.

[100] Regionales Rechenzentrum Erlangen: HPC services:
WWW.RRZE.UNI-ERLANGEN.DE, visited Oct 2003.

[101] D. Robinson, L. Foulds. Comparison of weighted labeled trees. in *Lecture Notes in Mathematics*, 748:119–126, Springer, Berlin,1979.

[102] F. Rodriguez, J.L. Oliver, A. Marin, J.R. Medina. The general stochastic model of nucleotide substitution. In *J. Theor. Biol.*, 142:485–501, 1990.

[103] M. Rosenberg, S. Kumar. Traditional phylogenetic reconstruction methods reconstruct shallow and deep evolutionary relationship equally well. In *Mol. Biol. Evol.*, 19:1823–1827, 2001.

[104] U. Roshan, B.M.E. Moret, T.L. Williams, T. Warnow. Performance of supertree methods on various data set decompositions. In *O.R.P. Bininda-Edmonds (editor) Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, 301–328, to be published. Preprint available at WWW.CS.UNM.EDU/˜TLW/PUBLICATIONS.HTML.

[105] N. Saitou, M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. In *Mol. Biol. Evol.*, 4(4):406–425, 1987.

[106] M.J. Sanderson, A.C. Driskell. The challenge of constructing large phylogenetic trees. In *Trends in Plant Science*, 8(8):374–378, 2003.

[107] M.J. Sanderson. The r8s software package: GINGER.UCDAVIS.EDU/R8S, visited Mar 2004.

[108] H.A. Schmidt, K. Strimmer, M. Vingron, A.v. Haeseler. TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. In *Bioinformatics*, 18:502–504, 2002.

[109] Seti@home project site: SETIATHOME.SSL.BERKELEY.EDU, visited Jul 2003.

[110] J. Setubal, J. Meidanis. Introduction to Computational Molecular Biology. PWS Publishing Company, Boston, 1997.

[111] H. Shimodaira, M. Hasegawa. Multiple comparisons of log-likelihoods with applications to phylogenetic inference. In *Molecular Biology and Evolution*, 16:1114–1116, 1999.

[112] A. Skourikhine. Phylogenetic Tree Reconstruction Using Self-Adaptive Genetic Algorithm. In *Proceedings of IEEE International Symposium on Bio-Informatics and Biomedical Engineering (BIBE'00)*, 2000.

[113] T.F. Smith, M.S. Waterman. Identification of common molecular subsequences. In *J. Mol. Biol.*, 147:195–197, 1981.

[114] P.H.A. Sneath, R.R. Sokal. In *Numerical Taxonomy*, 230–234, W.H. Freeman and Company, San Francisco, 1973.

[115] Q. Snell, M. Whiting, M. Clement, D. McLaughlin. Parallel Phylogenetic Inference. In *Proceedings of 13th Supercomputing Conference (SC2000)*, 2000.

[116] G. Stoesser, W. Baker, A.v.d. Broek, E. Camon, M. Garcia-Pastor, C. Kanz, T. Kulikova, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, N. Redaschi, P. Stoehr, M.A. Tuli, K. Tzouvara, R. Vaughan. The EMBL nucleotide sequence database. In *Nucl. Acid. Res.*, 30:21–26, 2002.

[117] K. Strimmer, V. Moulton. Likelihood analysis of phylogenetic networks using directed graphical models. In *Mol. Biol. Evol.*, 17:875–881, 2000.

[118] **A. Stamatakis**, T. Ludwig, H. Meier. RAxML-III: A Fast Program for Maximum Likelihood-based Inference of Large Phylogenetic Trees. In *Bioinformatics*, accepted for publication.

[119] **A. Stamatakis**, T. Ludwig, H. Meier. RAxML-II: A Program for Sequential, Parallel & Distributed Inference of Large Phylogenetic Trees. In *Concurrency and Computation: Practice and Experience*, accepted for publication.

[120] **A. Stamatakis**, M. Ott, T. Ludwig, H. Meier. DRAxML@home: A Distributed Program for Computation of Large Phylogenetic Trees.In *Future Generation Computer Systems (FGCS)*, accepted for publication.

[121] **A. Stamatakis**, T. Ludwig, H. Meier. The AxML Program Family for Phylogenetic Tree Inference. In *Concurrency and Computation: Practice and Experience*, accepted for publication.

[122] **A. Stamatakis**, T. Ludwig, H. Meier. Parallel Inference of a 10.000-taxon Phylogeny with Maximum Likelihood. In *Proceedings of Euro-Par 2004*, Pisa, Italy, accepted for publication.

[123] **A. Stamatakis**, T. Ludwig, H. Meier. Computing Large Phylogenies with Statistical Methods: Problems & Solutions. In *Proceedings of 4th International Conference on Bioinformatics and Genome Regulation and Structure (BGRS2004)*, Novosibirsk, Russia, accepted for publication.

[124] **A. Stamatakis**, T. Ludwig, H. Meier. New Fast and Accurate Heuristics for Inference of Large Phylogenetic Trees. In *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS2004)*, Proceedings on CD, Abstract on page 193, Santa Fe, New Mexico, April 2004.

[125] **A. Stamatakis**, T. Ludwig, H. Meier. A Fast Program for Maximum Likelihood-based Inference of Large Phylogenetic Trees. In *Proceedings of ACM Symposium on Applied Computing (SAC2004)*, 197–201, Nicosia, Cyprus, March 2004.

[126] **A. Stamatakis**, T. Ludwig, H. Meier. A Fast Program for Phylogenetic Tree Inference with Maximum Likelihood. In *Arndt Bode, Franz Durst, W erner Hanke, and Siegfried Wagner, editors, High Performance Computing in Science and Engineering*, Springer Verlag, accepted for publication.

[127] **A. Stamatakis**, T. Ludwig, H. Meier. RAxML: A Parallel Program for Phylogenetic Tree Inference. Poster abstract in *Proceedings of 2nd European Conference on Computational Biology (ECCB2003)*, 325–326, Paris, France, September 2003.

[128] **A. Stamatakis**, M. Lindermeier, M. Ott, T. Ludwig, H. Meier. DAxML: A Program for Distributed Computation of Phylogenetic Trees Based on

Load Managed CORBA. In *Proceedings of 7th International Conference on Parallel Computing Technologies (PaCT2003)*, Volume 2763 of Lecture Notes in Computer Science, 538–548, Springer Verlag, September 2003.

[129] **A. Stamatakis**, T. Ludwig, H. Meier. Neues vom Projekt ParBaum ... Parallele und verteilte Systeme und Algorithmen zur Berechnung grosser phylogenetischer Bäume mit Maximum-Likelihood (Parallel and Distributed Systems and Algorithms for the Inference of big Phylogenetic Trees with Maximum Likelihood). In *KONWIHR Quartl*, 34(1):4–7, May 2003.

[130] **A. Stamatakis**, T. Ludwig. Phylogenetic Tree Inference on PC Architectures with AxML/PAxML. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS2003)*, Proceedings on CD, Abstract on page 157, Nice, France, April 2003.

[131] **A. Stamatakis**, T. Ludwig, H. Meier, M.J. Wolf. Accelerating Parallel Maximum Likelihood-based Phylogenetic Tree Calculations using Subtree Equality Vectors. In *Proceedings of 15th Supercomputing Conference (SC2002)*, Proceedings on CD, Baltimore, Maryland, November 2002.

[132] **A. Stamatakis**, T. Ludwig, H. Meier. Adapting PAxML to the Hitachi SR8000-F1 Supercomputer. In *Arndt Bode, Franz Durst, Werner Hanke, and Siegfried Wagner, editors*, High Performance Computing in Science and Engineering, 453–466, Springer Verlag, October 2002.

[133] **A. Stamatakis**, T. Ludwig, H. Meier, M.J. Wolf. AxML: A Fast Program for Sequential and Parallel Phylogenetic Tree Calculations Based on the Maximum Likelihood Method. In *Proceedings of 1st IEEE Computer Society Bioinformatics Conference (CSB2002)*, 21–28, Stanford Univ., Palo Alto, California, August 2002.

[134] Standard Performance Evaluation Corporation (SPEC): WWW.SPEC.ORG, visited Apr 2004.

[135] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, W. Fischer. Parallel Implementation and Performance of fastdnaml - a Program for Maximum Likelihood Phylogenetic Inference. In *Proceedings of 14th Supercomputing Conference (SC2001)*, November 2001.

[136] C. Stewart, T. Tan, M. Buchhorn, D. Hart, D. Berry, Z. L., E. Wernert, M. Sakharkar, W. Fisher, D. McMullen. Evolutionary biology and computational grids. In *Technical report, IBM CASCON Computational Biology Workshop: Software Tools for Computational Biology*, 1999.

[137] K. Strimmer, A.v. Haeseler. Quartet Puzzling: A Maximum-Likelihood Method for Reconstructing Tree Topologies. In *Mol. Biol. Evol.*, 13:964–969, 1996.

[138] Sunhalle at TUM: WWWRBG.IN.TUM.DE, visited Apr 2004.

[139] Supercomputing Conference 2003 HPC challenge: WWW.SC-CONFERENCE.ORG/SC2003/TECH_HPC.PHP, visited Apr 2004.

[140] D.L. Swofford, G.J. Olsen, P.J. Wadell, D.M. Hillis. In *D.M. Hillis, C. Moritz, B.K. Mabel, editors, Molecular Systematics*, Phylogenetic Inference: 407–514, 1996, Sinauer Associates, Sunderland, MA.

[141] The Message Passing Interface (MPI) standard: WWW-UNIX.MCS.ANL.GOV/MPI, visited Jun 2004.

[142] The TreadMarks Distributed Shared Memory (DSM) System: WWW.CS.RICE.EDU/~WILLY/TREADMARKS/OVERVIEW.HTML, visited Jun 2004.

[143] J.D. Thompson, F. Plewniak, O. Poch. A comprehensive comparison of multiple sequence alignment programs. In *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[144] C. Tuffley, M. Steel. Links between Maximum Likelihood and Maximum Parsimony under a Simple Sodel of Site Substitution. In *Bull Math Biol*, 59(3):581–607, 1997.

[145] veryfastDNAml distribution: BIOWEB.PASTEUR.FR/SEQANAL/SOFT-PASTEUR.HTML, visited Jun 2004.

[146] T.L. Williams, B.M. Berger-Wolf, U. Roshan, T. Warnow. The relationship between maximum parsimony scores and phylogenetic tree topologies. In *Tech. Report*, TR-CS-2004-04, Department of Computer Science, The University of New Mexico, 2004.

[147] T.L. Williams, B.M.E. Moret. An Investigation of Phylogenetic Likelihood Methods. In *Proceedings of 3rd IEEE Symposium on Bioinformatics and Bioengineering (BIBE'03)*, 79–86, 2003.

[148] L. Wang, T. Jiang. On the complexity of multiple sequence alignments. In *J. Comp. Biol.*, 1(4):337–348, 1994.

[149] M. Wolf, S. Easteal, M. Kahn, B. McKay, L. Jermiin. TrExML: A Maximum Likelihood Program for Extensive Tree-space Exploration. In *Bioinformatics*, 16(4):383–394, 2000.

[150] XML-RPC project site: WWW.XMLRPC.COM, visited Apr 2004.

[151] Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. In *J. Mol. Evol.*, 39:306–314, 1994.

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

[153] C.M. Zmasek, S.R. Eddy. ATV: display and manipulation of annotated phylogenetic trees. In *Bioinformatics*, 17:383–384, 2002.

[154] E. Zuckerkandl, L. Pauling. Molecules as documents of evolutionary history. In *J. Theor. Biol.*, 8:357–366, 1965.