Phylogenetic Trees from Large Datasets

Inaugural – Dissertation

zur
Erlangung des Doktorgrades der
Mathematisch–Naturwissenschaftlichen Fakultät
der Heinrich–Heine–Universität Düsseldorf

vorgelegt von

Heiko A. Schmidt

aus Bonn

Düsseldorf

2003

Gedruckt mit der Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Heinrich-Heine-Universität Düsseldorf

Referent: Prof. Dr. Arndt von Haeseler
Korreferent: Prof. Dr. William Martin

Tag der mündlichen Prüfung: 28.07.2003

The Dachshund

Indeed, it was made on the plan of a bench for length and lowness. It seemed to be satisfied, but I thought the plan poor, and structurally weak, on account of the distance between the forward supports and those abaft. With age the dog's back was likely to sag; and it seemed to me that it would have been a stronger and more practicable dog if it had had some more legs.

Mark Twain (*Following the Equator*, 1895-1896)

Biologists must constantly keep in mind that what they see was not designed, but rather evolved.

Francis Crick (*What Mad Pursuit*, 1988)

Preface

This thesis deals with topics from Bioinformatics/Phyloinformatics. It touches the fields algorithmic complexity, molecular phylogenetics, sequence evolution, as well as parallel computing and, thus, is addressed to an interdisciplinary community. Although it is hardly possible to write a completely self-contained thesis, I tried to provide some basic information needed for members of either community to be able to more easily understand the topics discussed. Consequently, some information might be enclosed that seem trivial to, e.g., computer scientists but which biologists are possibly not familiar with and vice versa.

When writing up my thesis I decided, as it is custom in scientific presentations, to use the scientific 'we', which I prefer over the personal pronoun 'I'.

Parts of this thesis have been published in the following articles:

1. H. A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler (2002) TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18, 502-504.

2. H. A. Schmidt and A. von Haeseler (2002) Quartet Trees as a Tool to Reconstruct Large Trees from Sequences. In K. Jajuga, A. Sokolowski, and H.-H. Bock (eds.), *Data Analysis, Classification, and Related Methods, Studies in Classification, Data Analysis, and Knowledge Organization*, pages 379-388, Springer, Heidelberg, New York.

3. H. A. Schmidt and A. von Haeseler (2003) Maximum Likelihood Analysis Using TREE-PUZZLE. In A. D. Baxevanis, D. B. Davison, R. D. M. Page, G. Stormo, and L. Stein (eds.), *Current Protocols in Bioinformatics*, Unit 6.6, pages 6.6.1-6.6.25, Wiley and Sons, New York.

4. H. A. Schmidt, E. Petzold, M. Vingron, and A. von Haeseler (2003) Molecular Phylogenetics: Parallelized Parameter Estimation and Quartet Puzzling. *J. Parallel Distrib. Comput.*, 63, *in press*.

The TREE-PUZZLE package including developments presented in this thesis is freely available from `http://www.tree-puzzle.de`.

Contents

1. Overview **1**
1.1. Motivation . . . 1
1.2. Organization of this Thesis . . . 2

2. Molecular Phylogenetics: A General Introduction **5**
2.1. Biological Sequences and Molecular Evolution . . . 5
2.1.1. Types of Biological Data . . . 5
2.1.2. Sequence Evolution . . . 6
2.1.3. Multiple Sequence Alignment . . . 7
2.1.4. Modeling Molecular Evolution . . . 7
2.2. Phylogenetic Tree Reconstruction . . . 10
2.2.1. Notation on Phylogenetic Trees . . . 10
2.2.2. Types of Phylogenetic Methods . . . 14
2.2.3. Maximum Likelihood on Phylogenetic Trees . . . 14
2.2.4. Complexity of Phylogenetic Analysis . . . 17

3. The Quartet Puzzling Algorithm **19**
3.1. Introduction . . . 19
3.2. Methods from the TREE-PUZZLE Package . . . 20
3.2.1. *Likelihood Mapping* . . . 20
3.2.2. *Quartet Puzzling* . . . 22
3.2.2.1. *ML Step* . . . 23
3.2.2.2. *Puzzling Step* . . . 23
3.2.2.3. *Consensus Step* . . . 24

4. Improvement in Complexity of the Puzzling Step of QP **25**
4.1. Introduction . . . 25
4.2. Complexity Measures . . . 25
4.3. Complexity of the Original *Puzzling Step* . . . 27
4.4. More Efficient *Puzzling Step* Algorithms . . . 29
4.4.1. Split-Based *Puzzling Step* Algorithm . . . 29
4.4.2. Recursive *Puzzling Step* Algorithm . . . 31
4.4.2.1. Example . . . 32
4.4.2.2. Complexity . . . 34

4.4.3. Berry's MRCA-Based *Puzzling Step* Algorithm . . . 36
4.5. Runtime Analysis . . . 38
4.5.1. Datasets . . . 38
4.5.2. Benchmark Setup . . . 39
4.5.3. Results . . . 40
4.6. Discussion . . . 40

5. Parallelized Quartet Puzzling 43
5.1. Introduction . . . 43
5.2. Parameter Estimation for Evolutionary Models . . . 44
5.2.1. Algorithm: Estimating Model Parameters . . . 44
5.3. Datasets . . . 45
5.4. Runtime Analysis of the Sequential TREE-PUZZLE . . . 46
5.4.1. Runtime of the Parameter Estimation . . . 46
5.4.2. Runtime of the *Quartet Puzzling* Algorithm . . . 46
5.5. Parallelizing TREE-PUZZLE . . . 47
5.5.1. Parallelizing the Parameter Estimation . . . 47
5.5.2. Parallelizing *Quartet Puzzling* . . . 47
5.6. Scheduling Algorithms . . . 48
5.7. Efficiency of Parallel TREE-PUZZLE . . . 50
5.7.1. Benchmark Datasets and Setup . . . 50
5.7.2. Results for Parameter Estimation . . . 51
5.7.3. *ML Step* and *Puzzling Step* . . . 51
5.7.4. Overall Scaling of Parallel TREE-PUZZLE . . . 51
5.8. Discussion . . . 56

6. Large ML Trees from Sequences Using Quartets 57
6.1. Introduction . . . 57
6.2. The Modified *Quartet Puzzling* Algorithm . . . 57
6.2.1. The Algorithm ModPUZZLE . . . 58
6.3. Computational Aspects . . . 60
6.4. Some Practical Measures on the Method . . . 60
6.5. Discussion and Possible Extensions . . . 61

7. Phylogenetic Trees from Multiple Genesets with Missing Data 63
7.1. Introduction . . . 63
7.2. Methods to Combine Datasets . . . 64
7.2.1. *Low Level* Combination: Total Evidence . . . 64
7.2.2. *High Level* Methods . . . 64
7.2.2.1. Combining Equal-Sized Datasets: Consensus Methods . 64
7.2.2.2. Combining Overlapping Datasets: Supertree Methods . . 66
7.3. *Medium Level* Combined Phylogenetic Analysis . . . 68
7.3.1. Notation . . . 68
7.3.2. The Combined Quartet Method to Combine Genesets . . . 70

7.3.2.1. Combining the ML Quartets . . . 70
7.3.2.2. Computing the Overall Tree . . . 71
7.3.2.3. Assessing Whether Genesets Can Be Combined . . . 71
7.3.2.4. Overlap-Guided Puzzling Step . . . 72
7.3.2.5. Relative Majority Consensus . . . 73
7.4. The Phylogeny of the Grasses . . . 73
7.4.1. The Dataset . . . 73
7.4.2. Methods . . . 74
7.4.2.1. *Low Level* Combination . . . 74
7.4.2.2. *High Level* Combination . . . 74
7.4.2.3. *Medium Level* Combination . . . 74
7.4.3. Parameter Estimates from *Poaceae* Dataset . . . 76
7.4.4. Combinability of the *Poaceae* Dataset . . . 77
7.4.5. Reconstructed *Poaceae* Phylogeny . . . 78
7.4.5.1. *Total Evidence* Tree . . . 78
7.4.5.2. *MRP* Supertrees . . . 78
7.4.5.3. MinCut Supertrees . . . 82
7.4.5.4. Combined Quartet Tree . . . 82
7.4.6. Other Published *Poaceae* Phylogenies . . . 86
7.5. Discussion . . . 88
7.5.1. Problems of Dataset-Combining Methods . . . 88
7.5.2. Comparison of the Tree Topologies . . . 89
7.5.3. Conclusion . . . 90

8. Summary **93**

A. Variables and Functions **95**

B. Abbreviations **98**

Bibliography **101**

Acknowledgments **111**

List of Figures

1.1. Nucleotide Database Growth 2
1.2. Protein Database Growth 3

2.1. Sequence Evolution 6
2.2. DNA Substitution Rates 8
2.3. Tree Notation 11
2.4. Balanced and Linearized Tree Topologies 13

3.1. Binary Quartet Tree Topologies and Their Weights 20
3.2. Tree and an Induced Quartet Tree Topology 20
3.3. Partly and Unresolved Quartet Topologies and Their Weights 21
3.4. *Likelihood Mapping* 22
3.5. *Puzzling Step*: Finding the Insertion Branch in a 4-Tree 23
3.6. *Puzzling Step*: Finding the Insertion Branch in a 5-Tree 23

4.1. Complexity Notations 26
4.2. Penalty Sums Over the Tree 32

5.1. Parallelized Workflow of TREE-PUZZLE 48
5.2. Speedup and Runtime of Parameter Estimation 52
5.3. Speedup and Runtime of the *ML Step* 53
5.4. Speedup and Runtime of the *Puzzling Step* 54
5.5. Speedup and Runtime of the TREE-PUZZLE Program 55

6.1. Modified *Quartet Puzzling* Algorithm 59

7.1. Levels of Dataset Combination 65
7.2. Consensus Methods 66
7.3. Adams Consensus 66
7.4. MRP Coding Schemes 67
7.5. MinCut Supertree Algorithm 69
7.6. ModMinCut Supertree Algorithm 70
7.7. Overlap Graph of Gene Datasets 77
7.8. *Total Evidence* Phylogeny of the *Poaceae* Dataset 79
7.9. MRP-BR Phylogeny of the *Poaceae* Dataset 80
7.10. MRP-Pu Phylogeny of the *Poaceae* Dataset 81

7.11. MinCut supertree of the *Poaceae* Dataset 83
7.12. ModMinCut supertree of the *Poaceae* Dataset 84
7.13. Combined Quartet Phylogeny of the *Poaceae* Dataset 85
7.14. Extended Majority Consensus from All Intermediate Trees 87
7.15. Other *Poaceae* Trees 88

List of Tables

4.1. Quartets and Topologies for an Example 32
4.2. Runtime Analysis of *Puzzling Step* Algorithms 39

5.1. Runtime Profile for ML Parameter Estimation 46
5.2. Runtime Profile for the TREE-PUZZLE Parts 47

6.1. ModPUZZLE: Simulations with Increasing Overlap 61

7.1. Sequences in the *Poaceae* Dataset 75
7.2. Parameters of the *Poaceae* Datasets 76

List of Listings

4.1. Original Puzzling Step Algorithm 28
4.2. Split-based Puzzling Step Algorithm 30
4.3. Recursive Puzzling Step Algorithm 34
4.4. SubtreePenalty for the Recursive Algorithm 35
4.5. MRCA-Based Puzzling Step Algorithm 37
4.6. PropagatePenalties for the MRCA-based Algorithm 38

5.1. Estimating Model Parameters in TREE-PUZZLE 45

7.1. MinCut Supertree 69

1. Overview

1.1. Motivation

It is commonly accepted that contemporary genes, genomes, and organisms evolved from ancestors under the influence of natural selection. Consequently, the knowledge of the evolutionary tree behind their origin is crucial for understanding these entities. Evolutionary phylogenetics dealing with the reconstruction of those evolutionary relationships of organisms, genes, or gene families has become a basic application in many fields of research.

Knowledge about the relationships within gene families plays an important role in understanding, for example, the origins of biochemical pathways, regulatory mechanisms in cells as well as the development of complex systems. For example, knowing relationships between viruses is central for understanding their ways of infection and pathogenicity.

Phylogenetic analysis is usually performed on sets of aligned related sequences from the primary databases like EMBL/GenBank (Stoesser *et al.*, 2002; Benson *et al.*, 2003, nucleotide sequences) and SWISS-PROT/TrEMBL (Boeckmann *et al.*, 2003, protein sequences) or databases specialized on gene families like HOVERGEN (Duret *et al.*, 1994), SYSTERS (Krause *et al.*, 2002), the Ribosomal Database Project (RDP II, Maidak *et al.*, 2001), and the European Small Subunit rRNA Database (Van de Peer *et al.*, 2000b).

The development of efficient automated sequencing techniques and genome projects has produced a tremendously growing amount of data in these public databases (for a list see Baxevanis, 2003) during the recent decades. While a hand-curated high-quality database like SWISS-PROT grows about linearly (Fig. 1.2, dashed curve), other primary databases without the bottleneck of manual interaction like EMBL and TrEMBL show exponential growth rates (Figs. 1.1 and 1.2).

The enormous content of the public databases serves the need for larger datasets, since the increase of information is assumed to improve the accuracy of the results. The datasets compiled from the different data sources can increase in two dimensions, vertically and horizontally. Vertically means that the number of taxa or species is increased, horizontal growth describes the inclusion of more sites by obtaining either longer sequences or by adding different sequences to each taxon. The availability of large datasets motivates large scale projects like the assembly of the *tree of life*, a global phylogeny of all known organisms.

On the other hand, large datasets have led to severe difficulties, regarding runtime as well as memory requirements, for the efficient phylogenetic analysis of various types

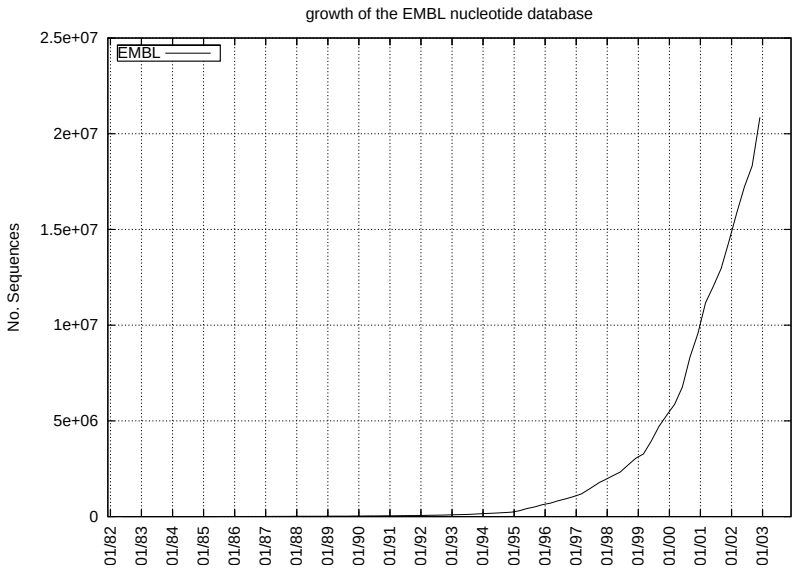


Figure 1.1.: Public nucleotide databases like the EMBL database show exponential growth, doubling their content every nine month

of large datasets. To be able to elucidate such humongous amounts of data, it is of utmost importance to design fast and efficient algorithms for molecular sequence analysis. This especially applies to maximum likelihood methods in phylogenetic analysis due to their enormous need of computational resources. To make maximum likelihood tree reconstruction available for different kinds of large datasets, this thesis introduces four different approaches. First, the speedup of phylogenetic analysis using efficient algorithms is studied. Second, the waiting time of phylogeny reconstruction is reduced by introducing parallel computing. Furthermore, a method is suggested that can be used to find the coarse phylogenetic structure for a large dataset. Finally, a novel method is proposed to combine overlapping datasets of different genes into one overall tree. All these approaches are based on *Quartet Puzzling* (Strimmer and von Haeseler, 1996), a quartet-based phylogeny reconstruction method.

1.2. Organization of this Thesis

After a general introduction to biological data and phylogenetic analysis (**Chapter 2**) the underlying quartet-based algorithms, namely, *Likelihood Mapping* (Strimmer and von Haeseler, 1997) and *Quartet Puzzling* (Strimmer and von Haeseler, 1996; Strimmer *et al.*, 1997) are delineated (**Chapter 3**). This thesis approaches the phylogenetic analysis of large datasets from four different angles described in **Chapters 4** to **7**.

Chapter 4: Chapter 4 introduces two new algorithms which reduce the level of complexity in the *puzzling step* from $O(n^5)$ to $O(n^4)$. The improved algorithms are

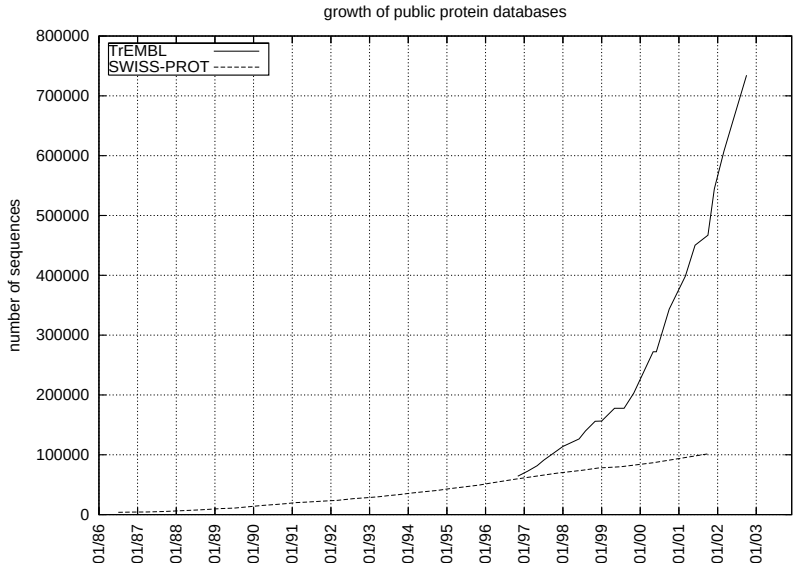


Figure 1.2.: While hand-curated protein databases like SWISS-PROT show only linear growth rates, the number of protein sequences derived from the nucleotide databases still grow exponentially (here TrEMBL, established in 1996).

compared to the original implementation and to another $O(n^4)$ algorithm suggested by Vincent Berry (Strimmer, personal communication).

Chapter 5: Due to their frequently still polynomial complexity, heuristics often do not sufficiently reduce the runtime of large dataset analysis. Hence, parallel computation has proven to be a valuable tool to decrease waiting time for the analysis. Chapter 5 describes the parallelization of the tree reconstruction in the TREE-PUZZLE package.

Chapter 6: The *maximum likelihood step* of *Quartet Puzzling* causes a major part of the runtime of tree reconstruction. Therefore, in Chapter 6 a modified *Quartet Puzzling* algorithm is suggested to decrease computation time by reducing the number of quartets used for phylogeny reconstruction.

Chapter 7: As mentioned, large datasets do not only consist of sequences from *many species* (vertically large). Since long sequences are considered to increase information in the analysis, sequences of different genes are analyzed simultaneously (horizontally large). Yet available sequences are not spread evenly among species of interest. Therefore, chapter 7 introduces a quartet-based method to combine incomplete genesets for phylogenetic reconstruction. This method is discussed in the context of *total evidence* and *SuperTree* methods.

At the end of the thesis the results will be summarized and a short outlook to future work will be given (**Chapter 8**).

2. Molecular Phylogenetics: A General Introduction

This chapter will present some basics of phylogenetic analysis. In the first section biological sequence data, the concept of molecular evolution, and a way to model the evolutionary process will be introduced. The basic notations on trees will be defined in the second part of the chapter, followed by the explanation of different types of phylogenetic methods, and a brief introduction to maximum likelihood phylogenies. Finally, some facts about the complexity of phylogenetic reconstruction will be given.

2.1. Biological Sequences and Molecular Evolution

2.1.1. Types of Biological Data

To study evolutionary relationships, very diverse biological data are used like morphological characters, binary data, genomic gene order, nucleotide, and protein sequence data.

The first data used were morphological characters of the species under interest. Morphological characters have the advantage to be easily obtainable by eye or by microscopy without molecular laboratory work.

In the early times of molecular phylogenetics, coarse grain genetic data like binary characters of presence and absence of restriction sites played an important role. Very recently, gene order data has been used to reconstruct phylogenetic relationships from the order of genome rearrangements, so-called breakpoint phylogenies (Blanchette *et al.*, 1997; Sankoff and Blanchette, 1998).

Today the vast majority of data in phylogenetic reconstruction is biological sequence data like nucleotide or protein sequences as stored in the public databases (see p. 1). One major advantage of biological sequence data is the increase of phylogenetically relevant information due to the large number of characters, that is, sites or residues, that can be obtained by sequencing.

Molecular sequences are coded as strings of literals from an alphabet $\mathbb{A}$ representing the order in which the building blocks are connected in the molecules. Nucleotide sequence data (DNA and RNA) are coded with an alphabet of the four nucleotides Adenine, Guanine, Cytosine, and Thymine in DNA or Uracil in RNA. Each nucleotide

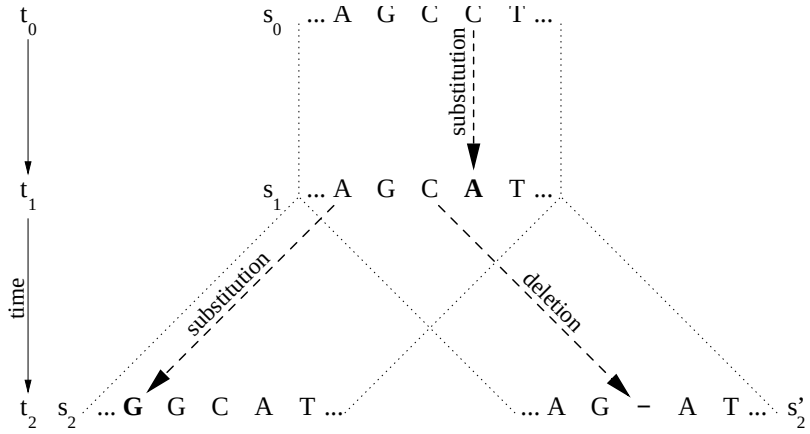


Figure 2.1.: An example for sequence evolution. Sequence s_0 evolves to s_1 in time $t_1 - t_0$. Then s_1 evolves divergently to s_2 and s_2' after a speciation/duplication event in time $t_2 - t_1$.

or base is denoted by its first letters, $\mathbb{A} = \{A, G, C, T/U\}$ (Liébecq, 1992). Nowadays nucleotide sequences can be efficiently obtained using fast (automated) sequencing techniques (Sanger *et al.*, 1977 and, e.g., Ansorge *et al.*, 1993).

Protein data is encoded over an alphabet of 20 amino acids (Liébecq, 1992). Each amino acid is represented as a one-letter or as a three-letter literal ('A' or 'Ala' for Alanine, 'P' or 'Pro' for Proline etc.). Although there has been a lot of work on protein sequencing (Edman, 1950; Sanger and Thompson, 1953) and its automation (Edman and Begg, 1967) since the 1950s, the indirect strategy of sequencing protein coding genes with the method of Sanger *et al.* (1977) and the subsequent translation into amino acids has proven to be faster and has become the usual way of deriving protein sequences.

2.1.2. Sequence Evolution

Before the division of any cell, plastid, or mitochondrion its genome has to be replicated to be inherited to the daughter cells or organelles. In spite of a proof reading machinery the process of copying is not error-free. Additionally, damages are introduced to the DNA by mutagens such as certain chemicals or UV light. Hence, the genomic sequence accumulates mutations as traces of evolutionary development. Although more complex mutations like rearrangements, duplications, and inversions are possible on chromosome level, only point mutations affecting single spots of the DNA are commonly considered. Point mutations are

substitutions – the exchange of one character state by another (Fig. 2.1)

insertions – inserting one or more characters

deletions – deleting one or more character usually indicated by gap characters '-' in the sequence. (Fig. 2.1)

Insertions and deletions cannot be distinguished when examining two contemporary sequences (e.g., s_2 and s_2' in Fig. 2.1). To decide whether an insertion or deletion has happened, one needs information on the sequence of the common ancestor (s_1 in Fig. 2.1) which, however, is usually unknown. For that reason *in*sertions and *del*etion are generalized as *indels*. Indels and substitutions generally apply to nucleotide as well as protein data. Although mutations happen at the genomic level, they can affect the amino acid sequence translated from protein coding genes. Nucleotide substitutions in protein coding genes that do not result in an amino acid substitution are called synonymous or silent mutations in contrast to non-synonymous mutations.

2.1.3. Multiple Sequence Alignment

Homologous sequences, that means, sequences related by a common ancestor sequence, are typically presented in a multiple sequence alignment (MSA). An alignment is a data matrix in which homologous characters, also called sites, of the sequences are aligned in the same column. For instance, the alignment of the sequences from Fig. 2.1 is

$$\begin{array}{lcccccc} s_0: & \ldots & A & G & C & C & T & \ldots \\ s_1: & \ldots & A & G & C & A & T & \ldots \\ s_2: & \ldots & G & G & C & A & T & \ldots \\ s_2': & \ldots & A & G & - & A & T & \ldots \end{array} \tag{2.1}$$

To present indels, an additional gap character '-' is used. Sequence alignments serve as input to almost every sequence analysis and programs like CLUSTALW (Thompson *et al.*, 1994) or DIALIGN (Morgenstern, 1999) are available to compute an MSA for a collection of sequences. However, there is still no standardized approach how to treat indels in the subsequent analysis. Typically columns with gaps are either discarded from the alignment, or indels are considered as 'wildcard' characters substituted by a distribution of character states. In the following we will for simplicity delete sites with indels.

2.1.4. Modeling Molecular Evolution

To reconstruct evolution, the evolutionary process is modeled as an evolutionary Markov process (EMP, Tavare, 1986; Müller and Vingron, 2000), also known as substitution model. An EMP is a time homogeneous, calibrated, stationary, reversible Markov process.

It is assumed that the frequencies π_ρ of the characters $\rho \in \mathbb{A}$ are in equilibrium and remained stationary during evolution and among the sequences (*stationarity*). The vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_{|\mathbb{A}|})$ of the character frequencies is called stationary distribution.

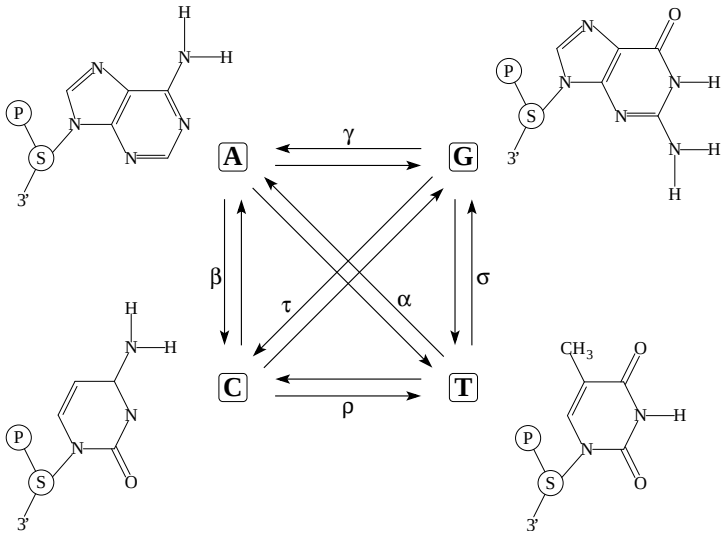


Figure 2.2.: DNA substitution rates

Since we infer evolution backward in time and generally have no means to derive ancestral sequences, mutations are modeled to be reversible, that is, mutations from ρ to σ are assumed to be equally likely as the back-mutation from σ to ρ (*reversibility*)

$$\pi_\rho P_{\rho\sigma}(t) = P_{\sigma\rho}(t)\pi_\sigma \qquad \rho, \sigma \in \mathbb{A}. \tag{2.2}$$

This assumption is also known as detailed balance.

Since evolution is modeled as a Markov process, it evolves without memory, that is, for the development from sequence s_1 to s_2 in Fig. 2.1 the character state of s_0 does not matter, only the state s_1 at t_1 is of importance. Furthermore, it is assumed that the Markov process acts on each character of the sequence independently from the others (*independence*).

The following applies to DNA as well as protein evolution but for simplicity we consider the smaller nucleotide alphabet $\mathbb{A} = \{A, C, G, T\}$ only. The probability of substitution of any character ρ by character σ in evolutionary time t is described by a transition probability matrix $\boldsymbol{P}$. Given the stationary distribution $\boldsymbol{\pi}$ and substitution rate matrix $\boldsymbol{R}$ (see Fig. 2.2 for substitution rates)

$$\boldsymbol{R} = \begin{array}{c} \begin{array}{cccc} A & C & G & T \end{array} \\ \begin{pmatrix} - & \beta & \gamma & \alpha \\ \beta & - & \tau & \rho \\ \gamma & \tau & - & \sigma \\ \alpha & \rho & \sigma & - \end{pmatrix} \end{array} \begin{array}{c} A \\ C \\ G \\ T \end{array} \tag{2.3}$$

an instantaneous rate matrix $\boldsymbol{Q}$ can be constructed such that

$$\boldsymbol{Q} = \begin{pmatrix} -\sum_{k\neq A} Q_{Ak} & \mu R_{AC}\pi_C & \mu R_{AG}\pi_G & \mu R_{AT}\pi_T \\ \mu R_{CA}\pi_A & -\sum_{k\neq C} Q_{Ck} & \mu R_{CG}\pi_G & \mu R_{CT}\pi_T \\ \mu R_{GA}\pi_A & \mu R_{GC}\pi_C & -\sum_{k\neq T} Q_{Gk} & \mu R_{GT}\pi_T \\ \mu R_{TA}\pi_A & \mu R_{TC}\pi_C & \mu R_{TG}\pi_G & -\sum_{k\neq T} Q_{Tk} \end{pmatrix} \tag{2.4}$$

Hence, for the row sums the following applies.

$$\sum_{\sigma\in\mathbb{A}} Q_{\rho\sigma} = 0 \qquad \text{for each } \rho \in \mathbb{A}. \tag{2.5}$$

The factor μ in Eq. 2.4 is calibrated such that

$$\sum_{\rho\neq\sigma} Q_{\rho\sigma} = -\sum_{\rho} Q_{\rho\sigma} = 1 \qquad \rho,\sigma \in \mathbb{A}. \tag{2.6}$$

The transition probability matrix $\boldsymbol{P}$ is constructed from $\boldsymbol{Q}$

$$\boldsymbol{P}(t) = e^{\boldsymbol{Q}t}. \tag{2.7}$$

Sometimes (cf. Müller and Vingron, 2000) μ is calibrated such that

$$\sum_{\rho\neq\sigma} \pi_\rho P_{\rho\sigma}(1) = 0.01 \qquad \rho,\sigma \in \mathbb{A}. \tag{2.8}$$

That means, evolutionary time is calibrated such that in time $t = 1$ (1 PAM) a sequence of length m accumulates $m/100$ ($= 1\%$) mutations.

$\boldsymbol{P}$ also has the following properties:

$$\sum_{\sigma\in\mathbb{A}} \pi_\rho P_{\rho\sigma}(t) = 1.0 \qquad \text{for each } t \leq 0 \text{ and } \rho \in \mathbb{A} \tag{2.9}$$

and

$$\lim_{t\searrow 0} P_{\rho\sigma}(t) = \begin{cases} 0.0, & \text{if } \rho \neq \sigma \\ 1.0, & \text{if } \rho = \sigma \end{cases} \qquad \rho,\sigma \in \mathbb{A}. \tag{2.10}$$

Note, that an EMP is uniquely described by $\boldsymbol{Q}$ or $\boldsymbol{P}(t)$, since $\boldsymbol{\pi}$ follows the equations $\boldsymbol{\pi Q} = 0$ and $\boldsymbol{\pi P}(t) = \boldsymbol{\pi}$ for all $t > 0$ (Tavare, 1986).

As mentioned above, Eq. 2.2 to Eq. 2.10 also apply for amino acids, but 20×20-matrices are used instead of 4×4-matrices, due to the larger alphabet size $|\mathbb{A}|$.

A number of models has been suggested both for nucleotide as well as amino acid evolution. The models vary in complexity regarding their parameters. Common DNA

models (listed in ascending complexity) are JC69 (Jukes and Cantor, 1969), Kimura-2-Parameter model (K2P, Kimura, 1980), HKY85 (Hasegawa *et al.*, 1985), TN93 (Tamura and Nei, 1993), and the General Time Reversible model (GTR, Rodriguez *et al.*, 1990) as used in Eq. 2.3 and Fig. 2.2.

To model amino acid evolution, matrices are designed for different scenarios. While the Dayhoff model (Dayhoff *et al.*, 1978) and JTT (Jones *et al.*, 1992) are generally applicable, VT (Müller and Vingron, 2000) and WAG (Whelan and Goldman, 2001) model distant evolution. Specialized matrices like mtREV (Adachi and Hasegawa, 1996a) and cpREV (Adachi *et al.*, 2000) model mitochondrial and chloroplast evolution respectively.

Transition probability matrices are applied to database searches and alignment construction. Furthermore, they are used to compute evolutionary distances between sequences and to evaluate trees for phylogenetic reconstruction in a maximum likelihood (ML) framework. For more detailed information on transition probability matrices see, e.g., Ewens and Grant (2001), Graur and Li (2000), Durbin *et al.* (1998), and Tavare (1986).

Additional to the transition probability matrix, other assumptions can be made to more realistically model the evolutionary process. It is known that sequence positions may evolve with different rates. To account for this fact, each site in an alignment may also get its own rate specific factor. Typically one assumes that rates are distributed according to a Γ-distribution, where one parameter α describes the amount of rate heterogeneity (Gu *et al.*, 1995; Yang and Kumar, 1994). Apart from that, methods have been suggested to estimate site-specific rates for each alignment column without assuming a Γ-distribution (e.g., Van de Peer *et al.*, 2000a; Meyer and von Haeseler, 2003, and references therein).

2.2. Phylogenetic Tree Reconstruction

Recent taxa and their genetic sequences evolved from ancestors. Hence, tree diagrams – phylogenetic trees – are a logical way to present evolutionary relationships. Before going into detail how phylogenetic trees are reconstructed, the basic notations on trees will be defined.

2.2.1. Notation on Phylogenetic Trees

The basic tree notations used throughout this thesis will be defined by means of the examples in Fig. 2.3. Certain characteristics of tree topologies will be explained using the trees in Fig. 2.4. If special notations are needed later on they will be defined in the corresponding chapters.

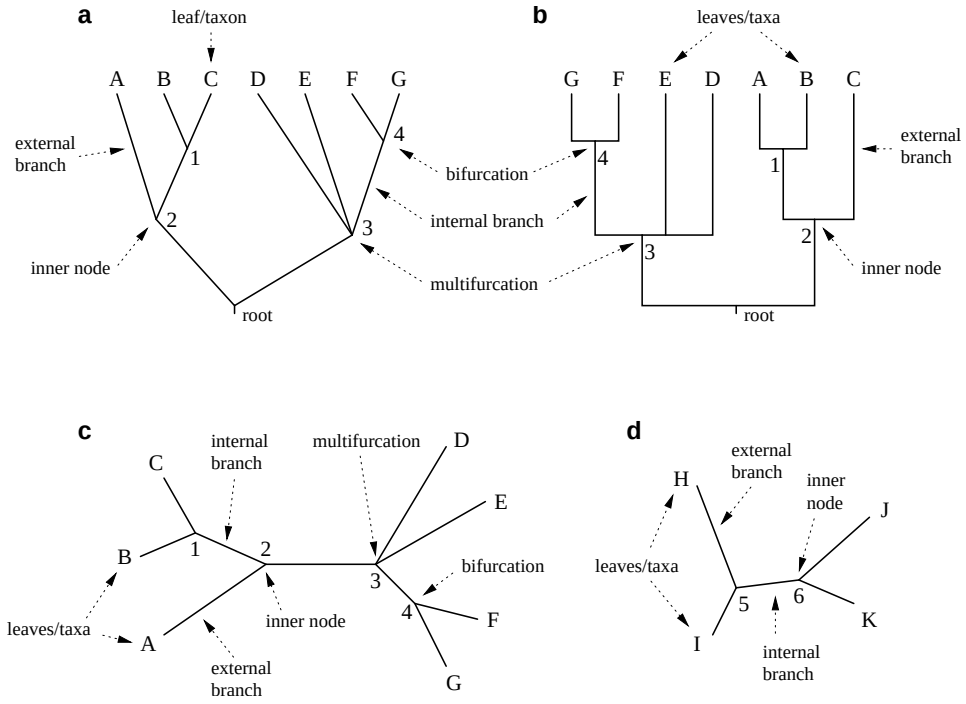


Figure 2.3.: Tree notation on different rooted tree diagrams (a,b), an unrooted tree (c), and unrooted quartet tree (d)

Trees, Nodes, and Edges A tree is an acyclic graph $T = (\mathcal{V}, \mathcal{E})$ consisting of a set of nodes $\mathcal{V}$ and a set $\mathcal{E}$ of edges linking two nodes each. If a trees has a root node $\in \mathcal{V}$ it is called a rooted tree represented by a directed acyclic graph (DAG) for which all edges $e \in \mathcal{E}$ are directed away from the root. Edges are denoted by their adjacent nodes, e.g., $e_{(1,2)}$ describes the edge linking nodes 1 and 2. In a rooted tree $e_{(\bullet,2)}$ describes the edge ending at node 2.

The degree of a node v is defined as the number of edges connected to v. External nodes, also called leaves, are connected to one branch only, they have degree 1. Nodes with degree > 1 are internal nodes. Except for the root node in rooted trees all inner nodes will have at least degree 3. A node with three adjacent branches (degree 3) is called a bifurcation. Nodes with degrees > 3 are called multifurcations (Fig. 2.3). A tree where all inner nodes have degree 3 is called a bifurcating or binary tree. Otherwise it is a multifurcating tree which is not completely resolved.

Evolutionary Trees, Leafsets, and Leaf-Labeled Trees In evolutionary trees, also called phylogenies, the nodes correspond to taxa and their ancestors. The edges, also called branches, depict the development between two nodes. Edges can be assigned weights to express evolutionary time. Evolutionary time is displayed as branch lengths (Fig. 2.3a,c). In Fig. 2.3b, only the vertical branch lengths depict time, while the horizontal part does not count.

Generally, sequences represent the taxa they are derived from. The leaves of the

phylogenetic tree are labeled by contemporary sequences (A to G in Fig. 2.3), while the internal nodes (nodes 1 to 4) correspond to the hypothetical ancestor sequences. Since we have no knowledge about ancestral sequences, evolutionary trees are leaf-labeled trees. Hence, the terms taxon, species, sequence, leaf, and external node are used interchangeably.

To access leafsets of trees we define the following. $T(\mathcal{S})$ is a tree with leafset $\mathcal{S}$, and $\mathcal{L}(T)$ describes the leafset of tree T. Hence, $\mathcal{L}\left(T(\mathcal{S})\right)$ is exactly $\mathcal{S}$.

A tree with n leaves is called an n-tree. The topology of an n-tree is denoted by T_n. Accordingly, the size of the leafset $|\mathcal{L}(T_n)|$ of an n-tree is n.

An n-tree has a maximal number of $n-2$ internal nodes and $2n-3$ edges if it is completely resolved. A completely unresolved tree comprises the minimal number of one internal node and n edges, i.e., it has only external edges.

Biological Root and Outgroup Rooting Since the sequence at the general ancestor is unknown the placement of the root often remains a problem. Therefore, the phylogeny reconstruction programs usually produce 'unrooted trees' like in Fig. 2.3c. If one wants to determine the order of evolutionary events additional information is necessary to place the root on the tree. One approach is *outgroup rooting*. Outgroups comprise one or more taxa that are known to have branched off the tree of life before the interesting group of taxa in the analysis. The root is then placed on the branch between the outgroup and the rest of the tree. The unrooted tree in Fig. 2.3c has been rooted by the outgroup sequences A, B, and C in the trees in Fig. 2.3a and b.

Splits and Bipartitions The internal edges induce bipartitions on the leafset, that means, when cut they split the set of leaves into to two disjoint sets. In Fig. 2.3, the edge $e_{(1,2)}$ between nodes 1 and 2 splits the leaf sets of the trees into the disjoint subsets $\{B, C\}$ and $\{A, D, E, F, G\}$. This bipartition or split is denoted as $BC|ADEFG$. In case of trees of four sequences this notation exactly describes the tree topology since there is only one internal branch possible (e.g., $HI|JK$ in Fig. 2.3d). To describe the subsets induced by a splitting edge e, we define

$$\overleftarrow{\mathbb{1}}_e = \{\text{taxa, on the left of edge } e\} \tag{2.11}$$

$$\overrightarrow{\mathbb{1}}_e = \{\text{taxa, on the right of edge } e\} \tag{2.12}$$

$$\overleftarrow{{}_r\mathbb{1}}_e = \{\text{taxa, on the same side as the root of edge } e\} \tag{2.13}$$

$${}_r\overrightarrow{\mathbb{1}}_e = \{\text{taxa, not on the same side as the root of edge } e\} \tag{2.14}$$

Although the sets $\overleftarrow{\mathbb{1}}_e$ and $\overrightarrow{\mathbb{1}}_e$ are not uniquely defined, the two notations are sufficient to describe the two set of leaves induced by an edge e.

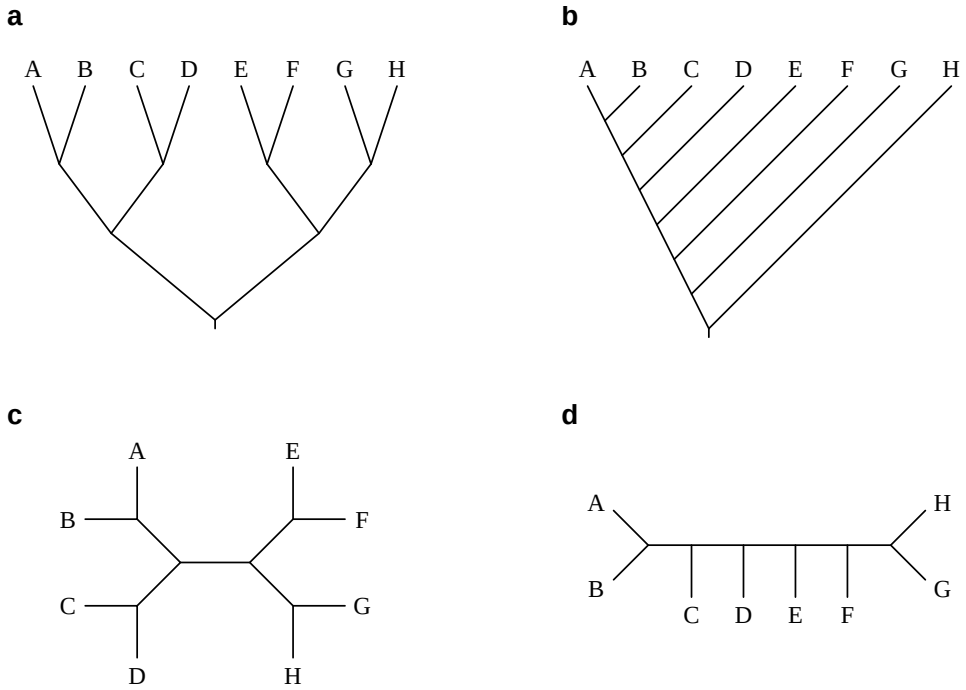


Figure 2.4.: Balanced and linearized tree topologies: balanced rooted tree (a), linearized rooted tree (b), balanced unrooted tree (c), linearized unrooted tree (d) also called caterpillar or comb tree

Rooted Trees, Ancestors, and Clusters In a rooted tree, $\mathrm{succ}(w)$ denotes the set of immediate descendants of node w, e.g., $\mathrm{succ}(3) = \{4, E, F\}$ in Fig. 2.3a and b.

Nodes in rooted trees allow the notation of common ancestors. The *most recent common ancestor* (MRCA) of a set of leaves or nodes is defined as the ancestral node furthest from the root that has all these leaves as descendants. In Fig. 2.3, the most recent common ancestor of the nodes $D, 4, F$ is node 3, i.e., $\mathrm{MRCA}(D, 4, F) = 3$, while $\mathrm{MRCA}(B, E) = \text{root}$.

The set of all descendants of an ancestral node is called a monophyletic group, or clade. Taxa A, B, and C are forming a monophyletic group while D, F, and G are not monophyletic. The set of descendant leaves of an internal node is also called the cluster of that node. The cluster of node 1 in Fig. 2.3 is $C^n(1) = \{B, C\}$. Since in rooted trees edges are directional, we also speak about the cluster of an edge, e.g., the cluster of the edge $e_{(2,1)}$ is also $C^e(e_{(2,1)}) = \{B, C\}$ which is identical to ${}_r\vec{\Gamma}_{e_{(2,1)}}$.

Binary Tree Topologies, Diameter, and Height Two topologies among the binary trees are extreme with respect to the diameter, the diameter of a tree being defined as the longest path between any two nodes in the tree. First, there are fully balanced trees (Fig. 2.4a,c) where all leaves have the same height. The height h is defined as the number of edges between that leaf and the root in rooted trees or that leaf and the central edge in unrooted trees. Second, there are linearized trees also called caterpillar or comb-like

trees, which are highly unbalanced (Fig. 2.4b,d). All other binary tree topologies are intermediates between these extremes. The number of binary unrooted n-trees $B(n)$ is

$$B(n) = \prod_{k=3}^{n}(2k-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!} \tag{2.15}$$

(Felsenstein, 1978). The number of possible rooted n-trees is $B(n+1)$, considering the root as an additional leaf.

2.2.2. Types of Phylogenetic Methods

A rich variety of tree reconstruction methods based on sequences has been developed (for an overview see Swofford *et al.*, 1996; Baxevanis *et al.*, 2002), which fall into three categories, (a) maximum parsimony methods (MP), (b) evolutionary distances (ED) between sequences, and (c) approaches applying the maximum likelihood (ML) principle. MP tries to find the tree which explains the data with the least mutations. ED methods calculate pairwise distances between the taxa and construct a tree from these distances. ML aims to find the tree that gains the maximum likelihood to have produced the underlying data. Most of these methods use objective functions to evaluate trees and aim to find trees optimal in respect to the objective functions.

In the following we will describe how trees a evaluated using maximum likelihood. ML analysis is statistically well founded and used in many fields of research (e.g., Cramer, 1989; Eliason, 1993; Mendel and Burrus, 1990; Ewens and Grant, 2001). Since ML has been made applicable for reconstructing phylogenies of molecular sequences in 1981 by Felsenstein, it has shown to give accurate results in practice and to be quite robust against several sources of error. Although phylogenetic methods applying ML are computationally very expensive, their use in phylogeny reconstruction has increased. They are a routine method in phylogenetic analysis.

2.2.3. Maximum Likelihood on Phylogenetic Trees

In this section ML analysis is described briefly, for more details on maximum likelihood phylogenies refer to Felsenstein (1981), Goldman (1990), and Swofford *et al.* (1996).

In ML analysis a likelihood value is used to describe the goodness of fit of the hypothesis H and the data A. The likelihood of the hypothesis H given the data A is defined as the probability of the data A which in turn is the product of the partial probabilities for each data sample A_c (Goldman, 1990)

$$L(H|A) = \Pr(A|H) = \prod_{c} \Pr(A_c|H). \tag{2.16}$$

The basic principle of ML estimation is that the best estimate of an unknown parameter of the hypothesis H is the one making the analyzed data most probable, which means, its likelihood value is maximized.

As data in molecular phylogenetics serves an alignment A comprising m columns of a set $\mathcal{S} = \{s_1, s_2, \dots s_n\}$ of n aligned homologous sequences like

	A_1	$\dots$	A_{c-1}	A_c	A_{c+1}	$\dots$	A_m
s_1	A	$\dots$	G	T	A	$\dots$	C
s_2	A	$\dots$	G	C	T	$\dots$	C
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
s_n	T	$\dots$	G	C	G	$\dots$	C

(2.17)

To compute the likelihood, we need a hypothesis H of the process how the data A came into existence. We assume that the contemporary sequences evolved according to a phylogenetic tree T under the influence of an EMP M (2.1.4). The phylogenetic tree T consists of an n-tree topology and a set of branch lengths t. t_{vw} denotes the length of the branch $e_{(v,w)}$ that connects the two nodes v and w. Each of the m alignment columns A_c is regarded to be an independent sample from the EMP acting along the tree. Although the hypothesis H includes both, the tree T and the EMP M, for simplicity we will consider the tree T only, we assume M to be fixed.

To evaluate T, its likelihood value is calculated. If the tree T (topology, branch lengths) and the character states at all nodes are known, the calculation is straightforward. For each data sample the likelihood is computed by multiplying the probabilities $P_{\rho\sigma}(t_{v,w})$ (Eq. 2.7) for all edges $e_{(v,w)}$, where ρ and σ are the character states at the adjacent nodes v and w and $t_{v,w}$ is the evolutionary time between these node. Finally, the likelihoods of all data samples are multiplied to gain the overall likelihood of the tree.

However, since we do not know the ancestral sequences we have to sum over all possible character states at the internal nodes. The possible states are the elements of the alphabet $\mathbb{A}$, that is, 20 amino acids for protein data or 4 nucleotides for DNA.

The likelihood $L_T = L(T|A)$ of the tree T is computed from the alignment A using the probability matrix $\boldsymbol{P}(t_e)$ for each edge e (Eq. 2.7) and the stationary distribution $\boldsymbol{\pi}$ of M as described by Felsenstein (1981). L_T is computed recursively in a post-order traversal of the tree, i.e., the probabilities of the states at the nodes will be computed after calculating the probabilities at the descendant nodes. The recursive calculation will proceed as follows.

Assume T a rooted n-tree. At the moment we only consider one column A_c of the alignment, which is possible under the assumption that the sites are independent. $A_{c,k}$ describes the character state of sequence s_k in that column.

Assuming node v to be a leaf representing sequence s_k, the likelihood $L_v(\rho)$ of the character state ρ at leaf v is determined from the alignment column since its state is known.

$$L_v(\rho) = \begin{cases} 1 & \text{if } \rho = A_{c,k} \\ 0 & \text{otherwise} \end{cases} \qquad \rho \in \mathbb{A}. \tag{2.18}$$

If node v is an inner node, the probability of state ρ at node v having given rise to states $\sigma \in \mathbb{A}$ at a descending node u is computed by summing up the products of probability $P_{\rho\sigma}(t_{v,u})$ that ρ is σ after time $t_{v,u}$ and the partial likelihood $L_u(\sigma)$ to find state σ at node u

$$L_v(\rho) = \sum_{\sigma \in \mathbb{A}} P_{\rho\sigma}(t_{v,u}) L_u(\sigma). \tag{2.19}$$

Since an inner node v has at least two descendants, the likelihood of any state ρ at node v is calculated by multiplying all probabilities for all succeeding nodes $w \in \mathrm{succ}(v)$

$$L_v(\rho) = \prod_{w \in \mathrm{succ}(v)} \left(\sum_{\sigma \in \mathbb{A}} P_{\rho\sigma}(t_{v,w}) L_w(\sigma) \right). \tag{2.20}$$

The likelihood $L_T^{A_c} = \Pr(A_c|T, M)$ of the alignment column A_c to be produced along tree T by an EMP M is the sum of the partial likelihoods of the states $\sigma \in \mathbb{A}$

$$L_T^{A_c} = \Pr(A_c|T, M) = \sum_{\sigma \in \mathbb{A}} \pi_\sigma L_{root}(\sigma). \tag{2.21}$$

From the likelihoods of the alignment columns, one computes the overall likelihood value of A for the tree T

$$L_T = \Pr(A|T, M) = \prod_{k=1}^{m} L_T^{A_k} \tag{2.22}$$

or the log-likelihood

$$\ell_T = \ln L_T = \sum_{k=1}^{m} \ln L_T^{A_k}. \tag{2.23}$$

This objective function enables the optimization of the parameters of the hypothesis. The branch lengths are optimized to find the maximum likelihood value of tree T under the assumption of a fixed model of evolution. Since optimizing all branch lengths at once is not feasible, Felsenstein (1981) suggested to optimize the branches one by one. This iterative task is accomplished by numerical methods such as Expectation Maximization (EM, Dempster *et al.*, 1977; Felsenstein, 1981) or Newton-Raphson-optimization (Press *et al.*, 1988; Olsen *et al.*, 1994) applying Felsenstein's (1981) *Pulley Principle*.

2.2.4. Complexity of Phylogenetic Analysis

Using an objective function like the one described in the previous section, a given tree can easily be evaluated. However, to find the optimal tree gaining the highest likelihood, one would have to examine all possible n-trees. As described in Eq. 2.15, the number of bifurcating unrooted trees $B(n)$ can be very large. For 10 sequences more than 2 million trees exist. For 55 taxa the trees outnumber the estimate of 10^{81} atoms in the known universe. Considering multifurcations makes the growth rate even more dramatic. Obviously, exhaustive tree searches are applicable for small data sets only.

To make things worse, the problem of finding an optimal phylogeny has been shown to be NP-complete for quite a number of reconstruction methods (Graham and Foulds, 1982; Foulds and Graham, 1982; Day and Sankoff, 1986; Day, 1987). NP-complete problems are those for which an efficient, i.e., polynomial-time algorithm is assumed to be impossible. In other words, it might be necessary to examine exponentially many trees to find the optimum. This should, in principle, also be true for ML methods.

In order to reduce the computational burden and to limit the vast number of trees to be examined heuristics have been suggested to search the tree space. Beside others a number of ML-based heuristics have been introduced such as stepwise insertion with local and global optimizations (DNAML, Felsenstein, 1981), the *Quartet Puzzling* algorithm (TREE-PUZZLE, Strimmer and von Haeseler, 1996; Schmidt *et al.*, 2002), and star decomposition (MOLPHY; Adachi and Hasegawa, 1996b). Recently, Bayesian approaches (MrBAYES, Huelsenbeck and Ronquist, 2001), genetic algorithms (Lewis, 1998), and simulated annealing (Salter and Pearl, 2001) have entered the field. Note, that heuristics cannot guarantee to find the best tree.

3. The Quartet Puzzling Algorithm

Since the novel methods and extensions that will be introduced throughout this thesis (chapters 4–7) are based on the *Quartet Puzzling* algorithm, the relevant methods implemented in the TREE-PUZZLE package will be described in this chapter. First the relevant notations on quartets are presented. Thereafter, the methods *Likelihood Mapping* and *Quartet Puzzling* will be explained.

3.1. Introduction

While most tree reconstruction methods aim at reconstructing the overall tree for n sequences by optimizing a global objective function, quartet methods break down this task into smaller parts. The methods attempt to reduce the problem of evaluating complex objective functions (cf. 2.2.3) for a lot of large trees to the computation of quartet trees, i.e. four-species trees, and to build the overall tree from the collection of quartet trees. The quartet tree topologies serve as building blocks of larger trees.

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be the set of n homologous (i.e. related) aligned sequences. $\mathcal{Q}$ denotes the set containing all $\binom{n}{4}$ quartets of $\mathcal{S}$. For each quartet $q = \{a, b, c, d\} \in \mathcal{Q}$ three binary tree topologies exist (Fig. 3.1). A quartet topology is uniquely described by the bipartition induced by the internal edge. Hence, quartet topologies are denoted by $ab|cd, ac|bd, ad|bc$ (Fig. 3.1), where $a, b, c, d \in \mathcal{S}$ are the labeled leaves of the trees.

Given a tree of n sequences, there are $\binom{n}{4}$ different quartets. For each quartet $q = \{a, b, c, d\}$ derived from an n-tree there is an induced unique quartet topology (Fig. 3.2). From the set of $\binom{n}{4}$ induced quartet topologies, the full tree can be unambiguously reconstructed (Bandelt and Dress, 1986). Since the overall tree is not known the set of quartet topologies has to be inferred from aligned sequences. Several criteria exist to find the set of quartet tree topologies supported by the input dataset. Possible methods are again evolutionary distances, the parsimony criterion, or the maximum likelihood framework.

If each constructed quartet tree agrees with the unknown induced quartet tree, then reconstructing the n-tree is straight–forward. However, the stochastic process of evolution makes it unlikely to reconstruct each quartet correctly. This, on the one hand, leads to the reconstruction of contradicting quartet topologies. Consider, for instance, $AB|CD$, $AB|CE$, and $AE|BD$ to be reconstructed quartet trees from sequence data. While the first two support a split $AB|CDE$, this is contradicted by the placement of taxa B and E in the third tree. Thus, it is not possible to unambiguously join these

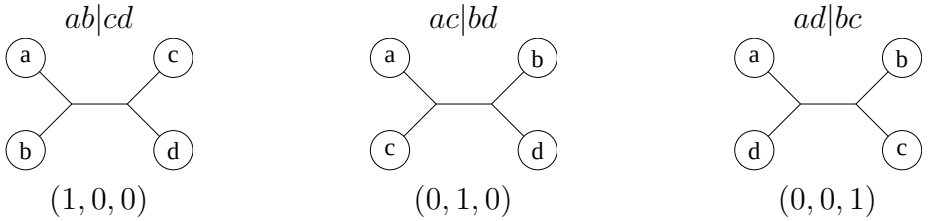


Figure 3.1.: The three different binary (fully resolved) tree-topologies for the quartet $q = \{a, b, c, d\}$. The generic weights ($w_{ab|cd}$, $w_{ac|bd}$, $w_{ad|bc}$) used for the least-square fit are given below the topologies.

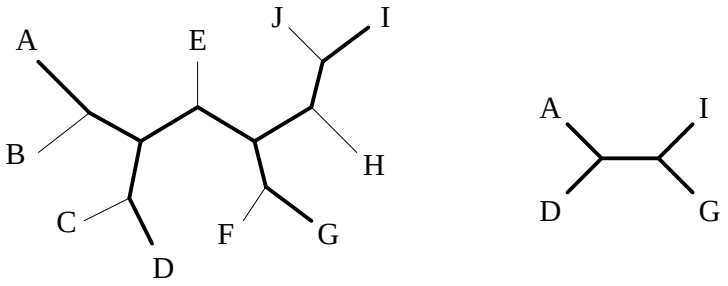


Figure 3.2.: An example tree with 10 leaves, and the induced topology for a quartet $\{A, D, G, I\}$.

trees. On the other hand, it might often not be possible to unequivocally resolve which topology is supported by the input data. Two or even all three topologies might gain almost equal support by the data. These so-called unresolved cases can be depicted by network or star tree topologies representing the neighborhood relations from more than one binary tree topology (Fig. 3.3). Most quartet methods, however, take into account only one best topology for each quartet.

A number of quartet-based tree reconstruction methods have been developed to construct trees from sets of 4-trees (e.g., Sattath and Tversky, 1977; Fitch, 1981; Bandelt and Dress, 1986; Dress *et al.*, 1986; Strimmer and von Haeseler, 1996; Ben-dor *et al.*, 1998; Ranwez and Gascuel, 2001). The developments and enhancements presented in this thesis are based on the quartet methods implemented in the TREE-PUZZLE package (Strimmer and von Haeseler, 1996; Schmidt *et al.*, 2002; Schmidt and von Haeseler, 2003).

3.2. Methods from the TREE-PUZZLE Package

3.2.1. Likelihood Mapping

While distance-based methods or the parsimony criterion can be used to decide which of the three informative tree topologies is best supported for one quartet, TREE-PUZZLE uses the maximum likelihood framework to select.

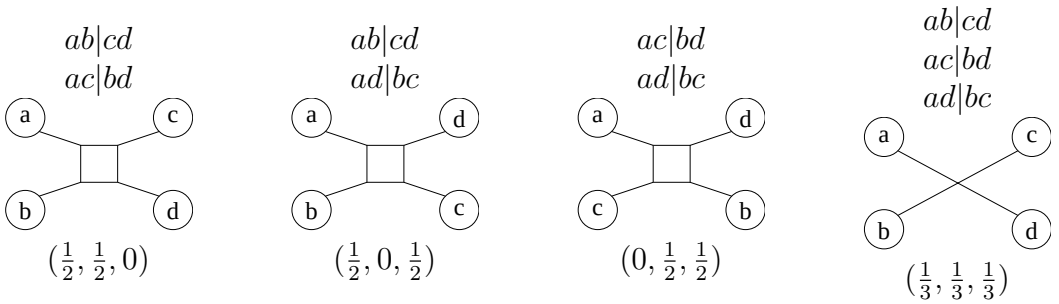


Figure 3.3.: The three different partially resolved tree-topologies and the completely unresolved one for the quartet $q = \{a, b, c, d\}$. The supported topologies combined in the respective scenario are given above, the generic weights ($w_{ab|cd}$, $w_{ac|bd}$, $w_{ad|bc}$) used for the least-square fit are given below the topologies.

For a quartet $q = \{a, b, c, d\}$ the trees $ab|cd$, $ac|bd$, and $ad|bc$ are evaluated. That is, maximum likelihood values

$$L_{ab|cd}, L_{ac|bd}, L_{ad|bc} \tag{3.1}$$

or the corresponding log-likelihoods

$$\ell_{ab|cd}, \ell_{ac|bd}, \ell_{ad|bc} \tag{3.2}$$

are computed for each topology according to 2.2.3.

To take into account that two or even all three topologies can receive almost identical support, Strimmer *et al.* (1997) computed the Bayesian weights or posterior probabilities of the likelihoods

$$p_\tau = \frac{L_\tau}{L_{ab|cd} + L_{ac|bd} + L_{ad|bc}} \qquad \tau \in \{ab|cd, ac|bd, ad|bc\}. \tag{3.3}$$

For equally supported quartet trees they decided randomly which topologies to choose. The rational behind this strategy is as follows. If one topology τ is clearly supported its weight is $p_\tau \approx 1.0$ while the other two are approximately zero. Such a quartet is called *fully resolved*. In the case of a *partly resolved quartet*, that is, two topologies τ and τ' are equally well supported their weights are $p_\tau \approx p_{\tau'} \approx 0.5$. If all three topologies get almost equal support (*unresolved quartet*) each of the weights will be approximately one third.

Accordingly, for the 7 possible scenarios (3 three fully resolved, 3 partly resolved, and 1 unresolved state; cf. Figs. 3.1 and 3.3) a set of generic Bayesian weights ($w_{ab|cd}$, $w_{ac|bd}$, $w_{ad|bc}$) are used to discriminate the scenarios. The generic weights are chosen 1, $\frac{1}{2}$, or $\frac{1}{3}$ as shown in Figs. 3.1 and 3.3. The supported quartet topologies are selected according to the scenario with the least square distance

$$D = \sum_{\tau \in \{ab|cd, ac|bd, ad|bc\}} (p_\tau - w_\tau)^2 \tag{3.4}$$

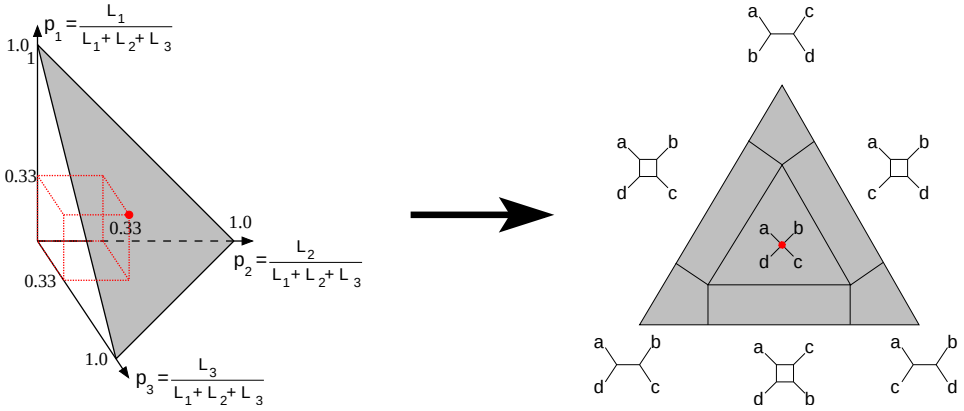


Figure 3.4.: How likelihood weights are plotted in a likelihood mapping diagram. Left side: likelihood weight plotted in a three-dimensional coordinate system. Right side: the simplex and its areas and the corresponding quartet topologies. The gray triangles are identical, only viewed from different angles.

between the generic weights w_τ and the posterior probabilities p_τ.

The posterior probabilities from Eq. 3.3 have the nice property of summing up to 1.0. This gave rise to a method called *likelihood mapping* (Strimmer and von Haeseler, 1997) that visualizes the quartet likelihoods. If three values which add up to 1.0 are plotted into a 3-dimensional coordinate system, all these points fall exactly into the triangle spanning between the coordinates $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. This triangle, also called a simplex, can be printed directly by changing the angle (see Fig. 3.4). The seven areas of the likelihood mapping diagram present the seven scenarios. Points in the corners show unambiguously supported quartet topologies. The rectangular areas display the partly resolved quartets, while the center comprises all unresolved quartets.

By plotting all or a large random subset of the quartets, *likelihood mapping* visualizes the phylogenetic information of a dataset. The more points fall into the middle of the diagram, the less suitable is the dataset for phylogenetic analysis. Moreover, by clustering the sequences into 2 to 4 clusters, the relationships and monophyly of these clusters can be examined. Each of the corners is assigned a certain cluster relationship, and the number of dots in that area shows the respective quartet support.

3.2.2. Quartet Puzzling

The tree reconstruction method, implemented in TREE-PUZZLE, first reads a multiple sequence alignment and estimates missing parameters according to the user specified model of evolution. The parameter estimation process will be described in chapter 5.

The actual tree reconstruction, the *Quartet Puzzling* algorithm, comprises three steps: (1) *maximum likelihood step* (*ML step*), (2) *puzzling step*, and (3) *consensus step* (Strimmer and von Haeseler, 1996; Strimmer *et al.*, 1997).

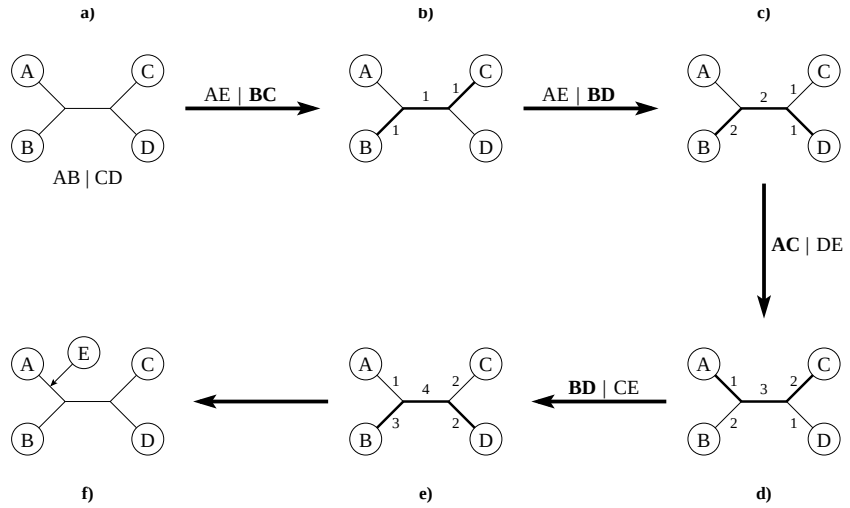


Figure 3.5.: Evaluation of the best insertion branch by penalizing. In this example $AE|BC$, $AE|BD$, $AC|DE$, and $BD|CE$ are the supported quartet topologies found in the previous step evaluating the quartets. The *penalty neighbors* are marked bold-face.

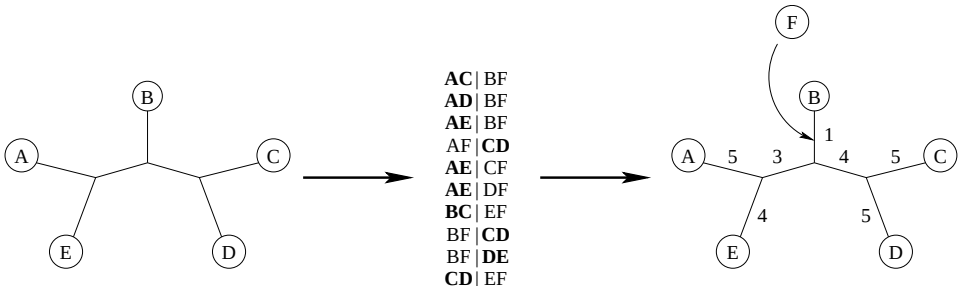


Figure 3.6.: Evaluation of the best insertion branch in a 5-tree by penalizing. The *penalty neighbors* in the quartet topologies are marked bold-face.

3.2.2.1. ML Step

In the *ML step*, the set of quartet topologies is computed which act as building blocks to reconstruct the overall tree topology. To do so, for each possible quartets $q \in \mathcal{Q}$ the three binary topologies are evaluated by ML (cf. 2.2.3). The supported topologies are chosen applying equations Eq. 3.3 and Eq. 3.4 (cf. 3.2.1). As described before, this approach also selects the second best or even all three quartet topologies if there is no clear support for a single topology (Strimmer *et al.*, 1997).

3.2.2.2. Puzzling Step

Starting from a supported topology of the first quartet $\{s_1, s_2, s_3, s_4\}$, sequences are added one after another as follows (Strimmer and von Haeseler, 1996; Strimmer *et al.*, 1997).

Assume X the next sequence to be added to the tree T_i that already contains the first i sequences. For all quartets of the type

$$q_X \in \{\, \{a, b, c, X\} \mid a, b, c \in \mathcal{L}(T_i),\ X \text{ to be inserted} \,\} = \mathcal{Q}_{(X,T_i)} \tag{3.5}$$

the supported topology is used to determine the optimal insertion point for sequence X (see Figs. 3.5 and 3.6 for examples). Assume $ab|cX$ the supported topology of a $q_X \in \mathcal{Q}_{(X,T_i)}$. Then for each such quartet $q_X \in \mathcal{Q}_{(X,T_i)}$ a penalty of 1 is added to every edge on the path between the two taxa a and b, which are not neighbors of X in the supported topology. This path between a and b is called *penalty path*, while a and b are called *penalty neighbors*. If more than one topology is supported for a quartet, one topology is chosen randomly. The sequence X is added to an randomly chosen edge $e \in \mathcal{E}_{min}$, where $\mathcal{E}_{min}$ is the set of edges with minimum penalty. The procedure of sequentially adding sequences is continued until an intermediate tree is reconstructed containing all n taxa.

The *puzzling step* itself is repeated several thousand times with different input orders.

3.2.2.3. Consensus Step

From the collection of all intermediate trees a majority-rule consensus tree is constructed, resolving all bipartitions which occur in more than 50% of the intermediate trees (for more details on consensus methods see 7.2.2.1 as well as McMorris and Neumann, 1983). The percent occurrence of the bipartitions among the intermediate trees gives a support value for the corresponding branch.

Finally, maximum likelihood branch lengths and the log-likelihood are computed for the consensus tree following the approach of Felsenstein (1981) as described in 2.2.3.

4. Improvement in Complexity of the Puzzling Step of QP

4.1. Introduction

Large growth rates of running times with increasing input size is a serious obstacle for the application of complex methods to large datasets.

When running time gets large more efficient algorithms are necessary. The enormous improvement that can be achieved by efficient algorithms has been impressively demonstrated in computer science textbooks, e.g., for sorting and searching algorithms (Cormen *et al.*, 2001; Knuth, 1997; Ottmann and Widmayer, 2002).

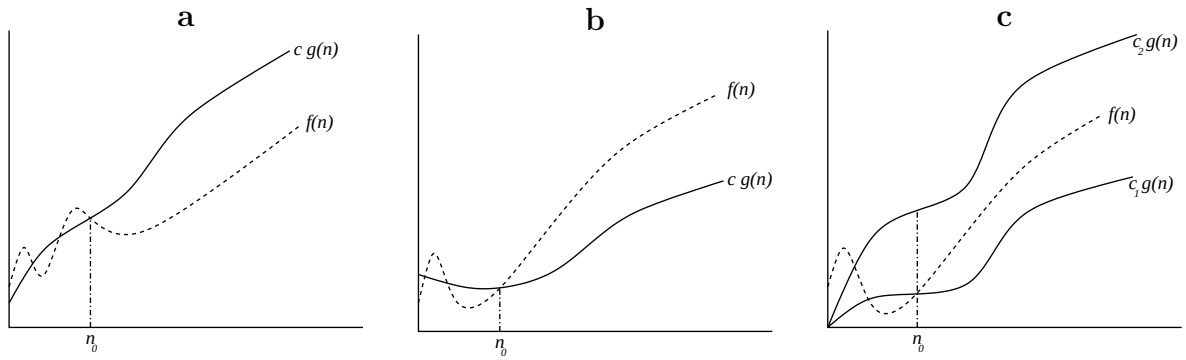
The efficiency of computer algorithms can be measured by running benchmarks on a computer to compare the performance of two algorithms. Benchmarks are a possible and necessary way to judge the efficiency of, e.g., a parallelized algorithm compared to the sequential version (cf. chapter 5). To measure the global efficiency of computer algorithms, benchmarks are by far too machine dependent. Hence, the general complexity of algorithms is scrutinized to reveal their efficiency. In case of algorithms with similar or equal complexity, benchmarks can help to find the preferable method.

In general two kinds of efficiency are investigated, the consumption of memory and runtime. Since one is interested in the consumption of resources for large input sizes, complexity measures have been introduced to describe the growth of computational and memory needs with respect to the input size.

This chapter is concerned with the complexity of the *puzzling step* introduced in chapter 3. For this purpose, the relevant complexity measures will be presented in the next section. We then will examine the complexity of the original *puzzling step*. Afterwards, three more efficient algorithms will be described, two novel ones and one suggested by Vincent Berry (Strimmer, personal communication). Finally, the runtime of all four algorithms will be compared and the difference will be discussed.

4.2. Complexity Measures

The complexity of an algorithm is usually described by asymptotic bounds using the O, Ω, and Θ-notation (Cormen *et al.*, 2001, chap. 3). The three notations are defined as follows.



$f(n) = O(g(n))$ $f(n) = \Omega(g(n))$ $f(n) = \Theta(g(n))$

Figure 4.1.: The three complexity notations O, Ω, and Θ

The *O-notation* describes an asymptotic upper bound of an algorithm and is defined by

$$\begin{aligned} O(g(n)) \quad = \quad \{f(n) : & \text{ there exist } c, n_0 \geq 0, \\ & \text{such that } 0 \leq f(n) \leq cg(n), \\ & \text{for all } n \geq n_0\} \end{aligned} \tag{4.1}$$

This equation means function $f(n)$ belongs to the set $O(g(n))$, if there exists a positive constant c, such that $cg(n)$ is an upper bound for $f(n)$ for all $n \geq n_0$ (Fig. 4.1a). $f(n)$ might, for example, describe the memory consumption of an algorithm.

The Ω-*notation* defines an asymptotic lower bound.

$$\begin{aligned} \Omega(g(n)) \quad = \quad \{f(n) : & \text{ there exist } c, n_0 \geq 0, \\ & \text{such that } 0 \leq cg(n) \leq f(n), \\ & \text{for all } n \geq n_0\} \end{aligned} \tag{4.2}$$

If there is a positive constant c, such that $cg(n)$ is a lower bound of $f(n)$ for all $n \geq n_0$ (Fig. 4.1b), $f(n)$ belongs to the set $\Omega(g(n))$.

The Θ-*notation* is defined as

$$\begin{aligned} \Theta(g(n)) \quad = \quad \{f(n) : & \text{ there exist } c_1, c_2, n_0 \geq 0, \\ & \text{such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \\ & \text{for all } n \geq n_0\} \end{aligned} \tag{4.3}$$

This means function $f(n)$ belongs to the set $\Theta(g(n))$, if there exist positive constants c_1, c_2, n_0, such that $f(n)$ can be 'sandwiched' between $c_1 g(n)$ and $c_2 g(n)$ for all $n \geq n_0$ (Fig. 4.1c). Note, if $f(n)$ belongs to $\Theta(g(n))$, this implies that it also belongs to $O(g(n))$ as well as to $\Omega(g(n))$.

O and Ω can be taken as the worst-case or best-case complexity, respectively.

Examples An algorithm that adds up all entries of an $(n \times n)$ matrix should be of order $\Theta(n^2)$, because every entry has to be touched exactly once.

Given a symmetric matrix, only $\frac{n^2+n}{2}$ entries, that is, one triangular part, have to be accessed. This, however, is still included in $\Theta(n^2)$.

Searching whether a zero contained in such a matrix, in worst case, all entries have to be touched once if there is no zero or one in the last entry. In the best case a zero is found in the first entry. Hence, such an algorithm should be of order $O(n^2)$ and $\Omega(1)$.

Most often the upper bound, denoted by the O-notation, is used to characterize an algorithm. Important growth rates for algorithms with respect to input size n are

$O(1)$	constant time (no growth dependent on input size n)
$O(\log n)$	logarithmic growth
$O(n)$	linear growth
$O(n \log n)$	$n \log n$ growth
$O(n^2)$	quadratic growth
$O(n^3)$	cubic growth
$O(n^x)$	polynomial growth
$O(x^n)$	exponential growth.

One algorithm is considered more efficient than another if it has lower a growth rate. For more details refer to computer science textbooks (e.g., Cormen *et al.*, 2001; Ottmann and Widmayer, 2002).

4.3. Complexity of the Original Puzzling Step

The original algorithm for the *puzzling step* (cf. 3.2.2) was implemented as described by Strimmer and von Haeseler (1996). A pseudo-code listing of the algorithm (Listing 4.1) will be used to examine its complexity. In the listing, the lines and loops important for the measurement of the complexity are numbered (e.g., 4.1c).

Length of Penalty Paths and the Diameter of a Tree Central to the original algorithm is the incrementation of the edge penalties (step 4.1c) on the path between two *penalty neighbors* a and b, the two leaves not being neighbors of the insertion taxon X in a preferred quartet topology (cf. 3.2.2 and Fig. 3.5).

The complexity of this step equals the length of the path between a and b, because direction vectors are used to look up the direction at any node to any other node in the tree. The entries of the vectors indicate directions to any internal or external node. The directions to the internal nodes are necessary to be able updating the direction to newly inserted edges easily. For the following it is assumed that tree T_i with i leaves has been constructed so far. The maximal path length between *penalty neighbors* is bounded by the *diameter of the tree* T_i, which is defined as the longest path between any two leaves of the tree.

Listing 4.1: Original Puzzling Step Algorithm

```
Data   : Set of taxa S = {s_1, s_2, ... s_n};
         Set of preferred quartet topologies for all quartets in S
Result : One binary tree T_n containing all n taxa in S
begin
    Set the sequences in S into a random order s_1, s_2, ... s_n;
    Set tree T_4 one preferred topology of {s_1, s_2, s_3, s_4};
    Initialize path directions in T_4;
4.1a for i = 4 to (n − 1) do
        X = s_{i+1};
4.1b    foreach quartet q_X ∈ Q_(X,T_i) do
            Let a and b be the penalty neighbors in the preferred topology of q_X;
4.1c        Add a penalty of 1 to all edge penalties Φ(ε) for all edges ε on the penalty
            path between a and b in T_i;
4.1d    Find the set of edges E_min(T_i) with minimal penalty;
        Insert X into one edge e ∈ E_min(T_i) ;
        Call the new tree T_{i+1};
4.1e    Update path directions in T_{i+1};
    return T_n;
end
```

The longest possible diameter for a tree with i leaves exists in a linearized tree topology (cf. Fig. 2.4d). Its diameter is exactly $i-1$, since all $i-3$ internal edges are on the path plus two external branches. Hence the upper bound for the diameter is $O(i)$.

The shortest possible diameter of a binary tree exists in a completely balanced tree (Fig. 2.4c). It is well-known (e.g., Ottmann and Widmayer, 2002) that the maximum *height* $h_{\max}$ of a balanced binary rooted tree (as in Fig. 2.4a) with i leaves is $h_{\max} \leq \lceil \log_2 i \rceil$, where the *height* h is defined as the number of edges between a node and the root. Hence, in this case the height h is bound by $\Theta(\log i)$. Therefore, the diameter of such rooted trees is twice its height, implying $\Theta(\log i)$ for the diameter. The same holds for the unrooted tree the diameter being one edge shorter because the central edge is not divided by a root (Fig. 2.4c). Accordingly, the complexity of step 4.1c incrementing the *penalty path* is bound by $O(i)$.

Runtime complexity Step 4.1c is repeated for each of the $\binom{i}{3}$ relevant quartets (loop 4.1b), leading to complexity $\binom{i}{3} \cdot O(i) \subset O(i^4)$ for loop 4.1b. Finding the set of minimal edges $\mathcal{E}_{min}(T_i)$ has complexity $O(i)$, since all $2n-3$ edges have to be examined (step 4.1d). After inserting X, the directions pointing to X and the new internal node have to be added by inserting the direction to the insertion edge in constant time per edge (step 4.1e. The direction vector of one of the adjacent nodes is copied to the newly inserted

node ($O(i)$). The linear complexity of this update (step 4.1d) is superseded by the $O(i^4)$ complexity of loop 4.1b. Since we add the n taxa sequentially (loop 4.1a), the overall runtime complexity of the original puzzling step algorithm is $O(n^5)$.

Memory complexity The memory needed for general purposes such as tree topologies and leaf set permutations are of linear complexity. This remains the same in all following algorithms and will, thus, not be considered any further. As mentioned above, direction vectors are used at all internal nodes to efficiently determine paths through the tree. The memory complexity of the direction vectors is $\Theta(n^2)$.

4.4. More Efficient Puzzling Step Algorithms

A main improvement in complexity compared to the original algorithm can be achieved by avoiding the tree traversal for each quartet during the insertion step.

Here two new algorithms are presented which reduce the complexity of the puzzling step. Both algorithms are based on a so-called *penalty neighborhood matrix* $\boldsymbol{N} = (N_{a,b})$ with $a, b \in \mathcal{L}(T_i)$, the leafset of T_i. The entries of $\boldsymbol{N}$ count the number of times two taxa are neighbors in the $\binom{i}{3}$ quartet tree topologies relevant to insert a next taxon X into T_i

$$N_{a,b} = |\{\, ab|cX \mid a, b, c \in \mathcal{L}(T_i)\}| \tag{4.4}$$

The two algorithms differ in the way the penalty $\boldsymbol{\Phi}(e)$ for an edge e is derived from the *penalty neighborhood matrix* $\boldsymbol{N}$.

Additionally, an algorithm suggested by Vincent Berry (Strimmer, personal communication)is presented that is based on the MRCA (most recent common ancestor) relationships of the *penalty neighbors*.

4.4.1. Split-Based Puzzling Step Algorithm

The following algorithm is sketched in Listing 4.2. To compute the penalty of an edge e only the sum of *penalty paths* passing through e induced by any two *penalty neighbors* on different sides of that edge have to be considered. The sum of all paths between any two *penalty neighbors* are computed in the *penalty neighborhood matrix* $\boldsymbol{N}$ (Eq. 4.4). Each incrementation has complexity $O(1)$ which is performed for the $\binom{i}{3}$ relevant quartets. Accordingly, computing $\boldsymbol{N}$ (loop 4.2b) has complexity $\Theta(i^3)$, since symmetrizing the matrix (loop 4.2c) needs $\Theta(i^2)$ assignments, which does not increase the complexity. $\boldsymbol{N}$ is symmetrized to avoid sorting every pair of leaves when accessing $\boldsymbol{N}$.

Listing 4.2: Split-based Puzzling Step

```
Data   : Set of taxa S = {s_1, s_2, ... s_n};
         Set of preferred quartet topologies for all quartets in S
Result : One binary tree T_n containing all n taxa in S
begin
    Set the sequences in S into a random order s_1, s_2, ... s_n;
    Set tree T_4 one preferred topology of {s_1, s_2, s_3, s_4};
    Initialize split vectors in T_4;
4.2a for i = 4 to (n − 1) do
        Clear (i × i) penalty neighborhood matrix N;
        X = s_{i+1};
4.2b    foreach quartet q_X ∈ Q_(X,T_i) do
            Let a = s_x and b = s_y (x < y) be the penalty neighbors in the preferred
            topology of q_X;
            Increment N_{a,b};
4.2c    foreach x < y ≤ i do N_{s_y,s_x} = N_{s_x,s_y};
4.2d    foreach edge e in T_i do
4.2e        foreach leaf a ∈ ⇂_e do
4.2f            foreach leaf b ∈ ↰_e do
                    Increment Φ(e) by N_{a,b};
4.2g    find the set of edges E_min(T_i) with minimal penalty;
        insert X into one edge e ∈ E_min(T_i);
        call the new tree T_{i+1};
4.2h    update split vectors in T_{i+1};
    return T_n;
end
```

The penalty $\mathbf{\Phi}(e)$ of an edge e is collected (loops 4.2e and 4.2f) summing up the joint penalties $N_{a,b}$ of each *penalty path* from a and b leading from any leaf a on the one side of e to any leaf b on the other side of e,

$$\mathbf{\Phi}(e) = \sum_{a \in \overrightarrow{\downarrow_e}} \sum_{b \in \overleftarrow{\downarrow_e}} N_{a,b}. \tag{4.5}$$

The computation of one $\mathbf{\Phi}(e)$ needs between $i-1$ for external edges and $\lceil (i/2)^2 \rceil$ additions, that means loops 4.2e and 4.2f together have complexity $O(i^2)$. The computation of the penalties for all $2i-3$ edges has overall complexity $O(i^3)$ (loop 4.2d).

Finding the set of minimal penalty edges $\mathcal{E}_{min}(T_i)$ (step 4.2g) is again of order $O(i)$, but can be collected while computing $\mathbf{\Phi}(e)$.

To be able to look up the splits at each edges, we use two split arrays of joint size n for each edge, which have to be updated once after an insertion (step 4.2h). After insertion we have to add the newly inserted taxon to the existing arrays in constant time per edge in a tree traversal, and all existing taxa have to be added to the cluster vectors at the two newly created edges again in $O(i)$.

Overall runtime complexity The highest complexity within the insertion loop 4.2a is $O(i^3)$ for the computation of all edge penalties as well as for the construction of the *penalty neighborhood matrix* $\boldsymbol{N}$, which supersede the other complexities. Since we add all taxa one by one, the algorithm's overall runtime complexity is $O(n^4)$.

Memory complexity In comparison to the original algorithm, the split-based algorithm requires additional memory for the *penalty neighborhood* $(n \times n)$*-matrix* $\boldsymbol{N}$ and cluster vectors with a joint size of n entries at every edge. The data structures both have complexity $\Theta(n^2)$. Hence, the memory complexity is equal for both algorithms. Note, the directional arrays of the original algorithm are not needed any more.

4.4.2. Recursive Puzzling Step Algorithm

As described above, the split-based algorithm reduces the complexity of the *puzzling step* by one order of magnitude. Yet further improvement can be achieved. While examining all $\binom{i}{3}$ relevant quartets is necessary for the insertion, computing the edge penalties can be solved more efficiently by using the following recursive approach (Listings 4.3 and 4.4).

The recursive algorithm again uses the *penalty neighborhood matrix* $\boldsymbol{N}$ which is constructed as before (Eq. 4.4). Additionally, the algorithm uses an *external node penalty vector* $\boldsymbol{E} = (E_a)$ containing the number of penalty paths starting at each leaf a. $\boldsymbol{E}$ is calculated as follows.

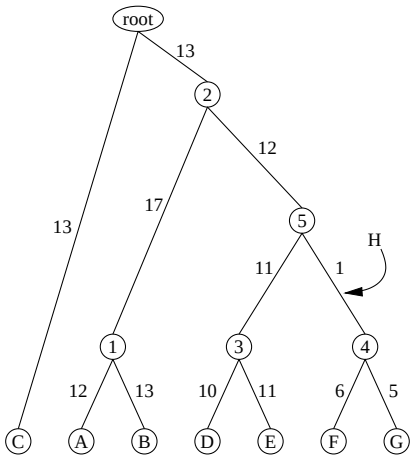
$$E_a = \sum_{b \neq a} N_{a,b} \qquad a, b \in \mathcal{L}(T_i) \tag{4.6}$$

Starting from $\boldsymbol{N}$ and $\boldsymbol{E}$, the edge penalties are computed via post-order traversal (calling the recursive function, Listing 4.4, in step 4.3d) in the recursive *puzzling step* algorithm (Listing 4.3). The tree T_i is assumed to be rooted arbitrarily at the external branch leading to s_1, the first sequence in the random permutation. Note, that the placement of the root does not change the result. Yet the root is necessary to introduce a starting point for the recursion.

Before analyzing the complexities of the algorithm sketched in Listings 4.3 and 4.4, the recursive procedure will be described in detail using an example.

Table 4.1.: Quartets $q_H \in \mathcal{Q}_{(H,T_7)}$ with $\mathcal{L}(T_7) = \{A, B, C, D, E, F, G\}$ (cf. Fig. 4.1) and their supported topologies for the example.

$\{A,B,C,H\}$	$AC\|BH^*$	$\{A,D,E,H\}$	$DE\|AH$	$\{B,C,G,H\}$	$BC\|GH$	$\{C,D,G,H\}$	$CD\|GH$
$\{A,B,D,H\}$	$AB\|DH$	$\{A,D,F,H\}$	$DF\|AH^*$	$\{B,D,E,H\}$	$DE\|BH$	$\{C,E,F,H\}$	$CE\|FH$
$\{A,B,E,H\}$	$AB\|EH$	$\{A,D,G,H\}$	$AD\|GH$	$\{B,D,F,H\}$	$BD\|FH$	$\{C,E,G,H\}$	$CE\|GH$
$\{A,B,F,H\}$	$AB\|FH$	$\{A,E,F,H\}$	$AE\|FH$	$\{B,D,G,H\}$	$BD\|GH$	$\{C,F,G,H\}$	$FG\|CH$
$\{A,B,G,H\}$	$AB\|GH$	$\{A,E,G,H\}$	$AE\|GH$	$\{B,E,F,H\}$	$BE\|FH$	$\{D,E,F,H\}$	$DE\|FH$
$\{A,C,D,H\}$	$AC\|DH$	$\{A,F,G,H\}$	$FG\|AH$	$\{B,E,G,H\}$	$BE\|GH$	$\{D,E,G,H\}$	$DE\|GH$
$\{A,C,E,H\}$	$AC\|EH$	$\{B,C,D,H\}$	$BC\|DH$	$\{B,F,G,H\}$	$FG\|BH$	$\{D,F,G,H\}$	$FG\|DH$
$\{A,C,F,H\}$	$AC\|FH$	$\{B,C,E,H\}$	$BC\|EH$	$\{C,D,E,H\}$	$DE\|CH$	$\{E,F,G,H\}$	$FG\|EH$
$\{A,C,G,H\}$	$AC\|GH$	$\{B,C,F,H\}$	$BC\|FH$	$\{C,D,F,H\}$	$CD\|FH$		



edge e	$CLP(C^e(e))$	$SMS(C^e(e))$	$\mathbf{\Phi}(e)$	cluster $C^e(e)$
$e_{(1,A)}$	12	0	12	$\{A\}$
$e_{(1,B)}$	13	0	12	$\{B\}$
$e_{(root,C)}$	13	0	13	$\{C\}$
$e_{(3,D)}$	10	0	10	$\{D\}$
$e_{(3,E)}$	11	0	11	$\{E\}$
$e_{(4,F)}$	6	0	6	$\{F\}$
$e_{(4,G)}$	5	0	5	$\{G\}$
$e_{(2,1)}$	25	8	17	$\{A,B\}$
$e_{(root,2)}$	57	44	12	$\{A,B,D,E,F,G\}$
$e_{(5,3)}$	21	10	11	$\{D,E\}$
$e_{(5,4)}$	11	10	1	$\{F,G\}$
$e_{(2,5)}$	32	20	12	$\{D,E,F,G\}$
root	70	70	0	$\{A,B,C,D,E,F,G\}$

Figure 4.2.: Edge penalties and penalty sums *SMS* and *CLP* computed recursively on the tree T_7 with $\boldsymbol{N}$ and $\boldsymbol{E}$ (Eq. 4.7) derived from the quartet topologies in Tab. 4.1.

4.4.2.1. Example

One insertion step of the recursive *Puzzling Step* algorithm will be exemplarily performed on the tree T_7 in Fig. 4.2 using the quartet topologies from Tab. 4.1.

According to Eqs. 4.4 and 4.6 the *penalty neighborhood matrix* $\boldsymbol{N}$ and its row sums, i.e., the *external node penalty vector* $\boldsymbol{E}$ are computed.

$$\boldsymbol{N} = \begin{array}{c} \begin{array}{ccccccc} \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} \end{array} \\ \begin{pmatrix} - & 4 & 5 & 1 & 2 & 0 & 0 \\ 4 & - & 4 & 2 & 2 & 1 & 0 \\ 5 & 4 & - & 2 & 2 & 0 & 0 \\ 1 & 2 & 2 & - & 5 & 0 & 0 \\ 2 & 2 & 2 & 5 & - & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & - & 5 \\ 0 & 0 & 0 & 0 & 0 & 5 & - \end{pmatrix} \end{array} \begin{array}{c} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \\ \text{F} \\ \text{G} \end{array} \qquad \boldsymbol{E} = \begin{array}{c} \begin{pmatrix} 12 \\ 13 \\ 13 \\ 10 \\ 11 \\ 6 \\ 5 \end{pmatrix} \end{array} \begin{array}{c} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \\ \text{F} \\ \text{G} \end{array} \tag{4.7}$$

Using E_a, which contains all penalties induced by *penalty paths* of leaf a, the penalty of the external edge $e_{(\bullet,a)}$ of leaf a can be set

$$\mathbf{\Phi}(e_{(\bullet,a)}) = E_a. \tag{4.8}$$

Climbing to an inner edge $e_{(\bullet,v)}$ that descends to an inner node v, the edge penalty $\mathbf{\Phi}(e_{(\bullet,v)})$ can be computed as follows. All *penalty paths* originating at any leaf of the edge's cluster $C^e(e_{(\bullet,v)})$ are added. The sum is called *cluster leaf penalty CLP*.

$$CLP(C^e(e_{(\bullet,v)})) = \sum_{a \in C^e(e_{(\bullet,v)})} E_a \tag{4.9}$$

Note, that all *penalty paths* that start and end within the cluster are counted twice. To obtain the true edge penalty we have to subtract all *penalty paths* remaining inside the cluster twice. This number is given by the *submatrix sum* of $\boldsymbol{N}$

$$SMS(C^e(e_{(\bullet,v)})) = \sum_{k_1 \in C^e(e_{(\bullet,v)})} \sum_{k_2 \in C^e(e_{(\bullet,v)})} N_{k_1,k_2}. \tag{4.10}$$

Consider edge $e_{(2,5)}$ from Fig. 4.2. The edge's cluster consists of the union of the descending edges' clusters, i.e., $C^e(e_{(2,5)}) = C^e(e_{(5,3)}) \cup C^e(e_{(5,4)})$ (cf. table in Fig. 4.2). Hence, $CLP(C^e(e_{(2,5)}))$ is efficiently computed by adding the *CLP*s of the descending branches, avoiding double calculations. Also $SMS(C^e(e_{(2,5)}))$ is computed more efficiently than summing up the whole submatrix of $\boldsymbol{N}$ colored grey in Eq. 4.7. The light grey parts have already been computed for the descendant edges. Thus, only one of the dark grey submatrices has to be summed up, because of $\boldsymbol{N}$ being symmetric:

$$\begin{aligned} SMS(C^e(e_{(2,5)})) &= SMS(C^e(e_{(5,3)})) + SMS(C^e(e_{(5,4)})) \\ &+ 2 \cdot \sum_{a \in C^e(e_{(5,3)})} \sum_{b \in C^e(e_{(5,4)})} N_{a,b} \end{aligned} \tag{4.11}$$

Finally, by subtracting *SMS* from *CLP* the edge penalty

$$\mathbf{\Phi}(e_{(2,5)}) = CLP(C^e(e_{(2,5)})) - SMS(C^e(e_{(2,5)})). \tag{4.12}$$

of edge $e_{(2,5)}$ is computed.

Listing 4.3: Recursive Puzzling Step

Data : Set of taxa $\mathcal{S} = \{s_1, s_2, \ldots s_n\}$;
Set of preferred quartet topologies for all quartets in $\mathcal{S}$
Result : One binary tree T_n containing all n taxa in $\mathcal{S}$
begin
  Set the sequences in $\mathcal{S}$ into a random order $s_1, s_2, \ldots s_n$;
  Set tree T_4 one preferred topology of $\{s_1, s_2, s_3, s_4\}$;
  Root T_{i-1} arbitrarily at the external edge of s_1, then called e_{root};
  Initialize cluster vectors in T_4;
4.3a **for** $i = 4$ ***to*** $(n-1)$ **do**
    Clear $(i \times i)$ *penalty neighborhood matrix* $\boldsymbol{N}$;
    Clear *external node penalty i-vector* $\boldsymbol{E}$;
    $X = s_{i+1}$;
4.3b **foreach** *quartet* $q_X \in \mathcal{Q}_{(X,T_i)}$ **do**
      Let $a = s_x$ and $b = s_y$ $(x < y)$ be the *penalty neighbors* in the preferred topology of q_X;
      Increment $N_{a,b}$;
4.3c **foreach** $x < y \leq i$ **do**
      $N_{s_x,s_y} = N_{s_y,s_x}$;
      Increment E_{s_y} by N_{s_y,s_x};
      Increment E_{s_x} by N_{s_y,s_x};
4.3d $\boldsymbol{\Phi}(e_{root}) =$ SubtreePenalty(e_{root}, T, $\boldsymbol{N}$, $\boldsymbol{E}$);
    if $\boldsymbol{\Phi}(e_{root}) \neq E_{s_1}$ **then error**
4.3e Find the set of edges $\mathcal{E}_{min}(T_i)$ with minimal penalty;
    Insert X into one edge $e \in \mathcal{E}_{min}(T_i)$ with minimal penalty;
    Call the new tree T_{i+1};
4.3f Update cluster vectors in T_{i+1};
  return T_n;
end

4.4.2.2. Complexity

Runtime Complexity The recursive algorithm again uses a neighborhood matrix $\boldsymbol{N}$ which is built and symmetrized the same way as before in $\Theta(i^3)$ (loop 4.3b). Symmetrizing $\boldsymbol{N}$ and computing the row sums $\boldsymbol{E}$ (loop 4.3c) has complexity $\Theta(i^2)$. External edge penalties $\boldsymbol{\Phi}(e_{(\bullet,a)}) = E_a$ are assigned in constant time for each of the i external edges (condition 4.4a).

Ascending to an internal edge ε the penalty $\boldsymbol{\Phi}(\varepsilon)$ is computed by adding the sums of the *penalty paths CLP* of the clusters $C^e(e_{(\bullet,l)})$ and $C^e(e_{(\bullet,r)})$ of the descending edges $e_{(\bullet,l)}$ and $e_{(\bullet,r)}$ (step 4.4e) in $O(1)$. From this sum the penalties (steps 4.4b to 4.4d) induced by paths among the members of the cluster $C^e(\varepsilon)$ of edge ε has to be subtracted (step

Listing 4.4: SubtreePenalty($e_{(v,w)}$, T, $\boldsymbol{N}$, $\boldsymbol{E}$)

Data : Current edge $e_{(v,w)}$ leading away from root, ending at node w;
Binary tree topology T;
Penalty Neighborhood Matrix $\boldsymbol{N}$;
External Node Penalty Vector $\boldsymbol{E}$

Result : Edge penalty $\boldsymbol{\Phi}(e_{(v,w)})$ of edge $e_{(v,w)}$;
Cluster $C^n(w)$;
Sum of paths $CLP(C^n(w))$ starting at leaves in cluster $C^n(w)$;
Sum of paths $SMS(C^n(w))$ within cluster $C^n(w)$

begin

4.4a **if** w = *external node* **then**
  $\boldsymbol{\Phi}(e_{(v,w)}) = E_w$;
  $CLP(\{w\}) = E_w$;
  $SMS(\{w\}) = 0$;
  return $\boldsymbol{\Phi}(e_{(v,w)}), C^n(w), CLP(C^n(w)), SMS(C^n(w))$;
else
  l = left descendant of w;
  $(\boldsymbol{\Phi}(e_{(w,l)}), CLP(C^n(l)), SMS(C^n(l)))$ =SubtreePenalty($e_{(w,l)}$, T, $\boldsymbol{N}$, $\boldsymbol{E}$);
  r = right descendant of w;
  $(\boldsymbol{\Phi}(e_{(w,r)}), CLP(C^n(r)), SMS(C^n(r)))$ = SubtreePenalty($e_{(w,r)}$, T, $\boldsymbol{N}$, $\boldsymbol{E}$);
4.4b $SMS(C^n(w)) = 0$;
4.4c **foreach** $a \in C^n(r)$ **do**
    foreach $b \in C^n(l)$ **do**
      Increase $SMS(C^n(w))$ by $N_{a,b}$;
4.4d $SMS(C^n(w)) = 2 \cdot SMS(C^n(w)) + SMS(C^n(l)) + SMS(C^n(r))$;
4.4e $CLP(C^n(w)) = CLP(C^n(l)) + CLP(C^n(r))$;
4.4f $\boldsymbol{\Phi}(e_{(v,w)}) = CLP(C^n(w)) - SMS(C^n(w))$;
  return $\boldsymbol{\Phi}(e_{(v,w)}), CLP(C^n(w)), SMS(C^n(w))$;

end

4.4f), an operation with complexity $O(i^2)$ executed once per edge. Although this step still has complexity $O(i^2)$ it reduces the runtime as described above. Finding the set of minimal penalty edges $\mathcal{E}_{min}(T_i)$ (step 4.2g) costs again $O(i)$, but can be collected immediately while computing $\boldsymbol{\Phi}(e)$.

After inserting the next leaf, the newly inserted inner edge has to get an initial cluster by copying it from the insertion edge in $O(i)$ (step 4.3f). In addition, the inserted leaf has to be added to the clusters of the edges on the path from the insertion edge to the root in $O(i)$ (step 4.3f).

The most expensive parts in the insertion loop 4.3a have complexity $O(i^3)$ per insertion, namely, the construction of $\boldsymbol{N}$ as well as computing the submatrix sums *SMS* recursively for all edges (SubtreePenalty, Listing 4.4, called in line 4.3d). Since we add all taxa one after the other, the overall complexity of the recursive algorithm is $O(n^4)$.

Memory complexity In comparison to the original algorithm, the recursive algorithm requires additional memory for the *penalty neighborhood* $(n \times n)$*-matrix* $\boldsymbol{N}$, the *external node penalty* n*-vector* $\boldsymbol{E}$, and a vector of n entries per edge to store the cluster of the edge. Hence, the recursive algorithm has memory complexity of $\Theta(n^2)$ as well.

4.4.3. Berry's MRCA-Based Puzzling Step Algorithm

Another $O(n^4)$ algorithm has been proposed by Vincent Berry (Strimmer, personal communication). Berry's algorithm calculates the edge penalties using the MRCA relations of the *penalty neighbors* and, hence, is referred to as MRCA-based *puzzling step* algorithm. In this algorithm, all *penalty paths* between two leaves a and b are collected in an n-vector at their MRCA. The entries of the vectors are then propagated down to the respective leaves in a pre-order traversal of the rooted tree (calling the recursive function Listing 4.6 in step 4.5f).

The Algorithm As in the recursive algorithm the tree T_i is rooted arbitrarily at the external branch leading to s_1. Again, the placement of the root does not change the result, but it is needed to introduce directions to define the MRCAs. In the beginning, the first tree and the MRCA matrix is set up in constant time. To add a sequence X to a previously built tree T_i, all $\binom{i}{3}$ relevant quartets containing X and three leaves already in T_i are examined (loop 4.5c). Let a and b be the *penalty neighbors* of such a quartet, then the entries of the *penalty propagation vector* $\boldsymbol{V}$ at the inner node MRCA(a, b) for the nodes a and b are incremented (loop 4.5c). Afterwards, the entries of the *penalty propagation vectors* an inner node are propagated down to the corresponding leaves in a pre-order traversal (called in step 4.5f). All entries of the propagation vector $\boldsymbol{V}$ are added to the entries at the descendant node's vector $\boldsymbol{V}$, if the corresponding leaf belongs to the cluster of this descendant node (steps 4.5d, 4.6a, and 4.6c). The penalty $\boldsymbol{\Phi}(e)$ of an edge e is computed by summing up all entries of $\boldsymbol{V}$ attached to the parental node of e which correspond to a leaf in the cluster $C^e(e)$ of e (steps 4.5e, 4.6b, and 4.6d). The taxon X is added at an edge with minimum penalty. (step 4.5g).

Runtime Complexity To initially fill the *penalty propagation vectors* requires evaluating all relevant quartets and, thus, has complexity $O(i^3)$ (loop 4.5c). Propagating penalties from one node to the other (steps 4.5d, 4.6a, and 4.6c) has complexity $O(i)$ as has the computation of the penalty of an edge (steps 4.5e, 4.6b, and 4.6d), due to the number of possible leaves in a cluster. Since the propagation is performed recursively on

Listing 4.5: MRCA-based Puzzling Step

Data : Set of taxa $\mathcal{S} = \{s_1, s_2, \ldots s_n\}$;
Set of preferred quartet topologies for all quartets in $\mathcal{S}$
Result : One binary tree T_n containing all n taxa in $\mathcal{S}$
begin
Set the sequences in $\mathcal{S}$ into a random order $s_1, s_2, \ldots s_n$;
Set tree T_4 one preferred topology of $\{s_1, s_2, s_3, s_4\}$;
Root T_4 arbitrarily in the external edge to s_1;
Initialize MRCA $(n \times n)$-matrix M for the rooted T_4;
4.5a **for** $i = 4$ ***to*** $(n-1)$ **do**
4.5b Clear MRCA penalty $(i-3 \times i-3)$-matrix $\boldsymbol{V}$;
$X = s_{i+1}$;
4.5c **foreach** *quartet* $q_X \in \mathcal{Q}_{(X,T_i)}$ **do**
Let s_x and s_y $(x < y)$ be the *penalty neighbors* in the preferred topology of q_X;
Increment $V_{s_x,\mathrm{MRCA}(s_x,s_y)}$;
Increment $V_{s_y,\mathrm{MRCA}(s_x,s_y)}$;
$l = \mathrm{succ}(root) \backslash s_1$;
4.5d **foreach** $d \in C^n(l)$ **do**
Increase $V_{d,l}$ by $V_{d,root}$;
4.5e Increase $\boldsymbol{\Phi}(e_{(\bullet,l)})$ by $V_{d,l}$;
4.5f $\boldsymbol{\Phi}$ = PropagatePenalties($e_{(\bullet,l)}$, $\boldsymbol{V}$, $\boldsymbol{\Phi}$);
4.5g Find the set of edges $\mathcal{E}_{min}(T_i)$ with minimal penalty;
Add X to one edge $e \in \mathcal{E}_{min}(T_i)$ with minimal penalty;
Call the new tree T_{i+1};
4.5h Update MRCA matrix for X with all leaves in $\mathcal{L}(T_i)$;
4.5i Update cluster vectors in T_{i+1};
return T_n;
end

all edges, the whole calculation of the $2i-3$ edge penalties has complexity $O(i^2)$. The set of minimal edges (loop 4.5g) is collected during the traversal in $O(1)$ at each edge.

After inserting X into T_i, X has to be added to the clusters of the nodes (step 4.5i) between the newly inserted inner node and the root (complexity $O(i)$). Also, X has to be added to the MRCA matrix (step 4.5h). The update follows the nodes between the newly inserted taxon and the root, where the current node is the MRCA of X and any leaf in the cluster of the current edge which was not in the cluster of the previous inner node. Since the clusters are known for the right and left descendant of any inner node and only i entries have to be added, the update has complexity $O(i)$. The largest complexity during one insertion step is $O(i^3)$ examining all relevant quartets (loop 4.5c),

Listing 4.6: PROPAGATEPENALTIES($e_{(\bullet,w)}$, $\boldsymbol{V}$, $\boldsymbol{\Phi}$)

```
Data   : Current edge e_(•,w) leading away from root, ending at node w;
         Tree topology T;
         MRCA penalty matrix V
Result : Edge penalty Φ(e_(•,w)) of edge e_(•,w)
begin
    l = left descendant of w;
4.6a | foreach a ∈ C^n(l) do
     |     Increase V_{a,l} by V_{a,w};
4.6b |     Increase Φ(e_(w,l)) by V_{a,l};
4.6c | r = right descendant of w;
     | foreach b ∈ C^n(r) do
     |     Increase V_{b,r} by V_{b,w};
4.6d |     Increase Φ(e_(w,r)) by V_{b,r};
     | if r internal node then PROPAGATEPENALTIES(e_(w,l), V, Φ);
     | if l internal node then PROPAGATEPENALTIES(e_(w,r), V, Φ);
end
```

which makes the overall runtime (for loop 4.5a) complexity $O(n^4)$.

Memory complexity The data structures maintained by the algorithm are the MRCA matrix with $\Theta(n^2)$ and the propagation vectors $\boldsymbol{V}$ at all inner nodes with $\Theta(n^2)$. Hence, the memory complexity of this algorithm is again $\Theta(n^2)$ as in all other *puzzling step* algorithms proposed in this chapter.

4.5. Runtime Analysis

All improved algorithms presented here have the same reduced runtime complexity of $O(n^4)$ while keeping the memory complexity at $\Theta(n^2)$. To measure the performance improvement of the different algorithms, benchmark tests have been performed for the original as well as the improved algorithms.

4.5.1. Datasets

To test the runtime performance of the implementation of the four different *puzzling step* algorithms, we used a variety of datasets with different numbers of taxa.

Datasets of different sizes were used: two biological amino acid datasets of elongation factors, the first consisting of size 24 (EF-24 Schmidt and von Haeseler, 2003) with 915

Table 4.2.: Results of the runtime analysis of the four different *puzzling step* algorithms: the original algorithm, the MRCA-based, split-based, and recursive modifications. The best runtime is marked bold-face.

dataset	seqs.	length	original	MRCA-based		split-based		recursive	
			t_{orig}	t_{MRCA}	$\frac{t_{MRCA}}{t_{orig}}$	t_{split}	$\frac{t_{split}}{t_{orig}}$	t_{recur}	$\frac{t_{recur}}{t_{orig}}$
EF-24	24	915aa	32s	20s	0.63	20s	0.63	**19s**	0.59
EF-50	50	1117aa	1159s	610s	0.53	589s	0.51	**557s**	0.48
GST-50	50	1065aa	1191s	582s	0.49	567s	0.48	**537s**	0.45
Sim-32-500	32	500nt	154s	91s	0.59	89s	0.58	**84s**	0.55
Sim-32-1000	32	1000nt	154s	91s	0.59	89s	0.58	**83s**	0.54
Sim-48-500	48	500nt	894s	477s	0.53	460s	0.51	**438s**	0.49
Sim-48-1000	48	1000nt	887s	472s	0.53	455s	0.51	**432s**	0.49
Sim-64-500	64	500nt	3146s	1614s	0.51	1541s	0.49	**1475s**	0.47
Sim-64-1000	64	1000nt	3100s	1548s	0.50	1480s	0.48	**1415s**	0.46

aligned columns and the other comprising 1117 aligned sites from 50 sequences (EF-50). As a third biological dataset cluster 63525 from the SYSTERS database (Version 3; Krause *et al.*, 2002, `http://systers.molgen.mpg.de/`) was used, consisting of 50 glutathione S-transferase proteins with 1065 aligned columns (GST-50).

Additionally, simulated datasets with different numbers of taxa (32, 48, 64) as well as different sequence lengths (500nt, 1000nt) were used (for more details on the simulated datasets refer to 5.3).

4.5.2. Benchmark Setup

To get even conditions, the initial random number seeds in any run were fixed, such that the runs only differed in the construction of the intermediate trees by the algorithms, but not in the choices made concerning quartet topologies or insertion edges. In each benchmark run 10,000 intermediate trees were computed. The benchmarks were performed several times and the runtimes were averaged and rounded to the next full second.

The benchmarks were computed on a LINUX workstation with an AMD Athlon XP 2100+ CPU and 512MB memory.

Since storing the splits is done at the end of each single puzzling step, a portion of the runtime is unaffected by the *puzzling step* algorithms. It was not possible to determine the exact split maintenance times, since – especially for small tree sizes – these are near or even below the resolution of the hardware clocks used for the measurements.

4.5.3. Results

The results of the benchmark tests (Tab. 4.2) show a clear reduction of the runtime by the three $O(n^4)$ algorithms of about 50% compared to the runtime of the original $O(n^5)$ algorithm. The best reduction was achieved by the recursive algorithm with 59-45% of the original algorithm's runtime. It is followed by the split-based algorithm with 63-48% runtime and then by Berry's MRCA-based algorithm with 63-49%. Although, the percentages are sometimes close to each other, the algorithms end up clearly ordered by their runtimes, namely the MRCA-based algorithm, the split-based algorithm, and the recursive algorithm, ordered by decreasing runtime for all benchmark datasets.

4.6. Discussion

The results in Tab. 4.2 convincingly demonstrate the positive effect of using more efficient algorithms. As described in 4.5.3 all $O(n^4)$-algorithms show reduced runtimes of about 50% compared to the original algorithm for all benchmark datasets. Furthermore, the three improved algorithms show a clear order of their runtimes. Although the implementation of the MRCA-based algorithm substantially reduced the runtime it was outperformed by the split-based as well as the recursive *puzzling step* algorithm. As expected, the split-based algorithm show longer runtime compared to the recursive implementation, which is due to the computational overhead calculating the edge penalties from the *penalty neighborhood matrix* $\boldsymbol{N}$ (cf. 4.4.2).

In particular, if we consider the operations add, assign, multiply, and subtract, computing the calculation of $SMS(C^e(e_{(2,5)}))$ (Eq. 4.11) in the example on page 32 needs four fix operations and four variable operations within the sums (variable means that the number depends on the cluster sizes). The calculation of $CLP(C^e(e_{(2,5)}))$ needs two fix operations. Two more operations are needed to compute $\boldsymbol{\Phi}(e_{(2,5)})$ (Eq. 4.12). In the end eight fix operations and four variable ones are needed, compared to twelve variable operations in the split-based algorithm. Note, that variable operations take longer time due to the counters and comparisons within the addition loops. Hence, although the numbers of operations counted are equal in this case a speedup can be gained. Additionally, a lot of dereferencing is avoided. The difference increases dramatically in larger trees. For instance, the penalty of an edge joining two clusters of size 5 and 6 in a tree with 30 leaves, needs 30 variable and again 8 fix operations, compared to $19 \cdot 11 = 209$ variable operations in the split-based algorithm.

Furthermore, the results from the simulations also show that the sequence lengths have only marginal effect on the runtimes measured. This is obvious, since the *puzzling step* is based on the ML quartet topologies. The fluctuation between the 500nt and 1000nt datasets are mainly to be attributed to a lower degree of information in the shorter sequences. Subsequently, more different intermediate trees are reconstructed from the less resolved quartets. This increases the number of occurring splits which leads to a slightly higher overhead maintaining the increased number of splits.

The results also show that the percentage of the remaining runtime decreases for growing numbers of taxa, which is due to the difference of one order of magnitude between the still polynomial complexities. The improved runtime complexity decreases the runtime of all datasets analyzed, and thus, enables the analysis of large datasets in less time.

5. Parallelized Quartet Puzzling

5.1. Introduction

The often exponential complexities of the applications lead to the use of heuristics. However, even heuristics sometimes do not succeed in reducing the runtime sufficiently to enable the analysis of large datasets within reasonable time. Therefore, parallel computing techniques emerged as valuable tools to reduce the runtime of applications.

Parallel computing has first been introduced to pairwise alignment algorithms for sequence comparison (Edmiston and Wagner, 1987; Lander *et al.*, 1988; Huang, 1989). Since the early 1990s, it has been applied to phylogenetic analysis (Hagstrom *et al.*, 1992; Olsen *et al.*, 1994, fastDNAml) by parallelizing the DNAML program (Felsenstein, 1981). Until today parallel processing has entered most areas of modern sequences analysis like database searching (Jülich, 1995; Pedretti *et al.*, 1999), RNA structure prediction (Nakaya *et al.*, 1995; Shapiro *et al.*, 2001), multiple sequence alignment (Kleinjung *et al.*, 2002), and recently also contig assembly (Ness *et al.*, 2002), and gene selection for tissue classification (Liu *et al.*, 2001).

Quite a variety of parallel platforms and techniques exist. Among the first parallel computers were vector computers which contain an array of processing elements (PEs) that are able to simultaneously execute the same instruction (SIMD computers – single instruction multiple data). More abundant nowadays are MIMD computers (multiple instructions multiple data) that are able to compute different tasks simultaneously on the different PEs/CPUs. The main types of MIMD platforms are multi-processor *shared-memory computers*, also known as *SMP* (scalable multi-processor), *distributed memory computers*, and *clusters of workstations* (*COWs*). While all CPUs of a *shared-memory computer* have access to a global (shared) memory, CPUs in distributed memory computers all have their own 'private' memory and communicate via a fast interconnection device. A *COW* comprises a number of workstations via a local network (e.g., Ethernet). Employing workstations with multiple processors, characteristics of both SMP and distributed memory platforms are combined.

Two main parallel programming paradigms exist, namely, *shared memory* and *message passing*. *Shared-memory* programming makes use of the shared memory either by accessing the same copy of data and variable with all processes or by running several subprocesses (threads) on the same program code in memory or both. In *message passing*, the processes communicate via explicit messages over the interconnection facilities. While the former is clearly restricted to shared memory architectures, the latter can be

implemented on all platforms mentioned. Thus, *message passing* gains high flexibility and portability. For more details see the review by Trelles (2001) on the use of parallel computing in bioinformatics and parallel platforms.

Although a number of parallelized phylogenetic programs have been published to date, mainly the DNAML program (Felsenstein, 1981) for the ML based analysis of DNA data was subject to parallelization attempts (Olsen *et al.*, 1994; Schmidt, 1996; Trelles *et al.*, 1998; Stewart *et al.*, 2001; Stamatakis *et al.*, 2002).

This chapter presents the efficient parallelization of the *Quartet Puzzling* algorithm implemented in TREE-PUZZLE (cf. chapter 3) employing Message Passing Interface (MPI, Snir *et al.*, 1998; Gropp *et al.*, 1998), a highly portable *de facto* standard in parallel programming. In the next section, the default parameter estimation of TREE-PUZZLE will be explained. The runtime profile of the sequential program will be analyzed, and the resulting parallelization scheme will be presented. The scalability of the parallel program will be tested, and the results will be discussed with respect to other parallelized ML-based tree reconstruction methods. Finally, a short outlook on further enhancements will be given.

5.2. Parameter Estimation for Evolutionary Models

As explained in 2.2.3, ML tree reconstruction methods require a model of the evolutionary process. The parameters that specify these substitution models need to be estimated from the data sets.

To account for the fact that positions in an alignment may evolve with different rates, each site may also get its own rate specific factor (cf. 2.1.4). Typically one assumes that rates are distributed according to a Γ-distribution (Gu *et al.*, 1995; Yang and Kumar, 1994), where one parameter α describes the amount of rate heterogeneity. Also, the shape parameter α must be estimated from the data.

The collection of parameters can either be specified by the user or can be estimated directly from a multiple sequence alignment. The parameter estimation is one of the challenging problems in maximum likelihood tree reconstruction. As for phylogenetic analysis in general, it is assumed that incorporating more data increases the accuracy of the estimates. In TREE-PUZZLE the following heuristic estimation algorithm is implemented (Strimmer, 1997).

5.2.1. Algorithm: Estimating Model Parameters

A single step of the heuristic estimation algorithm is described in Listing 5.1. With an initial or already optimized parameter set M a distance matrix $\boldsymbol{D}$ and from $\boldsymbol{D}$ a neighbor-joining tree T_{nj} (Saitou and Nei, 1987) with branch lengths is inferred. With T_{nj} a new estimate of M is computed. The ML estimation is performed as explained in

Listing 5.1: Estimating Model Parameters in TREE-PUZZLE

Data : $\mathcal{S}$ = a set of n aligned sequences
Result : Model parameters M
begin
  Assign biologically reasonable initial values to M;
  repeat
5.1a Estimate the ML distance matrix $\boldsymbol{D}$ for all $\frac{n(n-1)}{2}$ sequence pairs as described by Adachi and Hasegawa (1996b);
5.1b Construct a neighbor-joining tree T_{nj} from $\boldsymbol{D}$ (Saitou and Nei, 1987);
5.1c Fit the distances in $\boldsymbol{D}$ to T_{nj} by optimizing its branch lengths with the least squares method;
5.1d On the tree with fixed branch lengths compute the maximum likelihood estimate of parameters M;
  until *estimate of* M *does not change any more*;
  return M;
end

2.2.3, but assuming the tree T_{nj} fixed and optimizing the parameters of M to maximize the likelihood. This procedure is repeated until the estimate of M does not change any more.

The single estimation step as described in Listing 5.1 is repeated alternately for the parameters of the evolutionary model and rate heterogeneity parameters until the parameters set M converges or a maximal number of repetitions has been performed.

5.3. Datasets

For the analytic tree reconstructions as well as the benchmark tests we use simulated datasets as well as a biological one.

We generated balanced phylogenetic trees with 32, 48, and 64 taxa. For these trees alignments of length 500 and 1000 nucleotides (nt) where generated using Seq-Gen (version 1.2.6 Rambaut and Grassly, 1997) and assuming the HKY model (Hasegawa *et al.*, 1985) and Γ-distributed rates with $\alpha = 1.0$ introducing intermediate heterogeneity among sites (Gu *et al.*, 1995). To make the results of the datasets comparable, the tree lengths from root to the leaves were set to be 0.2 substitutions per site.

One might argue that using balanced trees influences the runtime behavior of the algorithm. Certainly, the runtimes of the parts differ relative to each other, but using other tree topologies shows similar behavior. Hence, balanced trees are chosen because of their simple scalability.

Table 5.1.: Runtime profile in percent of total runtime for ML parameter estimation.

Taxa n	length	distance matrix `mldistance`	least square fit `lslength`
32	500 nt	90.4 %	2.6 %
32	1000 nt	90.7 %	1.4 %
48	500 nt	88.7 %	6.6 %
48	1000 nt	90.9 %	3.5 %
64	500 nt	85.1 %	11.3 %
64	1000 nt	88.5 %	7.4 %
50	289 aa	85.1 %	> 0.1 %

In addition to the simulated data, 50 glutathione transferase sequences of the cluster 63,525 from the SYSTERS database (Version 3; Krause *et al.*, 2002, `http://systers.molgen.mpg.de/`) were used as a biological dataset. The alignment of the glutathione transferase proteins has a length of 289 amino acids (aa).

5.4. Runtime Analysis of the Sequential TREE-PUZZLE

Runtime profiling (Tabs. 5.1 and 5.2) has been performed on a variety of datasets, simulated as well as biological data to measure the time requirement of the different parts in the tree reconstruction. This is done to locate the parts which demand large portions of the overall runtime and, thus, are a prominent target for parallelization. Here the results are presented for those datasets which are used later for the benchmarks (for details, cf. section 5.7).

5.4.1. Runtime of the Parameter Estimation

Runtime profiling (Tab. 5.1) shows that the repeated computation of the $\frac{n(n-1)}{2}$ pairwise distances in step (5.1a) accumulates about 85-91% of the total computation time during parameter estimation. The second largest part with up to 11.3% calculates the least square branch lengths (step 5.1c) which performs $(2n-3)(n-1)$ independent computations per call.

5.4.2. Runtime of the Quartet Puzzling Algorithm

In the *QP* algorithm, the *ML step* and the *puzzling step* are by far most time-consuming (Tab. 5.2). Both steps consist of many independent tasks, the computation of trees for $\binom{n}{4}$ quartets and up to 50,000 computations of the intermediate trees. This makes both steps a prominent target for parallelization.

Table 5.2.: Runtime measurements for the different components of TREE-PUZZLE.

Taxa n	length	parameter estimation	number of quartets	*ML step*	# *puzzling steps*	*puzzling step*	*consensus step*
32	500 nt	6.87 %	35,960	51.40 %	10,000	41.05 %	0.68 %
32	1000 nt	8.45 %	35,960	65.14 %	10,000	25.84 %	0.56 %
48	500 nt	3.05 %	191,878	52.29 %	10,000	44.43 %	0.21 %
48	1000 nt	3.61 %	191,878	69.44 %	10,000	26.72 %	0.23 %
64	500 nt	0.96 %	628,649	29.83 %	25,000	68.99 %	0.21 %
64	1000 nt	1.27 %	628,649	43.15 %	25,000	55.53 %	0.05 %
50	289 aa	1.32 %	230,300	84.55 %	10,000	13.51 %	0.62 %

5.5. Parallelizing TREE-PUZZLE

For portability of the code as well as applicability to different parallel platforms including clusters of workstations (COWs), which are most common in molecular biology (Trelles, 2001), we decided to apply the message passing paradigm using the Message Passing Interface standard (MPI, Snir *et al.*, 1998; Gropp *et al.*, 1998). Master/worker setups in combination with dynamic scheduling algorithms are applied to gain flexibility and independence from the underlying parallel platform. The choice of the applied scheduling algorithms is discussed in section 5.6.

Due to serious differences in granularity (= the size of the independently computable units in the sequential implementation) between the parameter estimation and the *Quartet Puzzling* algorithm both parts are optimized separately.

5.5.1. Parallelizing the Parameter Estimation

To reduce computation time, we applied a master/worker setup for both steps (Fig. 5.1). The master process constructs the neighbor-joining trees (5.1b) and re-estimates the parameters (5.1d), while the computations of steps (5.1a) and (5.1c) are distributed among the worker processes.

5.5.2. Parallelizing Quartet Puzzling

Again a master/worker scheme is used. The master process keeps track of the quartets to be evaluated during the *ML step* and dispatches batches of tasks to the worker processes (Figure 5.1). To allow efficient transfer of the reconstructed quartet tree, the atomic batch size has to be a multiple of two since the preferred topologies of one quartet is stored in 4 bit.

In the *puzzling step* the master process tells the workers for how many permutations of taxa an intermediate tree needs to be reconstructed (Figure 5.1). The random permutations are based on SPRNG (`http://sprng.cs.fsu.edu/`), a parallel random number

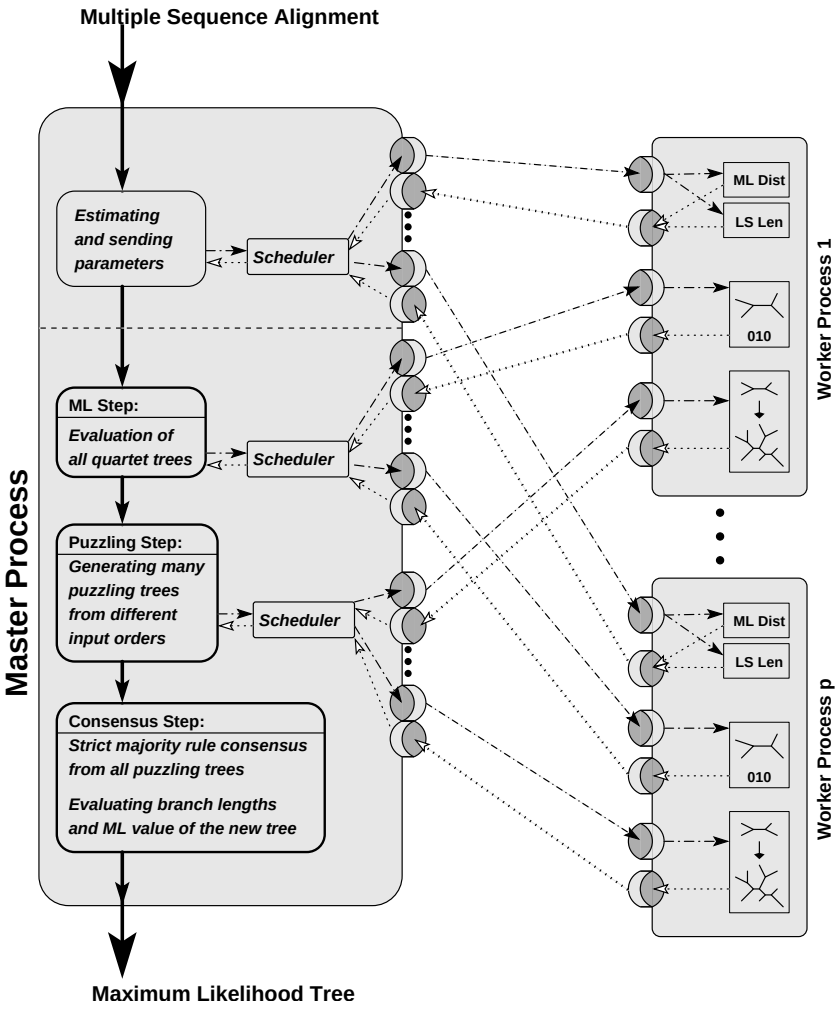


Figure 5.1.: Parallelized workflow of the TREE-PUZZLE program. The upper part of the master process controls the initialization and the parameter estimation where the least square branch length fitting (LS len) and ML distance estimation (ML dist) is performed by the worker processes. The lower part presents the parallel *Quartet Puzzling* algorithm.

generator, to avoid dependencies of the workers' generators. A list of all splits of the intermediate trees together with their absolute frequencies is returned to the master. The number of splits occurring during the *puzzling steps* depends on the quality of the data. Datasets with little phylogenetic information produce a higher number of different intermediate trees and, hence, a larger number of splits.

5.6. Scheduling Algorithms

To ensure an even work load of the worker processes, we applied dynamic scheduling algorithms (El-Rewini *et al.*, 1994). Dynamic scheduling algorithms are important on

heterogeneous COWs to produce a dynamically balanced work load. The effect of different scheduling algorithms on communication overhead and distribution of work load on the p worker processors is investigated on a heterogeneous COW comprising SUN workstations ranging from SPARC 20 to ULTRA 5.

For the parallelization scheme displayed in Figure 5.1 we tested algorithms which neither require knowledge about the network or allocation delays nor estimates of expected runtime. The following algorithms were evaluated. The notation follows Hagerup (1997) where the current status of the scheduler is denoted by Ξ and the return value is the size of the next batch to schedule:

- Self-Scheduling: $SS(\Xi) = 1$
- Static Chunking: $SC(\Xi) = \frac{NT}{NP}$
- Guided Self-Scheduling: $GSS(\Xi) = \frac{NR}{NP}$
- Trapezoid Self-Scheduling: $TSS(\Xi)$, TSS starts with batch size $\frac{NT}{2NP}$ decreasing linearly to a final value of 1,

where NT is the total number of tasks, NR is the number of tasks not yet scheduled, and p denotes the number of worker processes.

To test the performance of the scheduling algorithms, we ran the TREE-PUZZLE program several times with different scheduling algorithms on the heterogeneous SUN COW and analyzed the according GANTT-chart diagrams produced by the program. Since the presentation of the GANTT-charts is not possible we summarize the results.

Although the 'naive' approach, *self scheduling* (SS), which sends all tasks one-by-one, is able to produce a homogeneous work load, the communication and the subsequent overhead are very high which leads to delays when sending task batches and results.

Static chunking (SC) performs nicely if all processors work at the same speed and if the runtime of the tasks does not vary. In practice this is unrealistic for most parts of TREE-PUZZLE. SC has the advantage of minimizing communication, the batch of a worker can even easily be pre-computed. On heterogeneous COWs, however, the slowest processor becomes a bottleneck for the runtime of the whole set of tasks. Due to the very small computation times per call in the least square branch length inference of step (5.1c), the communication overhead has to be kept as small as possible. Hence, precomputed batch sizes of $\frac{NT}{NP}$, i.e. *static chunking*, are used. Only the initial data values are sent to the workers, and the results are then sent to the master process.

In preliminary tests, best performing were such scheduling algorithms which start with a large batch size, decreasing towards the end, like *trapezoid self scheduling* (TSS) and *guided self scheduling* (GSS). On heterogeneous COWs, GSS produced the most even workload. However, we observed that slowest processors became a bottleneck for the runtime, if they received the first (largest) batch of $\frac{NT}{NP}$.

Hence, we modified the GSS algorithm to schedule batch sizes of $GSS(\Xi) = \frac{NR}{2NP}$, to reduce the possible waste of computation time. The modified GSS algorithm was applied to schedule the computational more expensive pairwise distance computations in step (5.1a).

To apply GSS efficiently to the computationally most demanding parts of the program, *ML step* and *puzzling step*, an additional adaptation was necessary. An atomic batch size was introduced due to coding specificities of 4 bit per quartet to store the preferred quartet topologies from the *ML step* (cf. 5.5), to make efficient transmission of the results possible. The atomic size can also be used to keep the batch size at a certain level to reduce communication when only a few tasks are left to schedule.

Based on the modified GSS, the adaptation used for distributing the tasks of the *ML step* and the *puzzling step* determined the batch size by the formula

$$SGSS(\Xi) = \begin{cases} NR & \text{if } NR < ABS, \\ \left\lceil \frac{NR}{2NP} \right\rceil + ABS - \left(\left\lceil \frac{NR}{2NP} \right\rceil \bmod ABS \right) & \text{if } \left(\left\lceil \frac{NR}{2NP} \right\rceil \bmod ABS \right) > 0 \\ \left\lceil \frac{NR}{2NP} \right\rceil & \text{otherwise.} \end{cases} \quad (5.1)$$

The second line of Eq. 5.1 rounds to the next multiple of the atomic size ABS, if necessary. The first line schedules the rest NR if already smaller than the atomic size ABS. The adaptation produces a smoothed distribution of batch sizes, therefore it is referred to as *smooth guided self scheduling* (SGSS).

For *ML step* and *puzzling step* of the *QP* algorithm, we determined an atomic size of 4 tasks to be a good tradeoff between a desirably small atomic size and the reduction of communication overhead.

5.7. Efficiency of Parallel TREE-PUZZLE

When analyzing the efficiency of parallelized algorithms the scalability of the parallelized code with increasing numbers of processors is relevant. The scalability was evaluated on simulated and biological data for benchmarking.

5.7.1. Benchmark Datasets and Setup

To analyze the efficiency of the parallel implementation, we used all datasets described in 5.3.

For each dataset parallel TREE-PUZZLE was applied to obtain benchmarks for 2, 4, 8, 16, 32, 48, and 64 processors on a Cray T3E-1200 parallel computer. Note, we use processes and processors interchangeably because the queuing system guaranteed

that one processor executes exactly one process. Due to the local setup, the sequential version of TREE-PUZZLE did not run with the same priority as parallel jobs. Thus, the speedup is compared to a 2 processor run, i.e. 1 master and 1 worker process. The abscissa in the diagrams of Figures 5.3 to 5.5 shows the number of worker processors.

5.7.2. Results for Parameter Estimation

Fig. 5.2 shows that the parallel implementation of the parameter estimation with the modified GSS algorithm in step (5.1a) and SC in step (5.1c) does not scale for large numbers of processors. Although all calls of steps (5.1a) and (5.1c) accumulate over 90% of the running time (Tab. 5.1), every single call is very fast. The parallelized part is then interrupted by a sequential part. Due to this fine granularity, the result is not surprising. However, a slight increase in scalability can be observed with respect to taxa and sequence length. For the parallel estimation of parameters for datasets with $n > 200$ taxa the gain of runtime reduction is substantial.

The biological dataset with 50 taxa and 289 amino acids (aa) fits well with simulated data for 48 taxa and 1000 nt. The high computation time of the shorter aa dataset is attributed to the increased size of the alphabet (20 aa to 4 nucleotides).

5.7.3. ML Step and Puzzling Step

While the scalability of the parameter estimation is dissatisfactory, the parallel *ML step* (Fig. 5.3) and the parallel *puzzling step* (Fig. 5.4) show an almost perfect speedup indicating a very good performance of the parallelization strategy. Concerning the biological dataset the speedup is almost perfect in the *ML step*, but decreases in the *puzzling step* with growing numbers of processors.

While the runtime for the biological dataset is highest in *ML step*, the sequence length has no influence on the runtime of the *puzzling step* because only the quartet tree topologies are used here.

5.7.4. Overall Scaling of Parallel TREE-PUZZLE

Despite the disappointing scalability of the parallel parameter estimation and the additional sequential *consensus step*, the overall runtimes of the TREE-PUZZLE runs still show convincing scalability (Fig. 5.5). The scaling increases with growing numbers of taxa, and for 64 sequences of length 1000 nt the speedup is almost perfect. Although this could not be shown with the protein dataset used increasing the number of sequences should bring a better speedup for protein dataset as well.

As in the *ML step* and the parameter estimation, the computation time consumed by the non-parallelized *consensus step* increases from DNA to amino acid data because of the higher alphabet size (Fig. 5.3, Tab. 5.2).

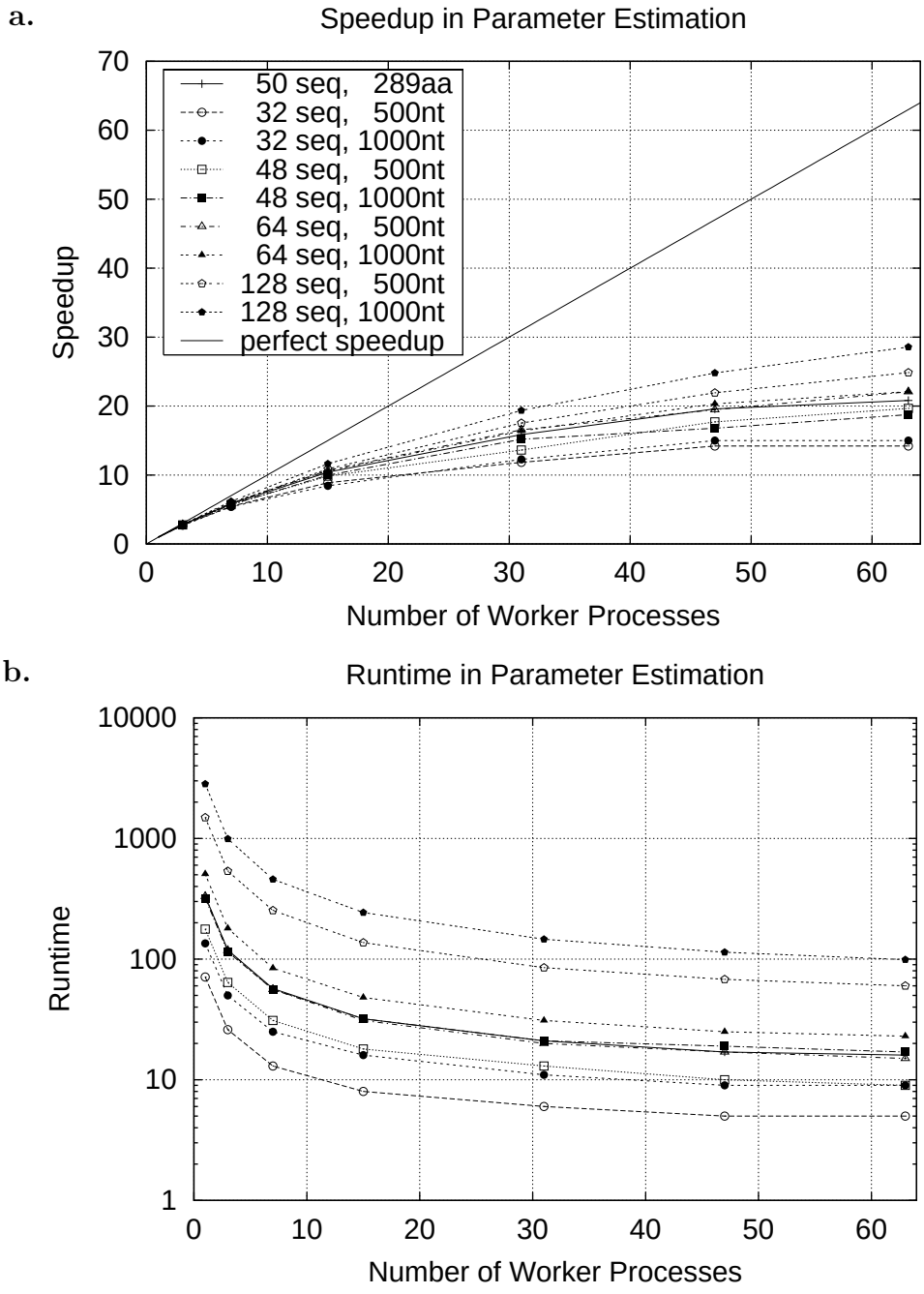


Figure 5.2.: Speedup (a) and runtime(b) of the parameter estimation for the different datasets.

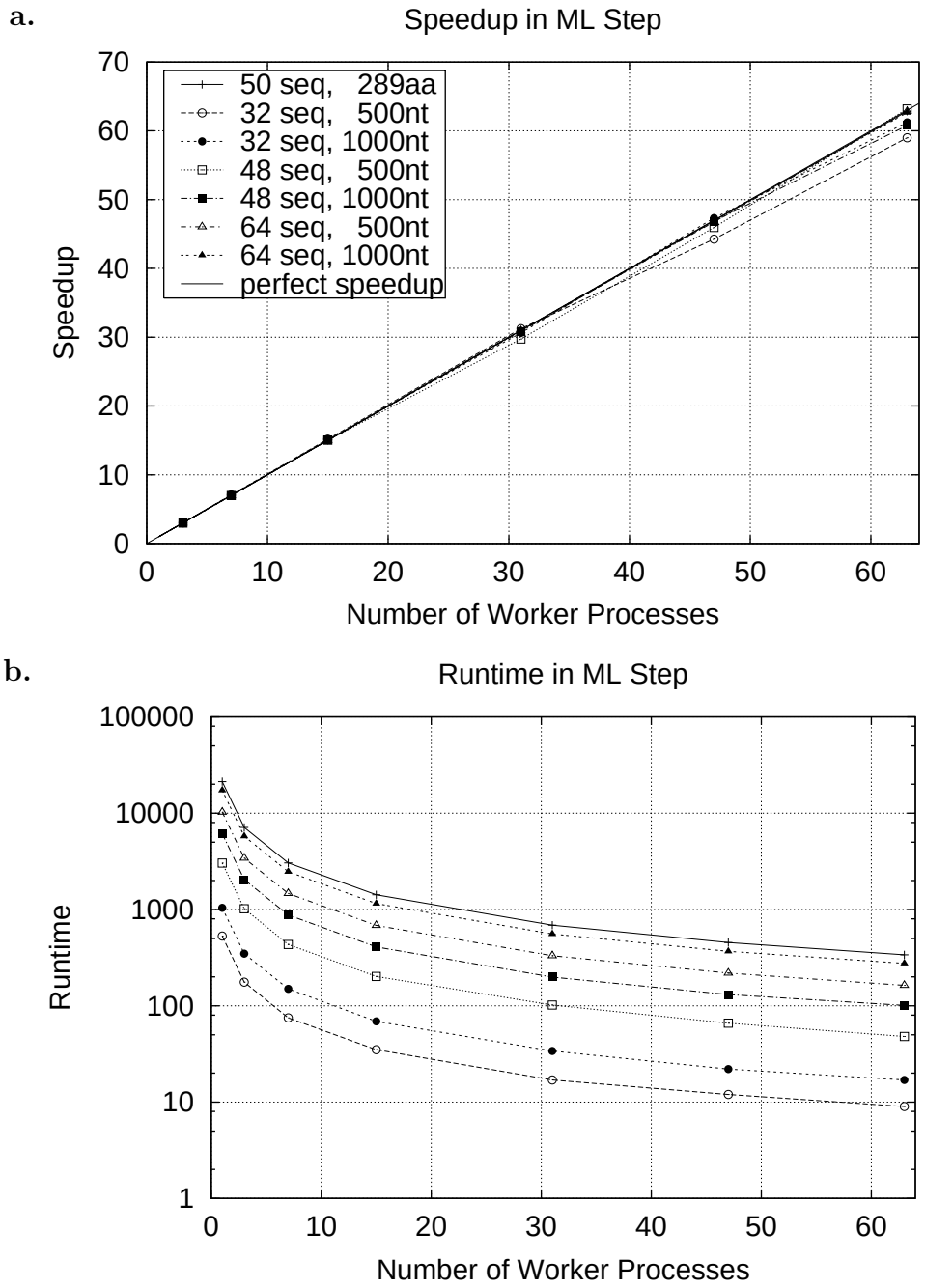


Figure 5.3.: Speedup (a) and runtime (b) of the *maximum-likelihood step* of *Quartet Puzzling* for the different datasets.

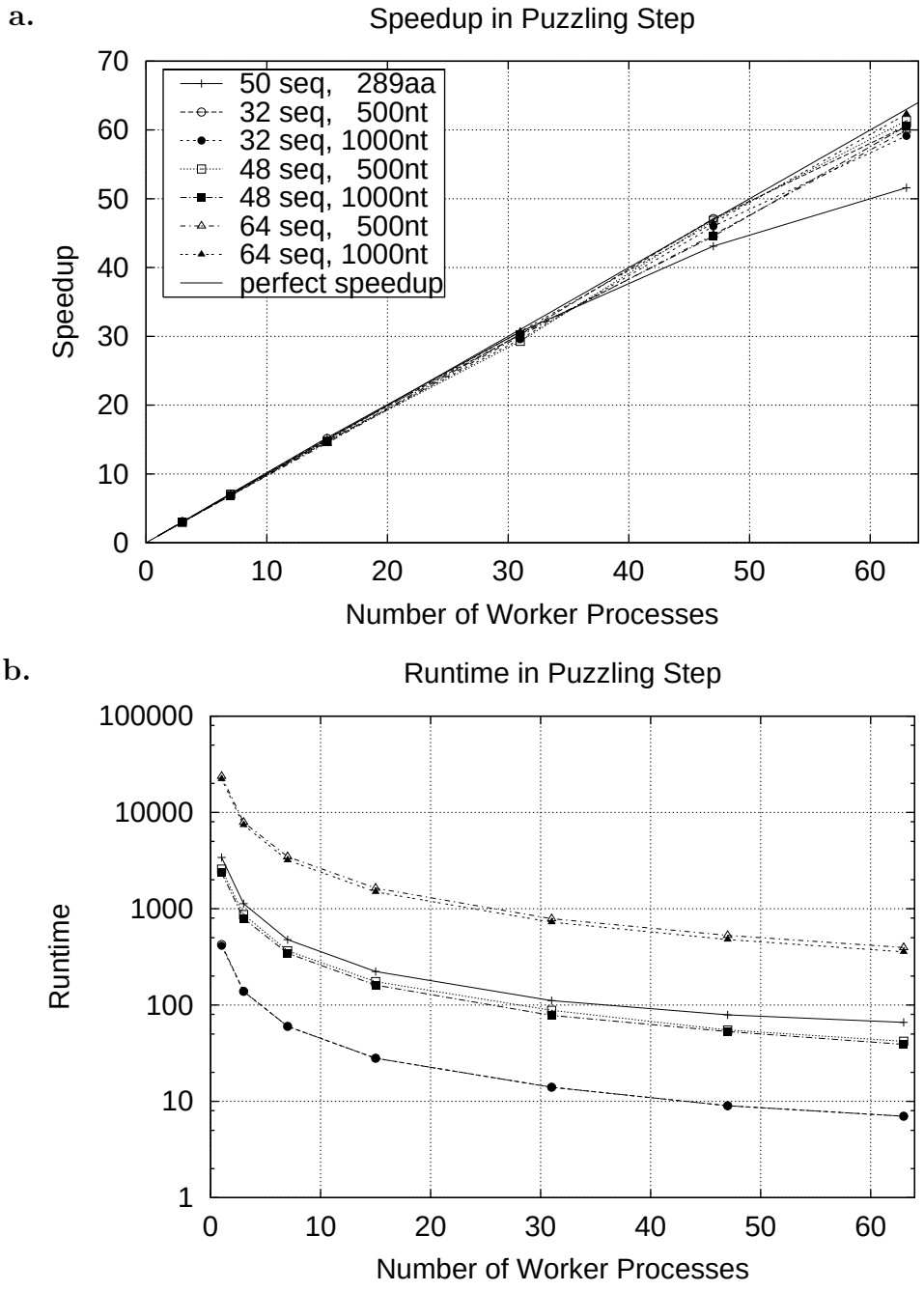


Figure 5.4.: Speedup (a) and runtime (b) of the *puzzling step* of *Quartet Puzzling* for the different datasets.

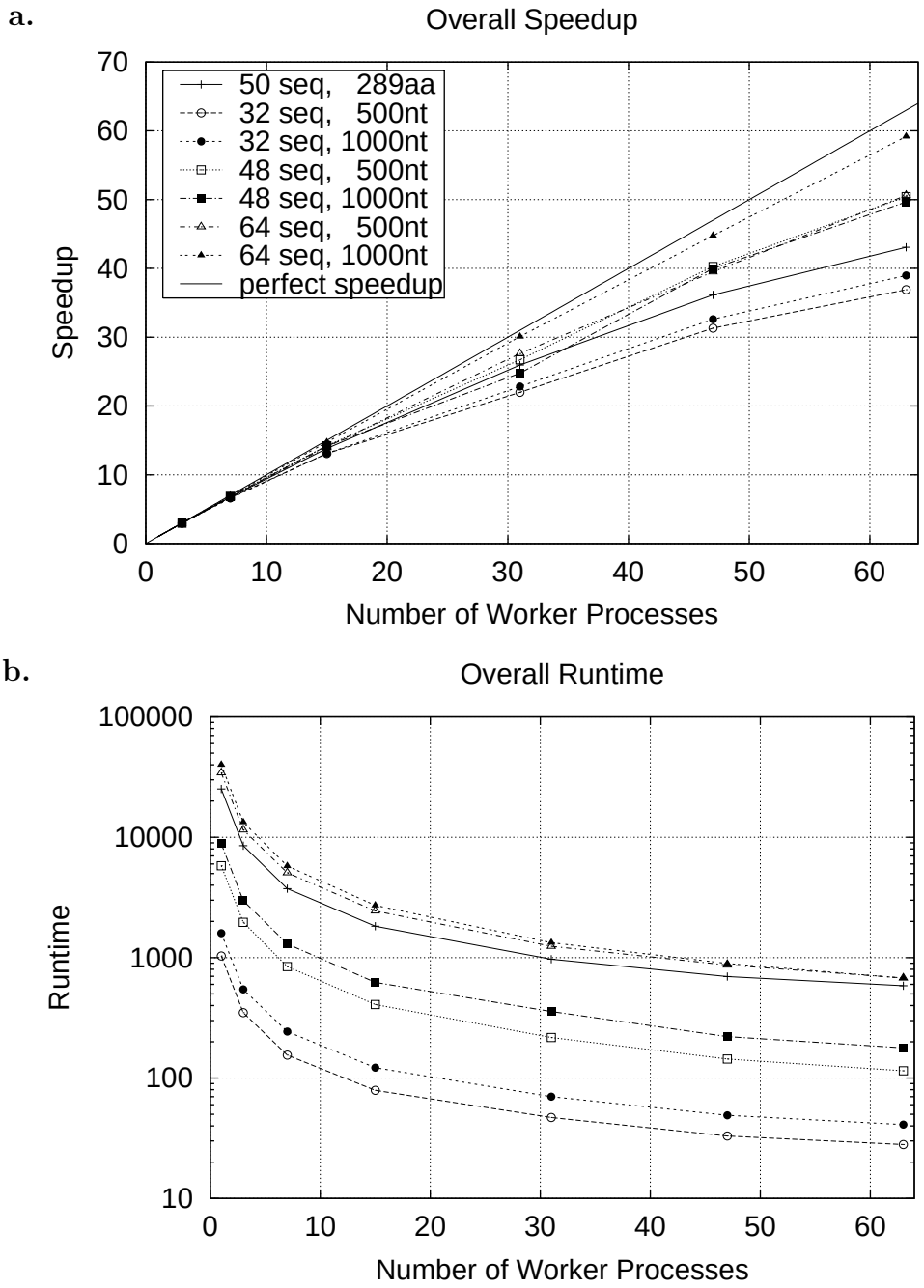


Figure 5.5.: Overall speedup (a) and runtime (b) of the parallelized TREE-PUZZLE program for the different datasets.

5.8. Discussion

To compute ML trees, it is necessary to estimate the underlying model parameters. This chapter illustrates a first attempt to speed up the estimation procedure by parallelizing the estimation heuristic implemented in TREE-PUZZLE. The combination of message passing with SC and a modified GSS leads to reduced computation time. However, the speedup is not even close to perfect. A factor of 17 for 32 processors will nevertheless enable biologists to study large datasets. For example, the estimation of evolutionary parameters and rate heterogeneity parameters for 215 rRNA sequences of length 2100 nt took only 8.6 minutes on a homogeneous 32-processor COW instead of 2.5 hours on a single processor. This reduction in computation time makes TREE-PUZZLE also a worthwhile stand-alone application to estimate evolutionary parameters. Thus in the future, sub-sampling strategies for parameter estimation (Meyer *et al.*, 1999; Meyer and von Haeseler, 2003) may no longer be necessary for moderately sized datasets. This is a first step to develop more efficient and scalable algorithms to support large phylogeny reconstructions.

The amount of data available naturally leads to large tree reconstruction projects, e.g., to reconstruct a tree for each of the 25,000 multi-sequence protein families in the SYSTERS database (Version 3, Krause *et al.*, 2002). For the smaller families the trees can be reconstructed using the sequential TREE-PUZZLE. However, for large families ($n \geq 100$ sequences) the parallelized version might be an alternative, because the presented parallelization provides for the first time a parallel tree reconstruction approach for amino acid sequences.

In the past the DNAML program (Felsenstein, 1981) was subject to parallelization attempts. Several parallel implementations were published showing moderate to well-scaling behavior (Olsen *et al.*, 1994; Schmidt, 1996; Trelles *et al.*, 1998; Stewart *et al.*, 2001; Stamatakis *et al.*, 2002).

This chapter presented the parallel version of the *QP* algorithm which is an integral part of the TREE-PUZZLE package (Schmidt *et al.*, 2002; Strimmer and von Haeseler, 1996). The parallelized *QP* shows convincing scalability. One advantage of the parallel TREE-PUZZLE is its simplicity. The master/worker setup with one master process for the scheduling is sufficient to handle all tasks and communication. No additional processes like foreman or dispatcher processes in the *fastDNAml* implementations (Olsen *et al.*, 1994; Stewart *et al.*, 2001) are necessary.

Since *Quartet Puzzling* is the only parallelized ML-based reconstruction method for protein phylogenies so far, it is a major contribution for the ML analysis of large protein datasets.

Although we have made considerable progress, more work needs to be done. The main drawback in scalability remains with the merging of the split lists from the workers into a consensus tree and the computation of ML branch lengths of the consensus tree. How to parallelize the computation of ML branch lengths remains an open problem.

6. Large ML Trees from Sequences Using Quartets

6.1. Introduction

As described before, the *ML step* and the *puzzling step* are the two steps demanding most of the runtime in the *Quartet Puzzling* procedure (a detailed runtime analysis of QP has been presented in 5.4). In chapter 4 we demonstrated that the complexity of the *puzzling step* can be reduced by one order of magnitude using an efficient $O(n^4)$ algorithm, which leads to substantial speedup. However, the complexity $\Theta(n^4)$ of the *ML step* cannot be reduced, since the *QP* algorithm requires the evaluation of all $3\binom{n}{4}$ possible quartet topologies. For datasets of 200 sequences or more the computation time required to evaluate all quartets is very large. For example, the evaluation of all quartet topologies for a red algae dataset of 215 sequences from the European small subunit (ssu) rRNA database (Van de Peer *et al.*, 2000b) assuming rate heterogeneity would take more than 6,500 hours on a SUN Enterprise 450 server.

Hence, a strategy is suggested that does not require the computation of all quartet trees dividing the set of sequences $\mathcal{S}$ into subsets. In order to ensure that our algorithm works, a minimal amount of pair-wise overlap, i.e., the number of sequences shared by the different subsets of the alignment is required. Once the quartet trees are computed, we proceed with a modified version of the *puzzling step* (cf. 3.2.2). An overall consensus tree is constructed from a series of so-called intermediate trees produced in the modified *puzzling step* algorithm. We show, that the resolution of the consensus tree critically depends on the number of quartets actually evaluated.

The outline of the chapter is as follows, in section 6.2 we briefly introduce the relevant additional notation followed by the explanation of the modified *Quartet Puzzling* algorithm for large datasets. The subsequent section gives some insights about the correlation between the resolution of the tree and the amount of computing time spend. The final section discusses applications and gives an outlook on further modifications of the proposed method.

6.2. The Modified Quartet Puzzling Algorithm

The *Quartet Puzzling* algorithm requires the computation of $\binom{n}{4}$ quartets, which is often not feasible if n is large. This chapter introduces a modified version of the algorithm

by computing only quartets for a collection of subsets $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \ldots, \mathcal{S}_k$ of $\mathcal{S}$. Then $\mathcal{Q}_g$ describes the set of all $\binom{|\mathcal{S}_g|}{4}$ quartets from $\mathcal{S}_g$ for all $g = 1, 2, \ldots k$. We require that

$$\bigcup_{i=1}^{k} \mathcal{S}_i = \mathcal{S}, \tag{6.1}$$

$$|\mathcal{S}_i| \geq 4, \text{ for } i = 1, \ldots, k \tag{6.2}$$

$$\left| \left\{ \bigcup_{j=1}^{i} \mathcal{S}_j \right\} \cap \mathcal{S}_{i+1} \right| \geq 3, \text{ for } i = 1, \ldots, k-1. \tag{6.3}$$

We call equation 6.3 the *overlap condition*. It ensures that we can assemble an n-species tree from the quartet trees. Note, that according to the *overlap condition* we would have to compute

$$\sum_{i=1}^{k} \binom{|\mathcal{S}_i|}{4} \tag{6.4}$$

quartets only, dividing $\mathcal{S}$ into k subsets.

6.2.1. The Algorithm ModPUZZLE

Before discussing some computational aspects, we will explain the algorithm for $k = 2$. If $k > 2$, then one has to apply the algorithm $k - 1$ times by setting

$$\mathcal{S}_1 = \bigcup_{i=1}^{j} \mathcal{S}_j \tag{6.5}$$

$$\mathcal{S}_2 = \mathcal{S}_{j+1} \tag{6.6}$$

for $j = 1, \ldots, k-1$. The order of the subsets is chosen randomly and that the *overlap condition* is fulfilled.

The ModPUZZLE algorithm works as follows (cf. also Figure 6.1)

step 1: Assume that an intermediate tree $T(\mathcal{S}_1)$ is reconstructed using the normal *puzzling step* algorithm (cf. 3.2.2 and 4.4).

step 2: Compute

$$\mathcal{S}_{1\cap 2} = \mathcal{S}_1 \cap \mathcal{S}_2.$$

The elements of $\mathcal{S}_{1\cap 2}$ are already in $T(\mathcal{S}_1)$.

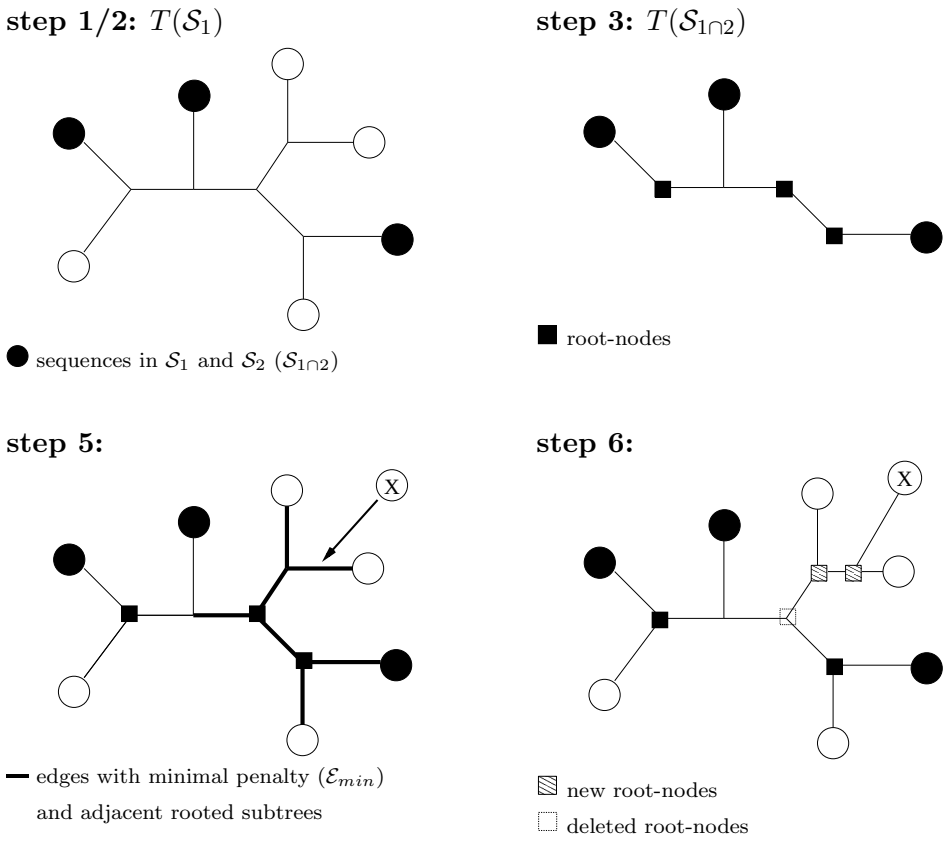


Figure 6.1.: The modified *Quartet Puzzling* algorithm. For details see text.

step 3: Derive the subtree $T(\mathcal{S}_{1\cap 2})$ from $T(\mathcal{S}_1)$, that has as leaf-set the sequences in $\mathcal{S}_{1\cap 2}$ and as internal node set all nodes from $T(\mathcal{S}_1)$ that are on a path connecting two leaves from $\mathcal{S}_{1\cap 2}$. Each node of degree 2 in $T(\mathcal{S}_{1\cap 2})$ is then a root of a subtree from $T(\mathcal{S}_1)$ with leaves in $\mathcal{S}_1$ but not in $\mathcal{S}_2$. Nodes of degree 2 are called root-nodes.

step 4: Pick a sequence $X \in \mathcal{S}_2 \backslash \mathcal{S}_1$ and compute the penalties of each edge in $T(\mathcal{S}_{1\cap 2})$ using the normal *puzzling step* algorithm (see 3.2.2 and 4.4).

step 5: Let $\mathcal{E}$ be the collection of all edges in $T(\mathcal{S}_{1\cap 2})$ with minimal penalty, then insert X randomly either on one edge $e \in \mathcal{E}_{min}$ or on an edge from a rooted subtree of $T(\mathcal{S}_1)$ whose root node has at least two edges in $T(\mathcal{S}_{1\cap 2})$ with minimal penalty.

step 6: If X is the inserted sequence, then update $T(\mathcal{S}_{1\cap 2})$ by adding X to the leaf-set and by adding all nodes of $T(\mathcal{S}_1)$ that lie on the path between X and the root-node R of the subtree of $T(\mathcal{S}_1)$, where X was inserted, to the set of root nodes. Finally, delete R from the root-node set.

step 7: Delete X from $\mathcal{S}_2 \backslash \mathcal{S}_1$. If $\mathcal{S}_2 \backslash \mathcal{S}_1 \neq \emptyset$ goto **step 4**, otherwise goto **step 8**.

step 8: Repeat **step 1** to **step 7** of the algorithm several times for different input orders of the subsets and the sequences within the subsets.

step 9: From this collection of trees the majority consensus tree is calculated.

6.3. Computational Aspects

To simplify matters, we assume that we partition the data in $k \leq n$ subsets of size $\nu = \frac{n}{k} + o$, where o defines the overlap between two neighboring sets, i.e. $|\mathcal{S}_i \cap \mathcal{S}_{(i \bmod k)+1}| = o$ for $i = 1, \ldots, k$, where we require that $o \geq 3$. Although not crucial for the ModPUZZLE algorithm, for the practical application but to add freedom to the repetitions of **steps 1** to **7**, we now construct the last subset $\mathcal{S}_k$ overlapping $\mathcal{S}_1$.

For $k = n$ and $o = 3$ we have to compute $n - 3$ quartets only. This, however, is not enough information to reconstruct the tree, even if the data were perfectly tree-like. Thus, one should try to increase o as much as possible, guided by the time one is willing to spend for the calculation of the

$$k\left(\binom{\nu - o}{4} + \binom{o}{4}\right) \tag{6.7}$$

possible quartets, where $o > 3$.

6.4. Some Practical Measures on the Method

In phylogenetic tree reconstruction one wants to reconstruct a fully resolved tree, i.e., all inner nodes of the tree have degree 3. However, due to the lack of a clear phylogenetic signal it is not always possible to achieve this goal (cf. 3.1). In these situations the tree reconstructed is not fully resolved. The amount of resolution can be measured by the number of splits, i.e. inner branches, that partition the sequences into two non-empty subsets. To analyze the influence of the size of the overlap between subsets on the resolution, we carried out a simulation study. To this end, we simulated the evolution of DNA sequences on a tree with 50 leaf-vertices using the Seq-Gen package (Rambaut and Grassly, 1997).

The ModPUZZLE algorithm was applied to the resulting data by randomly splitting the 50 sequences into $k = 5$ subsets of varying size and different overlap. Table 6.1 summarizes the results. The results show that one should try to maximize the overlap, since larger overlap enables the a higher resolution of the reconstructed tree. It shows that an overlap of 20 induces the computation of roughly 50% of all possible quartets. On the other hand, 39 from the 47 possible splits could be recovered, which provides a reasonably good resolution.

The ModPUZZLE algorithm was also applied to the alignment of all 215 red algae ssu rRNA sequences from the European small subunit rRNA database (Van de Peer *et al.*, 2000b) as a biological dataset. Due to the large number of quartets (86,567,815 possible

Table 6.1.: Results of the tree reconstruction using a simulated alignment of 50 DNA sequences with increasing overlap.

k subsets	ν sequences	o overlap	number of quartets	% of quartets	resolved splits	% of splits
5	13	3	3,230	1.4	2	4.3
5	15	5	6,800	2.6	7	14.9
5	20	10	23,175	10.1	23	48.9
5	30	20	112,800	49.0	39	83.0
full set	50		230,300	100.0	47	100.0

quartets) tests were only ran for a limited set of values for k and ν. Nevertheless, this large biological dataset shows similar characteristics as the simulated one. The resolution increases with the number of quartets available for the tree reconstruction.

The tests on the simulated data as well as the limited tests on biological data show that with the minimal amount of overlap ($o = 3$) almost no resolution of the trees can be gained. However, the resolution increases with the number of quartets used. Remarkable is the effect that the percentage of resolved splits grows faster than the percentage of quartets used (Tab. 6.1). Therefore it seems to be possible to reconstruct resolved trees even if one does not use all quartet trees. The potential runtime reduction achieving similar resolution, however, crucially depends on the amount of phylogenetic information present in the alignment.

6.5. Discussion and Possible Extensions

In this chapter a very simple algorithm has been suggested to reconstruct phylogenetic trees from large datasets. The method is based on a modified version of the *puzzling step* algorithm described in 3.2.2 and 4.4. The modified algorithm has the flexibility to adjust the amount of computing time one is willing to spend. If one is only interested in the coarse structure of the tree, then one needs to compute only very few quartets, thus obtaining a more or less unresolved tree. If one wants the fine details of the ramifications of the tree, one needs to compute a lot more quartets by increasing the overlap between the subsets.

But, as shown in Table 6.1, the percentage of resolved splits seems to grow faster than the amount of quartets used. This observation leads to a strategy how to analyze large datasets, which is not fully exploited here. Instead of randomly assigning sequences to the k subsets once in some kind of linear order, one could use more decompositions to build a network of subsets. To increase the resolution of the final tree, one may also use a data guided approach. For example, a threshold graph (Barthelemy and Guenoche, 1991; Huson *et al.*, 1999) based on the pairwise distances can be used as an indicator

how to group sequences. The applicability of this strategy needs to be analyzed by simulations.

Another extension of the algorithm is also possible. Instead of analyzing the gene tree of one set of aligned sequences, we may very well assume that each subset $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_k$ contains the collection of species for which a sequence alignment is available for one out of k different genes. We then can compute the tree for the entire set of species $\mathcal{S}$, based on k different genes, without requiring that one gene sequence is known for all n species in $\mathcal{S}$. The properties of this approach need to be investigated.

In chapter 7 a method will be described similar to the latter extension.

7. Phylogenetic Trees from Multiple Genesets with Missing Data

7.1. Introduction

The large amounts of molecular sequence data currently available serve two needs in the phylogenetic analysis of the relationship of species. On the one hand, the number of interesting species available for analysis grows (vertical growth). On the other hand, more different sequences per species get available (horizontal growth). Unfortunately, the number of sequences is not distributed evenly among species of interest. This often makes collecting a dataset for phylogenetic analysis a painful decision on the trade-off between the amount of taxa and the number of sequences.

While methods abound to infer phylogenies from a set of aligned sequences (Swofford *et al.*, 1996), only a small number of methods exists to combine data of different genes, proteins, or genomic areas for joint analysis.

There has been a large ongoing debate how to combine different datasets to reconstruct trees (cf. de Queiroz *et al.*, 1995, Bininda-Emonds, 2002, and Page and Holmes, 1998, chap. 8 for review). Two paradigms have been discussed, which we will classify by the 'distance' of the combination event from the underlying data into *low level* and *high level* methods (cf. Fig. 7.1). Note that *low* and *high* does not imply a quality rating.

The first paradigm is *total evidence* (also often called *combined* or *simultaneous analysis*) where the data is combined directly by concatenating the alignments. Hence, *total evidence* methods will be referred to as *low level methods* (cf. Fig. 7.1).

The other paradigm is the so-called *separate analysis* (also called *taxonomic congruence* and *consensus* or *supertree approach*). Such methods combine trees reconstructed separately from the single datasets. Since the combination takes place far from the underlying data, such methods are referred to as *high level methods* (cf. Fig. 7.1).

Both paradigms have their advantages and drawbacks (for review see de Queiroz *et al.*, 1995, Bininda-Emonds, 2002, and references therein). Major criticisms are, e.g., the problems to jointly model the evolutionary process for the concatenated dataset in *low level methods* and the loss of information in the *high level methods* since the underlying data is in general not considered when the trees are combined. These arguments gave rise to the design of an alternative method which will be proposed in this chapter.

The method presented here follows a *medium level* paradigm, according to the level of combination (cf. Fig. 7.1). Before particularizing the procedure (section 7.3), commonly

used *high* and *low level methods* are described (section 7.2). Then the new method is applied to a biological dataset in section 7.4. The results are discussed in the context of *total evidence* and *consensus/supertree methods*. Finally, problems will be discussed and an outlook on further improvement and extensions will be given.

7.2. Methods to Combine Datasets

7.2.1. Low Level Combination: Total Evidence

As mentioned above, *low level methods* are commonly referred to as *total evidence*, *combined* or *simultaneous analysis* methods. Claiming that all information (evidence) available should be used for phylogenetic analysis, all single datasets are combined into one single *'supermatix'* (Sanderson *et al.*, 1998) as shown in Fig. 7.1. This is achieved by concatenating all source alignments filling missing data with gap characters. The overall alignment is then used as input for phylogenetic analysis. This method seems unproblematic for complete datasets, where for each species one sequence in each source dataset exists. How *total evidence* approaches handle missing data crucially depends on the method used to reconstruct a phylogeny from the concatenated alignment. Since some programs exclude alignment columns with gaps, such programs will end up only with data from genes which are available for the whole set of species. Datasets with missing sequences are discarded.

7.2.2. High Level Methods

While *low level methods* combine the source datasets directly, *high level methods* construct a tree for each source dataset. The trees from the source datasets are then combined to an overall tree (Fig. 7.1). If all source datasets contain the full set of species, consensus techniques can be applied combining all the equally sized trees into one consensus tree. To combine sets of trees with unequal, but overlapping sets of species, so-called *supertree* methods have been developed which amalgamate the input trees into one overall *supertree*.

7.2.2.1. Combining Equal-Sized Datasets: Consensus Methods

Consensus methods aim to construct a consensus tree from a set of trees preserving common topological details of the source trees. Commonly applied consensus methods are strict consensus, majority-rule consensus (Margush and McMorris, 1981), semi-strict consensus (Bremer, 1990), and Adams consensus (Adams III, 1986).

Most consensus methods are based on splits or bipartitions (cf. 2.2.1) found in a majority of the source trees (M_l consensus *sensu* McMorris and Neumann, 1983, where

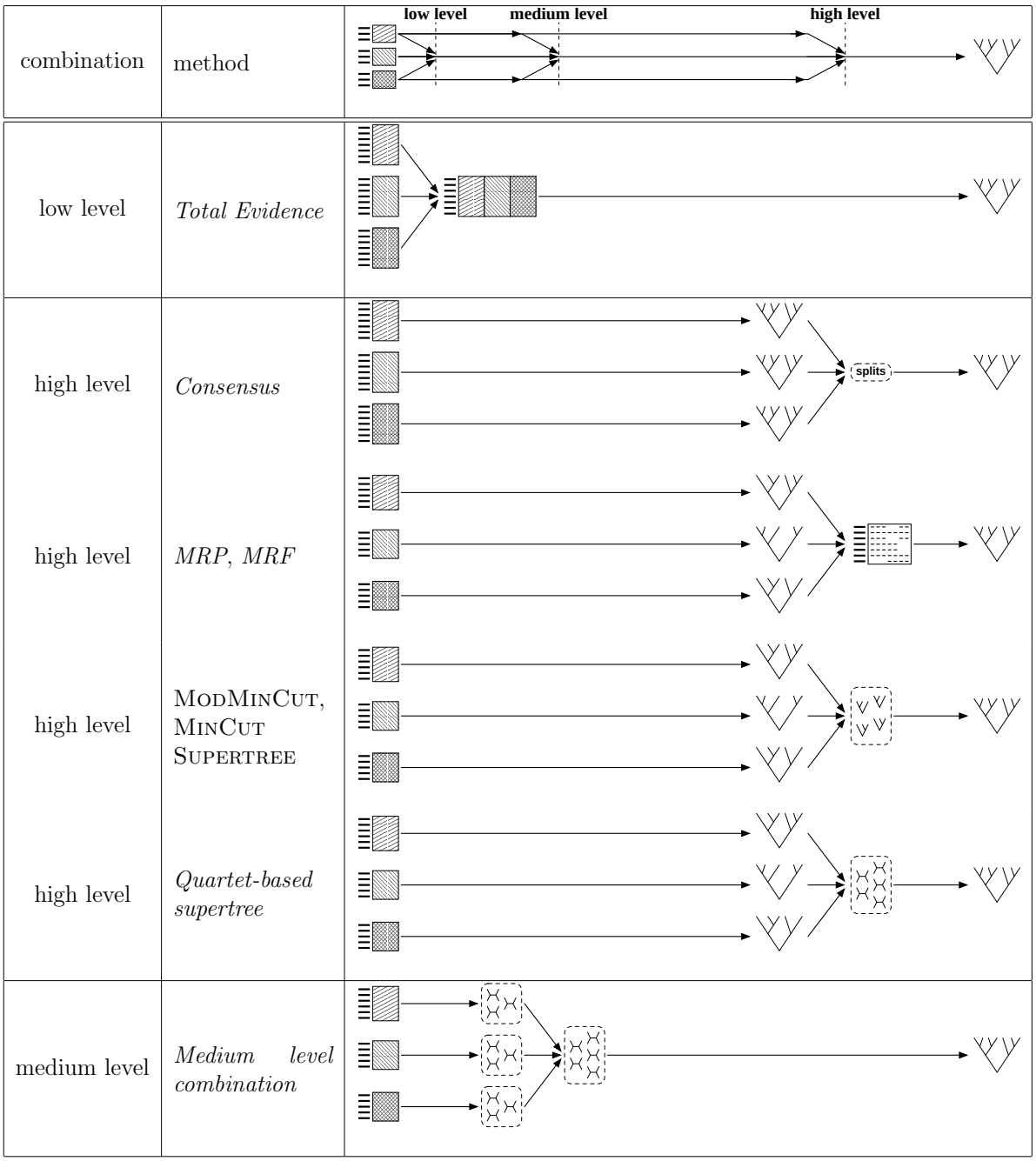


Figure 7.1.: Level of combination with regard to distance from the underlying datasets

l represents the percent occurrence of included splits). Sets of non-contradicting bipartitions are chosen to reconstruct the consensus tree.

The *majority-rule consensus* (Margush and McMorris, 1981) uses all splits occurring in the majority of input trees. Usually a *majority-rule consensus* is assumed (M_l consensus with $l > 0.5$, McMorris and Neumann, 1983). This means all splits occurring in more than 50% of the source trees (see Fig. 7.2). Considering $l > 0.5$ ensures that all splits

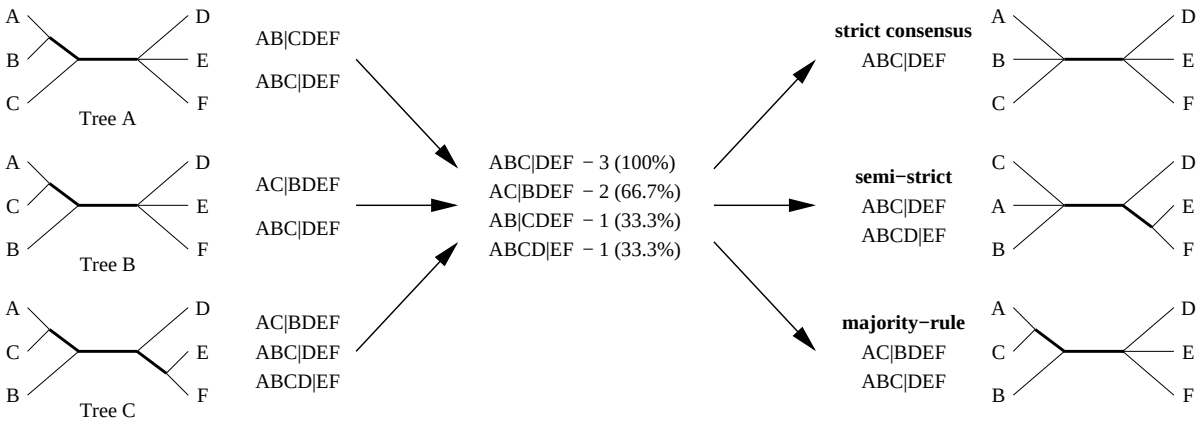


Figure 7.2.: Consensus methods

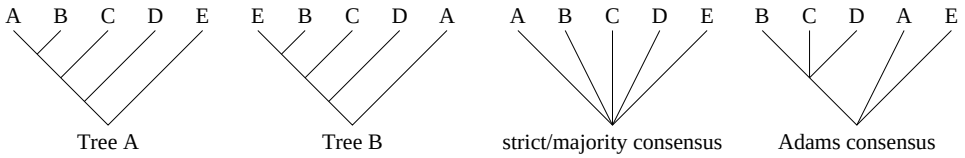


Figure 7.3.: Adams Consensus

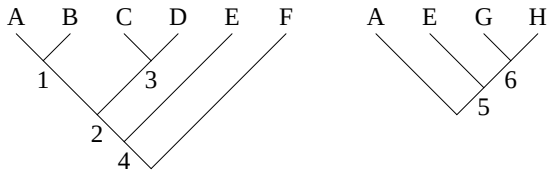
can be incorporated in a tree topology, because no two contradicting splits are able to occur in more than 50% of the input trees.

For the *strict consensus* tree only bipartitions are chosen that are found in all input trees ($M_{1.0}$, see Fig. 7.2). The *semi-strict consensus* (Bremer, 1990) contains all splits that are not contradicted in any of the input trees, e.g. the split $ABCD|EF$ is uncontradicted by the ABC-trifurcation in tree A and B in Fig. 7.2. Note that such splits can occur in less than half of the source trees. If all trees are binary trees *strict* and *semi-strict consensus* will always produce the same trees.

Contrary to most consensus methods which are based on bipartitions and their percent occurrence in the source trees, the *Adams consensus* is based on common nestings in trees, i.e., taxa frequently occurring together in the same subtree. This method tries to find groups of sequences that are commonly occurring together in the source trees (Adams III, 1986). *Adams consensus trees* can contain groups that cannot be found in any of the source trees. This makes it difficult to interpret. Yet it can be informative when there are sequences that are difficult to place. Such sequences are moved to the root of the *Adams consensus tree* (Fig. 7.3).

7.2.2.2. Combining Overlapping Datasets: Supertree Methods

Supertree methods have been developed to combine sets of overlapping trees into *'supertrees'* containing all leaves found in the source trees. Some of the methods are re-



Ragan/Baum

	1	2	3	4	5	6
A	1	1	0	1	0	0
B	1	1	0	1	?	?
C	0	1	1	1	?	?
D	0	1	1	1	?	?
E	0	0	0	1	1	0
F	0	0	0	0	?	?
G	?	?	?	?	1	1
H	?	?	?	?	1	1

Purvis

	1	2	3	4	5	6
A	1	1	0	1	0	**?**
B	1	1	0	1	?	?
C	0	1	1	1	?	?
D	0	1	1	1	?	?
E	**?**	0	**?**	1	1	0
F	**?**	**?**	**?**	0	?	?
G	?	?	?	?	1	1
H	?	?	?	?	1	1

Figure 7.4.: Coding schemes to encode tree topologies for MRP methods: Baum/Ragan and Purvis' scheme

stricted to rooted trees, based on the argument that the broad majority of trees published are rooted trees. Input trees cannot be produced from different datasets only, but can also be obtained from the literature.

Matrix Representation Methods A commonly used method to construct supertrees is *Matrix Representation using Parsimony* (MRP, Baum, 1992; Ragan, 1992), which uses binary coding to represent the input trees. The splits induced by all source trees are coded into a matrix with binary characters ('0', '1') with missing data ('?'). The binary matrix is then used to construct an overall tree (cf. Fig. 7.1) using *Maximum Parsimony* methods (Swofford *et al.*, 1996). The most abundant coding schemes are those independently suggested by Baum (1992) and Ragan (1992) and the modified scheme by Purvis (1995). Baum/Ragan code the bipartitions of an input tree assigning '1' to all leaves in the one part of the bipartition and '0' to the other that contains the root. All missing taxa are assigned the missing data character '?' (Fig. 7.4). Purvis (1995) differs in that only the sistergroup of a clade is assigned '0', while all other leaves of that part outside the clade are assigned '?' (Fig. 7.4).

To give input trees different weights in the analysis, different weighting schemes have been suggested (see Salamin *et al.*, 2002, Sanderson *et al.*, 1998, and references herein for more details).

Recently, another method has been proposed called *Matrix Representation using Flipping* (MRF, Chen *et al.*, 2003), which also uses a binary matrix coding of the trees as described above. The MRF supertree is constructed by 'flipping' conflicting character states from '1' to '0', or vice versa to produce a matrix that does not contain contradictions. Their optimal solution is the tree that needs the least 'flips'.

Direct Supertree Methods Another family of supertree methods use a more graph theoretical approach. Members of that family are the OneTree/BUILD algorithm (Aho *et al.*, 1981; Bryant and Steel, 1995), MinCut Supertree algorithm (Semple and Steel, 2000), and the modified MinCut Supertree algorithm (ModMinCut Supertree, Page, 2002). These algorithms take as input sets of rooted trees. The BUILD algorithm and the OneTree algorithm produce a supertree only if the input trees are compatible, i.e., the trees are not contradictory (cf. strict consensus in 7.2.2.1). Since this is almost never the case for biological data, these algorithms are mainly useful to test for compatibility of trees.

MinCut Supertree tries to yield a supertree applying a minimum cut algorithm to the graph $S_{\mathcal{T}}$ constructed from the set of input trees $\mathcal{T}$, following Listing 7.1 (see also Fig. 7.5 and Fig. 7.1). The notations follow Page (2002) – for further details see there.

Page (2002) suggested a modification (here called ModMinCut) to choose the minimum cuts, that takes into account preserving uncontradicted information from the source trees. The ModMinCut Supertree algorithm marks all edges in the graph $S_{\mathcal{T}}$ (step 7.1a) not contradicted by any tree in $\mathcal{T}$. If $S_{\mathcal{T}}$ has to be cut in step 7.1b minimum cuts are preferred which do not cut uncontradicted edges. Hence, the amount of uncontradicted information from the source trees is preserved in the supertree (cf. Fig. 7.6).

Quartet-Based Supertree Recently a quartet-based supertree method has been proposed by Robinson-Rechavi and Graur (2001). Instead of using a matrix representation, all source trees are decomposed into sets of quartets contained in the topologies (Fig. 7.1). Each quartet is weighted by the highest support value found on the path of the quartet's middle edge in any of the input trees (the authors used the TREE-PUZZLE package to compute the trees and support values). From this set of weighted quartets an overall tree is constructed with the quartet method implemented in the AllTree program (Bendor *et al.*, 1998). In a first step, this method computes all subsets of taxa and keeps those satisfying the most quartets. In a second step, it performs an exact search on each subset constructing an overall tree satisfying most quartets.

7.3. Medium Level Combined Phylogenetic Analysis

7.3.1. Notation

In the present chapter we assume a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of n species and k different genes to be used for combined analysis.

With $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ we denote subsets of $\mathcal{S}$, such that

$$|\mathcal{S}_g| \leq 4 \qquad \text{for each} \quad g \qquad \text{and} \qquad \bigcup_{g=1}^{k} \mathcal{S}_g = \mathcal{S}. \tag{7.1}$$

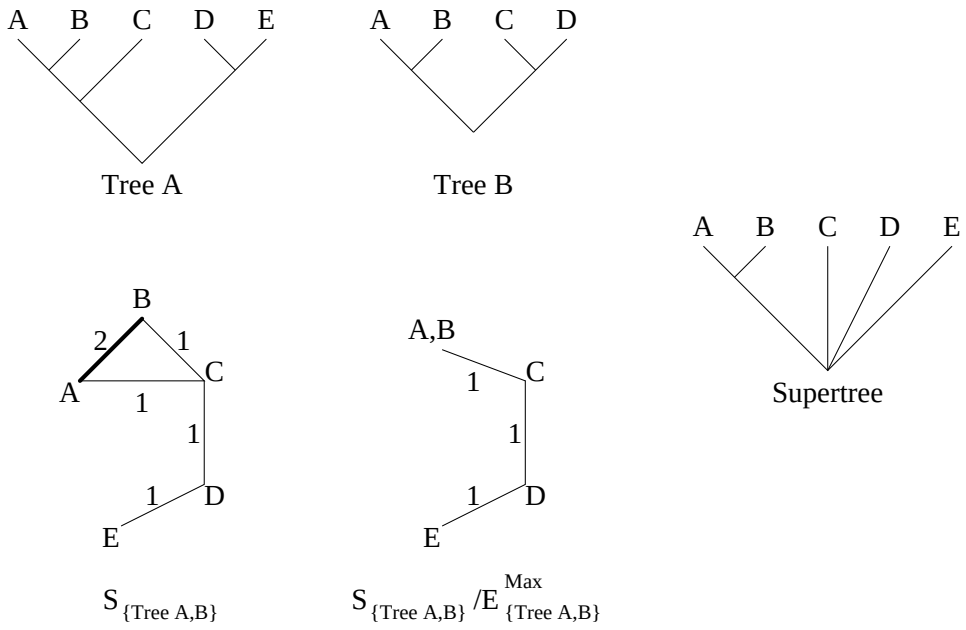


Figure 7.5.: The MinCut Supertree Algorithm. For more details, see text.

Listing 7.1: MinCut Supertree (Set of rooted trees $\mathcal{T}$, set of n leaves)

switch *number of leaves* n **do**
  case $(n = 1)$ **return** a single node x_1;
  case $(n = 2)$ **return** a tree with 2 leaves x_1 and x_2;
  otherwise

7.1a
    Construct the graph $S_{\mathcal{T}}$ by connecting each pair of taxa a and b by an edge $e_{(a,b)}$ if they are connected in at least one tree in $\mathcal{T}$ by a path that does not pass the root of that tree, i.e., MRCA$(a, b) \neq root$ (cf. Fig. 7.5, $S_{\{\text{TreeA,B}\}}$);
    Weight the edge $e_{(a,b)}$ by the number of trees supporting MRCA$(a, b) \neq root$;

if $S_{\mathcal{T}}$ *is a disconnected graph* **then**
  Let S_i be the components of $S_{\mathcal{T}}$;
else
  Construct $S_{\mathcal{T}}/E_{\mathcal{T}}^{Max}$ as follows: Merge all nodes a and b connected by an edge $e_{(a,b)}$ with maximum weight $|\mathcal{T}|$, i.e., supported by the whole set of trees and remove edge $e_{(a,b)}$ (cf. Fig. 7.5);
7.1b
  Remove all edges from $S_{\mathcal{T}}/E_{\mathcal{T}}^{Max}$ that are a minimum cut set of $S_{\mathcal{T}}$.
  Let S_i be the resulting components of $S_{\mathcal{T}}/E_{\mathcal{T}}^{Max}$;
foreach *component* S_i **do**
  $T_i =$ MinCut Supertree $(\mathcal{T}|S_i, L = S_i)$, where $\mathcal{T}|S_i$ denotes the trees from $\mathcal{T}$ with all leaves $r \notin S_i$ pruned;
Construct a new tree T by connecting the roots of T_i to a new tree;
return T

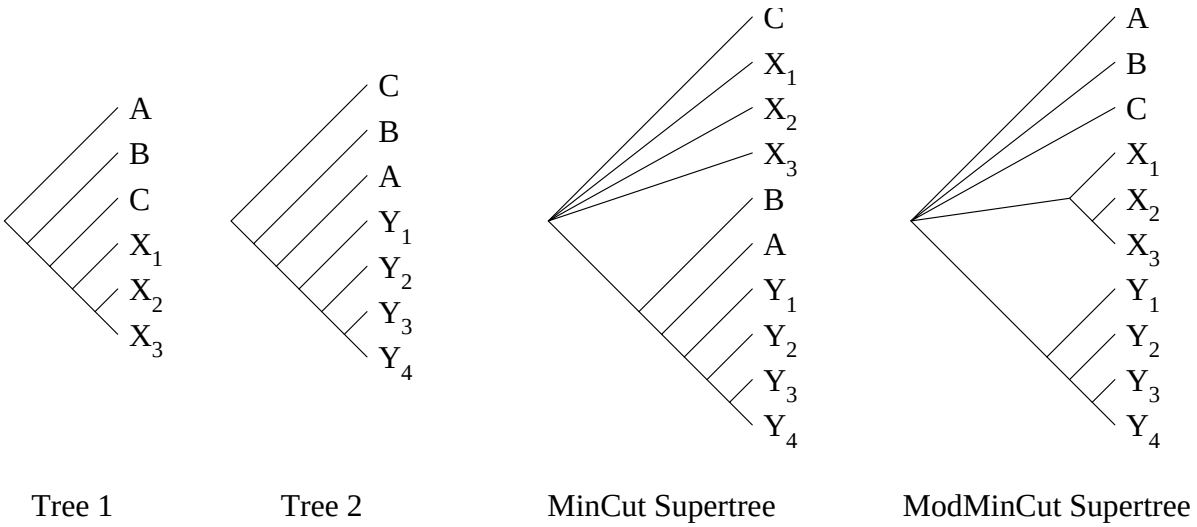


Figure 7.6.: The ModMinCut Supertree algorithm. For more details, see text.

Each $\mathcal{S}_g$ represents the subset of species for which a multiple alignment based on gene g is available. These subsets will be called genesets. Finally, $\mathcal{Q}_g, g = 1, \ldots, k$ represent the corresponding sets of possible quartets.

Instead of reconstructing the trees $T(\mathcal{S}_g)$ for each set $\mathcal{S}_g$, we will compute an overall tree $T(\mathcal{S})$ combining the information provided from the evaluation of the quartet sets $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k$ from the k different alignments. Note that a taxon is not represented by one sequence any more.

7.3.2. The Combined Quartet Method to Combine Genesets

7.3.2.1. Combining the ML Quartets

In the method proposed, the genesets will be combined on the level of the quartets. As a guideline to combine the quartets we use the log-likelihood $\ell^{(g)}_{ab|cd}$, $\ell^{(g)}_{ac|bd}$, and $\ell^{(g)}_{ad|bc}$ for quartet $\{a, b, c, d\} \in \mathcal{Q}_g$ on the geneset $\mathcal{S}_g$.

To combine the datasets for each geneset $\mathcal{S}_1$, $\mathcal{S}_2$, $\ldots \mathcal{S}_k$, all three log-likelihoods for each quartet in $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\ldots \mathcal{Q}_k$ are evaluated first (cf. *ML step* in 3.2.2).

Then, we compute

$$\ell_{ab|cd} = \sum_{g=1}^{k} \ell^{(g)}_{ab|cd} \tag{7.2}$$

$$\ell_{ac|bd} = \sum_{g=1}^{k} \ell^{(g)}_{ac|bd} \tag{7.3}$$

$$\ell_{ad|bc} = \sum_{g=1}^{k} \ell_{ad|bc}^{(g)} \tag{7.4}$$

for each quartet. For the sake of simplicity, the log-likelihoods $\ell_{\tau}^{(g)} = 0$ if the quartet with the sequences $\mathcal{L}(\tau)$ is not represented by alignment g.

Alternatively, one can also compute

$$\overline{\ell_{ab|cd}} = \frac{\ell_{ab|cd}}{|\{g : \{a,b,c,d\} \in \mathcal{Q}_g\}|} \tag{7.5}$$

$$\overline{\ell_{ac|bd}} = \frac{\ell_{ac|bd}}{|\{g : \{a,b,c,d\} \in \mathcal{Q}_g\}|} \tag{7.6}$$

$$\overline{\ell_{ad|bc}} = \frac{\ell_{ad|bc}}{|\{g : \{a,b,c,d\} \in \mathcal{Q}_g\}|}. \tag{7.7}$$

$\overline{\ell_{\tau}}$ can be viewed as the average support a quartet tree τ receives from the set of sequence alignments. In case that a quartet is not represented by any alignment the averages are set equal to zero.

7.3.2.2. Computing the Overall Tree

To reconstruct a phylogeny $T(\mathcal{S})$ based on the collection of log-likelihoods from Equations 7.2 to 7.4 or 7.5 to 7.7 we apply the PUZZLE idea from 3.2.2. The straightforward application of the *QP* algorithm is only possible if information exist for each quartet in $\mathcal{Q}$. This, for instance, is the case if at least one alignment $\mathcal{S}_g$ comprises all the species in $\mathcal{S}$.

If some quartets are missing, which usually happens if each $\mathcal{S}_g$ is a proper subset of $\mathcal{S}$, then these quartets are treated as unresolved. A quartet tree is selected randomly among the three possible topologies. In this case it is necessary to examine whether the overlapping genesets can be combined. This will be explained in the following.

7.3.2.3. Assessing Whether Genesets Can Be Combined

To combine two genesets $\mathcal{S}_i$ and $\mathcal{S}_j$ a minimum pairwise overlap of 3 sequence among the two subsets is required,

$$|\mathcal{S}_i \cap \mathcal{S}_j| \geq 3. \tag{7.8}$$

We call Eq. 7.8 the *pair-overlap condition*. The *pair-overlap condition* is different from the *overlap condition* (Eq. 6.3) in section 6.2 where the subsets were constructed from one complete set of sequences to ensure combinability of the subsets. Here the subsets

are genesets. Their sizes and overlap are defined by the availability of sequence data for the species in $\mathcal{S}$. Note, that not every pair of genesets might share a sufficient overlap.

To assess whether these gene datasets can be combined, we construct an overlap graph $G_{ovl} = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V}$ (genesets) and a set of undirected edges $\mathcal{E}$ (overlaps) with

$$\mathcal{V} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_k\} \tag{7.9}$$

$$\mathcal{E} = \{e_{(\mathcal{S}_i, \mathcal{S}_j)} \text{ for } |\{\mathcal{S}_i \cap \mathcal{S}_j\}| \geq 3\} \quad 1 \leq i < j \leq k. \tag{7.10}$$

The edge weights consist of the amount of overlap fulfilling the *pairwise-overlap condition* (see for an example Fig. 7.7). This representation of the datasets provides insight into several properties of the entire dataset.

If the graph consists of unconnected subgraphs, not all data is suited for a combined analysis. According to the *pairwise-overlap condition* gene datasets from different connected components cannot be used simultaneously in the same analysis because no quartet information is are available to reasonably guide the insertion of the sequences.

Genesets from one connected component, however, can be used for combined analysis. Although genesets might exist within a connected component which do not share sufficient overlap with each other to be combined, they can be linked by means of other genesets which first have to be added satisfying the *pairwise overlap condition*, to connect the two.

Note, that by applying the *overlap condition* from Eq. 6.3, sets from one connected component can gain an overlap ≥ 3 to the leafset $\mathcal{L}(T_i)$ of a tree T_i so far reconstructed from genesets of another connected component. Nevertheless, combining those sets will produce a very doubtful tree, because too little information is available to guide the insertion of sequences.

The combinability of the genesets from a connected component is characterized by two features of the *overlap graph*. A high connectivity within a component (each geneset shows overlap to many other geneset) results in network-like quartet information among the genesets. This reduces the need for mediating sets to connect non-overlapping genesets. Even more important is the size of the overlap. The higher the overlap, the more quartets are shared among two neighboring genesets. Consequently, a higher amount of information is available to guide the insertion of sequences from a geneset into a tree already reconstructed.

7.3.2.4. Overlap-Guided Puzzling Step

As described above, sufficient overlapping information is needed to reasonably insert a new leaf s_{i+1} into a tree T_i. Hence, instead of using a random permutation of leaves, the permutation has to follow certain restrictions. To satisfy the *pairwise overlap condition*, we apply a graph based procedure similar to Prim's minimum spanning tree (MST)

algorithm (see, e.g., Cormen *et al.*, 2001) assuming equal edge weights. We start with the leaves from a randomly picked geneset $\mathcal{S}' \in \{\mathcal{S}_1 \ldots \mathcal{S}_k\}$. We define a front set $\mathcal{F}$ containing all unused nodes (genesets) from the overlap graph that are connected by an edge to any geneset already used in the tree. From the sequences in $\mathcal{S}'$ we construct the tree $T(\mathcal{S}')$ applying the usual puzzling step algorithm (see section 3.2.2 and chapter 4). Then we randomly draw one geneset $\mathcal{S}''$ from $\mathcal{F}$. We remove $\mathcal{S}''$ from $\mathcal{F}$ and add the sets connected to $\mathcal{S}''$, but not yet used to $\mathcal{F}$. The sequences of $\mathcal{S}''$ are then added to the tree. Guided by the overlap graph we thus construct an intermediate tree by sequentially adding the genesets. As in the usual *puzzling step* many intermediate trees are constructed using different orders of leaves and genesets.

7.3.2.5. Relative Majority Consensus

Since missing data adds more ambiguity, this naturally hampers the construction of a resolved majority-rule consensus tree. We therefore decided to apply a majority consensus which is slightly different from the majority-rule consensus used in section 3.2.2. Instead of using only splits occurring in more than 50% (M_l with $l > 0.5$) of the intermediate trees, also splits below 50% are considered. The aim is to use as many splits as are supported by relative majority to extract a maximum uncontradicted information from the intermediate trees.

The relative majority consensus M_{rel} adds all congruent splits from the set of intermediate trees in descending order of occurrence. No splits will be accepted for the consensus that have the same or lower percentage occurrence as any incongruent split. This consensus procedure does not imply a fixed threshold but uses a variable percentage down to the first incongruence guided by the relative majority.

7.4. The Phylogeny of the Grasses

In the following, the grass (*Poaceae*) dataset provided by the GPWG (Grass Phylogeny Working Group, 2001, `http://www.ftg.fiu.edu/grass/gpwg/`) is reanalyzed with the new *medium level* method as well as with other methods for combined analysis.

7.4.1. The Dataset

We used all molecular sequence data from this dataset. There were three nuclear loci, NADH dehydrogenase, subunit F (*ndhF*), the internal transcribed spacer (ITS) of ribosomal DNA, and granule bound starch synthase I (GBSSI or *waxy*), as well as three chloroplast genes, ribulose 1,5-bisphosphate carboxylase/oxygenase, large subunit (*rbcL*), RNA polymerase II, β" subunit (*rpoC2*), and phytochrome B (*phyB*). The morphological and restriction data was omitted. The GPWG dataset consists of 'normal' taxa as well as of composite taxa represented by sequences from several species of one genus (30 taxa)

or several genera (6 taxa). The overall number of taxa in the analysis is $|\mathcal{S}| = 66$. The available sequences are listed in Tab. 7.1.

7.4.2. Methods

The new *medium level* approach is compared to other *high level* as well as *low level* methods. To this end, several tree reconstructions were performed using.

7.4.2.1. Low Level Combination

For *total evidence* analysis the alignments from the six genesets were concatenated into one large *supermatrix*. From this large alignment an overall tree was constructed with the TREE-PUZZLE program assuming the HKY model (Hasegawa *et al.*, 1985). The missing model parameters were inferred from the large alignment.

7.4.2.2. High Level Combination

For *high level methods*, first ML trees were reconstructed using the fastDNAml program (Olsen *et al.*, 1994; Stewart *et al.*, 2001). From the resulting set of six phylogenetic trees supertrees were constructed applying a number of *high level methods*.

Two *MRP* supertrees were built using the Baum/Ragan encoding as well as Purvis' scheme. The input trees were encoded using the SuperTree program (Version 0.8b) by Nicolas Salamin. From the two matrix representations the most parsimonious trees were constructed using the PARS program (see below). Equally parsimonious trees were combined to a strict consensus tree ($M_{1.0}$) with the CONSENSE program (see below).

In addition, supertrees were constructed with the MinCut (Semple and Steel, 2000) as well as the ModMinCut Supertree algorithm (Page, 2002). The reconstruction was performed by using Rod Page's SUPERTREE software (Version 0.2) via the web interface available at `http://darwin.zoology.gla.ac.uk`.

The choice of methods was guided by the availability of implementations or web interfaces. The programs CONSENSE and PARS belong to the PHYLIP package (Version 3.6a3, Felsenstein, 1993).

7.4.2.3. Medium Level Combination

The combined analysis was performed as described in 7.3.2. For each of the six genesets all possible quartet trees were evaluated. The HKY model (Hasegawa *et al.*, 1985) was assumed. Branch lengths were optimized in the computation of the ML quartets ('exact quartet' option in TREE-PUZZLE). Missing model parameters were inferred from the corresponding genesets. The resulting log-likelihoods were combined. The combined likelihoods were then used to determine the set of supported topologies. Finally, the set

Table 7.1.: Sequences available for analysis per taxon in the *Poaceae* dataset (Grass Phylogeny Working Group, 2001)

	ndhF	*phyB*	*rbcL*	*rpoC*	*waxy*	ITS
Flagellaria	X	X	X	-	-	-
Elegia[s]	X	-	X	-	-	-
Baloskion	X	-	X	-	-	-
Joinvillea[s]	X	X	X	X	-	X
Anomochloa	X	X	X	-	X	-
Streptochaeta[s]	X	X	-	-	-	X
Pharus[s]	X	X	-	-	X	X
Guaduella	X	-	X	-	-	-
Puelia	X	X	X	-	-	-
Eremitis	X	X	-	-	X	-
Pariana	X	X	-	-	X	-
Lithachne[s]	X	X	X	-	-	X
Olyra[s]	X	X	-	X	-	-
Buergersiochloa	X	X	-	-	-	-
Pseudosasa[g]	X	X	X	X	-	-
Chusquea[s]	X	X	X	-	X	X
Streptogyna	X	X	-	-	-	-
Ehrharta[s]	X	X	-	X	-	X
Oryza	X	X	X	X	X	X
Leersia[s]	X	-	X	-	-	X
Phaenosperma	X	-	-	-	-	-
Brachyelytrum	X	-	-	-	-	X
Lygeum	X	X	-	X	X	X
Nardus	X	X	-	X	-	X
Anisopogon	X	X	-	X	-	X
Ampelodesmos	X	-	-	-	-	X
Stipa[s]	X	-	X	X	-	X
Nassella[s]	X	X	-	-	-	X
Piptatherum[s]	X	-	-	-	-	X
Brachypodium[s]	X	X	-	-	-	X
Melica[s]	X	X	-	-	X	X
Glyceria[s]	X	X	-	-	X	X
Diarrhena[s]	X	X	-	-	-	X
Avena[g]	X	X	X	X	-	X
Bromus[s]	X	X	X	X	-	X
Triticum[g]	X	X	X	-	X	X
Aristida[s]	X	X	X	X	-	X
Stipagrostis	X	-	X	X	-	X
Amphipogon[s]	X	-	X	X	-	X
Arundo	X	-	X	X	-	X
Molinia[g]	X	X	X	X	-	X
Phragmites	X	X	X	X	-	X
Merxmuellera m.	X	-	X	X	X	X
Karoochloa	X	-	X	X	X	X
Danthonia[s]	X	X	X	X	-	X
Austrodanthonia	X	-	X	X	X	X
Merxmuellera r.	X	-	X	X	X	X
Centropodia	X	-	X	X	X	X
Eragrostis[s]	X	X	X	X	-	X
Uniola	X	-	-	-	-	-
Zoysia[s]	X	-	-	-	-	-
Distichlis	X	-	-	-	-	-
Pappophorum[g]	X	-	X	X	-	-
Spartina[s]	X	-	-	X	-	X
Sporobolus[s]	X	X	-	-	-	X
Eriachne[s]	-	-	X	-	-	X
Micraira[s]	X	-	-	X	-	X
Thysanolaena	X	X	X	X	-	X
Gynerium	X	-	X	X	-	X
Chasmanthium[s]	X	X	X	X	-	X
Zeugites	X	-	-	-	-	-
Danthoniopsis[s]	X	X	-	-	X	-
Panicum[s]	X	X	-	X	-	X
Pennisetum[s]	X	X	X	X	X	X
Miscanthus[g]	X	X	X	X	X	X
Zea	X	X	X	X	X	X

[s] composite taxon, represented by sequences from several species

[g] composite taxon, represented by sequences from several genera

Table 7.2.: The *Poaceae* dataset; estimated parameters and quartet statistics

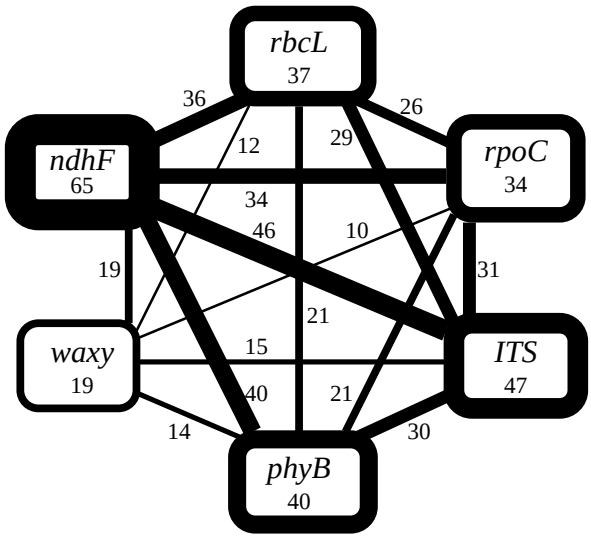
	ndhF	*phyB*	*rbcL*	*rpoC*	*waxy*	ITS	combined	tot. evid.
sequence origin	chloroplast	nucleus	chloroplast	chloroplast	nucleus	nucleus	–	mixed
# sequences	65	40	37	34	19	47	–	66
alignment length	2210	1182	1344	777	773	322	–	6608
constant sites [1]	10.1%	5.6%	45.3%	2.4%	54.2%	48.4%	–	0%
A content	27.3%	21.5%	27.1%	40.5%	21.3%	18.7%	–	26.4%
C content	16.5%	26.9%	19.3%	14.6%	29.4%	31.6%	–	20.0%
G content	17.6%	29.2%	24.9%	28.2%	33.8%	33.1%	–	23.0%
T content	38.7%	22.3%	28.7%	16.7%	15.5%	16.6%	–	30.6%
test failed[2]	0	4	0	1	2	2	–	43
ts:tv[3] parameter	2.14	1.87	1.69	2.31	1.11	1.58	–	1.77
ts:tv[3] S.E.	0.08	0.08	0.12	0.23	0.08	0.11	–	0.04
ts:tv[3] ratio	1.93	1.85	1.66	2.98	1.05	1.45	–	1.73
Y:R ts ratio[4]	1.33	0.96	0.82	0.21	0.63	0.85	–	1.01
average distance	0.077	0.160	0.053	0.107	0.147	0.154	–	0.087
# quartets	677040	91390	66045	46376	3876	178365	720720	720720
resolved	94.41%	92.91%	94.57%	71.49%	94.09%	85.98%	91.87%	94.77%
partly	5.00%	5.77%	4.98%	9.00%	5.62%	10.23%	4.23%	4.17%
unresolved	0.60%	1.33%	0.45%	19.52%	0.28%	3.79%	0.08%	1.06%
missing							3.82%	

[1] including gapped positions
[2] sequences failing χ^2 test on composition
[3] transition:transversion
[4] pyrimidine:purine transition ratio

of quartet topologies and the overlap graph were fed to the modified TREE-PUZZLE program to reconstruct the combined tree. 100,000 *puzzling steps* were performed to build an M_{rel} consensus (cf. section 7.3.2.5).

7.4.3. Parameter Estimates from Poaceae Dataset

The *Poaceae* dataset used for combined phylogenetic reconstruction consisting of *ndhF*, *phyB*, *rbcL*, *rpoC*, *waxy*, and ITS has a total of 6608 aligned positions. The different genesets show substantial differences in their features as well as in the estimated parameters (cf. Fig. 7.2). The number of species in the datasets ranges from 19 in *waxy* to 65 in *ndhF* (cf. also Fig. 7.7). The sequences are closely related resulting in a low average distance from 0.053 (*rbcL*) to 0.160 (*phyB*) and a high percentage of constant sites in the case of *ITS* (48.4%), *rbcL* (45.3%), and *waxy* (54.2%). The nucleotide frequencies differ strongly between the genes. The estimated parameters for the HKY model (Hasegawa *et al.*, 1985) also vary a lot. The transition:transversion ratios range from 1.05 (*waxy*) to 2.98 (*rpoC*), while the transition ratio between pyrimidines and purines varies from 0.21 (*rpoC*) to 1.33 (*ndhF*). Furthermore, the fact only 23 (35%) of the 66 sequences in the concatenated dataset pass the χ^2 test on the sequence composition (Fig. 7.2) shows that the overall character frequencies might be inappropriate for a majority of the sequences. This as well as the variation of the parameters clearly demonstrates that finding one single parameter set that matches the joint dataset can be problematic when doing *'simultaneous analysis'*.



	ndhF	*rbcL*	*rpoC*	*ITS*	*phyB*	waxy
ndhF	•	36	34	46	40	19
rbcL		•	26	29	21	12
rpoC			•	31	21	10
ITS				•	30	15
phyB					•	14
waxy						•

Figure 7.7.: Overlap graph of the gene datasets: Line width is chosen to represent the number of sequences in the overlap or the dataset, respectively. The table shows the amount of pairwise overlap.

The quartet resolution of the genesets, i.e., the amount of fully resolved quartets, is high (up to 94.57%), except for *rpoC*, where only 71.49% of the possible quartets could be resolved completely.

The resolution of the combined quartet set with 91.87% completely resolved quartets was rather high. This shows that the combination of the datasets did not lead to a substantial reduction of the quartet resolution. Yet a high number of resolved quartets does not exclude contradiction among the inferred quartet topologies. Missing data among the overlapping genesets induces a total of 3.82% of the 720,720 quartets for which no phylogenetic information is available.

The parameters estimated from the concatenated alignment (Tab. 7.2) also present substantial differences compared to the parameters inferred from the separate genesets. While the average distance among the concatenated sites is still low (0.087), the constant sites have vanished. This is attributed to the gaps introduced to the alignment to fill up the genesets to 66 sequences.

7.4.4. Combinability of the Poaceae Dataset

The overlap graph in Fig. 7.7 was constructed from the overlap among the genesets which are listed in Tab. 7.1. The line widths represent the amount of sequences in the genesets (frames) or the amount of overlap between two adjacent genesets (edges), respectively.

The overlap graph consists of one connected component. This means that all genesets can be used together in a combined analysis. The connectivity of the graph is high, every

geneset is connected to every other geneset by an overlap > 3. In general, the overlap of any two genesets is higher than half the size of the smaller geneset. This means that more than 50% of the taxa contained in a geneset is also contained in a bigger geneset.

According to the *pairwise-overlap condition* and the high connectivity of the graph the genesets can be combined. Note, that this judgment only takes into account the physical overlap of the genesets and, thus, determines only the technical combinability of the dataset.

7.4.5. Reconstructed Poaceae Phylogeny

The taxonomy and the nomenclature follow the Grass Phylogeny Working Group (2001). For more details about the phylogeny of the grasses refer to the article.

7.4.5.1. Total Evidence Tree

The *total evidence* tree of the grasses (*Poaceae*) based on the concatenated GPWG dataset using TREE-PUZZLE for the simultaneous analysis is displayed in Fig. 7.8. The tree is rooted by the outgroup taxa *Flagellaria*, *Baloskion*, *Elegia*, and *Joinvillea*. Then the so-called early branching taxa branch off the tree. First the *Anomochloid* subfamily which is shown to be a monophyletic sister group of *Pharus*. Next, the monophyletic *Puelioid* subfamily forms a sister group to the main group of the grasses. Within the grasses six subfamilies establish the PACCAD clade. The PACCAD subfamilies *Aristidoids*, *Danthonioids*, and a main part of the *Panicoids* could be recovered. *Aristidoids* and *Danthonioids* together form a monophyletic group.

The three other grass subfamilies, *Bambusoids*, *Erhartoids*, and *Pooids*, form monophyletic subtrees. These three subfamilies are reported to form a monophyletic group, the so-called BEP clade (Grass Phylogeny Working Group, 2001) for different datasets. The BEP clade itself, however, could not be revealed.

Two other *total evidence* grass trees based on the *MP* criterion have been published (Grass Phylogeny Working Group, 2001; Salamin *et al.*, 2002) and will be presented later in section 7.4.6.

7.4.5.2. MRP Supertrees

The two MRP supertrees present different topologies (Figs. 7.9 and 7.10). The first tree was reconstructed from a matrix encoded according to Baum/Ragan. 29 most-parsimonious tree topologies have been found and joined by strict consensus (cf. Baum, 1992; Ragan, 1992). The resulting tree (MRP-BR) could establish most of the grass subfamilies (Fig. 7.9). Starting from the root, the early branching taxa branch off in the order *Anomochloids*, *Pharus*, and then *Puelioids*. Within the grass subtree, the

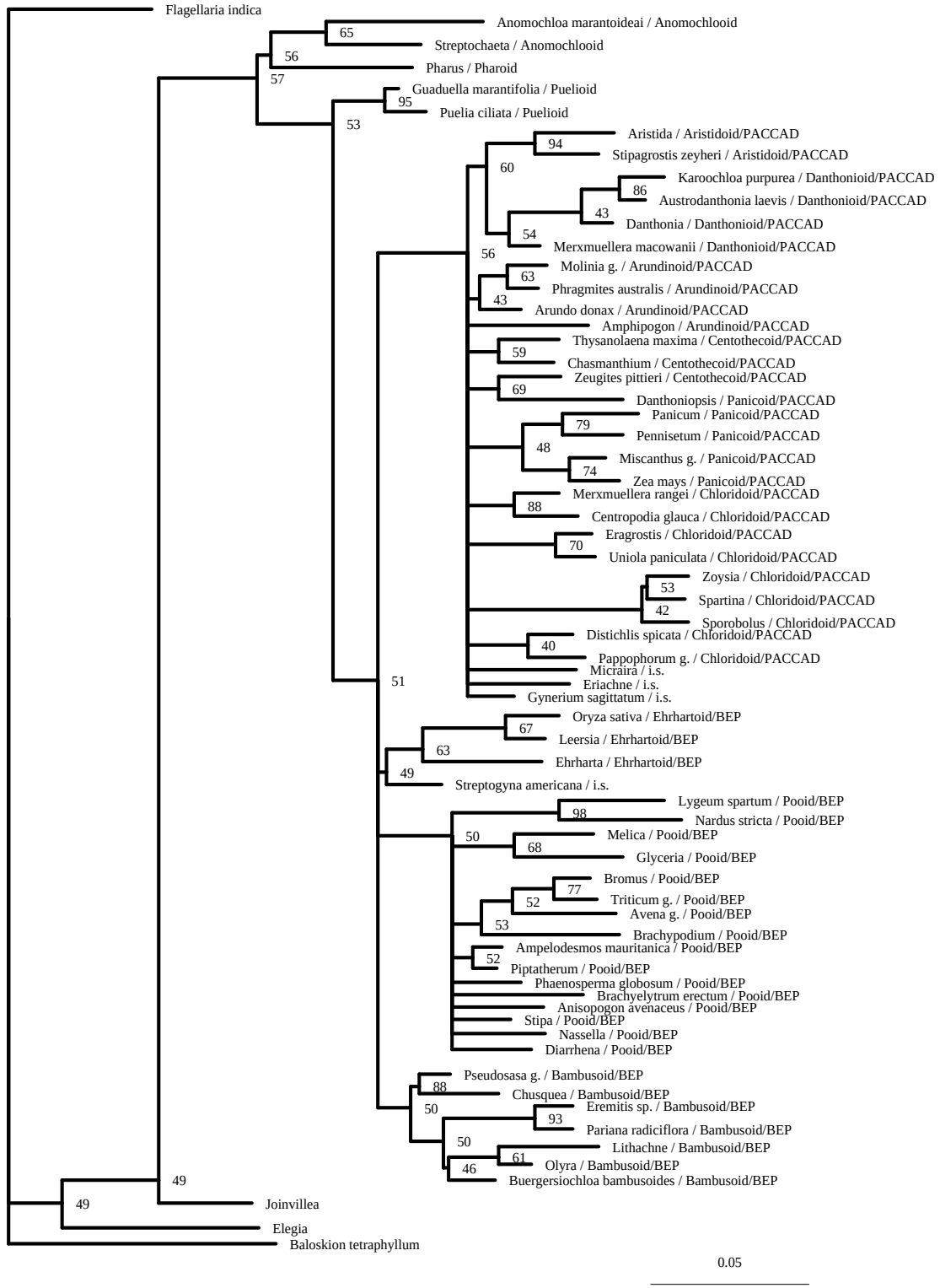


Figure 7.8.: *Total Evidence Poaceae* phylogeny of the 66 taxa from the GPWG dataset. For the analysis HKY model has been assumed, missing parameters have been estimated from the joint datasets. The numbers at the internal nodes represent the percentage of the corresponding congruent splits.

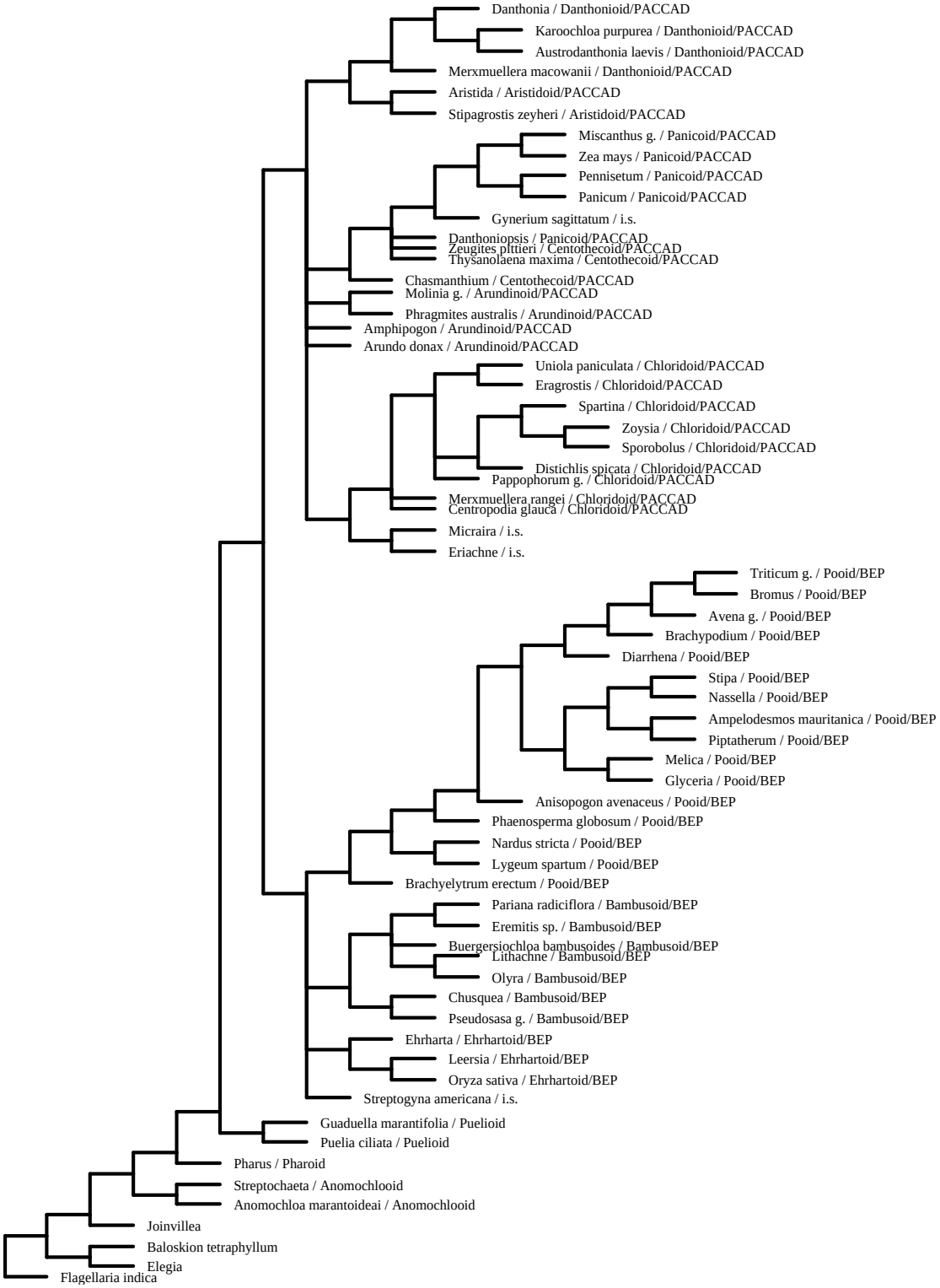


Figure 7.9.: MRP phylogeny of the 66 taxa from the *Poaceae* dataset based on ML trees and the Baum/Ragan coding scheme (MRP-BR).

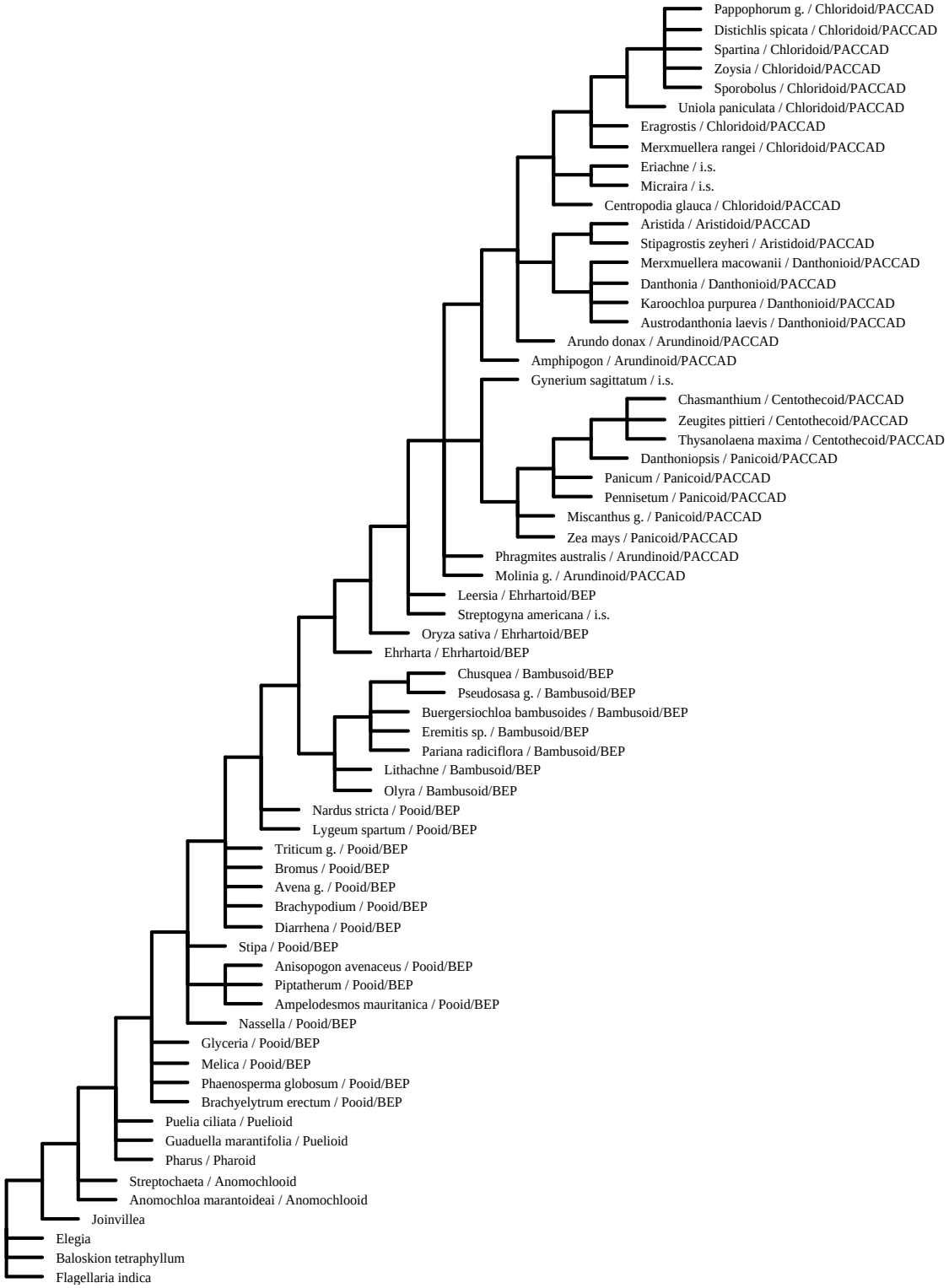


Figure 7.10.: MRP phylogeny of the 66 taxa from the *Poaceae* dataset based on ML trees and Purvis' coding scheme (MRP-Pu).

PACCAD clade is again monophyletic. The PACCAD subfamilies *Chloridoids*, *Aristidoids*, and *Danthonioids* could be established where the latter two form a monophyletic group. The branching pattern of *Arundinoids*, *Panicoids*, and *Centothecoids* remains unresolved. The BEP clade consisting of the monophyletic *Bambusoids*, *Erhartoids*, *Pooids* could be recovered whereas their branching order remained unresolved.

Using Purvis' encoding one most parsimonious tree was reconstructed (MRP-Pu, Fig. 7.10), which showed a by and large linearized topology. The PACCAD clade is recovered in which the *Chloridoids*, *Centothecoids*, *Aristidoids*, and *Danthonioids* remain monophyletic. The members of the *Arundinoids* and *Panicoids* are placed on the two main PACCAD branches in a linearized order not providing separate subtrees. The taxa of the BEP clade subfamilies are inserted along the main branch of grass subtree. Only the *Bambusoids* subtree could be recovered.

7.4.5.3. MinCut Supertrees

The MinCut Supertree produced a long caterpillar tree (Fig. 7.11) that only maintains the *Danthonioid* subfamily at the top and a monophyletic group for the two early branching *Anomochloids*. In the linearized tree, the early branching taxa are separated from the outgroup taxa and from the grass subtree by edges above and below.

The modifications suggested by Page (2002) in the ModMinCut Supertree algorithm lead to a biologically reasonable tree (Fig. 7.12). Not only the PACCAD clade and the BEP clade are recovered, but also nearly all subfamilies could be established in the highly resolved tree. In the PACCAD clade, the *Panicoids* are joined by the *Centothecoids* which do not show a monophyletic group. Then the other subfamily subtrees join in the order *Arundinoids*, *Danthonioids*, *Chloridoids*, and *Aristidoids*. Within the BEP clade *Bambusoids* and *Pooids* are sister groups followed by the *Erhartoids*. The early branching taxa branch off from the root in the order *Anomochloids*, *Pharus*, and *Puelioids*.

7.4.5.4. Combined Quartet Tree

The reconstructed phylogeny of the grasses (*Poaceae*) based on the combined quartets is given in Fig. 7.13. Rooted by the outgroup taxa (*Flagellaria*, *Baloskion*, *Elegia*, and *Joinvillea*), the so-called early branching taxa *Puelioids*, *Anomochloids*, and *Pharus* are placed in a multifurcation at the root of the grass subtree. The *Puelioids* and *Anomochloids* form monophyletic groups. Within the grass subtree a group of six subfamilies establishes the so-called PACCAD clade. Three of these subfamilies of the PACCAD clade, the *Aristidoids*, *Arundinoids*, and *Danthonioids*, could be reconstructed, while from the *Panicoids*, *Centothecoids*, *Chloridoids* only parts could be established as a monophyletic group.

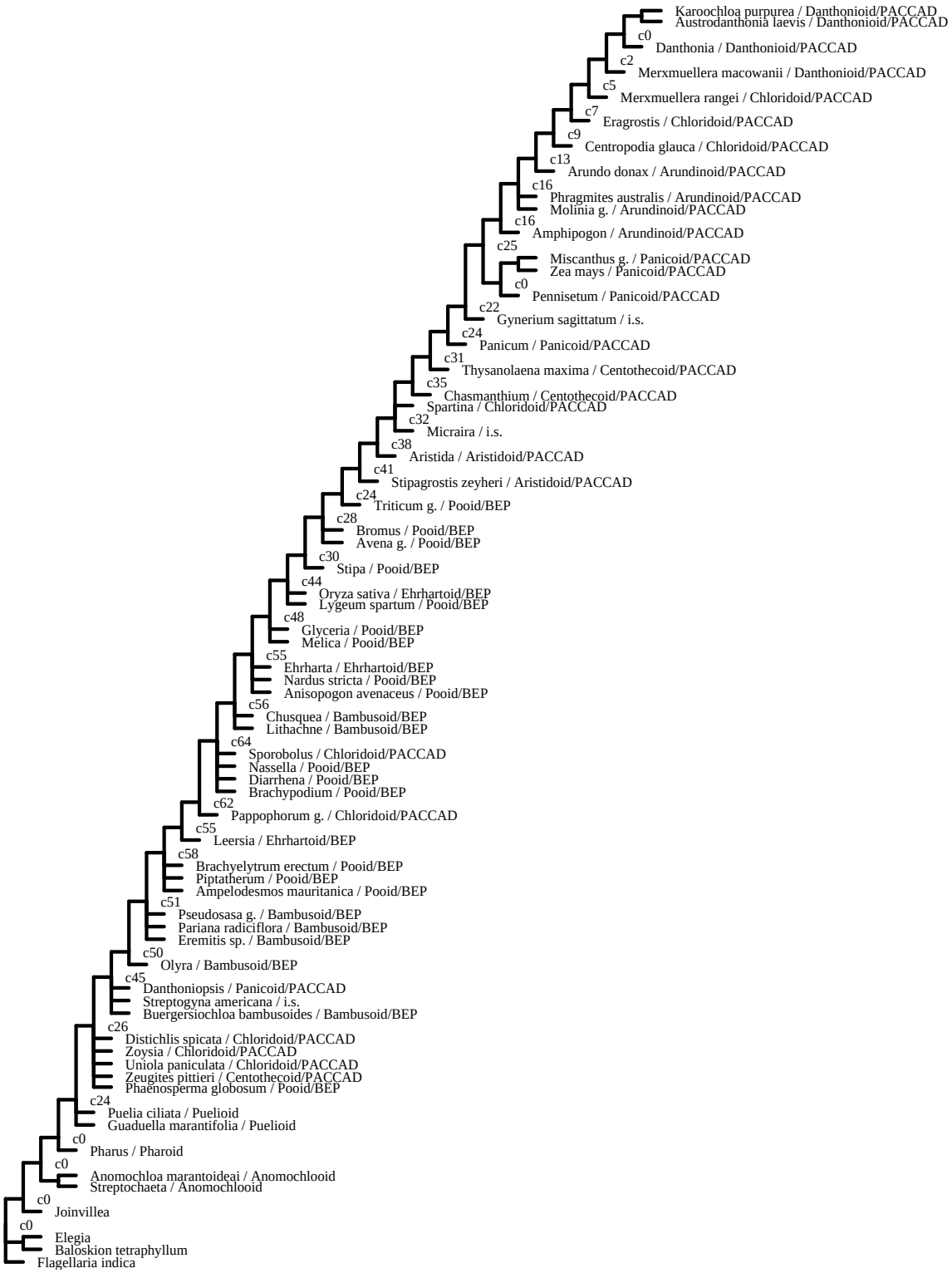


Figure 7.11.: MinCut supertree of the 66 taxa from the *Poaceae* dataset based on the 8 separate ML trees. The numbers at the inner nodes present the number of edges in the minimal cut set.

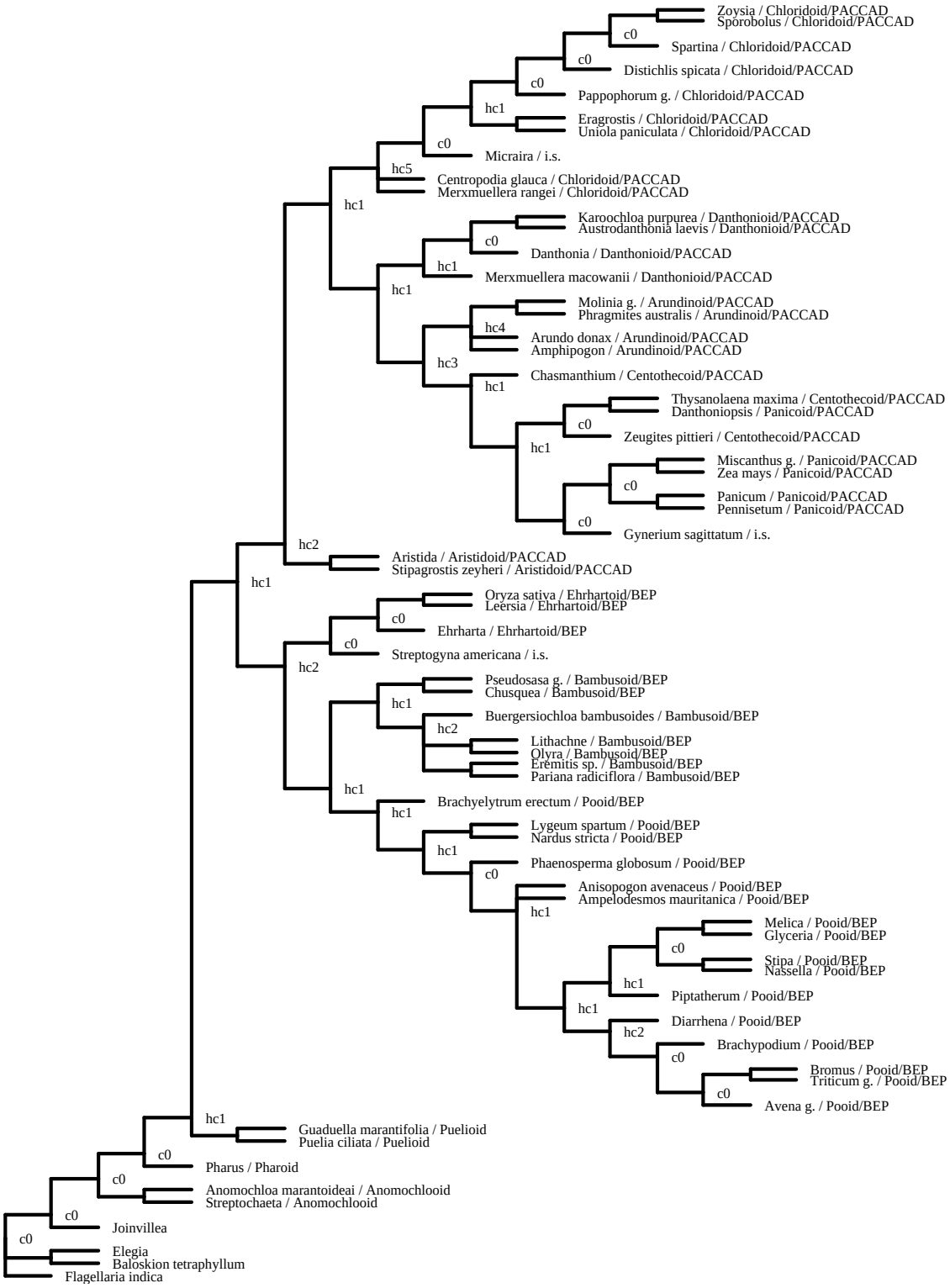


Figure 7.12.: ModMinCut supertree of the 66 taxa from the *Poaceae* dataset based on the 8 separate ML trees. The numbers at the inner nodes present the number of cuts or hierarchical cuts weighted as described by Page (2002).

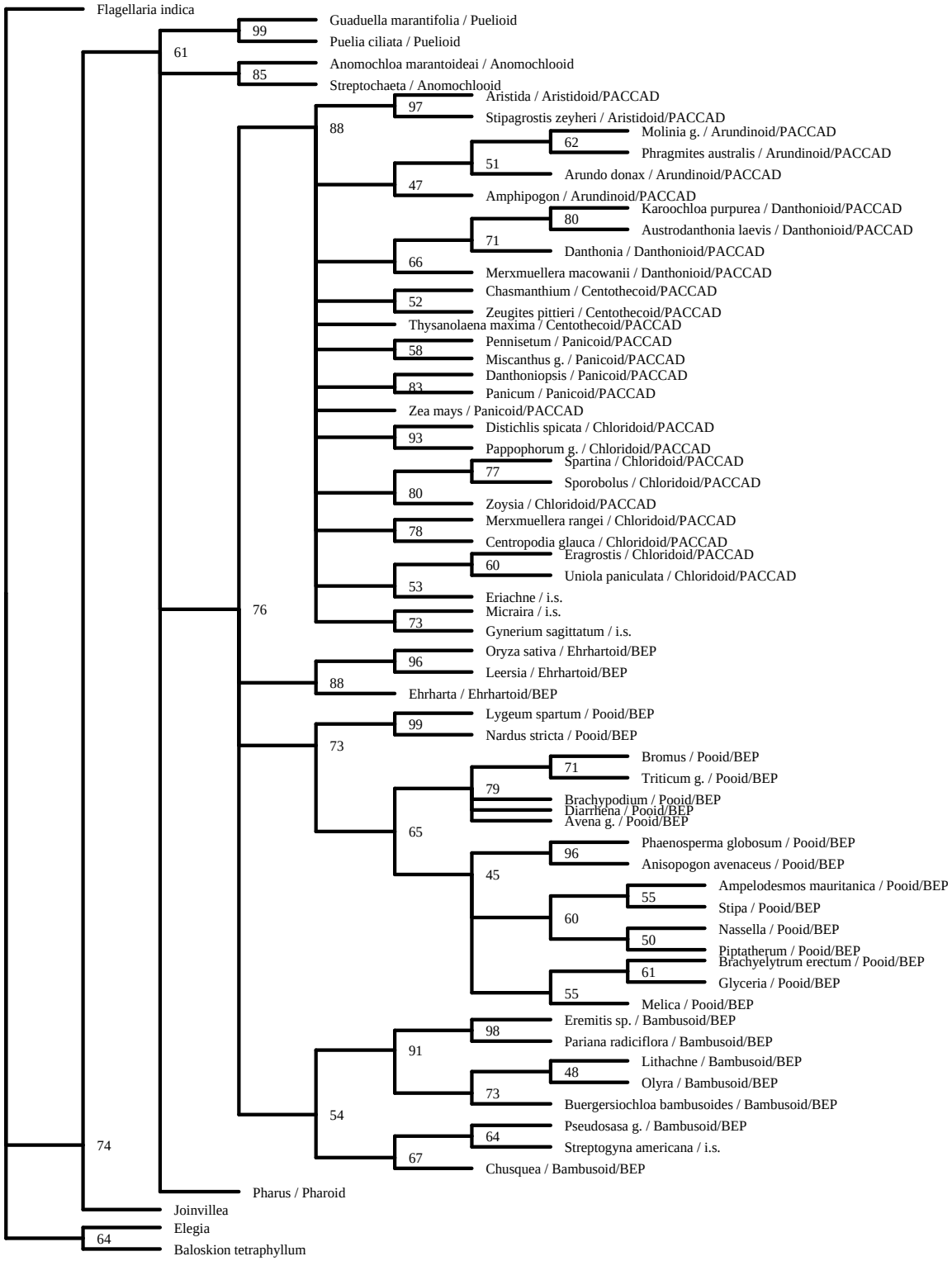


Figure 7.13.: Combined Quartet phylogeny of the 66 taxa from the *Poaceae* dataset. For the analysis the HKY model has been assumed, missing parameters have been estimated separately from the according datasets. The numbers at the internal nodes represent the percentage of the corresponding congruent splits.

From the BEP clade all three subfamilies, *Bambusoids*, *Erhartoids*, and *Pooids*, could be established. The BEP clade, however, could not be reconstructed with the combined quartet method.

Another consensus method, the so-called extended majority consensus, from the CONSENSE program (PHYLIP 3.6a3, Felsenstein, 1993) has been applied by others (Ben-dor *et al.*, 1998; Ranwez and Gascuel, 2001) to construct a binary tree from the set of intermediate trees. The extended majority consensus is a greedy consensus method that adds all splits in descending order of percent occurrence to build a fully resolved tree. All splits that contradict already accepted splits are discarded. We call this approach M_{ext}.

Applying M_{ext} to the set of 100,000 intermediate trees produces a fully resolved tree as anticipated (Fig. 7.14). In addition to the groups resolved by the M_{rel} consensus, this tree reveals all PACCAD subfamilies, with the exception of the placement of *Zea mays*. The M_{ext} consensus tree presents a monophyletic BEP clade with *Bambusoids* and *Erhartoids* being sister groups. The branching order of the early branching taxa is resolved as *Puelioids*, *Anomochloids*, and finally *Pharus*.

7.4.6. Other Published Poaceae Phylogenies

Different versions of the *Poaceae* dataset have been analyzed by others (Grass Phylogeny Working Group, 2001; Salamin *et al.*, 2002). Since the *Poaceae* data is maintained by the GPWG, it has grown over the last years. Therefore, the datasets used in publications differ in size (66 taxa in Grass Phylogeny Working Group, 2001 and 61 taxa in Salamin *et al.*, 2002) as possibly in the distribution of genes per taxon. Nevertheless, the published results will be discussed here, because the authors also used other methods to reconstruct combined phylogenies.

Three trees from the two publications (two *total evidence* trees and one supertree) are sketched in Fig. 7.15. The GPWG *total evidence* tree was reconstructed with *maximum parsimony* from 8 datasets covering 66 taxa. In contrast to our study they used the additional morphology and restriction data. Salamin *et al.* (2002) analyzed the 8 datasets with a total of 61 taxa. Their *total evidence* tree was also constructed by MP, while the MRP supertree (the best out of several different MRP trees) is based on a Baum/Ragan encoding weighted by bootstrap values from the input trees.

All these trees show a similar overall topology having the early branching taxa at the root of the grass subtree. All four support a monophyletic PACCAD clade. Also *Bambusoids*, *Erhartoids*, and *Pooids* are shown as monophyletic subfamilies. Concerning the order of the three subfamilies within the BEP clade, all three possible topologies have been found. The GPWG *total evidence* tree groups together *Bambusoids* and *Ehrhartoids*. The *total evidence* tree of Salamin *et al.* (2002) forms a *Bambusoid/Pooid* group while their supertree supports a cluster of *Ehrhartoids* and *Pooids*. This shows the high amount of uncertainty in the data. The subfamily order within the PACCAD clade remains an open question. All trees exhibit different branching orders, although most of

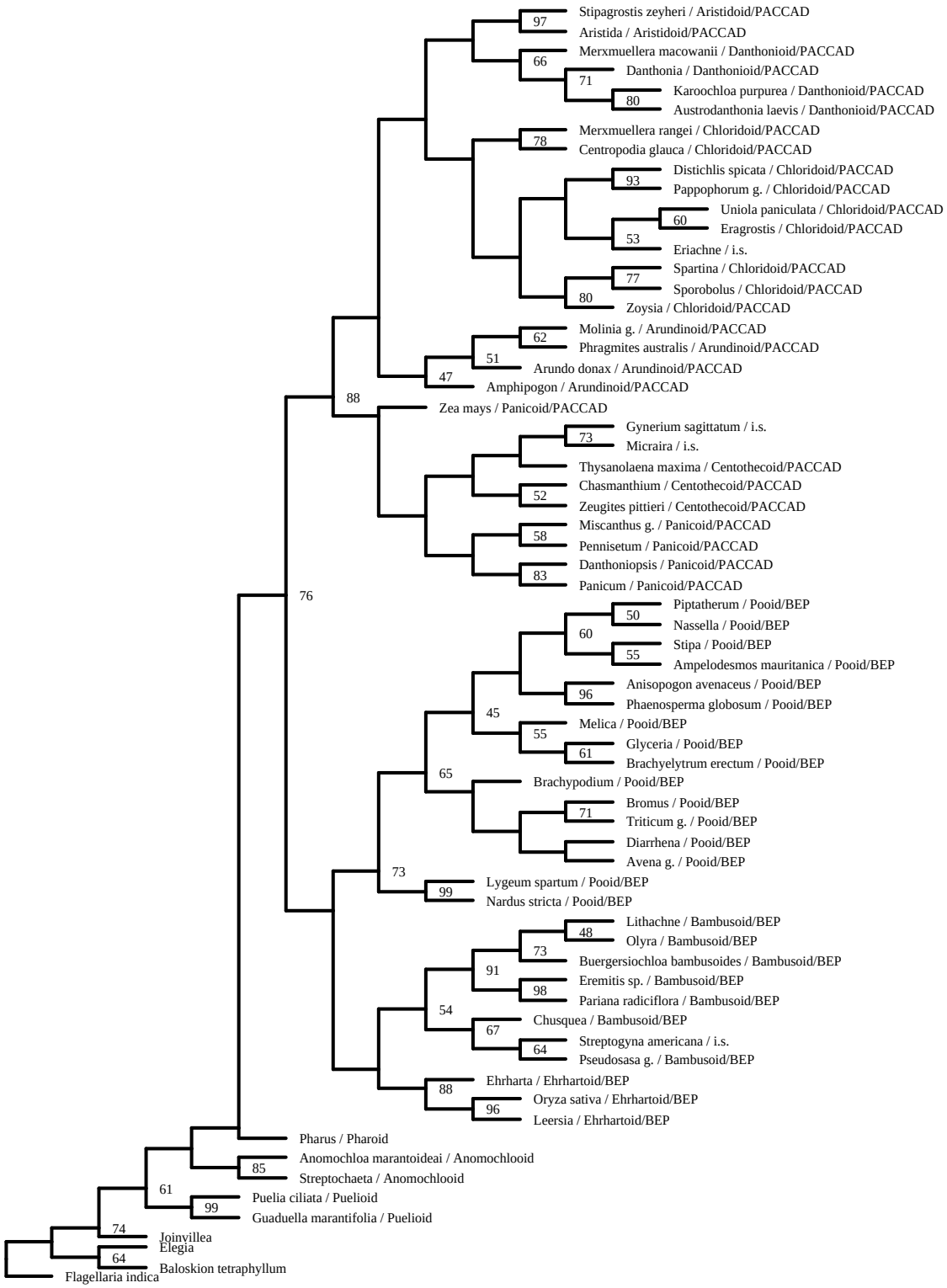


Figure 7.14.: Extended majority consensus of all intermediate trees from the combined reconstruction. The numbers at the internal nodes represent the percentage of the corresponding congruent splits.

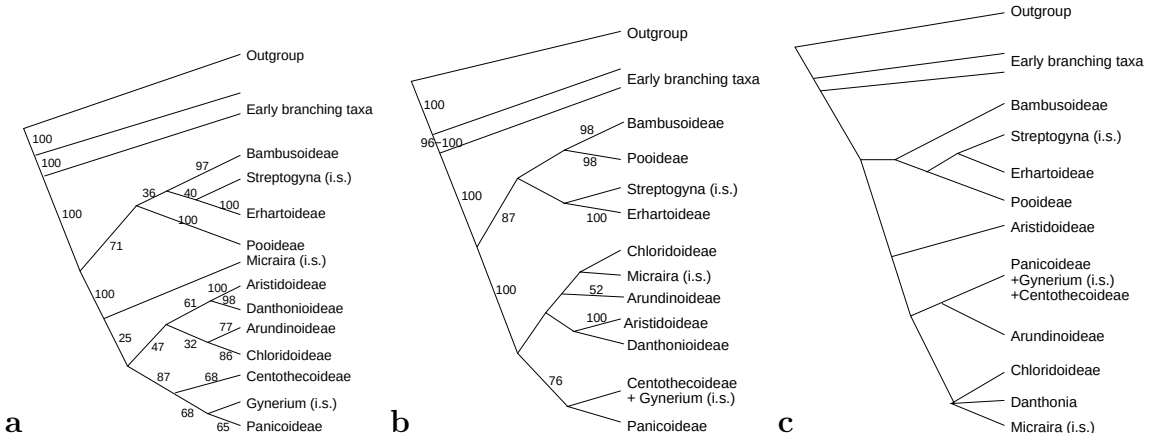


Figure 7.15.: *Poaceae* phylogenies from other publications: **(a)** *total evidence* tree from Grass Phylogeny Working Group (2001); **(b)** *total evidence* tree, and **(c)** supertree from Salamin *et al.* (2002). The numbers represent bootstrap values.

the subfamilies are displayed as monophyletic. In the GPWG and Salamin *et al.*'s *total evidence* trees, *Aristidoids* and *Danthonioids* form sister groups. The same holds for *Centothecoids* and *Panicoids*. The topology of the PACCAD subtree in Salamin *et al.*'s supertree differs strongly from the others.

7.5. Discussion

7.5.1. Problems of Dataset-Combining Methods

There is a long ongoing debate on the question whether *low level* or *high level* combination should be preferred (e.g. de Queiroz *et al.*, 1995, and references therein). A main argument against *total evidence* methods is the problem of how to choose an appropriate model of evolution for the concatenated dataset. It is well-known that different areas of the genome develop under different evolutionary constraints. It is hard if not impossible to choose one appropriate model for different sequences coding for proteins, structural or functional RNAs, and 'function-less' parts of the genome. Missing data also causes problems because many methods discard alignment columns containing gaps. In this case only few or even no alignment columns remain for the analysis.

Since *consensus* and *supertree methods* reconstruct trees separately for each sequence alignment, they can easily use different evolutionary models matching the different constraints of the genesets. *Supertree methods* have been developed to handle the problem of missing data. For the reconstruction of an overall tree, *consensus* and *supertree methods* are using meta data like Matrix Representation derived from the input trees, or they use the trees directly. The sequence data is usually not used in this process. Although this

is an advantage when input trees obtained from publications are combined, it discards valuable information when the underlying sequence data is available.

It has been stated by Bininda-Emonds (2002, 275) that *"the inherent loss of information from using the source trees is a necessary trade-off to be able to combine all possible sources of phylogenetic information."* However, the *medium level method* proposed in this chapter combines the datasets at the quartet level guided by the likelihood of underlying data and, thus, tries to minimize the loss of information in the combination process. Furthermore, the method offers the possibility to use appropriate models of evolution for the different sources of data. Yang (1996) has shown that combined analysis of datasets is possible even if the parameters as well as the branch lengths differ among the datasets. In general, it should also be possible to combine likelihoods of different types of data like DNA, amino acid, and restriction data. A prerequisite for combining such different data sources is the availability of applicable ML models to compute the quartet topologies. The necessity to adequately normalize the substitution models for different types of data remains to be elucidated.

7.5.2. Comparison of the Tree Topologies

The maximum likelihood framework is the usual way to test whether tree topologies are significantly different and which tree might be supported best by the data. A number of possible tests have been suggested (for review see Goldman *et al.*, 2000).

Yet such tests require a model of evolution as well as a dataset that enables a comparison of topologies. In our example only overlapping incomplete genesets are available. Substantial variation has been found among the parameter sets that have been estimated from the genesets (cf. section 7.4.3). Consequently, as discussed above, it might be impossible to choose one parameter set that is adequate for the overall dataset. Moreover, for testing evolutionary hypotheses it is highly questionable to use a concatenated dataset with such a large amount of missing data. The concatenated GPWG dataset contains 37.85% gaps and missing characters, while the amount of ambiguous characters per sequence range from 5.42% (*Zea mays*) up to 76.41% for *Eriachne*.

Since this framework seems not applicable (cf. also Novacek, 2001), the comparison of the presented trees which are based on various optimality criteria relies on the taxonomic classification proposed in a joint effort by a group of experts, the Grass Phylogeny Working Group (2001).

The essence of the classification is the following. The PACCAD clade is monophyletic consisting of the six monophyletic subfamilies *Aristidoids*, *Danthonioids*, *Chloridoids*, *Centothecoids*, *Arundinoids*, *Panicoids* as well as the three species *incertae sedis* (*i.s.*[1]). The species *i.s.* are *Eriachne*, *Micraira*, and *Gynerium*. Three grass subfamilies, namely, *Bambusoids*, *Erhartoids*, and *Pooids* are reported to form a monophyletic group for different datasets, the so-called BEP clade. Finally, three early branching subfamilies, the

[1]latin for 'of uncertain placement'

Anomochloids, *Pharoids*, and *Puelioids* were described. This comprehensive subfamily classification consists of eleven previously published subfamilies and the new *Danthonioid* subfamily (see Grass Phylogeny Working Group, 2001). This taxonomic classification is based on the review of a large number of publications about *Poaceae* as well as a *Total evidence* tree and the separate genetrees from the GPWG dataset.

Examining the reconstructed trees shows that most methods can more or less resolve the taxonomic structure of the grasses. Taxa from the unresolved subfamilies are generally placed in multifurcations which do not contradict the taxonomic classification. Only MinCut Supertree and MRP-Pu distribute members of unrevealed subfamilies throughout the constructed tree. The methods which where able to reconstruct best the grass subfamilies, PACCAD, and BEP clade were ModMinCut Supertree, the M_{ext} consensus from the combined quartet intermediate trees, and the two MP-based *total evidence* trees (Fig. 7.15).

Although the M_{ext} consensus gave a biologically reasonable result, its application should not generally be recommended. It has been suggested by Kluge and Kolf (1993) that bold, i.e., completely resolved, hypothesis are to be preferred over 'safe' consensus trees, since the safest tree would be the unresolved one. In order to achieve a completely resolved tree, however, the M_{ext} consensus accepts splits which would have been outvoted by more abundant splits if the latter would not have been discarded due to earlier contradictions. Using the M_{rel} consensus instead avoids this problem, since it does not resolve the topology beyond the first incongruence. This agrees more with de Queiroz *et al.* (1995, 664), stating that *"one might want a tree in which all clades have a certain level of support."*

7.5.3. Conclusion

We have presented a new quartet-based *medium level method* to combine multiple sequence datasets with missing data for phylogeny reconstruction. The results obtained from the GPWG *Poaceae* dataset show that this method performs comparable to other methods for phylogeny reconstruction from overlapping datasets with missing data. Although the resulting tree was not fully resolved, it did not contradict the taxonomic classification by the Grass Phylogeny Working Group (2001).

The modifications to the MinCut Supertree suggested by Page (2002, ModMinCut Supertree) have shown to substantially increase the result (cf. 7.4.5.3). Yet, none of the method to analyze incomplete phylogenetic data presented here has proven to be the final choice. Facing the problem that it is unlikely to have complete molecular and morphological datasets in the near future, methods combining overlapping incomplete datasets remain a valuable tool to reconstruct evolution from multiple datasets (cf. also Bininda-Emonds, 2002). The development and improvement of methods for the combined analysis of multiple dataset should be encouraged. Tree reconstruction using *combined quartets* provides a new tool for such work.

The judgment of the efficiency of dataset combining methods highly relies on biological knowledge. Since the 'true' tree is unknown in general for these complex types of datasets, we suggest to perform simulations to study the reliability of such methods. Furthermore, the influence of composite taxa should be examined in more detail. Recently, Malia Jr. *et al.* (2003) presented results on misleading effects of composite taxa on *total evidence* trees.

The *medium level* approach using combined quartets also offers a possibility to analyze heterogeneous datasets as follows. Fast evolving datasets might not be suitable to reconstruct deep phylogenies due to saturation, but might contain valuable information of the relationship of closely related species. Such datasets can be decomposed into separate group-specific 'genesets' which are then connected by datasets from slower evolving genes.

In addition to tree reconstruction, the set of combined quartet tree likelihoods can also be used for *likelihood mapping* analysis to visualize, for instance, the combined quartet support for specified cluster relationships (cf. 3.2.1).

8. Summary

Efficient sequencing techniques lead to exponentially growing amounts of molecular sequence data resulting mainly in two kinds of large datasets: vertically (large number of taxa) as well as horizontally large datasets (long or many sequences per taxon). The eminent need for efficient methods to analyze such datasets motivated the development and improvement of several approaches based on the *Quartet Puzzling* algorithm which were proposed in this thesis.

Accelerated Puzzling Step (chapter 4): Two efficient $O(n^4)$ algorithms, one split-based and a recursive algorithm, were developed to accelerate the *puzzling step* since it demands a major runtime portion of the tree reconstruction by *Quartet Puzzling*. Runtime tests show that the new methods outperform the original as well as Berry's algorithm. The best performing algorithm is based on the recursive computation of the edge penalties. It requires about half the runtime of the original $O(n^5)$ algorithm.

Parallelized Quartet Puzzling (chapter 5): To reduce the overall waiting time for the reconstruction of large trees, a parallel implementation was developed. For the parallel workflow we investigated several scheduling algorithms. At last, a GSS-based scheduling algorithm, modified to our needs, was devised to produce an even workload on heterogeneous workstation clusters which are the most abundant parallel platforms in molecular biology. Benchmark tests of the parallel implementation showed convincing scaling behavior for large datasets.

Phylogenetic Trees from Less Quartets (chapter 6): A second component of the *QP* algorithm which demands a large part of computation time is the *ML step*. An approach is suggested to reduce the number of required quartets. To that end the dataset is decomposed into overlapping subsets from which the sets of possible quartets are assembled into an overall tree applying a modified *puzzling step* algorithm called ModPUZZLE. Results show that the resolution one can gain depends on the overlap between the subsets which can be determined by the number of quartets one is willing to compute. The approach can be applied to reconstruct the rough tree topology of large datasets.

Combined Quartet Method for Overlapping Datasets (chapter 7): We presented a procedure to reconstruct overlapping datasets from different genes (genesets) with

missing data. In contrary to *total evidence* approaches our method combines the datasets on a so-called *medium level* allowing to use evolutionary parameters specific for each geneset. The combination of the data is performed on the quartet level guided by the likelihoods computed from the sequences. Proceeding this way tries to minimize the loss of primary information before combining the genesets, a common criticism of the *supertree* methods. Different methods to combine genesets were applied to the biological dataset of the grasses. With regard to the GPWG taxonomic classification of the grasses, the combined quartet method performs comparable to the other methods.

A. Variables and Functions

$\overleftarrow{\urcorner}_e$ – taxa, on the left of edge e

$\overrightarrow{\ulcorner}_e$ – taxa, on the right of edge e

$\overleftarrow{r\urcorner}_e$ – taxa, on the same side as the root of edge e

${}_r\overrightarrow{\ulcorner}_e$ – taxa, not on the same side as the root of edge e

$\emptyset$ – empty set

α – shape parameter of a Γ-distribution

$\mathbb{A}$ – character alphabet

a – a leaf label

A – alignment

A_c – column c of alignment A

A_{cj} – character state in alignment column c of sequence j

ABS – atomic batch size (scheduling)

b – a leaf label

$B(n)$ – number of possible binary trees with n leaves

c – a leaf label

$C^e(\ldots)$ – cluster of edge ...

$C^n(\ldots)$ – cluster of node ...

$CLP(\ldots)$ – sum of all penalty paths beginning at any leaf in cluster ...

d – a leaf label

D – square distance

$\boldsymbol{D}$ – distance matrix

$Deg(\ldots)$ – degree of node ...

ε – an edge of the *penalty path*

e – an edge

$e_{(\bullet,a)}$ – edge ending at node a in a rooted tree

$e_{(a,b)}$ – edge connecting nodes a and b

$\mathcal{E}$ – set of edges

$\mathcal{E}_{min}$ – edge set with minimum penalty

$\mathcal{E}_{min}(T)$ – edge set with minimum penalty in tree T

$\boldsymbol{E}$ – external node penalty vector

$E(\ldots)$ – penalty induced by external node ...

EMP – evolutionary Markov process

$\mathcal{F}$ – front set of nodes

g – a gene; a geneset index

$G = (\mathcal{V}, \mathcal{E})$ – graph

G_{ovl} – overlap graph

GST – glutathione S-transferase

h – height of a node

$h_{\max}$ – maximal height in a tree

H – hypothesis

i – general purpose variable; insertion counter

j – general purpose variable

k – number of subsets/genesets

ℓ_T – log-likelihood of T

$\mathcal{L}(T)$ – leaf set of tree T

l – a node or leaf

$L(\ldots)$ – likelihood of ...

$L_i(j)$ – likelihood of state j at node i

L_T – likelihood of T

$L_T^{A_c}$ – likelihood of T in alignment column A_c

μ – calibration factor for substitution matrix $\boldsymbol{Q}$

m – number of sites, sequence length

M – evolutionary model

M_l – l-majority consensus

M_{ext} – extended majority consensus (PHYLIP)

M_{rel} – relative majority consensus

MRCA(...) – most recent common ancestor of ...

ν – size of subsets

n – number of sequences

$\boldsymbol{N}$ – penalty neighborhood matrix

$N_{a,b}$ – entry of matrix $\boldsymbol{N}$

NT – number of tasks (scheduling)

NP – number of processes (scheduling)

NR – number of un-scheduled tasks (scheduling)

o – overlap between subsets

$\Omega(\ldots)$ – complexity measure, lower bound

$O(\ldots)$ – complexity measure, upper bound

$\Phi(\ldots)$ – penalty of edge ...

π – frequency vector, stationary distribution

π_i – frequency of state i

p_i – Bayesian weight of topology i

$\boldsymbol{P}$ – substitution probability matrix

$P_{\rho\sigma}$ – entry of $\boldsymbol{P}$ for substitution $\rho \rightarrow \sigma$

$P_{\rho\sigma}(t)$ – probability of substitution $\rho \rightarrow \sigma$ in time t

$Pr(\ldots)$ – probability of ...

q – quartet

$\mathcal{Q}$ – set of all possible quartets from $mathcalS$

$\mathcal{Q}_i$ – set of all possible quartets from $mathcalS_i$

$\mathcal{Q}_{(X,T_i)}$ – set of all quartets containing X and three sequences from T_i

$\boldsymbol{Q}$ – instantaneous rate matrix

Q_{ij} – entry of $\boldsymbol{Q}$ for substitution $i \rightarrow j$

ρ – a character state

$\boldsymbol{R}$ – substitution rate matrix

r – a node or leaf

R – root node

R_{ij} – entry of $\boldsymbol{R}$ for substitution $i \rightarrow j$

σ – a character state

$\mathcal{S}$ – set of sequences

$\mathcal{S}_i$ – subset of sequences

$\mathcal{S}_{1\cap 2}$ – intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$

s_i – ith sequence

$S_{\mathcal{T}}$ – graph from the input trees $\mathcal{T}$ (MinCut)

$S_{\mathcal{T}} \backslash E_{\mathcal{T}}^{Max}$ – graph from the input trees $\mathcal{T}$ without $E_{\mathcal{T}}^{Max}$ (MinCut)

$SMS(\ldots)$ – sum of all penalty paths beginning at any leaf in cluster ..., while remaining within the cluster

succ(...) – direct descendants of node ...

$\Theta(\ldots)$ – complexity measure, upper and lower bound

t – time

$t_{(a,b)}$ – length of edge $e_{(a,b)}$ in evolutionary time

t_{MRCA} – runtime (MRCA-based puzzling step)

t_{orig} – runtime (original puzzling step)

t_{recur} – runtime (recursive puzzling step)

t_{split} – runtime (split-based puzzling step)

T – a tree, tree topology

$\mathcal{T}$ – set of trees

$\mathcal{T}|S_i$ – the trees from $\mathcal{T}$ with all leaves $r \notin S_i$ pruned (MinCut)

$T(\mathcal{S})$ – tree containing leaf set $\mathcal{S}$

T_n – n-tree, tree with n leaves

T_{nj} – neighbor-joining tree

u – a node or leaf

v – a node or leaf

$\boldsymbol{V}$ – penalty propagation vector

$\mathcal{V}$ – set of nodes/vertices

w – a node or leaf

w_i – generic weight of topology i

x – general purpose variable; index

X – taxon to insert next

Ξ – current status of a scheduler

y – general purpose variable; index

z – general purpose variable; index

B. Abbreviations

aa – amino acids

A – Adenine

ANSI – American National Standard Institute

BR – Baum/Ragan encoding

C – Cytosine

COW – Cluster of Workstations

cpREV – reversible model for chloroplast data

CPU – Central Processing Unit, Processor

DNA – Deoxyribonucleic Acid

ED – Evolutionary Distances

EF – Elongation Factor

EMBL – European Molecular Biology Laboratories

EMP – evolutionary Markov process

G – Guanine

GBSSI – Granule Bound Starch Synthase I

GPWG – Grass Phylogeny Working Group

GTR – General Time Reversible model

GSS – Guided Self-Scheduling

HKY – Hasegawa-Kishino-Yano model

HKY85 – Hasegawa-Kishino-Yano model

Indel – Insertions and Deletions

i.s. – incertae sedis (lat.) – of uncertain placement

ITS – internal transcribed spacer

JC69 – Jukes-Cantor model

JTT – Jones-Taylor-Thornton model

K2P – Kimura 2-Parameter model

`lslength` – least square branch length estimation

LS len – least square branch length estimation

MIMD – Multiple Instruction – Multiple Data

MinCut – Minimum Cut

ML – Maximum Likelihood

`mldistance` – ML distance estimation

ML dist – ML distance estimation

ModMinCut – Modified Minimum Cut

MP – Maximum Parsimony

MPI – Message Passing Interface

MRCA – Most Recent Common Ancestor

MRF – Matrix Representation using Flipping

MRP – Matrix Representation using Parsimony

MSA – Multiple Sequence Alignment

MST – Minimum Spanning Tree

mtREV – reversible model for mitochondrial data

nt – nucleotides

ndhF – NADH dehydrogenase, subunit F

PC – Personal Computer

PE – Processing Element

phyB – phytochrome B

Pu – Purvis' encoding

QP – Quartet Puzzling

rbcL – ribulose 1,5-bisphosphate carboxylase/oxygenase, large subunit

RDP – Ribosomal Database Project

RNA – Ribonucleic acid

rRNA – ribosomal ribonucleic acid

rpoC – RNA polymerase II, β" subunit

rpoC2 – RNA polymerase II, β" subunit

SC – Static Chunking (scheduling)

SGSS – Smooth Guided Self-Scheduling

SIMD – Single Instruction – Multiple Data

SMP – Scalable Multi-Processor

SPRNG – Scalable Parallel Random Number Generator

SS – Self-Scheduling

ssu rRNA – small subunit rRNA

T – Thymine

TN93 – Tamura-Nei model

ts – transition

TSS – Trapezoid Self-Scheduling

tv – transversion

U – Uracil

UV – ultraviolet

VT – Variable Time model

WAG – Whelan-and-Goldman model

Bibliography

Adachi, J. and Hasegawa, M. (1996a) Model of amino acid substitution in proteins encoded by mitochondrial DNA. *J. Mol. Evol.*, **42**, 459–468.

Adachi, J. and Hasegawa, M. (1996b) *MOLPHY Version 2.3 – Programs for Molecular Phylogenetics Based on Maximum Likelihood*, vol. 28 of *Computer Science Monographs*. Institute of Statistical Mathematics, Minato-ku, Tokyo.

Adachi, J., Waddell, P. J., Martin, W. and Hasegawa, M. (2000) Plastid genome phylogeny and a model of amino acid substitution for proteins encoded by chloroplast DNA. *J. Mol. Evol.*, **50**, 348–358.

Adams III, E. N. (1986) N-trees as nestings: Complexity, similarity, and consensus. *J. Classif.*, **3**, 299–317.

Aho, A. V., Sagiv, Y., Szymanski, T. G. and Ullman, J. D. (1981) Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.*, **10**, 405–421.

Ansorge, W., Zimmermann, J., Erfle, H., Hewitt, N., Rupp, T., Schwager, C., Sproat, B., Stegemann, J. and Voss, H. (1993) Sequencing reactions for ALF (EMBL) automated DNA sequencer. *Methods Mol. Biol.*, **23**, 317–356.

Bandelt, H.-J. and Dress, A. (1986) Reconstructing the shape of a tree from observed dissimilarity da ta. *Adv. Appl. Math.*, **7**, 309–343.

Barthelemy, J. P. and Guenoche, A. (1991) *Trees and Proximity Representations*. Interscience Series in Discrete Mathematics and Optimization, Wiley, New York.

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

Baxevanis, A. D. (2003) The molecular biology database collection: 2003 update. *Nucleic Acids. Res.*, **31**, 1–12.

Baxevanis, A. D., Davison, D. B., Page, R. D. M., Stormo, G. and Stein, L. (eds.) (2002) *Current Protocols in Bioinformatics*. Wiley and Sons, New York, USA.

Ben-dor, A., Chor, B., Graur, D., Ron, O. and Pelleg, D. (1998) Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships. *J. Comput. Biol.*, **5**, 377–390.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. and Wheeler, D. L. (2003) GenBank. *Nucleic Acids Res.*, **31**, 23–27.

Bininda-Emonds, O. R. P. (2002) MRP supertree construction in the consensus setting. In *Bioconsensus: Proceedings of Tutorial and Workshop on Bioconsensus II*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, page N.N.

Blanchette, M., Bourque, G. and Sankoff, D. (1997) Breakpoint phylogenies. In *Proceedings of the Genome Informatics Workshop VIII*, pages 25–34, Universal Academy Press.

Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.-C., Estreicher, A., Gasteiger, E., Martin, M. J., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S. and Schneider, M. (2003) The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res.*, **31**, 365–370.

Bremer, K. (1990) Combinable component consensus. *Cladistics*, **6**, 369–372.

Bryant, D. and Steel, M. (1995) Extension operations on sets of leaf-labeled trees. *Adv. Appl. Math.*, **16**, 425–453.

Chen, D., Diao, L., Eulenstein, O., Fernández-Baca, D. and Sanderson, M. J. (2003) Flipping: A supertree construction method. *DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences*, **000**, to appear.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001) *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, Second edn..

Cramer, J. S. (1989) *Econometric Applications of Maximum Likelihood Methods*. Cambridge University Press, Cambridge, UK.

Day, W. H. E. (1987) Computational complexity of inferring phylogenies from dissimilarity matrices. *Bull. Math. Biol.*, **49**, 461–467.

Day, W. H. E. and Sankoff, D. (1986) Computational complexity of inferring phylogenies by compatibility. *Syst. Zool.*, **35**, 224–229.

Dayhoff, M. O., Schwartz, R. M. and Orcutt, B. C. (1978) A model of evolutionary change in proteins. In Dayhoff, M. O. (ed.), *Atlas of Protein Sequence Structure*, vol. 5, pages 345–352, National Biomedical Research Foundation, Washington DC.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977) Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statist. Soc.*, **39**, 1–39.

Dress, A., von Haeseler, A. and Krüger, M. (1986) Reconstructing phylogenetic trees using variants of the four-point condition. *Studien zur Klassifikation*, **17**, 299–305.

Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. (1998) *Biological sequence analysis - Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge.

Duret, L., Mouchiroud, D. and Gouy, M. (1994) HOVERGEN, a database of homologous vertebrate genes. *Nucleic Acids Res.*, **22**, 2360–2365.

Edman, P. (1950) Method for determinatin of the amino acid sequence in peptides. *Acta Chem. Scand.*, **4**, 283–290.

Edman, P. and Begg, G. (1967) A protein sequenator. *Eur. J. Biochem.*, **1**, 80–91.

Edmiston, E. and Wagner, R. A. (1987) Parallelization of the dynamic programming algorithm for comparison of sequences. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 78–80, Pennsylvania, Penn State Press.

El-Rewini, H., Lewis, T. G. and Ali, H. H. (1994) *Task Scheduling in Parallel and Distributed Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

Eliason, S. R. (ed.) (1993) *Maximum Likelihood Estimation: Logic and Practice*. Quantitative Applications in the Social Sciences, Sage Publications, Newbury Park.

Ewens, W. J. and Grant, G. R. (2001) *Statistical Methods in Bioinformatics: An Introduction*. Springer Verlag, New York, USA.

Felsenstein, J. (1978) The number of evolutionary trees. *Syst. Zool.*, **27**, 27–33.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, **17**, 368–376.

Felsenstein, J. (1993) *PHYLIP (Phylogeny Inference Package) version 3.5c*. Department of Genetics, University of Washington, Seattle, Distributed by the author.

Fitch, W. M. (1981) A non-sequential method for constructing trees and hierarchical classifications. *J. Mol. Evol.*, **18**, 30–37.

Foulds, L. R. and Graham, R. L. (1982) The Steiner problem in phylogeny is NP-complete. *Adv. Appl. Math.*, **3**, 43–49.

Goldman, N. (1990) Maximum likelihood inference of phylogenetic trees, with special reference to a poisson process model of DNA substitution and to parsimony analyses. *Syst. Zool.*, **39**, 345–361.

Goldman, N., Anderson, J. P. and Rodrigo, A. G. (2000) Likelihood-based tests of topologies in phylogenetics. *Syst. Biol.*, **49**, 652–670.

Graham, R. L. and Foulds, L. R. (1982) Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Math. Biosci.*, **60**, 133–142.

Grass Phylogeny Working Group (2001) Phylogeny and subfamilial classification of the grasses (poaceae). *Ann. Mo. Bot. Gard.*, **88**, 373–457.

Graur, D. and Li, W.-H. (2000) *Fundamentals of Molecular Evolution*. Sinauer Associates, Sunderland, Massachusetts, Second edn..

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W. and Snir, M. (1998) *MPI: The Complete Reference - The MPI Extensions*, vol. 2. The MIT Press, Cambridge, Massachusetts, Second edn..

Gu, X., Fu, Y.-X. and Li, W.-H. (1995) Maximum likelihood estimation of the heterogeneity of substitution rate among nucleotide sites. *Mol. Biol. Evol.*, **12**, 546–557.

Hagerup, T. (1997) Allocating independent tasks to parallel processors: An experimental study. *J. Parallel Distrib. Comput.*, **47**, 185–197.

Hagstrom, R., Matsuda, H., Overbeek, R., Olsen, G. and Woese, C. (1992) Inferring relationships among microorganisms by using maximum likelihood. The Concurrent Supercomputing Consortium Annual Report FY1991-1992, California Institute of Technology, Pasadena, CA, USA.

Hasegawa, M., Kishino, H. and Yano, T.-A. (1985) Dating of the human–ape splitting by a molecular clock of mitochondrial DNA. *J. Mol. Evol.*, **22**, 160–174.

Huang, X. (1989) A space-efficient parallel sequence comparison algorithm for a message-passing multiprocessor. *Int. J. Parallel Program.*, **18**, 223–239.

Huelsenbeck, J. P. and Ronquist, F. (2001) MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, **17**, 754–755.

Huson, D. H., Nettles, S. M. and Warnow, T. J. (1999) Disk-covering, a fast-converging method for phylogenetic reconstruction. *J. Comput. Biol.*, **6**, 369–386.

Jones, D. T., Taylor, W. R. and Thornton, J. M. (1992) The rapid generation of mutation data matrices from protein sequences. *Comput. Appl. Biosci.*, **8**, 275–282.

Jukes, T. H. and Cantor, C. R. (1969) Evolution of protein molecules. In Munro, H. N. (ed.), *Mammalian Protein Metabolism*, vol. 3, pages 21–123, Academic Press, New York.

Jülich, A. (1995) Implementations of BLAST for parallel computers. *Comput. Appl. Biosci.*, **11**, 3–6.

Kimura, M. (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, **16**, 111–120.

Kleinjung, J., Douglas, N. and Heringa, J. (2002) Parallelized multiple alignment. *Bioinformatics*, **18**, 1270–1271.

Kluge, A. G. and Kolf, A. J. (1993) Cladistics: What's in a word? *Cladistics*, **9**, 183–199.

Knuth, D. E. (1997) *Sorting and Searching*, vol. 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, Third edn..

Krause, A., Haas, S. A., Coward, E. and Vingron, M. (2002) SYSTERS, GeneNest, SpliceNest: Exploring sequence space from genome to protein. *Nucleic Acids Res.*, **30**, 299–300.

Lander, E., Mesirov, J. P. and W., T. (1988) Protein sequence comparison on a data parallel computer. In *Proceedings of the 1988 International conference on Parallel Processing*, pages 257–263, Pennsylvania, Penn State Press.

Lewis, P. O. (1998) A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Mol. Biol. Evol.*, **15**, 277–283.

Liébecq, C. (ed.) (1992) *Biochemical Nomenclature and Related Documents*. Portland Press, London, Second edn..

Liu, J., Iba, H. and Ishizuka, M. (2001) Selecting informative genes with parallel genetic algorithms in tissue classification. In *Proceedings of the Genome Informatics Workshop XII*, vol. 12 of *Genome Informatics*, pages 14–23, Universal Academy Press.

Maidak, B. L., Cole, J. R., Lilburn, T. G., Parker Jr., C. T., Saxman, P. R., Farris, R. J., Garrity, G. M., Olsen, G. J., Schmidt, T. M. and Tiedje, J. M. (2001) The RDP-II (ribosomal database project). *Nucleic Acids Res.*, **29**, 173–174.

Malia Jr., M. J., Lipscomb, D. L. and Allard, M. W. (2003) The misleading effects of composite taxa in supermatrices. *Mol. Phylogenet. Evol.*, **27**, 522–527.

Margush, T. and McMorris, F. R. (1981) Consensus n-trees. *Bull. Math. Biol.*, **43**, 239–244.

McMorris, F. R. and Neumann, D. A. (1983) Consensus functions defined on trees. *Math. Soc. Sci*, **4**, 131–136.

Mendel, J. M. and Burrus, C. S. (1990) *Maximum-Likelihood Deconvolution: A Journey into Model-Based Signal Processing*. Springer Verlag, New York.

Meyer, S. and von Haeseler, A. (2003) Identifying site-specific substitution. *Mol. Biol. Evol.*, **20**, 182–189.

Meyer, S., Weiss, G. and von Haeseler, A. (1999) Pattern of nucleotide substitution and rate heterogeneity in the hypervariable regions I and II of human mtDNA. *Genetics*, **152**, 1103–1110.

Morgenstern, B. (1999) DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, **15**, 211–218.

Müller, T. and Vingron, M. (2000) Modeling amino acid replacement. *J. Comput. Biol.*, **7**, 761–776.

Nakaya, A., Yamamoto, K. and Yonezawa, A. (1995) RNA secondary structure prediction using highly parallel computers. *Comput. Appl. Biosci.*, **11**, 685–692.

Ness, S. R., Terpstra, W., Krzywinski, M., Marra, M. A. and Jones, S. J. M. (2002) Assembly of fingerprint contigs: Parallelized FPC. *Bioinformatics*, **18**, 484–485.

Novacek, M. J. (2001) Mammalian phylogeny: Genes and supertrees. *Curr. Biol.*, **11**, R573–R575.

Olsen, G. J., Matsuda, H., Hagstrom, R. and Overbeek, R. (1994) fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, **10**, 41–48.

Ottmann, T. and Widmayer, P. (2002) *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, Heidelberg, Fourth edn..

Page, R. D. M. (2002) Modified mincut supertrees. In *Proceedings of the 2nd Workshop on Algorithms in Bioinformatics (WABI 2002)*, vol. 2452 of *Lecture Notes in Computer Science*, pages 537–551, New York, Springer.

Page, R. D. M. and Holmes, E. C. (1998) *Molecular Evolution: A Phylogenetic Approach*. Blackwell Science, Oxford.

Pedretti, K. T., Casavant, T. L., Braun, R. C., Scheetz, T. E., Birkett, C. L. and Roberts, C. A. (1999) Three complementary approaches to parallelization of local BLAST service on workstation clusters. In *Proceedings of the 5th International Conference on Parallel Computing Technologies (PaCT-99)*, vol. 1662 of *Lecture Notes in Computer Science*, pages 271–282, New York, Springer.

Van de Peer, Y., Baldauf, S. L., Doolittle, W. F. and Meyer, A. (2000a) An updated and comprehensive rRNA phylogeny of (crown) eukaryotes based on rate-calibrated evolutionary distances. *J. Mol. Evol.*, **51**, 565–576.

Van de Peer, Y., De Rijk, P., Wuyts, J., Winkelmans, T. and De Wachter, R. (2000b) The European small subunit ribosomal RNA database. *Nucleic Acids Res.*, **28**, 175–176.

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1988) *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge.

Purvis, A. (1995) A composite estimate of primate phylogeny. *Philos. Trans. R. Soc. Lond. Ser. B*, **348**, 405–421.

de Queiroz, A., Donoghue, M. J. and Kim, J. (1995) Separate versus combined analysis of phylogenetic evidence. *Annu. Rev. Ecol. Syst.*, **26**, 657–681.

Ragan, M. A. (1992) Phylogenetic inference based on matrix representation of trees. *Mol. Phylogenet. Evol.*, **1**, 53–58.

Rambaut, A. and Grassly, N. C. (1997) Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, **13**, 235–238.

Ranwez, V. and Gascuel, O. (2001) Quartet-based phylogenetic inference: Improvements and limits. *Mol. Biol. Evol.*, **18**, 1103–1116.

Robinson-Rechavi, M. and Graur, D. (2001) Usage optimization of unevenly sampled data through the combination of quartet trees: An eutherian draft phylogeny based on 640 nuclear and mitochondrial proteins. *Isr. J. Zool.*, **47**, 259–270.

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

Saitou, N. and Nei, M. (1987) The neighbor–joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.

Salamin, N., Hodkinson, T. R. and Savolainen, V. (2002) Building supertrees: An empirical assessment using the grass family (poaceae). *Syst. Biol.*, **51**, 136–150.

Salter, L. A. and Pearl, D. K. (2001) Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. *Syst. Biol.*, **50**, 7–17.

Sanderson, M. J., Purvis, A. and Henze, C. (1998) Phylogenetic supertrees: Assembling the trees of life. *TREE*, **13**, 105–109.

Sanger, F., Nicklen, S. and Coulsen, A. R. (1977) DNA sequencing with chain–terminating inhibitors. *Proc. Natl. Acad. Sci. USA*, **74**, 5463–5467.

Sanger, F. and Thompson, E. O. P. (1953) The amino-acid sequence in the glycyl chain of insulin: 1. the identification of lower peptides from partial hydrolysates. *Biochem. J.*, **53**, 353–365.

Sankoff, D. and Blanchette, M. (1998) Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.*, **5**, 555–570.

Sattath, S. and Tversky, A. (1977) Additive similarity trees. *Psychometrika*, **42**, 319–345.

Schmidt, H. A. (1996) *Parallelisierung phylogenetischer Methoden zur Untersuchung der Crown-Group-Radiation*. Diplomarbeit, Universität zu Köln.

Schmidt, H. A. and von Haeseler, A. (2002) Quartet trees as a tool to reconstruct large trees from sequences. In Jajuga, K., Sokołowski, A. and Bock, H.-H. (eds.), *Data Analysis, Classification, and Related Methods*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 379–388, Springer, Heidelberg, New York.

Schmidt, H. A. and von Haeseler, A. (2003) Maximum likelihood analysis using TREE-PUZZLE. In Baxevanis, A. D., Davison, D. B., Page, R. D. M., Stormo, G. and Stein, L. (eds.), *Current Protocols in Bioinformatics*, pages 6.6.1–6.6.25, Wiley and Sons, New York, USA.

Schmidt, H. A., Petzold, E., Vingron, M. and von Haeseler, A. (2003) Molecular phylogenetics: Parallelized parameter estimation and quartet puzzling. *J. Parallel Distrib. Comput.*, **63**, in press.

Schmidt, H. A., Strimmer, K., Vingron, M. and von Haeseler, A. (2002) TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, **18**, 502–504.

Semple, C. and Steel, M. (2000) A supertree method for rooted trees. *Discr. Appl. Math.*, **105**, 147–158.

Shapiro, B. A., Jin Chu Wu, Bengali, D. and Potts, M. J. (2001) The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. *Bioinformatics*, **17**, 137–148.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. (1998) *MPI: The Complete Reference - The MPI Core*, vol. 1. The MIT Press, Cambridge, Massachusetts, Second edn..

Stamatakis, A. P., Ludwig, T., Meier, H. and Wolf, M. J. (2002) AxML: A fast program for sequential and parallel phylogenetic tree computations based on the maximum likelihood method. In *Proceedings of the 1st IEEE Computer Society Bioinformatics Conference - CSB2002*, pages 21–28, Palo Alto.

Stewart, C. A., Hart, D., Berry, D. K., Olsen, Gary J.and Wernert, E. A. and Fischer, W. (2001) Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference. In *Proceedings of the International Conference on High Performance Computing and Communications - SC2001*, pages 191–201.

Stoesser, G., Baker, W., van den Broek, A., Garcia-Pastor, M., Kanz, C., Kulikova, T., Leinonen, R., Lin, Q., Lombard, V., Lopez, R., Mancuso, R., Nardone, F., Stoehr, P., Tuli, M. A., Tzouvara, K. and Vaughan, R. (2002) The EMBL nucleotide sequence database: major new developments. *Nucleic Acids Res.*, **31**, 17–22.

Strimmer, K., Goldman, N. and von Haeseler, A. (1997) Bayesian probabilities and quartet puzzling. *Mol. Biol. Evol.*, **14**, 210–213.

Strimmer, K. and von Haeseler, A. (1996) Quartet puzzling: A quartet maximum–likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, **13**, 964–969.

Strimmer, K. and von Haeseler, A. (1997) Likelihood–mapping: A simple method to visualize phylogenetic content of a sequence alignment. *Proc. Natl. Acad. Sci. USA*, **94**, 6815–6819.

Strimmer, K. S. (1997) *Maximum Likelihood Methods in Molecular Phylogenetics*. Ph.D. thesis, Ludwig–Maximilians–Universität, München, Germany.

Swofford, D. L., Olsen, G. J., Waddell, P. J. and Hillis, D. M. (1996) Phylogeny reconstruction. In Hillis, D. M., Moritz, C. and Mable, B. K. (eds.), *Molecular Systematics*, pages 407–514, Sinauer Associates, Sunderland, Massachusetts, Second edn..

Tamura, K. and Nei, M. (1993) Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Mol. Biol. Evol.*, **10**, 512–526.

Tavare, S. (1986) Some probabilistic and statistical problems on the analysis of DNA sequences. *Lec. Math. Life Sci.*, **17**, 57–86.

Thompson, J. D., Higgins, D. G. and Gibson, T. J. (1994) CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions–specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Trelles, O. (2001) On the parallelisation of bioinformatics applications. *Brief. Bioinform.*, **2**, 181–194.

Trelles, O., Ceron, C., Wang, H. C., Dopazo, J. and Carazo, J. M. (1998) Application note: New phylogenetic venues opened by a novel implementation of the DNAml algorithm. *Bioinformatics*, **14**, 544–545.

Whelan, S. and Goldman, N. (2001) A general empirical model of protein evolution derived from multiple protein families using a maximum likelihood approach. *Mol. Biol. Evol.*, **18**, 691–699.

Yang, Z. (1996) Maximum-likelihood models for combined analyses of multiple sequence data. *J. Mol. Evol.*, **42**, 587–596.

Yang, Z. and Kumar, S. (1994) Approximate methods for estimating the pattern of nucleotide substitution and the variation of substitution rates among sites. *Mol. Biol. Evol.*, **13**, 650–659.

Acknowledgments

First and foremost, I wish to thank my supervisor Arndt von Haeseler. I am grateful for his excellent advise and collaboration. Furthermore I want to thank him as well as Ulrike and Jule Friedrichs for their extraordinary hospitality during my many visits in Leipzig and Düsseldorf.

I want to thank Martin Vingron for introducing me to Arndt von Haeseler in the context of a joint project as well as for giving me the possibility the work in the pleasant and stimulating environment of his research groups at the DKFZ in Heidelberg and the Max-Planck-Institute (MPI) for Molecular Genetics in Berlin.

Also I want to thank Bill Martin for accepting the task to read this thesis as a 'Korreferent'.

Special thanks go to Heike Hennig-Schmidt for taking the burden of reading loads of text unrelated to her own field of research as well as to Antje Krause for patience to repeatedly listening to and discussing ideas.

I thank Korbinian Strimmer for a lot of help when getting used to his TREE-PUZZLE source code and for communicating Vincent Berry's MRCA-based algorithm.

I thank Ekkehard Petzold for his close collaboration on parameter estimation issues.

I thank Jens Stoye, Sven Rahmann, and Tobias Müller for fruitful discussions on algorithmic and technical issues.

I appreciated very much the cleverly designed computing environment at the MPI for Molecular Genetics maintained by Wilhelm Rüsing, Peter Marquardt, and others from the Computing Support group.

Furthermore I want to thank all friends, collaborators, and colleagues who gave support in any way, especially Antje Pleß, Christine Steinhoff, and all members of the After-Lunch-Walk Crew.

Financial support from the Deutsche Forschungsgemeinschaft and the Max-Planck-Gesellschaft is gratefully acknowledged.

Finally, I want to thank the John-von-Neumann-Institute for Computing (NIC) in Jülich and the Max-Planck-Institute for Molecular Genetics in Berlin for access to computing facilities.