Effect of DNA Composition Bias on Models of Evolution

Andreas Gabril Bernauer

May 4, 2003

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

2004

APPROVAL PAGE

Master of Science Thesis

Effect of DNA Composition Bias on Models of Evolution

Presented by

Andreas Gabril Bernauer

Major Advisor _____
J. Peter Gogarten

Associate Advisor _____
Don Guk Shin

Associate Advisor _____
Linda D. Strausbaugh

University of Connecticut

2004

Contents

1 Introduction **1**

1.1 Continuous Markov models of amino acid evolution 2

1.2 Estimating Markov models 4

2 Materials and Methods **6**

2.1 Sequence data 6

2.2 Estimating the instantaneous rate matrices 12

2.3 Adjusting for not closely related sequences 15

2.4 Approximation of the instantaneous rate matrix 18

3 Results **21**

3.1 Sequence data 21

3.2 Instantaneous rate matrices 22

4 Discussion **38**

A Mathematics **41**

A.1 Derivation of $\mathbf{P}(T)$ 41

B Distributing tasks on a set of machines **43**

C Sequence data **54**

C.1 Count matrices 54

List of Tables

2.1 List of Cyanobacteria 7

2.2 List of Firmicutes (a) 8

2.3 List of Firmicutes (b) 9

2.4 List of Actinobacteria 10

3.1 Relative frequencies of occurence 23

3.2 Universal instantaneous rate matrix Cyanobacteria–Firmicutes 27

3.3 Universal instantaneous rate matrix Cyanobacteria–Actinobacteria 28

3.4 Universal instantaneous rate matrix Firmicutes–Actinobacteria 29

3.5 Comparison of two universal rate matrices 31

3.6 Comparison of two universal rate matrices *(continued)* 32

3.7 Comparison of two universal rate matrices *(continued)* 33

3.8 Comparison of two universal rate matrices *(continued)* 34

3.9 Comparison of two universal rate matrices *(continued)* 35

3.10 Comparison of two universal rate matrices *(continued)* 36

4.1 Boundaries of the model 40

B.1 Specification of client–server protocol. 44

B.2 Further commands understood by the server 44

C.1 Count matrix between Cyanobacteria and Firmicutes. 55
C.2 Count matrix between Cyano- and Actinobacteria. 56
C.3 Count matrix between Firmicutes and Actinobacteria. 57

List of Figures

3.1 Approximation of derivatives 24
3.2 Correction functions 25

B.1 Example client script. 45
B.2 Example script that starts the server 46

Chapter 1

Introduction

In this thesis I study how a bias in DNA composition affects models of amino acid sequence evolution. Models of amino acid evolution are used in various ways, *e.g.* for amino acid sequence alignment, reconstruction of protein phylogenies, inferring ancestral protein sequences, searches in protein databases and simulation of protein evolution (*e.g.* see [28] for a review).

The model most widely used are continuous Markov models ([1, 7, 8, 14, 19], [23] for review of models of molecular evolution). A continuous Markov model has a finite set of states and describes the probabilities of changing from a state i to a state j (which may be the same as state i) after some time T. In the case of modelling amino acid sequence evolution, the set of states are the 20 amino acids and the Markov models describes the probabilites of their substitution over time. This means, that all sites of an amino acid sequence are treated independently from each other [7].

Reconstructing the tree of life has been and still is a major target of molecular evolution (*e.g.* see [31] for a review). The tree of life is considered to describe the evolution of all living species and their relationships. Based on a model of evolution, the task is to find a tree that best describes the evolution that led to the nucleotide or amino acid sequences we find today. What a "best" description is, depends on the method that is used for the reconstruction (see [18] for

a review on current methods).

For trees based on proteins, the reconstruction method uses some model of amino acid evolution. This model is usually applied on all species of the tree, assuming that this model approximates molecular evolution well in every species. However, it is known that the DNA composition of species can vary over a large range. The DNA composition is measured in the percentage of two of the four the nucleotides used in DNA: guanine (G) and cytosine (C). Hence, we speak of the G+C content of an organism. Virtually all organisms use the same code to construct their proteins based on their DNA. It is unclear, whether, and if yes, how a different DNA composition—that is, a different G+C content—may affects the composition of proteins of the organisms. If there is such an effect, the tree reconstruction methods have to be adjusted for this to compensate this effect.

In this thesis, I study the effect of a bias in DNA composition on the models of amino acid evolution. For this, I use species that are known to have very different G+C contents, construct Markov models of evolution and compare them. The species are from three phyla in the kingdom of bacteria: Cyanobacteria, which have mostly an average G+C content, Firmicutes, which have a very low G+C content and Actinobacteria, which have a very high G+C content. Firmicutes are also known under the name low G+C gram positive bacteria, and Actinobacteria are also known under the name high G+C gram positive bacteria. I derive Markov models of evolution for each pair of phyla and study to what extent the three models differ from each other.

1.1 Continuous Markov models of amino acid evolution

As mentioned above, continuous Markov models are used to describe the evolution of amino acid sequences. For each amino acid in a sequence, the Markov model describes the probability of its substitution by another amino acid over time. The probabilities for a continuous Markov model are given by an *instantaneous rate matrix* $\mathbf{Q}$. The instantaneous rate matrix describes

the rate of change from amino acid i to amino acid j, *i.e.* it describes how the probability for such a change alters as time elapses.

When the probabilities for the changes between the amino acids are written in a matrix $\mathbf{P}(T)$—with $\mathbf{P}_{ij}(T)$ denoting the change from amino acid i to j over time T—then the probability of a change for time T plus a small time $\mathrm{d}T$ is given by the instantaneous rate matrix as follows (the symbol $\mathbf{I}$ denotes the identity matrix):

$$\begin{aligned}\mathbf{P}(T+\mathrm{d}T) &= \mathbf{P}(T) + \mathbf{P}(T)\mathbf{Q}\mathrm{d}T \\ &= \mathbf{P}(T)(\mathbf{I} + \mathbf{Q}\mathrm{d}T)\end{aligned} \tag{1.1}$$

This equation can be solved (see Appendix A.1) to give:

$$\mathbf{P}(T) = e^{T\mathbf{Q}} \tag{1.2}$$

Given this definition, we can see two properties of the probability matrix $\mathbf{P}(T)$:

$$\mathbf{P}(T+S) = e^{(T+S)\mathbf{Q}} = e^{T\mathbf{Q}}e^{S\mathbf{Q}} = \mathbf{P}(T)\mathbf{P}(S) \tag{1.3}$$

$$\mathbf{P}(nT) = e^{nT\mathbf{Q}} = \left(e^{T\mathbf{Q}}\right)^n = \mathbf{P}(T)^n \tag{1.4}$$

The first property (1.3) states that the way the model reaches time $T+S$ does not affect the probabilities for that time: the probability calculated for the complete time $T+S$ is the same as first considering the probability for time T and then for time S. The model is said to be "memoryless" as the probabilities only depend on the current state and the time of consideration, and not on how the model reached the current state.

The second property (1.4) is a continuation of the first one on multiple time periods T. It states that the probability after n steps of time T is the same as the probability after time T raised to the power of n (*i.e.* n times matrix multiplication of $\mathbf{P}(T)$ with itself).

Note that the time T and the instantaneous matrix $\mathbf{Q}$ are closely linked: double the time

and half the rate will yield the same results ($T\mathbf{Q} = (T/\gamma)(\gamma\mathbf{Q})$ for any γ). This means that we cannot measure absolute time. Instead, time is usually measured in expected substitutions per site (see 2.2).

1.2 Estimating Markov models

Basically, there are two possible ways to come up with the instantaneous rate matrix of a continous Markov model: derive it theoretically or empirically. For nucleotide sequences, theoretical parametrization of the instantaneous rate matrix [10, 15, 20, 21] has been widely used, while for amino acid models empirical models are usually favored. The pioneers for empirical models for amino acid substitutions are Dayhoff's famous PAM matrices [7, 8], which various authors later adapted (*e.g.* Jones *et al.* [19], Gonnet *et al.* [14], Adachi and Hasegawa [1]).

Dayhoff and coworkers used very similar amino acid sequences to derive an empirical model for amino acid sequence evolution. When sequences are very similar ($> 85\%$), it is assumed that only little time has elapsed in their evolution from their most recent common ancestor. This little evolutionary time has two consequences. First, we expect that only a neglectible amount of multiple substitutions have happened, *i.e.* every observed substitution is the result of a single substitution event (*e.g.* L →V) rather than one of multiple substitution events, which take more time (*e.g.* L→A →W →V). Second, the transition matrix is approximately the same as the sum of the identity matrix and the instantaneous rate matrix: if we start from $T = 0$ and let only a small amount of time t elapse, we have

$$\begin{aligned} \mathbf{P}(0+t) &\approx \mathbf{P}(0+\mathrm{d}T) \\ &= \mathbf{P}(0)(\mathbf{I}+\mathbf{Q}\mathrm{d}T) && (1.1) \\ &= \mathbf{I}+\mathbf{Q}\mathrm{d}T && \text{as } \mathbf{P}(0)=\mathbf{I} \end{aligned} \tag{1.5}$$

Therefore, empirically deriving the transition probability matrix of a Markov model gives

also a good estimate of the underlying instantanteous rate matrix. This is the approach which is usually taken to derive empirical Markov models of sequence evolution.

Chapter 2

Materials and Methods

2.1 Sequence data

The National Center for Biotechnology Information (NCBI) hosts the GenBank database that collects publicly available nucleotide and amino acid sequences [3], including whole genomes of numerous bacteria. I downloaded the whole genomes from GenBank's FTP server[1] as available on November 2003. Other sites hosting cyanobacterial genomes (*e.g.* the Cyanobase[2]) did not contain genomes that weren't already submitted to Genbank. I included the plastids of a bacterium only if the bacterial genome was also available. The complete lists of the bacteria used in this study are given in tables 2.1, 2.2, 2.3 and 2.4.

Bacterium	Accession number	total genes	used genes	G+C content
Cyanobacteria				
Gloeobacter violaceus	NC_005125	4430	600	62.00
Nostoc sp. PCC 7120	NC_003272	5366	678	41.35
Nostoc sp. PCC 7120 plasmid pCC7120alpha	NC_003276	385	30	40.51
Nostoc sp. PCC 7120 plasmid pCC7120beta	NC_003240	186	23	40.23
Nostoc sp. PCC 7120 plasmid pCC7120delta	NC_003273	66	2	41.61
Nostoc sp. PCC 7120 plasmid pCC7120epsilon	NC_003270	31	3	40.86
Nostoc sp. PCC 7120 plasmid pCC7120gamma	NC_003267	90	15	41.03
Nostoc sp. PCC 7120 plasmid pCC7120zeta	NC_003241	5	0	44.18
Prochlorococcus marinus str. MIT 9313	NC_005071	2273	104	50.74
Prochlorococcus marinus subsp. marinus str. CCMP1375	NC_005042	1882	0	36.44
Prochlorococcus marinus subsp. marinus str. CCMP1375	AE017126	1882	80	36.44
Prochlorococcus marinus subsp. pastoris str. CCMP1378	NC_005072	1716	54	30.80
Synechococcus sp. WH 8102	NC_005070	2526	129	59.41
Synechocystis sp. PCC 6803	NC_000911	3167	340	47.72
Thermosynechococcus elongatus BP-1	NC_004113	2475	237	53.92
Average		1787	164	45.06

Table 2.1: List of Cyanobacteria used in the analysis. The columns contain the name of the Cyanobacterium or its plasmid, its accession number in the GenBank database, the total number of protein encoding genes, the number of these genes used in the analysis and the G+C content of the organism. The last row contains the average total and und used amount of genes and the average G+C content. See 3.1 for total numbers. Note that Prochlorococcus marinus subsp. marinus str. CCMP1375 happened to make it twice into the database under different accession numbers. However, only one of the copies contributed to the analysis, as can be seen in the “used genes” column. It is excluded from the averages.

Bacterium	Accession number	total genes	used genes	G+C content
Firmicutes				
Bacillus anthracis str. Ames	NC_003997	5311	195	35.38
Bacillus cereus ATCC 14579	NC_004722	5234	228	35.28
Bacillus cereus ATCC 14579 plasmid pBClin15	NC_004721	21	0	38.09
Bacillus halodurans	NC_002570	4066	362	43.69
Bacillus subtilis	NC_000964	4112	246	43.52
Clostridium acetobutylicum ATCC824	NC_003030	3672	230	30.93
Clostridium acetobutylicum plasmid pSOL1	NC_001988	176	21	30.91
Clostridium perfringens	NC_003366	2660	136	28.57
Clostridium perfringens plasmid pCP13	NC_003042	63	2	25.50
Clostridium tetani E88	NC_004557	2373	129	28.75
Enterococcus faecalis V583	NC_004668	3113	109	37.53
Lactobacillus plantarum WCFS1	NC_004567	3051	128	44.47
Lactococcus lactis subsp. lactis	NC_002662	2321	46	35.33
Listeria innocua Clip11262	NC_003212	2981	97	37.44
Listeria innocua plasmid pLI100	NC_003383	80	7	35.52
Listeria monocytogenes strain EGD	NC_003210	2855	76	37.98
Mycoplasma gallisepticum R	NC_004829	726	10	31.45
Mycoplasma genitalium	NC_000908	484	2	31.69
Mycoplasma penetrans	NC_004432	1037	29	25.72

Table 2.2: List of Firmicutes (also known as low G+C gram positive bacteria) used in the analysis. See table 2.1 for column descriptions. The table is continued in table 2.3.

Bacterium	Accession number	total genes	used genes	G+C content
Firmicutes *(continued)*				
Mycoplasma pneumoniae	NC_000912	689	11	40.01
Mycoplasma pulmonis	NC_002771	782	4	26.64
Oceanobacillus iheyensis HTE831	NC_004193	3500	229	35.68
Staphylococcus aureus strain Mu50	NC_002758	2714	17	32.88
Staphylococcus aureus subsp. aureus Mu50 plasmid VRSAp	NC_002774	34	1	28.93
Staphylococcus aureus subsp. aureus MW2	NC_003923	2632	54	32.83
Staphylococcus aureus subsp. aureus N315	NC_002745	2593	0	32.84
Staphylococcus aureus subsp. aureus N315 plasmid pN315	NC_003140	31	1	28.70
Staphylococcus epidermidis ATCC 12228	NC_004461	2419	55	32.10
Streptococcus agalactiae 2603V/R	NC_004116	2124	22	35.65
Streptococcus agalactiae NEM316	NC_004368	2134	45	35.63
Streptococcus mutans UA159	NC_004350	1960	39	36.83
Streptococcus pneumoniae	NC_003028	2094	16	39.70
Streptococcus pneumoniae R6	NC_003098	2043	39	39.72
Streptococcus pyogenes	NC_002737	1697	12	38.51
Streptococcus pyogenes MGAS315	NC_004070	1865	2	38.59
Streptococcus pyogenes SSI-1	NC_004606	1861	27	38.55
Streptococcus pyogenes strain MGAS8232	NC_003485	1845	6	38.55
Thermoanaerobacter tengcongensis strain MB4T	NC_003869	2588	303	37.57
Ureaplasma urealyticum	NC_002162	614	10	25.50
Average		2066	76	34.70

Table 2.3: List of Firmicutes (also known as low G+C gram positive bacteria) used in the analysis, continued from table 2.2. See table 2.1 for column descriptions. The last row contains the average total and und used amount of genes and the average G+C content. See 3.1 for total numbers.

Bacterium	Accession number	total genes	used genes	G+C content
Actinobacteria				
Bifidobacterium longum NCC2705	NC_004307	1729	256	60.12
Corynebacterium diphtheriae	NC_002935	2320	185	53.48
Corynebacterium efficiens YS-314	NC_004369	2950	243	63.14
Corynebacterium glutamicum ATCC 13032	NC_003450	2993	245	53.81
Mycobacterium bovis subsp. bovis AF2122/97	NC_002945	3953	241	65.63
Mycobacterium leprae strain TN	NC_002677	2720	56	57.80
Mycobacterium tuberculosis CDC1551	NC_002755	4187	63	65.61
Mycobacterium tuberculosis H37Rv	NC_000962	3927	9	65.61
Streptomyces avermitilis MA-4680	NC_003155	7575	798	70.72
Streptomyces coelicolor A3(2)	NC_003888	7825	840	72.12
Streptomyces coelicolor A3(2) plasmid SCP1	NC_003903	356	10	69.06
Streptomyces coelicolor A3(2) plasmid SCP2	NC_003904	34	1	72.12
Tropheryma whipplei str. Twist	NC_004572	808	46	46.33
Tropheryma whipplei TW08/27	NC_004551	784	4	46.31
Average		3012	214	61.56

Table 2.4: List of Actinobacteria (also known as high G+C gram positive bacteria) used in the analysis. See table 2.1 for column descriptions. The last row contains the average total and und used amount of genes and the average G+C content. See 3.1 for total numbers.

Estimating a model of evolution (see section 2.2 below) can only be done with sequences whose evolutionary history reflects speciation events as opposed to a history of gene duplication within a single organism. The former sequences are said to be *orthologous*, the latter *paralogous* [12]. As the evolutionary history of a gene is in general unknown, feasible approximations are necessary. In this study, two sequences are considered orthologous, if they are each other's top hit in a similarity search in a database. I use BLAST [2, 25] for the similarity search. To reduce the probability of random mutual hits, I used an evalue of 10^{-4} as an cutoff value. With this, the probability of two sequences being considered orthologous just by chance is at most $10{-}8$ (but usually lower). The search for orthologous sequences results in a list of pairs of sequences of which each sequence is from a different phylum (*i.e.* I don't consider orthologous sequences within a phylum).

To increase the reliability of the alignments (see below), I used protein sequences published in the SWISS-PROT database [4]. SWISS-PROT is a well maintained database containing well annotated proteins, thus it provides a reliable basis for sequence comparisons. I used the version in FASTA format [24] published along with a version of BLAST [2, 25] on NCBI's ftp server[3] on January 21, 2004.

For each sequence of a sequence pair, I searched the SWISS-PROT database with BLAST with a cutoff evalue of 10^{-8} and added the shared hits to the sequence pair. I dropped pairs for which there were no shared hits. I created a multiple alignment of the sequence pair and its shared hits in the SWISS-PROT database in such a way that it can be considered reliable, *i.e.* it aligns only positions that really evolved from each other. I dropped pairs for which the reliable alignment was less than 200 bp long. This prevents substitution rates that look artificially high just because the alignment is short.

The reliable alignments were done by a program called `clustalw2.pl` from J. Gogarten [13]. It creates two multiple alignments using CLUSTALW (v1.83) [27]: one in the usual way and one with the sequences reversed (*i.e.* the sequence starts with its last amino acid and ends

with its first). Then the program compares the two resulting multiple alignments and outputs only the columns which are the same in both alignments, dropping columns that contain gaps. Note that this does not imply that all the amino acids in the column are the identical. Comparing the usual alignment with the reversed one ensures that only alignment positions are considered that are reliable. The result of the alignment are two sequences of equal length, where each sequence has lost some of its columns.

The task of searching the SWISS-PROT database and generating the alignments needs a lot of computational power. For this purpose, I wrote a program that distributes arbitrary tasks on a set of machines. The program is described in appendix B.

2.2 Estimating the instantaneous rate matrices

Dayhoff and coworkers estimated their Markov models of amino acid sequence evolution by inferring ancestral states for each site of amino acid sequences. As it is not always possible to infer the ancestral states unambiguously, modern approaches create alignments of sequences and count the substitutions that are observed there [14, 19]. From a count matrix of substitutions the methods derive the transition probability of the Markov model which implies the instantaneous rate matrix as shown in (1.5). I use the modern approach to estimate the Markov model. However, both methods, base on closely related sequences, which I clearly don't have. This issue is addressed by the next section 2.3.

Given a set of pairwise alignments, where in each pairwise alignment the two sequences x and y have the same length and don't have any gaps, I derive the Markov model as follows. First, I create the count matrix $\mathbf{C}^{(x,y)}$ with entries

(2.1) $\mathbf{C}_{ij}^{(x,y)} =$ number of sites where sequence x has amino acid i and sequence y has amino acid j,

where a site means a position in the alignment. The entries of the count matrix are positive integers. $\mathbf{C}_{ij}$ denotes the ith row and the jth column of the matrix.

Note that in general $\mathbf{C}^{(x,y)} \neq \mathbf{C}^{(y,x)}$. However, as I don't know the direction of evolution, *i.e.* I don't know if amino acid i was replaced by j or if j was replaced by i, I only consider the symmetrized version of the count matrices

$$\begin{aligned} \mathbf{C} &= \frac{1}{2}(\mathbf{C}^{(x,y)} + \mathbf{C}^{(y,x)}) \\ &= \frac{1}{2}(\mathbf{C}^{(x,y)} + \mathbf{C}_T^{(x,y)}), \end{aligned} \tag{2.2}$$

where the subscript $_T$ indicates transposition of the matrix (*i.e.* $\mathbf{A}_{ij} = \mathbf{B}_{ji}$ for $\mathbf{A} = \mathbf{B}_T$). Symmetrizing the count matrix also ensures time reversibility of the Markov model.

Along with the count matrix comes the vector of occurences $\mathbf{c} = (c_1, \ldots, c_{20})$ with:

$$\begin{aligned} c_i &= \text{amount of amino acid } i \text{ in the alignment} \\ &= \sum_{j=1}^{20} \mathbf{C}_{ij} + \mathbf{C}_{ji}. \end{aligned}$$

The entries of the vector of occurences are nonnegative integers. From the vector of occurences, I obtain the vector of amino acid frequencies $\pi = (\pi_1, \ldots, \pi_{20})$ with

$$\begin{aligned} \pi_i &= \frac{\sum_{i=1}^{20} \mathbf{C}_{ij}}{\sum_{i,j=1}^{20} \mathbf{C}_{ij}} \\ &= \frac{c_i}{2\sum_{i,j=1}^{20} \mathbf{C}_{ij}}, \end{aligned}$$

The vector of amino acid frequencies describes the fraction of an amino acid in the pairwise alignments, thus the elements of the vector are between 0 and 1 and sum up to 1. It is assumed that these frequencies remain constant over time, *i.e.* they are the equilibrium frequencies of the Markov model.

From the symmetrized count matrix, I finally obtain the mutability matrix $\mathbf{M}$ with elements

$$\mathbf{M}_{ij} = \frac{\mathbf{C}_{ij}}{\sum_{i=1}^{20} \mathbf{C}_{ij}}. \tag{2.3}$$

The entry $\mathbf{M}_{ij}$ of the mutability matrix is the probability of seeing amino acid j in the on of the sequences of a pairwise alignment, given that we see amino acid i in the other sequence. Therefore, each entry $\mathbf{M}_{ij}$ falls between 0 and 1 and each row of the mutability matrix sums up to 1. In other words, the mutability matrix is a stochastic matrix.

As I derived the mutability matrix from the symmetrized count matrix, the mutability matrix is also *time reversible*, that is the model can't tell the direction of evolution: evolving from sequence A to sequence B is the same as evolving from sequence B to sequence A. The model is said to fulfill the *detailed balance equation*

$$\pi_i \mathbf{M}_{ij} = \pi_j \mathbf{M}_{ji}, \tag{2.4}$$

which says that the probability from going from state i to state j is the same as going from state j to state i.

Although Dayhoff did not mention it originally, the mutability matrix is the transition probability matrix of the Markov model, *i.e.* $\mathbf{M} = \mathbf{P}(T)$ for some time unknown time T. In the case of closely related sequences this unknown time is supposed to be small enough for the approximation shown in (1.5). In my case of divergent sequences the approximation can't be done and I will have to adjust for this. This is covered by the next section. But first I will make two further definitions that I will need later.

Given the vector of frequencies and the mutability matrix, the average substitution frequency is:

$$\mathcal{D}(\mathbf{M}) = 1 - \sum_{i=1}^{20} \pi_i \mathbf{M}_{ii}, \tag{2.5}$$

The average substitution frequency $\mathcal{D}(\mathbf{M})$ is the probability that the model does not stay in its current state. It is a measure for the observed distance between the sequences of the set of alignments: the more time has elapsed, the more substitutions are expected and vice versa. Thus, we can define time in terms of number of expected or observed substitutions. If the rate matrix is scaled such that its mean rate is 1, then each substitution event corresponds to a time unit.

$$\bar{\mathbf{Q}} = \frac{\mathbf{Q}}{\sum_i^{20} \pi_i Q_{ii}} \tag{2.6}$$

This is the suggested method of N. Goldmann [22] to present instantaneous rate matrices, as it easily allows for comparisons.

2.3 Adjusting for not closely related sequences

Henikoff and Henikoff [16] used blocks of conserved regions of a protein family to estimate scores for amino acid alignment. Although in principal similar to the method described in the previous section, their resulting matrix has not been used for phylogenetic reconstruction, as it misses an underlying model of evolution. The reason for this is that the sequences from which the blocks of conserved regions was obtained were not closely related.

Veerassamy *et al.* [29] derived a model of evolution from the BLOCKS database [17] that Henikoff and Henikoff used. They implemented this model which they call PMB in the PROML and PROTDIST v3.6 programs of the PHYLIP package [11], a collection of programs for phylogenetic reconstruction. Bullerwell *et al.* [5] used the PMB model to derive a phylogeny of mitochondrial genes in fission yeasts. Smith *et al.* [26] applied the method of Veerassamy *et al.* on modeling the evolution of RNA.

The problem with divergent sequences is that the time that has elapsed in the evolution of the sequences is too large that the approximation of the transition probability matrix and

the instantaneous rate matrix in (1.5) holds, *i.e.* we can only get estimates for $\mathbf{Q}T$ but not for $\mathbf{Q}\mathrm{d}T$. The idea presented in Veerassamy *et al.* [29] is to derive a correction formula that yields an estimate $\hat{T}$ for the actual evolutionary distance given the observed distance D by solving a differential equation that involves the derivate of D with respect to T. Dividing $\mathbf{Q}T$ by $\hat{T}$ will yield an estimate for $\mathbf{Q}$. Getting the estimates works as follows, as also shown in Smith *et al.* [26].

Given a set of pairwise alignments of divergent sequences, I determine for each pairwise alignment the percentage of dissimilarity. The percentage of dissimilarity is the fraction of alignment sites of the whole alignment that contain different amino acids. Then I slide a window of some window length w over the range of dissimilarity. Each window creates a cluster of alignments whose dissimilarities falls in the window range. For the first window, the range is from 0 to w, for the next window from 0.02 to $(w + 0.02)$, for the next from 0.04 to $(w + 0.04)$ and so on. The last window covers the range from $(1 - w)$ to 1. Most alignments will be in multiple clusters; in fact, consecutive clusters will usually contain almost the same set of alignments.

For each cluster c I calculate the mutability matrix $\mathbf{M}_c(T_c)$ and the observed distance $\mathcal{D}(\mathbf{M}_c(T_c))$ as shown in the previous section. Note that both the mutability matrix and the observed distance depend on some unknown actual evolutionary distance T_c. The aim is to derive a correction function that estimates T_c given the known observed distance. For this, I calculate the derivative of $\mathcal{D}(\mathbf{M}_c(T_c))$, using the 5-point formula for numerical differentiation (*e.g.* [6]):

$$\frac{\mathrm{d}}{\mathrm{d}x}f(x_0) = \frac{1}{12h}\left(f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)\right) \tag{2.7}$$

The 5-point formula estimates the derivative of the function f at the point x_0 by considering values of f around x_0 seperated by a small intervall h. It can be derived by a Taylor expansion of f to the fourth derivative and has an error only in order of $O(h^4)$, therefore yielding good

numerical approximations. With $f = \mathcal{D}$, $x_0 = T_c$ and $h = 0.01T_c$ we get

$$\frac{\mathrm{d}\mathcal{D}(\mathbf{M}(T_c))}{\mathrm{d}T} = \frac{1}{12(0.01T_c)}[\mathcal{D}(\mathbf{M}(T_c - 0.02T_c)) - 8\mathcal{D}(\mathbf{M}(T_c - 0.01T_c)) + 8\mathcal{D}(\mathbf{M}(T_c + 0.01T_c)) - \mathcal{D}(\mathbf{M}(T_c + 0.02T_c))] \tag{2.8}$$

Using equation (1.4) we can calculate the mutability matrices for the various times around T_c and get

$$T_c\frac{\mathrm{d}\mathcal{D}(\mathbf{M}(T_c))}{\mathrm{d}T} = \frac{1}{0.12}\mathcal{D}(\mathbf{M}(T_c)^{0.98}) - 8\mathcal{D}(\mathbf{M}(T_c)^{0.99}) + 8\mathcal{D}(\mathbf{M}(T_c)^{1.01}) - \mathcal{D}(\mathbf{M}(T_c)^{1.02}) \tag{2.9}$$

The right hand side of this equation contains only the known values $\mathbf{M}_c$ and the function $\mathcal{D}$. The powers can be calculated by diagonalizing $\mathbf{M}_c$ If we plot the approximation of the derivative against the observed distance, we can approximate the points by a polynomial function f of the observed distance (see figure 3.1).

For notational convenience I define

$$D = \mathcal{D}(\mathbf{M}(T)) \tag{2.10}$$

Using the polynomial approximation function f, we get a differential equation

$$T\frac{\mathrm{d}D}{\mathrm{d}T} = f(D) = \sum_{i=0}^{n} a_i D^i \tag{2.11}$$

We know that at small evolutionary distances, the observed and the actual evolutionary distance is approximately the same and there are no multiple substitutions:

$$\lim_{T\to 0} D = 0 \quad \text{and} \quad \lim_{T\to 0} \frac{\mathrm{d}D}{\mathrm{d}T} = 1. \tag{2.12}$$

Taking the limit for $T \to 0$ on both sides of equation (2.11) lets us conclude that $a_0 = 0$ as the left hand side's limit is 0.

If the degree of the approximation function f is low enough, the differential equation (2.11) can be solved and yields a correction function $\mathcal{C}$ for the actual evolutionary distance

$$\hat{T} = \mathcal{C}(D) \tag{2.13}$$

Note that the correction formula covers all clusters, *i.e.* for each pair of phyla there will be a correction formula. With the estimate $\hat{T}$ of the actual evolutionary distance, we can find an approximation for the instantaneous rate matrix $\mathbf{Q}$ of the data set.

2.4 Approximation of the instantaneous rate matrix

From equation (1.2) we know that

$$\mathbf{M}_c = e^(\mathbf{A}_c T_c) \tag{2.14}$$

for some instantaneous rate matrix $\mathbf{A}_c$. With the estimate for the average of the actual evolutionary distance, we can calculate an instantaneous rate matrix for each cluster c:

$$A_c = \frac{\ln(\mathbf{M}_c)}{\hat{T}_c} \tag{2.15}$$

The calculation of the logarithm can be numerical unstable for a transition matrix $\mathbf{M}_c$ that is not close enough to the identity matrix.[4] To detect such a case, we can relate how much the exponential using the calculated logarithm of $\mathbf{M}_c$ differs from the real $\mathbf{M}_c$. Veerassamy *et al.* [29] suggested the following criteria:

$$\frac{\|\exp(A_c \hat{T}_c) - \mathbf{M}_c\|}{\|\mathbf{M}_c\|} < \text{tol} \tag{2.16}$$

where $\|.\|$ denotes the norm of a matrix as the largest singular value and tol is 1000 times the distance between the 1.0 and the next floating point number representable by the computing machine. On the machine I used tol is 10^{-6}.

Each cluster will result in an instantaneous rate matrix A_c. If everything were perfect, there had to be a universal rate matrix $\mathbf{U}$ so that for each cluster the following holds:

$$\mathbf{M}_c = \exp(\mathbf{U}T_c) \tag{2.17}$$

However, we have only different approximations to $\mathbf{U}$, each with different accuracy. Finding a set of weights w_c with $\sum_c w_c = 1$ and calculating

$$\mathbf{U} = \sum_c w_c \mathbf{A}_c \tag{2.18}$$

so that the error

$$\sum_c \frac{\|\exp(\mathbf{U}\hat{T}_c) - \mathbf{M}_c\|}{\|\mathbf{M_c}\|} \tag{2.19}$$

is minimized will yield an approximation to $\mathbf{U}$.

This problem is in general difficult as the space of possible weights w_c is very big. Smith *et al.* [26] avoided this problem by selecting the instantaneous rate matrix $\mathbf{A}_c$ for which the sum in (2.19) is minimal, *i.e.* all but one weights are zero. I used a random search through the space of the weights and selected the set of weights for which (2.19) was minimal.

Notes

[1] `ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/`

[2] `http://www.kazusa.or.jp/cyanobase`

[3] `ftp://ftp.ncbi.nih.gov/blast/db/FASTA/swissprot.gz`

[4] To be more precise, the matrix has to fall within the convergence radius of the logarithm series. This means there must exist some $n \in \mathbb{N}$ such that $\|(\mathbf{P} - 1)^n\| < 1$ for some matrix norm $\|.\|$.

Chapter 3

Results

3.1 Sequence data

Including plasmids, there were 15 whole genomes of Cyanobacteria available at the NCBI site (containing 26 480 genes, of which 2 295 made it into the analysis, covering 1 014 666 amino acids), 14 genomes of Actinobacteria (42 161, 2 997, 1 329 833) and 39 genomes of Firmicutes (80 555, 2 946, 1 255 671). It is interesting to note that although for every phylum there was a different amount of genomes and thus genes available, each phylum contributed approximately equally to the amount of genes and base pairs of the analysis.

The initial BLAST search between the three phyla resulted in 235 319 top hits of which 23 620 were mutual hits, *i.e.* there were 11 810 pairs of genes considered orthologous: 3 715 between Cyanobacteria and Firmicutes, 3 309 between Cyanobacteria and Actinobacteria and 4 786 between Firmicutes and Actinobacteria.

In the reliable alignments between Cyanobacteria and Firmicutes I counted 676 596 amino acids and observed 453 294 changes, between Cyano- and Actinobacteria 709 912 amino acids and 468 970 changes, and between Firmicutes and Actinobacteria 1 123 782 and 731 366 changes.

As already expected, the species vary in the G+C content. While the Cyanobacteria have

an average G+C content of about 45%, the G+C content of the Firmicutes is much lower at about 35%. The Actinobacteria are even farther away from the Cyanobacteria with an average G+C content of about 62%. The highest G+C content in this study has the Actinobacterium *Streptomyces coelicolor* with 72%, while the Firmicute *Ureaplasma urealyticum* has the lowest with 26% together with a plasmid of *Clostridium perfringens*. The dataset therefore offers both the variaty in G+C content as well as the group specificity of this variaty that I want for the following analysis.

3.2 Instantaneous rate matrices

As I used a window that slides over the range of dissimilarity among sequences, there are no single count matrices on which the instantaneous rate matrices are based on. However, appendix C displays the three count matrices for each pairwise comparison of the phyla.

Vector of frequencies While the count matrices are not of interest, the vector of frequencies π for each comparison is, as it reflects an estimate for the equilibrium frequencies of the Markov model. The equilibrium frequencies are displayed in table 3.1 along with the frequencies that Dayhoff *et al.* [8] and Jones *et al.* [19] found i their own analysis of other proteins.

Approximation of the derivatives I could approximate each of the derivatives of the three pairs of phyla with a correlation coefficient of $R^2 = 0.9987$ or better. The approximation functions for the derivatives are all similar to each other:

$$f_{CH}(D) = -0.4100x^3 - 0.5660x^2 + x \tag{3.1}$$

$$f_{CL}(D) = -0.4062x^3 - 0.5710x^2 + x \tag{3.2}$$

$$f_{LH}(D) = -0.3753x^3 - 0.5936x^2 + x \tag{3.3}$$

Amino-acid	Cyano–Lowgc	Cyano–Highgc	Lowgc–Highgc	Cyano	Lowgc	Highgc	Dayhoff	JTT
Ala (A)	0.082	0.114	0.101	0.097	0.074	0.127	0.087	0.077
Arg (R)	0.045	0.065	0.051	0.056	0.036	0.067	0.041	0.051
Asn (N)	0.042	0.027	0.035	0.034	0.046	0.023	0.040	0.043
Asp (D)	0.048	0.052	0.053	0.046	0.050	0.057	0.047	0.052
Cys (C)	0.009	0.009	0.008	0.010	0.008	0.008	0.033	0.020
Gln (Q)	0.039	0.037	0.030	0.045	0.032	0.028	0.038	0.041
Glu (E)	0.061	0.054	0.057	0.055	0.065	0.052	0.050	0.062
Gly (G)	0.070	0.082	0.078	0.075	0.069	0.087	0.089	0.074
His (H)	0.019	0.022	0.021	0.020	0.019	0.023	0.034	0.023
Ile (I)	0.076	0.051	0.065	0.063	0.084	0.045	0.037	0.053
Leu (L)	0.112	0.117	0.105	0.123	0.101	0.109	0.085	0.091
Lys (K)	0.055	0.029	0.045	0.037	0.067	0.025	0.081	0.059
Met (M)	0.021	0.018	0.023	0.018	0.026	0.019	0.015	0.024
Phe (F)	0.046	0.038	0.043	0.043	0.049	0.035	0.040	0.040
Pro (P)	0.041	0.050	0.041	0.048	0.035	0.049	0.051	0.051
Ser (S)	0.059	0.054	0.057	0.057	0.061	0.053	0.070	0.069
Thr (T)	0.054	0.057	0.059	0.053	0.055	0.061	0.058	0.059
Trp (W)	0.013	0.016	0.014	0.016	0.012	0.016	0.010	0.014
Tyr (Y)	0.036	0.027	0.032	0.030	0.040	0.025	0.030	0.032
Val (V)	0.073	0.082	0.080	0.073	0.072	0.090	0.065	0.066

Table 3.1: Relative frequencies of occurence for each pair of phyla, for each phylum by itself and as published by Dayhoff *et al.* (PAM-Matrix) [8] and by Jones *et al.* (JTT-Matrix) [19] for comparison. Frequencies are based only on the genes that contributed to the estimation of the model. The most remarkable difference is that for Cystein (C): while both Dayhoff *et al.* and Jones *et al.* observed a frequency over 0.020, this analysis only observed frequencies below 0.10. See text for further comparisons and discussion. *Cyano* Cyanobacteria; *Lowgc* Firmicutes; *Highgc* Actinobacteria.

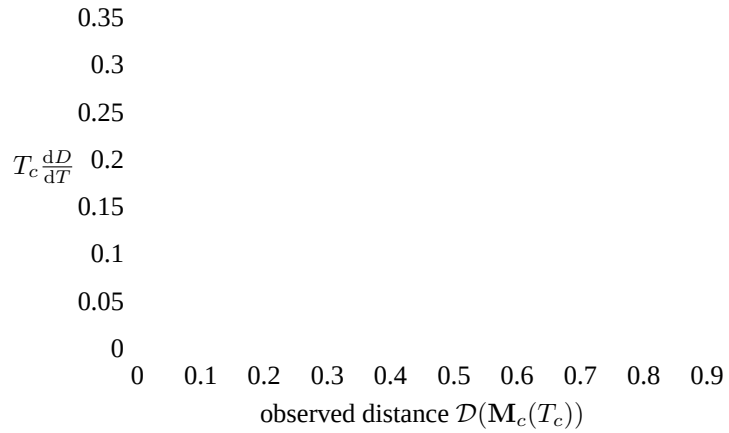

Figure 3.1: Approximation of the derivates for the Cyanobacteria–Firmicutes alignments. The circles are the data points that result from calculating the right hand side of equation (2.9) for each observed distance, while the dashed line shows the approximation function.

f_{CH} approximates the derivative resulting from the alignments of Cyanobacteria and Actinobacteria, f_{CL} for Cyanobacteria and Firmicutes, and f_{LH} for Firmicutes and Actinobacteria.

As an example, figure 3.1 shows the approximation of the derivative for the case of the Cyanobacteria–Firmicutes.

Solving the differential equations With the approximation function we get a differential equation for each pair of phyla

$$T\frac{\mathrm{d}D}{\mathrm{d}T} = f_{CH}(D) \tag{3.4}$$

$$T\frac{\mathrm{d}D}{\mathrm{d}T} = f_{CL}(D) \tag{3.5}$$

$$T\frac{\mathrm{d}D}{\mathrm{d}T} = f_{LH}(D) \tag{3.6}$$

The differential equations are separable and thus can be solved. They yield the following

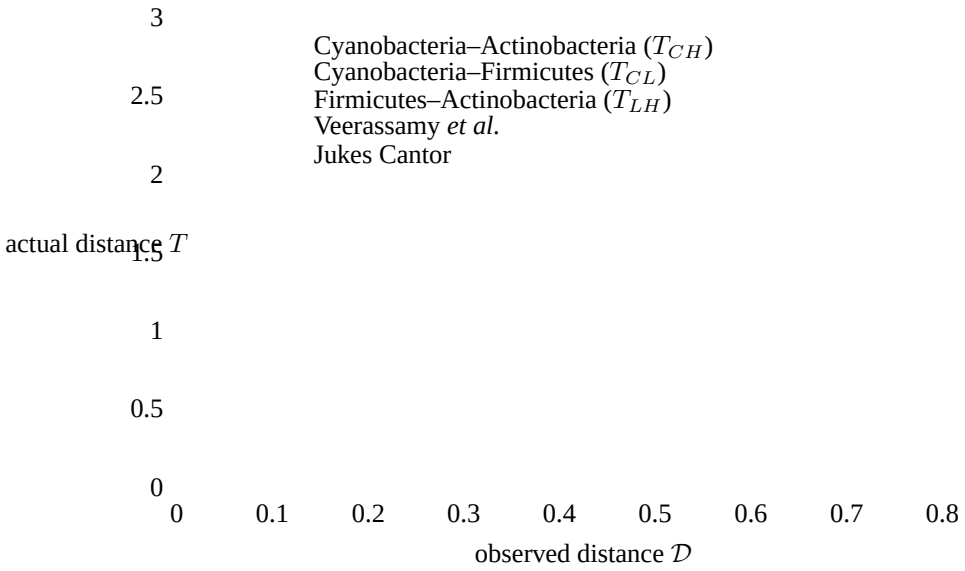

Figure 3.2: Correction functions for estimating the actual evolutionary distance based on the observed distance. Displayed are the correction functions for all three pairs of phyla (Cyanobacteria–Actinobacteria solid line, Cyanobacteria–Firmicutes widely dashed line, Firmicutes–Actinobacteria small dashed line) along with the correction function of Veerassamy *et al.* [29] (dotted line) and the formula of Jukes and Cantor for amino acids [20] (interrupted line). The correction functions for all three comparisons are virtually the same. They are all more severe than the correction function of Veerassamy *et al.* and even more than the one of Jukes and Cantor.

correction functions[5]:

$$\hat{T}_{CL}(D) = \frac{5.654D}{(2.398 + D)^{-0.2979}(8.137 - 8D)^{-0.7021}} \tag{3.7}$$

$$\hat{T}_{CH}(D) = \frac{3.185D}{(48.44 + 20D)^{-0.2956}(1.016 - D)^{-0.7044}} \tag{3.8}$$

$$\hat{T}_{LH}(D) = \frac{1.331D}{(2.605 + D)^{-0.2820}(1.023 - D)^{-0.7180}} \tag{3.9}$$

which are displayed in figure 3.2 against the correction functions of Veerassamy *et al.* [29] and Jukes and Cantor [20]. We see that the correction is more severe, which is not suprising as the species involved are supposed to be very distantly related; we therefore expect that numerous multiple substitutions have happend for which the correction formula must correct.

Deriving the universal rate matrices Given the correction formula I can derive a set of instantaneous rate matrices for each of the three pairs of phyla. With the proper set of weights, I determine the universal rate matrix for each pair of phyla as outlined above. Choosing a set of weights in opposition to just picking the universal rate matrix among the instantantaneous ones turned out to be better in terms of a lower difference as defined in equation (2.19). The universal rate matrices for the evolution between Cyanobacteria and Firmicutes is displayed in table 3.2, the one for the evolution between Cyano- and Actinobacteria in table 3.3 and the one for the evolution between Firmicutes and Actinobacteria in table 3.4.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	−	145	165	154	295	326	761	1066	55	343	613	601	268	175	335	1961	780	37	147	1329
R	276	−	568	143	19	979	707	248	307	246	392	4025	160	86	155	481	361	85	262	245
N	364	687	−	1986	74	723	850	819	641	94	198	982	76	117	174	1542	778	32	247	125
D	275	136	1545	−	10	559	2802	405	203	6	30	544	26	35	290	759	346	14	64	93
C	2767	86	301	44	−	153	$\underline{-111}$	326	119	765	972	122	309	377	94	1550	1321	31	302	1846
Q	744	1207	757	721	38	−	3028	266	658	319	618	2911	419	91	155	715	706	74	238	255
E	1076	532	537	2223	$\underline{-17}$	1851	−	249	201	81	227	1427	17	30	423	586	515	32	138	275
G	1244	155	437	268	42	138	203	−	64	38	122	242	25	69	117	697	170	29	42	106
H	242	711	1225	500	59	1238	625	246	−	176	386	1010	111	404	203	545	317	94	1282	217
I	381	152	51	1	78	159	68	35	44	−	4137	77	640	555	87	151	382	58	111	5515
L	467	159	70	12	84	207	127	80	70	2832	−	247	1110	1108	96	99	257	104	230	1235
K	997	3573	748	504	24	2077	1689	335	385	116	530	−	119	39	529	837	718	60	109	303
M	1024	323	137	58	134	681	37	82	97	2170	5453	286	−	894	112	325	668	199	283	1152
F	319	97	102	36	77	74	48	109	175	941	2777	42	458	−	70	132	191	483	2056	654
P	698	178	168	342	23	143	648	208	95	163	260	682	60	79	−	645	379	39	102	319
S	3097	391	1091	665	274	475	647	923	196	205	198	823	133	107	468	−	2511	33	237	152
T	1242	305	541	307	213	477	577	230	111	537	524	685	278	161	285	2536	−	27	160	1485
W	237	289	95	49	19	211	148	152	131	341	877	237	332	1622	118	146	113	−	1504	363
Y	359	341	276	85	81	254	244	85	716	240	735	162	175	2591	116	372	243	543	−	382
V	1522	149	64	66	241	117	223	101	57	5506	1822	215	341	388	175	111	1074	64	179	−

Table 3.2: Universal instantaneous rate matrix for the evolution between Cyanobacteria and Firmicutes, standardized to the mean rate 1 as suggested by Nick Goldmann [22] (see text). Displayed are the entries mutltiplied by 10^4. The rows contain the amino acids that are substituted. The diagonal elements are determined by the constraint that the rows must sum up to zero. Negative entries are underlined. The total amount of negative rate is $-128 \cdot 10^{-4}$ which makes 0.064% of the total absolute rate represented in this matrix, which is 20.16 (numbers disregard the diagonal entries.)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	−	510	168	323	307	529	776	1120	108	258	781	334	178	145	511	1636	798	73	117	1347
R	898	−	288	265	46	1480	782	373	411	204	579	2233	108	81	270	486	596	125	138	127
N	716	694	−	1935	42	404	673	1072	645	82	160	570	76	83	180	1414	1022	11	194	156
D	700	326	996	−	21	715	2464	587	271	62	25	320	47	31	394	760	500	8	64	17
C	3524	308	107	115	−	−5	−69	349	201	281	910	−34	304	484	150	1093	1020	122	227	1842
Q	1680	2661	293	1046	−1	−	2769	519	685	52	654	1214	409	49	364	765	1054	79	163	457
E	1592	931	335	2374	−11	1818	−	353	240	105	204	786	9	22	417	573	613	45	48	261
G	1541	293	355	374	41	226	231	−	115	6	156	144	39	48	173	741	189	39	32	135
H	563	1204	804	659	88	1125	586	439	−	133	424	414	149	415	288	650	370	164	1184	206
I	571	260	47	63	55	34	111	11	58	−	4455	12	646	534	79	75	422	107	75	7415
L	775	331	37	11	76	211	100	112	83	2049	−	118	909	838	149	123	295	147	191	1730
K	1284	4826	526	587	−7	1427	1453	410	290	22	439	−	114	30	478	567	826	64	124	385
M	1090	325	111	126	159	759	23	173	181	1790	5355	175	−	872	66	355	828	125	231	1162
F	439	140	55	43	124	50	30	104	243	738	2545	23	436	−	78	94	266	564	2049	782
P	1180	352	96	416	31	266	459	285	128	80	345	285	25	58	−	626	473	40	51	335
S	3563	605	741	755	207	526	602	1174	269	79	263	337	130	69	589	−	2836	33	139	202
T	1613	677	488	463	176	669	598	279	145	392	598	424	275	173	413	2609	−	55	110	1596
W	533	515	19	29	75	178	159	198	234	344	1079	124	166	1375	127	114	202	−	1139	253
Y	494	335	194	127	81	215	96	97	949	141	802	132	157	2784	96	277	236	661	−	412
V	1836	94	51	9	217	195	173	136	51	4640	2349	142	266	353	199	122	1086	45	135	−

Table 3.3: Universal instantaneous rate matrix for the evolution between Cyano- and Actinobacteria. See table 3.2 for display information. The total amount of negative rate is $-133 \cdot 10^{-4}$ which makes less than 0.062% of the total absolute rate represented in this matrix, which is 20.70 (numbers disregard the diagonal entries.)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	−	58	266	194	252	360	803	1017	107	453	577	805	250	187	351	1881	830	28	174	1358
R	139	−	567	143	43	909	856	161	419	336	323	4576	185	113	107	488	387	83	262	77
N	789	819	−	2150	43	614	577	1002	730	43	196	556	111	54	192	1467	977	28	156	132
D	377	133	1395	−	11	506	2779	517	203	56	62	653	17	42	253	720	431	11	84	30
C	3207	263	178	67	−	10	−158	360	115	335	846	−231	296	324	84	1265	859	72	252	2051
Q	1241	1510	720	908	3	−	2736	365	604	70	600	2001	437	51	315	760	721	47	167	372
E	1416	725	348	2537	−22	1388	−	294	229	109	148	1317	46	−2	384	511	637	25	113	226
G	1299	95	433	342	36	134	213	−	79	58	80	303	51	61	96	674	184	25	43	67
H	515	974	1190	504	43	848	626	303	−	151	335	890	153	447	157	519	403	86	1236	125
I	700	260	22	47	41	29	97	69	48	−	4099	−154	684	394	108	40	417	66	64	7030
L	571	160	66	36	66	179	81	62	75	2710	−	232	1212	1032	90	95	284	94	205	1314
K	1837	5066	434	773	−40	1330	1698	537	419	−208	506	−	37	−33	569	509	999	94	17	529
M	1056	374	157	38	98	528	107	171	131	1839	4851	56	−	877	69	294	810	71	248	1271
F	443	132	43	53	60	35	−2	114	226	618	2468	−36	509	−	69	98	226	445	1935	833
P	867	128	157	322	17	225	529	183	80	169	224	612	38	72	−	616	390	20	87	300
S	3541	437	943	689	187	410	526	978	200	64	176	412	132	75	465	−	2553	46	137	196
T	1467	330	577	389	119	365	639	252	145	476	494	761	344	168	277	2397	−	33	120	1483
W	205	284	67	47	39	97	101	141	128	299	694	298	120	1336	59	182	137	−	1391	286
Y	558	408	168	137	63	153	207	107	823	135	660	20	192	2616	112	233	217	619	−	550
V	1703	42	56	17	203	138	158	66	32	5763	1618	300	377	439	153	132	1056	48	215	−

Table 3.4: Universal instantaneous rate matrix for the evolution between Firmicutes and Actinobacteria. See table 3.2 for display information. The total amount of negative rate is $-886 \cdot 10^{-4}$ which makes 0.4% of the total absolute rate represented in this matrix, which is 20.23 (numbers disregard the diagonal entries).

Comparing the universal rate matrices While we can take *each* model and use it for phylogenetic tree reconstruction, the question remains, if it is appropriate to use a single model for all species that have such a high variaty in DNA composition as the species in this study. To do so, I compare the three universal rate matrices by taking their elementwise quotients. As they all are normalized to some common mean rate (namely 1), the different equilibrium frequencies won't affect this analysis.

Table 3.5 shows the elementwise quotient of the rate matrices from Cyanobacteria–Actinobacteria alignments and Cyanobacteria–Firmicute alignments. Table 3.6 shows the inverse elementwise quotient. There are great difference between the two rate matrices. Most remarkable the transitions between I and D in 3.5, which have a ten to seventy fold higher rate. The transition between W and C in the same table is four times higher. In the upper left corner are a lot of entries indicating transitions with two to three times higher rate. Entries with five to seven times higher rates can be found in table 3.6, too, *e.g.* V →D, G →I or I →K. Especially the replacement by a K seems to be favored, as almost the whole column shows a rate 1.5 to 2.5 times larger.

Comparing the instantaneous rates of the Firmicutes–Actinobacteria and the Cyanobacteria–Firmicutes alignment (tables 3.7 and 3.8) also reveals increased rates. The peaks aren't as high, though: the highest ones are ten (G →I) and seven (C →K) times increased rates in table 3.7 and eight (A →R) and seven (K →Y) times increased rates in table 3.8. Table 3.8 also shows more rates with more than 1.5 times higher rates, especially in replacing by W.

The last pair of comparison, the alignments Cyanobacteria–Actinobacteria and Firmicutes–Actinobacteria (tables 3.9 and 3.10) contains some very increased rates. In table 3.9 the rates between C and Q are about fifteen times higher and Y →K eight times, while in table 3.10 the rates between I and D show nine and fifty times higher rates. Despite this high peak, higher rates are sparse in this table compared to the other rate matrix quotients.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	35	10	21	10	16	10	11	20	8	13	6	7	8	15	8	10	20	8	10
R	33	–	5	19	24	15	11	15	13	8	15	6	7	9	17	10	16	15	5	5
N	20	10	–	10	6	6	8	13	10	9	8	6	10	7	10	9	13	3	8	13
D	25	24	6	–	20	13	9	14	13	103	8	6	18	9	14	10	14	6	10	2
C	13	36	4	26	–	0	6	11	17	4	9	< 0	10	13	16	7	8	40	8	10
Q	23	22	4	15	0	–	9	20	10	2	11	4	10	5	23	11	15	11	7	18
E	15	18	6	11	6	10	–	14	12	13	9	6	5	7	10	10	12	14	4	10
G	12	19	8	14	10	16	11	–	18	2	13	6	16	7	15	11	11	13	8	13
H	23	17	7	13	15	9	9	18	–	8	11	4	13	10	14	12	12	18	9	10
I	15	17	9	757	7	2	16	3	13	–	11	2	10	10	9	5	11	18	7	13
L	17	21	5	9	9	10	8	14	12	7	–	5	8	8	16	12	11	14	8	14
K	13	14	7	12	< 0	7	9	12	8	2	8	–	10	8	9	7	12	11	11	13
M	11	10	8	22	12	11	6	21	19	8	10	6	–	10	6	11	12	6	8	10
F	14	14	5	12	16	7	6	9	14	8	9	6	10	–	11	7	14	12	10	12
P	17	20	6	12	13	19	7	14	13	5	13	4	4	7	–	10	12	10	5	10
S	12	15	7	11	8	11	9	13	14	4	13	4	10	6	13	–	11	10	6	13
T	13	22	9	15	8	14	10	12	13	7	11	6	10	11	15	10	–	21	7	11
W	22	18	2	6	40	8	11	13	18	10	12	5	5	8	11	8	18	–	8	7
Y	14	10	7	15	10	8	4	11	13	6	11	8	9	11	8	7	10	12	–	11
V	12	6	8	1	9	17	8	13	9	8	13	7	8	9	11	11	10	7	8	–

Table 3.5: Comparison of the universal rate matrices derived from Cyanobacteria–Actinobacteria alignments and Cyanobacteria–Firmicutes alignments. Displayed is the elementwise quotient multiplied by 10 and rounded to the nearest integer. Cells are color encoded as follows: 0 to 15 , 16 to 20 , 21 to 25 , 26 to 30 , > 30 . Cells with negative entries were ommitted and marked with < 0. See discussion for reasoning and discussion. The diagonal elements were omitted for clarity.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	3	10	5	10	6	10	10	5	13	8	18	15	12	7	12	10	5	13	10
R	3	–	20	5	4	7	9	7	7	12	7	18	15	11	6	10	6	7	19	19
N	5	10	–	10	18	18	13	8	10	11	12	17	10	14	10	11	8	29	13	8
D	4	4	16	–	5	8	11	7	8	1	12	17	6	11	7	10	7	17	10	55
C	8	3	28	4	–	< 0	16	9	6	27	11	< 0	10	8	6	14	13	3	13	10
Q	4	5	26	7	< 0	–	11	5	10	61	9	24	10	19	4	9	7	9	15	6
E	7	6	16	9	16	10	–	7	8	8	11	18	18	14	10	10	8	7	29	11
G	8	5	12	7	10	6	9	–	6	64	8	17	6	14	7	9	9	8	13	8
H	4	6	15	8	7	11	11	6	–	13	9	24	7	10	7	8	9	6	11	11
I	7	6	11	0	14	47	6	33	8	–	9	64	10	10	11	20	9	5	15	7
L	6	5	19	11	11	10	13	7	8	14	–	21	12	13	6	8	9	7	12	7
K	8	7	14	9	< 0	15	12	8	13	54	12	–	10	13	11	15	9	9	9	8
M	9	10	12	5	8	9	16	5	5	12	10	16	–	10	17	9	8	16	12	10
F	7	7	19	8	6	15	16	11	7	13	11	18	11	–	9	14	7	9	10	8
P	6	5	17	8	7	5	14	7	7	20	8	24	24	14	–	10	8	10	20	10
S	9	6	15	9	13	9	11	8	7	26	8	24	10	15	8	–	9	10	17	8
T	8	5	11	7	12	7	10	8	8	14	9	16	10	9	7	10	–	5	15	9
W	4	6	50	17	3	12	9	8	6	10	8	19	20	12	9	13	6	–	13	14
Y	7	10	14	7	10	12	25	9	8	17	9	12	11	9	12	13	10	8	–	9
V	8	16	13	75	11	6	13	7	11	12	8	15	13	11	9	9	10	14	13	–

Table 3.6: Comparison of the universal rate matrices derived from Cyanobacteria–Firmicutes alignments and Cyanobacteria–Actinobacteria alignments. The color coding is 0 to 15 | 16 to 20 | 21 to 25 | 26 to 30 | > 30. See table 3.5 for display information.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	4	16	13	9	11	11	10	19	13	9	13	9	11	10	10	11	7	12	10
R	5	–	10	10	23	9	12	6	14	14	8	11	12	13	7	10	11	10	10	3
N	22	12	–	11	6	9	7	12	11	5	10	6	15	5	11	10	13	9	6	11
D	14	10	9	–	10	9	10	13	10	93	21	12	6	12	9	9	12	8	13	3
C	12	31	6	15	–	1	14	11	10	4	9	< 0	10	9	9	8	6	23	8	11
Q	17	13	10	13	1	–	9	14	9	2	10	7	10	6	20	11	10	6	7	15
E	13	14	6	11	13	7	–	12	11	13	6	9	27	< 0	9	9	12	8	8	8
G	10	6	10	13	8	10	11	–	12	15	7	13	20	9	8	10	11	9	10	6
H	21	14	10	10	7	7	10	12	–	9	9	9	14	11	8	10	13	9	10	6
I	18	17	4	561	5	2	14	19	11	–	10	< 0	11	7	12	3	11	11	6	13
L	12	10	9	30	8	9	6	8	11	10	–	9	11	9	9	10	11	9	9	11
K	18	14	6	15	< 0	6	10	16	11	< 0	10	–	3	< 0	11	6	14	16	2	17
M	10	12	11	7	7	8	29	21	14	8	9	2	–	10	6	9	12	4	9	11
F	14	14	4	15	8	5	0	10	13	7	9	< 0	11	–	10	7	12	9	9	13
P	12	7	9	9	7	16	8	9	8	10	9	9	6	9	–	10	10	5	9	9
S	11	11	9	10	7	9	8	11	10	3	9	5	10	7	10	–	10	14	6	13
T	12	11	11	13	6	8	11	11	13	9	9	11	12	10	10	9	–	12	8	10
W	9	10	7	10	20	5	7	9	10	9	8	13	4	8	5	12	12	–	9	8
Y	16	12	6	16	8	6	8	12	11	6	9	1	11	10	10	6	9	11	–	14
V	11	3	9	3	8	12	7	7	6	10	9	14	11	11	9	12	10	7	12	–

Table 3.7: Comparison of the universal rate matrices derived from Firmicutes–Actinobacteria alignments and Cyanobacteria–Firmicutes alignments. The color coding is 0 to 15 , 16 to 20 , 21 to 25 , 26 to 30 , > 30 . See table 3.5 for display information.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	25	6	8	12	9	9	10	5	8	11	7	11	9	10	10	9	13	8	10
R	20	–	10	10	4	11	8	15	7	7	12	9	9	8	14	10	9	10	10	32
N	5	8	–	9	17	12	15	8	9	22	10	18	7	22	9	11	8	11	16	9
D	7	10	11	–	10	11	10	8	10	1	5	8	16	8	11	11	8	12	8	31
C	9	3	17	7	–	157	7	9	10	23	11	< 0	10	12	11	12	15	4	12	9
Q	6	8	11	8	149	–	11	7	11	46	10	15	10	18	5	9	10	16	14	7
E	8	7	15	9	8	13	–	8	9	7	15	11	4	< 0	11	11	8	13	12	12
G	10	16	10	8	12	10	9	–	8	7	15	8	5	11	12	10	9	12	10	16
H	5	7	10	10	14	15	10	8	–	12	12	11	7	9	13	11	8	11	10	17
I	5	6	23	0	19	55	7	5	9	–	10	< 0	9	14	8	37	9	9	17	8
L	8	10	11	3	13	12	16	13	9	10	–	11	9	11	11	10	9	11	11	9
K	5	7	17	7	< 0	16	10	6	9	< 0	10	–	32	< 0	9	16	7	6	65	6
M	10	9	9	15	14	13	3	5	7	12	11	51	–	10	16	11	8	28	11	9
F	7	7	24	7	13	21	< 0	10	8	15	11	< 0	9	–	10	14	8	11	11	8
P	8	14	11	11	14	6	12	11	12	10	12	11	16	11	–	10	10	19	12	11
S	9	9	12	10	15	12	12	9	10	32	11	20	10	14	10	–	10	7	17	8
T	8	9	9	8	18	13	9	9	8	11	11	9	8	10	10	11	–	8	13	10
W	12	10	14	10	5	22	15	11	10	11	13	8	28	12	20	8	8	–	11	13
Y	6	8	17	6	13	17	12	8	9	18	11	80	9	10	10	16	11	9	–	7
V	9	36	12	38	12	8	14	15	18	10	11	7	9	9	11	8	10	13	8	–

Table 3.8: Comparison of the universal rate matrices derived from Cyanobacteria–Firmicutes alignments and Firmicutes–Actinobacteria alignments. The color coding is 0 to 15 , 16 to 20 , 21 to 25 , 26 to 30 , > 30 . See table 3.5 for display information.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	87	6	17	12	15	10	11	10	6	14	4	7	8	15	9	10	26	7	10
R	65	–	5	19	11	16	9	23	10	6	18	5	6	7	25	10	15	15	5	16
N	9	8	–	9	10	7	12	11	9	19	8	10	7	15	9	10	10	4	12	12
D	19	25	7	–	20	14	9	11	13	11	4	5	28	7	16	11	12	7	8	6
C	11	12	6	17	–	< 0	4	10	18	8	11	1	10	15	18	9	12	17	9	9
Q	14	18	4	12	< 0	–	10	14	11	8	11	6	9	10	12	10	15	17	10	12
E	11	13	10	9	5	13	–	12	10	10	14	6	2	< 0	11	11	10	18	4	12
G	12	31	8	11	11	17	11	–	15	1	20	5	8	8	18	11	10	15	7	20
H	11	12	7	13	21	13	9	14	–	9	13	5	10	9	18	13	9	19	10	17
I	8	10	21	13	13	12	11	2	12	–	11	< 0	9	14	7	19	10	16	12	11
L	14	21	6	3	11	12	12	18	11	8	–	5	8	8	17	13	10	16	9	13
K	7	10	12	8	2	11	9	8	7	< 0	9	–	31	< 0	8	11	8	7	74	7
M	10	9	7	33	16	14	2	10	14	10	11	32	–	10	10	12	10	18	9	9
F	10	11	13	8	21	14	< 0	9	11	12	10	< 0	9	–	11	10	12	13	11	9
P	14	28	6	13	18	12	9	16	16	5	15	5	7	8	–	10	12	20	6	11
S	10	14	8	11	11	13	11	12	13	13	15	8	10	9	13	–	11	7	10	10
T	11	21	8	12	15	18	9	11	10	8	12	6	8	10	15	11	–	17	9	11
W	26	18	3	6	20	18	16	14	18	11	16	4	14	10	22	6	15	–	8	9
Y	9	8	12	9	13	14	5	9	12	10	12	65	8	11	9	12	11	11	–	8
V	11	23	9	5	11	14	11	21	16	8	15	5	7	8	13	9	10	9	6	–

Table 3.9: Comparison of the universal rate matrices derived from Cyanobacteria–Actinobacteria alignments and Cyanobacteria–Firmicutes alignments. The color coding is 0 to 15 , 16 to 20 , 21 to 25 , 26 to 30 , > 30 . See table 3.5 for display information.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	–	1	16	6	8	7	10	9	10	18	7	24	14	13	7	12	10	4	15	10
R	2	–	20	5	9	6	11	4	10	16	6	20	17	14	4	10	6	7	19	6
N	11	12	–	11	10	15	9	9	11	5	12	10	15	7	11	10	10	26	8	8
D	5	4	14	–	5	7	11	9	8	9	25	20	4	13	6	9	9	14	13	18
C	9	9	17	6	–	< 0	23	10	6	12	9	68	10	7	6	12	8	6	11	11
Q	7	6	25	9	< 0	–	10	7	9	13	9	16	11	10	9	10	7	6	10	8
E	9	8	10	11	20	8	–	8	10	10	7	17	50	< 0	9	9	10	6	23	9
G	8	3	12	9	9	6	9	–	7	98	5	21	13	13	6	9	10	6	14	5
H	9	8	15	8	5	8	11	7	–	11	8	21	10	11	5	8	11	5	10	6
I	12	10	5	7	7	8	9	63	8	–	9	< 0	11	7	14	5	10	6	9	9
L	7	5	18	32	9	8	8	5	9	13	–	20	13	12	6	8	10	6	11	8
K	14	10	8	13	56	9	12	13	14	< 0	12	–	3	< 0	12	9	12	15	1	14
M	10	12	14	3	6	7	46	10	7	10	9	3	–	10	10	8	10	6	11	11
F	10	9	8	12	5	7	< 0	11	9	8	10	< 0	12	–	9	10	9	8	9	11
P	7	4	16	8	5	8	12	6	6	21	6	22	15	12	–	10	8	5	17	9
S	10	7	13	9	9	8	9	8	7	8	7	12	10	11	8	–	9	14	10	10
T	9	5	12	8	7	5	11	9	10	12	8	18	12	10	7	9	–	6	11	9
W	4	6	35	16	5	5	6	7	5	9	6	24	7	10	5	16	7	–	12	11
Y	11	12	9	11	8	7	22	11	9	10	8	2	12	9	12	8	9	9	–	13
V	9	4	11	20	9	7	9	5	6	12	7	21	14	12	8	11	10	11	16	–

Table 3.10: Comparison of the universal rate matrices derived from Firmicutes–Actinobacteria alignments and Cyanobacteria–Actinobacteria alignments. The color coding is 0 to 15 , 16 to 20 , 21 to 25 , 26 to 30 , > 30 . See table 3.5 for display information.

Notes

[5]These are not the true solutions for the given differential equations. The factors in the denominators should read $(8D - 8.137)$ instead of $(8.137 - 8D)$, etc. However, in the range $[0, 0.9]$ under consideration here, the displayed correction functions are still close enough solutions to the given differential functions; their real part differs only slightly more than the precision of the computing machine (data not shown).

Chapter 4

Discussion

I showed that there are highly different rates of amino acid substitution between Cyanobacteria, Actinobacteria and Firmicutes, thus a single model of evolution for all species in the tree of life might not be suitable to get correct results. A further indication is that the amino acids frequencies observed here differ from those of previous studies on other proteins. While Dayhoff *et al.* and Jones *et al.* mostly observed the same frequency for each amino acids (with the greatest difference being the one of Lysin (K) with 0.022) the frequencies observed here differ from theirs partly greatly. The most remarkable difference is the frequency of Cystein (C) which is about only a third to a half from their frequency. Furthermore, there are several frequencies well above 0.100 which have not been reported by them, *e.g.* Leucin (L), especially for the Cyanobacteria, and Alanin (A) with a peak of 0.127 for Actinobacteria. However, these amino acids are among the most frequent in their analysis, too.

The comparison of the elementwise quotients may suggest higher differences because of approximation and sampling errors. Especially, the seventy times higher rate for D $\rightarrow$I observed in table 3.5 is suspicious and might due to poor sampling of the involved amino. However, evaluating the count matrices reveals that both D and I are not less represented than other amino acids, neither in terms of frequency nor in terms of observed changes. Thus, we can

expect that the order of the rates is close to the real rate.

As noticed above, all phyla contributed approximately equally to the analysis in amounts of counted amino acids, although each phylum started with highly different number of amino acids. A reason for this might be that the set of genes on which the analysis is based on is not only shared between two phyla, but between all three phyla.

The approximation functions and the correction formulas are approximately the same as used in Veerassamy *et al.* [29]. Veerassamy also used the swapping in the denominator of the correction formula, yet without reasoning.

Rate matrices Unfortunately, the rate matrices turn out to have negative entries. Such matrices are called *pseudorate matrices* as they fullfill only two of the three necessary mathematical properties to be a rate matrix of a Markov model: their row sums are zero and their diagonal entries are negative, but their off-diagonal entries are not all positive. In contrast to rate matrices, pseudo rate matrices may yield negative probabilities for small time periods. Determining a rate matrix for a given transition matrix is known to the literature as the *embeddability problem* (*e.g.* [9]): Given a stochastic matrix $\mathbf{P}$, does there exist a rate matrix $\mathbf{Q}$ such that $\mathbf{P} = e^{\mathbf{Q}}$? When this is the case, $\mathbf{P}$ is said to be *embeddable*. According to Devauchelle [9], the problem is solved for small matrices (up to 3×3), but open for arbitrary matrices.

Devauchelle also proofs that there always exists a lower bound τ_0 with the property that for every $\tau \geq \tau_0$ the matrix $\mathbf{P}(\tau) = e^{\mathbf{Q}\tau}$ has only positive entries. Devauchelle gives a sufficient condition for this and a monotonically decreasing function of τ that will eventually reach this condition. I concluded from his proof, that the lower bound τ_0 can be found via the well known interval search: starting with some value $\tau_i = \tau_1$ one determines if $\mathbf{P}(\tau_1)$ has negative entries. If yes, one sets $\tau_{i+1} := 2\tau_i$ and repeats with the new value τ_{i+1}. Otherwise one sets $\tau_{i+1} := 1/2(\tau_i - \tau_{i-1})$ and repeats, using zero for the first run. As the function of τ that reaches the condition is monotonically decreasing, this algorithm converges to τ_0.

Using this algorithm, I found approximate values for τ_0 for each phylum pair. With the

	Cyano–Lowgc	Cyano–Highgc	Lowgc–Highgc
τ_0	33.3%	23.8%	23.8%
D	27.5%	20.8%	20.7%
$\min D$	25.6%	23.6%	21.6%
$\max D$	87.4%	87.6%	87.7%

Table 4.1: Boundaries of the model. The table shows the minimum time τ_0 for which the according model yields positive probabilities and the according observed distance D (as defined in (2.5)). The rows labeled with $\min D$ and $\max D$ display the minimum and maximum observed distance in the according alignment set. Time is measured in expected substitutions per 100 sites.

correction formulas described above, I can give the analogous lower limit for the observed distance. The values are shown in table 4.1. We see that the minimum observed distance at which we get positive probabilities lies below the minimum observed distance that was actually observable in the dataset from which the probabilities were estimated; except for the Cyanobacteria–Firmicutes pair. Although it might be desirable that the model could make estimates below the minimum observed distance present, this boundary is reasonable. The maximum observed distance in the dataset is probably an upper bound, too, as we can't expect to get reliable estimates at this level of divergence. However, if we stay within the given boundaries, we can expect to get a reasonable approximation for model of amino acid evolution.

Appendix A

Mathematics

A.1 Derivation of $\mathbf{P}(T)$

This section describes how we get from the definition of the continuous Markov process

$$\mathbf{P}(T + \mathrm{d}T) = \mathbf{P}(T) + \mathbf{P}(T)\mathbf{Q}\mathrm{d}T \tag{1.1}$$

to the form

$$\mathbf{P}(T) = e^{T\mathbf{Q}} \tag{1.2}$$

(1.1) is basically a first order differential equation. Rearrangement leads to

$$\frac{\mathbf{P}(T + \mathrm{d}T) - \mathbf{P}(T)}{\mathrm{d}T} = \mathbf{P}(T)\mathbf{Q} \tag{A.1}$$

Taking the limit for $\mathrm{d}T \to 0$ results in getting the derivative on the left hand side:

$$\mathbf{P}'(T) = \mathbf{P}(T)\mathbf{Q} \tag{A.2}$$

This differential equation can be solved using Taylor's theorem that states that every function f can be written as

$$f(x) = f(a) + \sum_{i=1}^{\infty} \frac{f^{(i)}(a)x^i}{i!} \tag{A.3}$$

provided the derivatives $f^{(i)}(a)$ (*i.e.* the ith derivative) exist and if x lies in a certain interval.

We know that there can't be any change if no time elapses, thus $\mathbf{P}(0) = \mathbf{I}$. Continously

taking the derivative on both sides of (A.2) gives

$$\begin{aligned} \mathbf{P}''(T) =& \mathbf{P}'(T)\mathbf{Q} = \mathbf{P}(T)\mathbf{Q}^2 \\ \mathbf{P}'''(T) =& \mathbf{P}''(T)\mathbf{Q} = \mathbf{P}(T)\mathbf{Q}^3 \\ & \vdots \\ \mathbf{P}^{(i)}(T) =& \mathbf{P}(T)\mathbf{Q}^i \end{aligned} \tag{A.4}$$

Using Taylor's theorem (A.3) with $a = 0$ and the initial condition $\mathbf{P}(0) = \mathbf{I}$ results in

$$\begin{aligned} \mathbf{P}(T) &= \mathbf{P}(0) + \sum_{i=1}^{\infty} \frac{\mathbf{P}^{(i)}(0)T^i}{i!} \\ &= \mathbf{I} + \sum_{i=1}^{\infty} \frac{\mathbf{P}(0)Q^iT^i}{i!} && \text{using (A.4)} \\ &= \mathbf{I} + \sum_{i=1}^{\infty} \frac{\mathbf{P}(0)(QT)^i}{i!} \end{aligned}$$

which is the definition of the exponential function:

$$\mathbf{P}(T) = e^{T\mathbf{Q}}.$$

The Taylor series for the exponential function is known to converge for any T (e.g. [30]).

Appendix B

Distributing tasks on a set of machines

The repeated BLAST searches and multiple reliable alignments that were performed during the analysis of the data need a lot of computational power. For this purpose, I used a set of machines at the University of Connecticut and distributed the tasks on them. As the machines did not implement some protocol for distributing jobs, I wrote the program `Distributor` that does this. Distributing tasks on various machines is difficult in general, but in this case, the tasks did not depend on each other, *e.g.* the BLAST search for one sequence is independent from the BLAST search of another sequence.

The `Distributor` implements a client server architecture: a server knows about a list of data items that are supposed to be processed by a program and clients on the machines periodically ask the server for new data items, process them until the server signals that all data items have been processed.

I implemented the `Distributor` as a pair of perl modules, `Distributor.pm` and `Collector.pm`. While `Distributor.pm` contains the functions necessary for the server, `Collector.pm` is supposed to be used by the clients to poll the server.

Protocol specification

The communication between client and server follows a simple protocol that is displayed in table B.1. The client contacts the server sending `HELO`, which the server answers with `HELO`, too. The client then sends the command `RECV` upon which the server send the client one or more lines that start with `DATA` and contain the data items. It signals the list of the data items with `DEND`. If there no data items left on the server, the server sends `QUIT` (maybe instead of `DEND`). Instead of the `HELO` command the client can also send the commands displayed in table B.2. If the server does not understand a command it sends `ABRT` to the client along with a human readable reasoning and closes the connection. Sending no command (*i.e.* an empty line) is the proper way for the client to make the server closing the connection.

Client	Command	Server
Contacts the server for new data items	⇒ HELO <id>	
	⇐ HELO <id>	Acknowledges the connection
Performs the command for new data items	⇒ RECV	
	⇐ DATA <data item>	Sends the data to the client. More than one data item can be send per connection (see below).
	⇐ DEND	Signals end of data list and closes the connection.

Table B.1: Specification of part of the protocol between the client and the server. The commands displayed here are used for the regular case when the client needs more data items to process. <id> is an identification number that the server assigned to the client and intended for later use. <data item> is a data item on a single line. The direction in which the commands are sent is indicated by ⇒ (client to server) and ⇐ (server to client). They are not part of the command but only for illustration.

Command	Server action
STAT	Answers with lines of statistical information. Each line starts with STAT, the last line says ENDS. The client can proceed with any command.
DOWN <password>	If the password is the one expected by the server (see below), the server closes the connection and shuts down.

Table B.2: Further commands understood by the server. The statistical output of the server gives information about the amount of data items delivered so far and is intended to be read by humans.

```
#!/usr/bin/perl -w

use Collector;
my $hostname = ‘hostname‘;
chomp $hostname;
print STDERR "janify.pl on $hostname started.\n";

my ($remote_host, $remote_port) = @ARGV;
$result = wrapper(’$HOME/andreas/research/jan/clustalw2.pl’,
                  $remote_host, $remote_port);

print STDERR "janify.pl on $hostname: wrapper returned: $result\n";
```

Figure B.1: Example script that is started by the server as specified in figure B.2. The script prints a message that it was started. It receives the hostname and port of the server as command line arguments and tells the `wrapper` function of the `Collector.pm` module to use J. Gogarten's `clustalw2.pl` program for the multiple sequence alignments. The `wrapper` function repeatedly calls the program with the the data items given by the server until no data items are left. It returns a status code indicating success or failure of the process.

Example program

An example perl program that starts the server is displayed in figure B.2. It first initializes some variables and defines the `nextItem` function. This function is used by the server to receive the next data item. The server is started by the function `runDistribution` of the `Distributor.pm` module. Its arguments are explained in the caption of the figure and the description of the modules in the next section.

The corresponding client script `janify.pl` is displayed in figure B.1. It accepts the hostname and port of the server on its command line and starts the `wrapper` function of the `Collector.pm` module which is explained in the next section. The wrapper function repeatedly calls J. Gogarten's `clustalw2.pl` program that performs the reliable alignments.

```
#!/usr/bin/perl -w

use strict;
use Distributor;

my ($hostname, $hostport) = ("c1n1.math.uconn.edu", 49876);
my @nodes = qw ();
for my $c (1..4) {
  for my $n (1..7) {
    my $node = "c${c}n${n}.math.uconn.edu";
    push @nodes, $node, $node;  # use both processors
  }
}

my @items = <>;
my $alldelivered = 0;
sub nextItem {
  my $nextline;
  if (!$alldelivered && ($nextline = shift @items)) {
    chomp $nextline;
    return "\$HOME/andreas/research/$nextline";
  } else {
    return undef;
    $alldelivered = 1;
  }
}

runDistribution($hostname, $hostport, "neynex", 1,
                \&nextItem, "gogarten", @nodes,
                "nice andreas/research/bin/janify.pl");
```

Figure B.2: Example script that starts the server. This is the script I used to create the reliable alignments for each sequence pairs. The `@nodes` array is loaded with the names of the machines I used. The script receives data items (that are filenames that contain sequences) on stdin and loads them into the `@nodes` array. The `&nextItem` function is periodically called by the server and returns the next file name or `undef` if there are no file names anymore. `runDistribution` is a function exported by the `Distributor.pm` module. It launches the server and the clients on the remote machines and takes the following arguments: hostname and hostport of the server, the password for shutting down the server, the number of data items to be sent per client connection (*i.e.* after the `RECV` command), a reference to a function that returns the next single data item, the username on the remote machines, a array of machine names and the command to execute on each remote machine. `janify.pl` is a client script and is shown in figure B.1. The password is not a real security measure as it is transferred in plain over the network. It is rather intended to make sure the presumed server is shut down.

Module descriptions

The server module The `Distributor.pm` module implements the server along with the protocol as described above. It is derived from the perl module `Net::Server::Single`, which implements a single threaded server. Using a single threaded server avoids the problems arising by concurrent execution of the `nextItem` function or access to global variables, which otherwise had to be serialized in some or another anyway.

The module first inherits from `Net::Server::Single` and defines some global variables that are used later and the prototype for `myfork`. `myfork` takes two references to two functions and their argument arrays, forks the process and calls each function with its arguments in the parent or the child process, respectively.

```
#!/usr/bin/perl -w

package Distributor;
require Exporter;

use strict;
use Net::Server::Single;
use Errno qw(EAGAIN);

our @ISA = qw(Net::Server::Single Exporter);
our @EXPORT = qw (runDistribution);
our @EXPORT_OK = qw();
our $VERSION = 0.1;

my $serverHost;         # server host name
my $serverPort;         # server host port
my $serverPassword;     # server host shutdown password
my $dataPerCon;         # data items to deliver per request
my $nextData;           # function that returns next data string
my $username;           # name on every remote machine
my $nodes;              # array of nodes that we connect to
my $remoteCommand;      # command to launch on remote machine (node)
sub myfork($$);
```

Next it defines `runDistribution`, the interface function for launching the server along with the clients. It accepts arguments as described above.

```
sub runDistribution ($$$$$$$) {
  ($serverHost, $serverPort, $serverPassword,
   $dataPerCon , $nextData, $username, $nodes, $remoteCommand) = @_;
```

The function forks into a process that starts the server

```
  sub server () {
    Distributor->run(syslog_ident => "PerlDistributer",
                     port => $serverPort,
                     host => $serverHost,
                     listen => 10,
```

```
                    group => "nobody",
                    user => "nobody");
  }
```

and one that calls the client. The server is started by the `run` function of `Net::Server::Single`. The client process itself forks again several times, giving rise to a series of processes that connect to the remote machine via ssh and start the specified client script. The stdout and stderr output of each client is logged into a seperate logfile.

```
  sub clients ($){
    my $pid = shift;

    sub ssh ($$$$){
      my $pid = shift;
      my ($logfile, $errlogfile, $node) = @_;
      my @ssh_command = ( "ssh",
                          "$username$node",
                          $remoteCommand,
                          $serverHost,
                          $serverPort);
      my $logfileMark = "*** " . scalar(localtime) .
                           ": New subprocess started.\n";
      open(LOGFILE, ">>$logfile")
                                       or die "Couldn't open $logfile: $!\n";
      open(ERRLOGFILE, ">>$errlogfile")
                                    or die "Couldn't open $errlogfile: $!\n";
      print LOGFILE $logfileMark;
      print ERRLOGFILE $logfileMark;
      open(STDOUT, ">>&LOGFILE")  or die "Couldn't dup STDOUT: $!\n";
      open(STDERR, ">>&ERRLOGFILE")
                                          or die "Couldn't dup STDERR: $!\n";
      exec {"ssh"} @ssh_command
        or die "Could not exec ssh: $!\n";
    }

    sub nothing () { }

    for my $node (@$nodes) {
      print "Launching on node: $node\n";
      my @ssh_args = ($node . ".log",
                      $node . ".err.log",
                      $node);
      myfork &nothing , @{[]}, &ssh, @ssh_args;
    }
  }
  myfork &server, @{[]}, &clients, @{[]};
}   # end of runDistribution
```

The `myfork` function forks into two processes, calling the parent function in the parent process with its arguments and the child function in the child process with its process identification number (PID) and its arguments. `myfork` takes care of errors of the fork system and that the child does not return to the parent process.

```
sub myfork ($$) {
  my ($parentSub, $parentArg, $childSub, $childArg) = @_;
 FORK: {
    local $SIG{CHLD} = "IGNORE";
    my $pid=fork;
    if ($pid) {                        # parent
      &$parentSub(@$parentArg);
    } elsif (defined $pid) { # child
      unshift @$childArg, $pid;
      &$childSub(@$childArg);
      exit;                        # don't let child fall back
                                   # into main code
    } elsif ($! == EAGAIN) { # recoverable fork error
      sleep 5;
      redo FORK;
    } else {
      die "Can't fork: $!\n";
    }
  }
}
```

Clients on various platforms are known to send different end of line indicators. This is taken care of the helping function `netChomp` that removes the end of line characters used in the Net.

```
sub netChomp ($) {
  my $ref = shift;
  $$ref =~ s/\r?\n$//;
}
```

The core function `process_request` of the server handles the requests of the clients. It overrides the function of the `Net::Server::Single` module that handles the client requests and implements the protocol as specified above. The function is called with stdin and stdout bound to the socket to the client.

First, `process_request` sets a timer that causes the connection to being closed after 30 seconds. This prevents the system of being halted completely if a client dies during the transaction of the data items. The actual transaction is evaluated in an `eval` block whose return value `@_` is checked afterwards and an appropriate message is printed.

```
my $dataDone = 0;              # data items delivered so far
my $finished = 0;              # next_arg returned undef.
sub process_request {
  my $self = shift;
  my $timeout = 30;            # 30 seconds timeout
  my $previous_alarm = alarm($timeout);
  eval {

    local $SIG{ALRM} = sub { die "Timed Out!\n" };

      <BLOCK omitted here, see below.>
```

```
  };

  alarm($previous_alarm);

  if ( $@=~/timed out/i ) {
    print STDOUT "Timed Out.\r\n";
    return;
  } elsif ($@ =~ /no input/i) {
    print STDOUT "ABRT empty line\r\n";
    return;
  } elsif ($@ ne "") {
    print $nextData, "\r\n";
    print STDOUT "Internal error: $@\r\n";
    return;
  }
}
```

The omitted block is described here. Basically, it goes step by step trough the specified protocol. It reads a line from the client, matches it with one of the next possible commands and proceeds accordingly. The first command must be either `HELO` followed by some arbitrary number, `STAT` or `DOWN` followed by a password. If it is neither one, the server closes the connection with an `ABRT message`. The `BLOCK` label is used to repeat the protocol, *e.g.* after an `STAT` command, or to exit it prematurely.

```
  my $input;
  BLOCK:
    {      # block to break out via last;
      $input = <STDIN>;
      die "No input\n" unless defined $input;
      my $clientID;
      netChomp \$input;
      if ($input =~ /^HELO\s+(\d+)$/i) {       # HELO
        $clientID = $1;
        print "HELO $clientID Go ahead.\r\n";
      } elsif ($input =~ /^STAT$/i) {              # STAT
        print "STAT $dataDone data items delivered so far.\r\n";
        print "STAT All data items delivered.\r\n" if ($finished);
        print "ENDS\r\n";
        redo BLOCK;
      } elsif ($input =~ /^DOWN\s+(\w+)$/) {   # DOWN
        if ($1 eq $serverPassword) {
          print "DOWN Server shutting down.\r\n";
          $self->server_close;
        } else {
          print "ABRT Not a valid command ($input) or intended quit.\r\n";
          last BLOCK;
        }
      } else {
        print "ABRT Not a valid command ($input) or intended quit.\r\n";
        last BLOCK;
```

```
}
```

After the preliminaries the next command must be the RECV command. The server answers the RECV command by $dataPerCon many DATA lines, each appended by the result of a call of the &nextItem function. $dataPerCon and &nextItem were passed as arguments to runDistribution. The DATA lines are ended by DEND line. If the nextData indicates that there are no more data items, QUIT is sent instead.

```
alarm($timeout);
$input = <STDIN>;
die "No input\n" unless defined $input;
netChomp \$input;
if ($input =~ /^RECV\s*/i) {           # RECV
  for (1..$dataPerCon) {
    if ( my $next_arg = &$nextData() ) { # next DATA
      print "DATA $next_arg\r\n";
      $dataDone++;
      alarm($timeout);
    } else {                           # no more data: QUIT
      print STDERR "All data items delivered.\r\n"  if (!$finished);
      $finished = 1;
      print "QUIT All data items delivered.\r\n";
      last BLOCK;
    }
  }
  print "DEND\r\n";       # DEND: end of current  data set
} else {
  print "ABRT Not a valid command ($input) or intended quit.\r\n";
  last BLOCK;
}
}      # end of block
```

The client module Although not absolutely necessary, a Perld module for a client will ease the client's repeated task of contacting the server and downloading data items from it. I implemented the most often used functions in Collector.pm

The basic function is doConnect which establishes a connection to the server. It uses the Perl module IO::Socket::INET for this task. doConnect accepts the hostname and port of the server as its arguments. Optionally, another argument can specify the number of times doConnect should try to establish the connection until it gives (more or less silently) up.

```
package Collector;
require Exporter;

our @ISA=qw(Exporter);
our @EXPORT=qw(readData doConnect wrapper);
our $VERSION = 0.1;

use IO::Socket::INET;
```

```
my $verbose = 0; # internal use

sub doConnect {
  my ($remote_host, $remote_port, $retries) = @_
    or die "Wrong number of arguments to doConnect (@_)\n";
  $retries = 0 if (!defined($retries));

  my $socket;
  while ($retries >=0) {
    $socket = IO::Socket::INET->new(PeerAddr => $remote_host,
    PeerPort => $remote_port,
    Proto => "tcp",
    Type => SOCK_STREAM);
    last if ($socket);
    warn "Could not connect to $remote_host:$remote_port ($!);            ↩
                                                    $retries retries left.";
    sleep 1;
    $retries--;
  }
  return $socket;
}
```

Based on `doConnect`, `readData` reads the data items from the server and returns them as an array. “No data items left” is signalled by returning an undefined array.

```
sub readData {
  my ($remote_host, $remote_port) = @_;
  my $socket = doConnect($remote_host, $remote_port)
    or die "Cannot connect to server, giving up.\n";

  print $socket "HELO 1\r\n";
  my $answer = <$socket>;
  my @result;
  if ($answer =~ /^HELO 1/) {
    print "Got answer '$answer'\n" if ($verbose);
    print $socket "RECV\r\n";
    while (($answer = <$socket>) =~ /^DATA (.*)\r\n/) {
      print "Got answer '$answer'\n" if ($verbose);
      push @result, $1;
    }
  }
  return @result;
}
```

As the protocol works over regular sockets, the client program doesn’t need not be a Perl program. The `wrapper` allows the usage of arbitrary programs that accept their input from the command line. `wrapper` repeatedly calls `readData` to get the data items. It then passes them to an arbitrary program. Depending on how this program will be called and how long it takes to get started, this approach may add a significant amount of overhead. However, it does not restrict you from implementing the protocol yourself, which is not a hard one anyway.

```
sub wrapper {
  my $command = shift
    or die "Command missing\n";
  my ($remote_host, $remote_port) = @_;
  # Usually, "hgt.mcb.uconn.edu"  49876
  my @data;
  my $atLeastOnce = 0;
  while (@data = readData($remote_host, $remote_port)) {
    system "$command @data";
    $atLeastOnce = 1;
  }
  return $atLeastOnce;
}
```

Using the Distributor module I could—as expected—significantly decrease the waiting time from several days on my single machine to a single day on the set of machines at the University of Connecticut to which I had access to.

The Distributor is surely improvable, *e.g.* a desirable feature would be to repeatedly take notes which data items have been processed already and save this information in case the jobs are aborted so that it can be resumed later at the last save point. Furthermore, it does not aid in collecting the results from the nodes. This was not an issue for me, as the machines I used shared a common file system. For debugging puposes, a more sophisticated approach can be taken instead of splitting stderr and stdout into two different files and not timestamping them. There are also a series of minor improvements imaginable, *e.g.* different usernames on different machins or balancing the workload depending on the speed and success of each individual machine. However, these weren’t restrictive issues for my tasks.

Appendix C

Sequence data

C.1 Count matrices

The count matrices that cover the whole range of alignments are displayed in table C.1 for the phylum pair Cyanobacteria and Firmicutes, table C.2 for the phylum pair Cyano- und Actinobacteria and in table C.3 for the phylum pair Firmicutes and Actinobacteria.

As already mentioned in section 3.2, these are not the count matrices used to estimate the instantaneous rate matrices, as there is no single count matrix in a sliding window approach. However, the matrices summarize the observed changes in amino acid sequence between the phyla.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
F	55495	30500	28243	32470	6108	26212	41016	47576	12708	51415	76099	37041	14482	31390	27419	39582	36448	9128	24207	49057
C	37357	20726	20445	20554	4478	20882	28290	21108	8766	36283	46199	28247	11514	20974	14733	29994	26346	5842	15707	34849
A	9069	641	975	923	347	786	1857	2039	285	1740	1892	1736	583	787	790	2770	1693	142	653	2029
R	609	4887	915	627	48	845	1248	549	360	584	833	3215	248	310	310	833	696	119	452	541
N	426	376	3899	1093	44	384	821	590	273	321	327	991	132	188	192	881	576	33	312	293
D	485	307	1321	5958	41	496	2059	638	271	229	287	984	107	173	286	836	573	38	247	263
C	339	55	109	45	815	69	59	121	42	222	255	113	91	126	42	261	211	20	117	321
Q	717	806	883	920	46	2665	1925	480	393	562	775	2016	252	271	273	822	678	99	386	445
E	710	578	1008	1854	45	967	6363	535	280	440	564	1771	162	235	371	927	726	65	360	488
G	1740	415	1064	803	112	423	848	13234	231	482	616	952	197	348	335	1381	746	96	309	521
H	196	239	443	264	26	292	360	155	1971	215	268	485	84	237	101	274	214	50	436	178
I	843	286	316	216	120	254	463	274	133	7566	3519	594	807	1127	204	526	794	131	563	3557
L	1650	597	670	440	255	665	892	592	310	6432	14950	1307	1962	3054	455	928	1229	356	1338	3789
K	501	940	789	665	41	596	1204	373	216	405	496	4397	171	219	279	658	540	45	276	395
M	312	110	110	80	28	129	140	95	51	711	1030	192	1484	349	48	176	230	52	175	447
F	460	149	289	158	89	189	232	197	185	1343	1915	324	441	5208	148	346	382	257	1624	845
P	759	351	498	680	58	383	960	492	190	522	637	1050	180	260	6343	878	667	70	287	521
S	1752	486	1204	944	154	551	1129	1129	282	694	797	1268	280	377	545	4794	1774	74	472	693
T	1137	418	791	639	138	473	961	541	204	1077	979	1118	352	431	369	1669	5051	65	374	1227
W	150	119	115	75	22	99	124	116	82	304	493	202	146	643	60	148	127	1643	501	244
Y	298	212	287	178	43	181	270	158	294	492	679	370	196	1215	110	311	303	229	4250	418
V	1605	299	405	309	203	351	652	415	167	4781	2916	750	658	1051	372	764	1224	131	581	7104

Table C.1: Count matrix for the interphylum comparison of Cyanobacteria and Firmicutes. The first two lines marked with ‘F’ and ‘C’ show the total number of observed amino acids and changes, respectively. The count matrix is not symmetrized; the amino acids of the Cyanobacteria are on the left, the amino acids of the Firmicutes on the top.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
F	80771	46290	19145	36982	6733	26051	38278	58252	15407	36069	82814	20559	12815	26873	35737	38311	40135	11373	18914	58403
C	53669	30040	13835	23200	4943	21031	26626	27490	10643	27541	48564	16135	10193	17731	18983	29295	29613	7097	12096	40245
A	13551	1641	460	1309	361	685	1473	2787	411	749	2025	528	406	487	1343	2227	1822	233	334	2559
R	1804	8125	333	859	56	872	1105	1042	482	259	889	1090	175	181	607	808	1045	179	193	680
N	948	729	2655	1236	54	350	671	1022	350	141	383	302	97	124	345	823	728	58	175	374
D	1275	758	503	6891	48	431	1633	1103	318	103	329	307	82	107	607	760	708	50	111	342
C	653	140	31	81	895	51	72	177	60	116	309	32	66	121	96	211	234	37	57	365
Q	1776	1928	298	1172	54	2510	1589	903	526	230	823	634	173	176	577	801	934	122	170	639
E	2013	1511	358	2385	35	925	5826	1040	359	196	605	586	122	131	773	851	922	115	140	635
G	3062	888	417	942	112	364	655	15381	296	228	674	292	145	201	654	1147	775	142	179	696
H	527	610	210	372	36	211	343	340	2382	73	301	113	89	167	237	300	278	95	238	209
I	1843	635	150	295	142	269	376	421	210	4264	4103	188	590	742	384	422	886	237	290	4867
L	3763	1474	286	534	280	579	683	1074	472	3198	17125	398	1520	1985	942	921	1561	552	755	5575
K	1212	2167	288	672	37	533	867	646	291	167	554	2212	100	123	497	558	671	94	136	440
M	586	215	55	96	53	176	117	181	77	404	1232	78	1311	256	126	181	318	77	125	651
F	957	404	123	184	103	172	200	346	315	712	2302	116	374	4571	240	291	464	528	1137	1242
P	1636	743	190	734	54	312	626	774	217	221	645	272	91	144	8377	724	726	92	132	637
S	3206	1122	472	1169	152	479	890	1709	359	305	852	358	193	238	965	4508	1926	116	217	849
T	2318	1003	402	865	126	459	781	932	289	489	1044	345	250	261	708	1613	5261	108	179	1520
W	414	293	38	87	37	93	105	172	115	149	597	67	74	377	114	109	167	2138	299	298
Y	681	442	133	235	73	190	210	235	515	249	839	115	150	1029	187	289	329	466	3409	545
V	3155	678	178	398	221	355	528	717	232	2502	3506	261	492	671	611	682	1427	191	317	9079

Table C.2: Count matrix for the interphylum comparison of Cyano- and Actinobacteria. The first two lines marked with ‘F’ and ‘C’ show the total number of observed amino acids and changes, respectively. The count matrix is not symmetrized; the amino acids of the Cyanobacteria are on the left, the amino acids of the Actinobacteria on the top.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
F	113375	56847	39307	59811	8778	34055	64424	87856	23506	73413	118013	51116	25940	47889	46402	64584	65997	16013	36306	90150
C	75215	37525	27961	36127	6310	26397	43454	38320	15718	53269	69703	39610	19970	30359	23538	47522	46837	8939	22610	61982
A	19080	1262	460	1020	484	680	1162	3469	411	969	2137	535	466	509	1427	2789	2181	220	347	3011
R	1487	9661	317	682	51	657	882	838	508	233	725	1095	151	189	589	680	834	175	172	570
N	2273	1716	5673	2798	107	878	1390	2316	772	307	757	703	209	263	752	1802	1624	154	310	656
D	2166	1269	968	11842	59	797	2751	1814	479	198	508	568	104	149	987	1369	1216	102	194	523
C	753	113	49	75	1234	40	46	197	48	132	340	20	59	106	77	264	231	41	48	467
Q	1964	1926	441	1171	48	3829	1644	972	542	249	872	731	225	188	670	923	979	124	203	655
E	3993	2659	744	4133	64	1764	10485	1781	717	391	1002	1183	236	204	1521	1622	1778	180	262	1008
G	4222	806	552	1138	167	436	635	24768	291	283	677	371	178	232	736	1646	984	171	167	750
H	781	865	337	582	49	402	446	516	3894	135	430	238	88	208	306	445	437	137	302	289
I	4039	1121	289	466	328	493	695	901	334	10072	9328	361	1395	1588	862	901	1947	455	653	10622
L	4664	1555	390	619	347	762	836	1221	516	4355	24155	484	2145	2613	1038	1060	1974	703	940	6592
K	3953	6520	844	2058	82	1651	2800	1998	903	519	1494	5753	298	298	1590	1729	2130	243	340	1357
M	1361	491	156	231	104	356	256	429	168	998	2981	160	2985	616	250	417	753	161	255	1450
F	1846	639	204	308	182	264	307	636	501	1362	4643	176	718	8765	455	555	838	966	1846	2250
P	1619	593	199	595	41	285	528	714	204	208	569	257	101	113	11432	731	679	78	106	576
S	5999	1594	959	1731	315	790	1382	2927	484	523	1287	662	318	348	1466	8531	3225	222	317	1378
T	3819	1413	638	1243	249	685	1084	1481	441	869	1640	589	438	442	1155	2863	9580	193	293	2335
W	338	238	35	68	28	80	88	169	97	135	546	49	70	375	101	125	158	3537	320	279
Y	1480	972	295	489	146	364	521	560	1007	565	1820	252	319	2226	446	628	695	1012	6848	1244
V	4919	938	297	499	353	486	759	939	302	4060	5133	369	859	996	914	1046	2304	303	494	14084

Table C.3: Count matrix for the interphylum comparison of Firmicutes and Actinobacteria. The first two lines marked with ‘F’ and ‘C’ show the total number of observed amino acids and changes, respectively. The count matrix is not symmetrized; the amino acids of the Firmicutes are on the left, the amino acids of the Actinobacteria on the top.

Bibliography

[1] J Adachi and M Hasegawa. Model of amino acid substitution in proteins encoded by mitochondrial dna. *J Mol Evol*, 42:459–68, Apr 1996.

[2] SF Altschul, W Gish, W Miller, EW Myers, and DJ Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–10, Oct 5 1990.

[3] DA Benson, I Karsch-Mizrachi, DJ Lipman, J Ostell, and DL Wheeler. Genbank: update. *Nucleic Acids Res*, 32, Jan 2004.

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

[5] CE Bullerwell, J Leigh, L Forget, and BF Lang. A comparison of three fission yeast mitochondrial genomes. *Nucleic Acids Res*, 31:759–68, Jan 15 2003.

[6] RL Burden and JD Faires. *Numerical Analysis*. PWS Kent Publishing, Boston, 4 edition, 1998.

[7] MO Dayhoff, RV Eck, and CM Park. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure*, volume 5. 1972.

[8] MO Dayhoff, RM Schwartz, and BC Orcutt. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure*, volume 5 of *suppl. 3*. 1978.

[9] C Devauchelle, A Grossmann, A Hénaut, M Holschneider, M Monnerot, JL Risler, and B Torrésani. Rate matrices for analyzing large families of protein sequences. *J Comput Biol*, 8:381–99, 2001.

[10] J Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *J Mol Evol*, 17:368–76, 1981.

[11] J Felsenstein. Phylip (phylogeny inference package) version 3.5c. Distributed by the author. `http://evolution.genetics.washington.edu/phylip.html`, 2004. Department of Genetics, University of Washington, Seattle.

[12] WM Fitch. Homology a personal view on some of the problems. *Trends Genet*, 16:227–31, May 2000.

[13] Jan Gogarten. Reliable alignments with clustalw. Kindly provided by the author, September 22 2003.

[14] GH Gonnet, Cohen MA, and Benner SA. Exaustive matching of the entire protein sequence database. *Science*, 256:1443–5, Jun 5 1992.

[15] M Hasegawa, H Kishino, and T Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *J Mol Evol*, 22:160–74, 1985.

[16] S Henikoff and JG Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89:10915–9, Nov 15 1992.

[17] S Henikoff, JG Henikoff, and S Pietrokovski. Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, 15:471–9, Jun 1999.

[18] M Holder and PO Lewis. Phylogeny estimation: traditional and bayesian approaches. *Nat Rev Genet*, 4:275–84, Apr 2003. Review.

[19] DT Jones, WR Taylor, and JM Thornton. The rapid generation of mutation data matrices from protein sequences. *Comput Appl Biosci*, 8:275–82, Jun 1992.

[20] TH Jukes and CR Cantor (editor HN Munro). *Mammalian protein metabolism*, chapter Evolution of protein molecules, pages 21–132. Academic Press, New York, NY, 1969.

[21] M Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J Mol Evol*, 16:111–20, Dec 1980.

[22] C Kosiol and N Goldmann. The different versions of the dayhoff rate matrix. unpublished, personal communication, 2004.

[23] P Liò and N Goldman. Models of molecular evolution and phylogeny. *Genome Res*, 8:1233–44, Dec 1998.

[24] WR Pearson and DJ Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–8, Apr 1988.

[25] Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, and Lipman DJ. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–402, Sep 1 1997.

[26] AD Smith, TW Lui, and ER Tillier. Empirical models for substitution in ribosomal rna. *Mol Biol Evol*, 21:419–27, March 2004. Epub 2003 Dec 05.

[27] JD Thompson, DG Higgins, and TJ Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22:4673–80, Nov 11 1994.

[28] JL Thorne. Models of protein sequence evolution and their applications. *Curr Opin Genet Dev*, 10:602–605, 2000.

[29] S Veerassamy, A Smith, and ER Tillier. A transition probability model for amino acid substitutions from blocks. *J Comput Biol*, 10:997–1010, 2003.

[30] Eric W. Weisstein. Maclaurin series. `http://mathworld.wolfram.com/MaclaurinSeries.html`, 1999. From MathWorld–A Wolfram Web Resource.

[31] YI Wolf, IB Rogozin, NV Grishin, and EV Koonin. Genome trees and the tree of life. *Trends Genet*, 18:472–9, Sep 2002. Review.