High-performance approaches to the comprehensive computation and evaluation of signatures in bacterial sequence datasets

Kai Christian Bader

TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Rechnertechnik und Rechnerorganisation / Parallelrechnerarchitektur

High-performance approaches to the comprehensive computation and evaluation of signatures in bacterial sequence datasets

Kai Christian Bader

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. H. M. Gerndt

Prüfer der Dissertation:

1. Chr. Grothoff, Ph.D. (UCLA)
2. Univ.-Prof. Dr. A. Bode
3. Univ.-Prof. Dr. B. Rost

Die Dissertation wurde am 01.10.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.02.2013 angenommen.

Abstract

This thesis details two new computational methods for the comprehensive search for sequence- and group-specific oligonucleotide signatures in whole genome or marker gene sequence datasets. Designing primers and probes for molecular diagnostic methods depends on the identification of these signatures, the short binding sites on genome or marker gene sequences. CaSSiS, the implementation of this thesis, combines the powerful inexact sequence search capabilities of the ARB PT-Server search index with the structured storage of signature-to-sequence relations. With PLEASE, a client-server application was evaluated which facilitates the access to signature candidates.

This thesis deals with the challenge that the signatures should reliably be detectable within target groups (i.e. have a high group coverage), but not interact with any non-targets that might be in the same sample. The two CaSSiS algorithms both allow strict and relaxed (fault-tolerant) search conditions within user-defined constraints. CaSSiS was successfully tested with multiple versions of the ARB SILVA database, the largest collection of annotated aligned SSU-rRNA sequences of almost full length (> 900 nt).

The first algorithm CaSSiS-BGRT tackles these problems with the newly designed data structure "Bipartite Graph Representation Tree" (BGRT). Results are sorted by their degree of specificity. All signatures guarantee a defined weighted mismatch value as a measurement for the Hamming distance to non-target sequences. The problem, that the search for signatures becomes computationally expensive when working with large collections of deeply hierarchically clustered target (and non-target) sequences was successfully addressed by using bounding methods. The CaSSiS-BGRT structure allows the comprehensive signature computation as well as single queries of freely defined groups.

With our second algorithm CaSSiS-LCA, we present an even more runtime and memory efficient solution for the comprehensive *in silico* search for promising signature candidates even under relaxed search conditions. This is done by not using the intermediate storage as BGRT data structure. Instead, signatures are directly added into the phylogenetic tree structure. As a result, only comprehensive computations are possible, but at a significantly faster runtime and with far less memory consumption.

Even with the optimizations that were applied in the implementation of the two algorithms, processing the amounts of data expected in the future will still be a struggle. The DUP System was used to address a major issue that is expected: the memory limitation of the available search indices, in our case the ARB PT-Server. Multi-stream pipelines were used to distribute

the work load onto multiple nodes within a cluster. The speedups achieved with distributed indices were by themselves clearly not sensational; But they show, that the combination of new algorithmic approaches for the signature selection and the use of distributed index structures allow us to process unreduced datasets and deep hierarchical clusterings we could not process before.

By partitioning the search index in a cluster and additionally using a distributed CaSSiS-LCA topology, we estimate that the CaSSiS-LCA implementation should be able to process genome data of virtually arbitrary size. It would mostly be limited by the size of the structure which is used to store the resulting signature candidates on each node. This structure can then be merged in a final step to gather all results and produce the signature candidate lists.

Besides the command line tool CaSSiS, the client-server application PLEASE was implemented to allow end users to (re-)evaluate the signature candidates without the need for powerful computer architectures. A new, intuitive graphical interface allows group selection within a loaded phylogenetic tree. Single requests are usually processed within a few seconds. The result is a list of signatures with maximum coverage (sensitivity) for each entry within the range of allowed non-target matches, and their thermodynamic characteristics.

Acknowledgements

This work was supported by the Bayerische Forschungsstiftung (BFS) as part of the NANOBAK project (AZ767-07).

Parts of this thesis have already been published. Chapter 3, describing the CaSSiS-BGRT algorithm, is an extended version of an article [12] published in Bioinformatics, that was joint work with Christian Grothoff and Harald Meier. In a bachelor project supervised by me, Sebastian Wiesner implemented an OpenMP-parallelized version of the CaSSiS-BGRT traversal algorithm. Section 3.4 is based on his final report. Chapter 4 is an extended version of an article [10] published in the ACM Journal of Experimental Algorithmics. It is a collaboration with Prof. Mikhail J. Atallah and Christian Grothoff and describes the CaSSiS-LCA algorithm. Chapter 5 on the subject of distributed signature matching is an extended version of a LNCS "Network and Parallel Computing" article [11]. It was joint work with Tilo Eißler, Nathan Evans, Prof. Chris GauthierDickey, Christian Grothoff, Krista Grothoff, Jeff Keene, Harald Meier, Craig Ritzdorf, and Prof. Matthew J. Rutherford. The PLEASE software application (Chapter 6) is based on the ARB Probe Library, developed by Tina Lai, Lothar Richter, and Ralf Westram in 2003. It was refactored and extended in two students projects that were supervised by me: the final project of Peter Klimpt in 2008, and the diploma thesis of Tianxiang Lu in 2011. I am very grateful for the collaboration with all of these co-authors and thank them for their contributions to this thesis.

This work was carried out while I was a research assistant and Ph.D. student at the Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR) at the TU München. I would like to thank Prof. Arndt Bode, head of the chair, for providing me such supportive work conditions. I am particularly grateful to Harald Meier and Christian Grothoff. Harald supported me since my undergraduate studies and he inspired the topic of my dissertation. In Christian, I found a mentor who helped me both professionally and personally get the thesis done. I would like to thank all my colleagues, especially Tilo Eißler, Wolfgang Ludwig, and Ralf Westram, for their suggestions, their support, stimulating discussions, and amusing coffee breaks.

I would like to thank my son Adam Vinzenz for distracting me when I needed a break from my thesis. My family and my friends were an important support for me during my time as a Ph.D. student. And last, but not least, I especially thank my wife Angelika for her love and for being the most supportive person I can imagine.

Figure 1: **Thanks for the distraction, Adam!** Comics used with permission by "Piled Higher and Deeper", Jorge Cham, `www.phdcomics.com`. Originally published 1st – 8th June, 2012.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Structure of the thesis . . . 4

1.3 Molecular biology and genetics . . . 4

1.3.1 Nucleic acids . . . 5

1.3.2 Ribosomal RNA (rRNA) . . . 7

1.3.3 Oligonucleotide primers and probes . . . 7

1.4 Taxonomic classification and phylogenetic trees . . . 8

1.5 Publicly available sequence collections . . . 10

1.6 Outlook . . . 12

2 Related Computational Methods in Bioinformatics **13**

2.1 Thermodynamic prediction and filtering . . . 14

2.1.1 Introduction . . . 14

2.1.2 GC-content . . . 15

2.1.3 Simple melting temperature predictions . . . 15

2.1.4 Nearest-Neighbor predictions . . . 16

2.1.5 Gibbs free energy ΔG° . . . 18

2.2 Approximate Search . . . 19

2.2.1 Search indices . . . 20

2.2.2 The ARB PT-Server and MiniPT . . . 22

2.2.3 Approximate string matching . . . 24

3 The CaSSiS-BGRT Approach **27**

3.1 Introduction . . . 28

3.2 Material and Methods . . . 29

3.2.1 Extraction and evaluation of signature candidates . . . 30

3.2.2 Organizing signature candidates by specificity . . . 32

3.2.3 Determination of valuable signatures . . . 33

3.2.4 Testing conditions . . . 35

3.3 Results . . . 36

3.3.1 Performance of search index and signature candidate evaluation . . . 36

3.3.2 Performance of the BGRT-generation . . . 37
3.3.3 Performance of the BGRT-traversal . . . 37
3.3.4 Signature search space reduction . . . 38
3.3.5 Comparison to other approaches . . . 40
3.3.6 Evaluation of the computed signature collection . . . 42
3.4 Parallelizing the BGRT traversal with OpenMP . . . 45
3.4.1 OpenMP Implementation . . . 46
3.4.2 Benchmark setup and results . . . 47
3.5 Discussion . . . 48

4 The CaSSiS-LCA Approach 55
4.1 Introduction . . . 55
4.2 Related Work . . . 57
4.2.1 Insignia . . . 58
4.2.2 CaSSiS-BGRT . . . 59
4.3 The CaSSiS-LCA Algorithm . . . 60
4.3.1 Perfect Match Algorithm . . . 61
4.3.2 Partial Group Coverage . . . 62
4.3.3 Allowing at most k Outgroup Hits . . . 64
4.4 Implementation and Results . . . 67
4.5 Discussion . . . 73
4.6 Conclusion . . . 74

5 Distributed Signature Matching 77
5.1 Introduction . . . 77
5.2 The DUP System . . . 79
5.2.1 Related Work . . . 80
5.2.2 The DUP Assembly Language . . . 81
5.2.3 DUP System Architecture . . . 83
5.2.4 Generic DUP Stages . . . 84
5.2.5 DUP Programming Philosophy . . . 84
5.3 Material and Methods . . . 86
5.3.1 Testing conditions . . . 86
5.3.2 Adapting the PT-Server for DUP . . . 87
5.4 Results . . . 87
5.5 Conclusion . . . 89

6 Web-based Signature Evaluation with PLEASE 93
6.1 Introduction . . . 93
6.2 Implementation . . . 94
6.2.1 The PLEASE Probe Library . . . 96
6.2.2 The PLEASE Probe Match . . . 100

6.2.3 The PLEASE Client . . . 101
6.3 Results and Discussion . . . 102

7 Conclusion and Future Work 105
7.1 Conclusion . . . 105
7.2 Future Work . . . 106

Appendix 111

A Supplementary Material 111
A.1 Test systems . . . 111
A.2 Signature evaluation . . . 112
A.2.1 Computing the signature dataset . . . 112
A.2.2 Comparison with relevant published signatures . . . 113

B The CaSSiS software package 117
B.1 Availability . . . 118
B.1.1 Building CaSSiS from source . . . 118
B.1.2 Binary packages . . . 119
B.2 CaSSiS command line interface . . . 119
B.3 The CaSSiS result files . . . 122
B.4 CaSSiS graphical user interface . . . 123
B.5 CaSSiS bgrt2Graphviz . . . 123
B.6 Changelog . . . 125
B.7 License for CaSSiS . . . 133
B.8 License for MiniPT . . . 146

Bibliography 149

List of Figures

1 Thanks for the distraction, Adam! . . . VI

1.1 Schema of the CaSSiS pipeline . . . 3
1.2 RNA and DNA structure . . . 5
1.3 Two representations of a phylogenetic tree . . . 10
1.4 Sequence database growth 1 . . . 11
1.5 Sequence database growth 2 . . . 12

2.1 Sample DNA oligonucleotide . . . 15
2.2 Hydrogen bonds between nucleotides . . . 15
2.3 Digital search trees . . . 21
2.4 Depth-limited suffix trie . . . 22

3.1 Schematic of the CaSSiS-BGRT pipeline . . . 30
3.2 CaSSiS-BGRT input data structure . . . 31
3.3 Hamming distance schemata . . . 32
3.4 BGRT construction . . . 33
3.5 Final BGRT structure . . . 35
3.6 CaSSiS-BGRT: Build time and peak memory consumption . . . 37
3.7 BGRT statistics . . . 38
3.8 Efficiency of the bounding methods . . . 39
3.9 Memory consumption of the BGRT structure . . . 40
3.10 Sample signature that cannot be derived from sequence data . . . 42
3.11 GC-content distribution . . . 43
3.12 Melting temperature distribution . . . 44
3.13 Percentage of completely covered sequences and groups . . . 46
3.14 Speedup of the parallelized BGRT traversal . . . 50
3.15 CaSSiS-BGRT workflow . . . 51

4.1 Schematic of the CaSSiS-LCA pipeline . . . 56
4.2 Number of organisms matched per signature . . . 61
4.3 Illustration of the Perfect Match Algorithm . . . 63
4.4 Illustration of the Partial Match Algorithm . . . 65

4.5 Illustration of the Partial Match Algorithm with outgroup hits 68
4.6 Illustration of one iteration in the foreach loop of CaSSiS-LCA 69
4.7 Overall runtime of the CaSSiS-BGRT and CaSSiS-LCA implementations 71
4.8 Comparison: memory consumption of CaSSiS-BGRT and CaSSiS-LCA 72
4.9 Comparison: runtime of CaSSiS-BGRT and CaSSiS-LCA (512, 000 sequences) . . 73

5.1 The DUP specification 82
5.2 Simplified grammar for the DUP Assembly language 82
5.3 Example of a configuration of the DUP System 84
5.4 DUP specification for multiple identical ARB PT-Servers 87
5.5 PT-Server querying using replicated datasets 88
5.6 DUP specification for the partitioned configuration with PT-Server slices 89
5.7 PT-Server querying using dataset partitions 90
5.8 Speedup of string matches for replicated PT-Servers 91

6.1 ARB Probe Library (ARB-PL) client/server structure 95
6.2 PLEASE client/server structure 95
6.3 E-R-Model of the PLEASE database 99
6.4 The PLEASE Probe Library UI 102
6.5 PLEASE Probe Match UI 103

7.1 Schema of the CaSSiS pipeline 106

B.1 CaSSiS homepage 117
B.2 Screenshot: CaSSiS user interface 123
B.3 Signature computation result 125
B.4 Circular visualization of a BGRT 126
B.5 Unrooted visualization of a BGRT 127

List of Tables

1.1 IUPAC nomenclature 6
1.2 Examples for taxonomic ranks 9

2.1 Thermodynamic nearest-neighbor parameters 18
2.2 Specificity and Sensitivity 24

3.1 CaSSiS-BGRT Result Array 34
3.2 CaSSiS-BGRT — Test datasets 36
3.3 Exponential growth in the number of possible *k*-mer signatures 41
3.4 Number of edges and signatures in the bigraph 41
3.5 Comparison to other approaches 45
3.6 Runtime of the OpenMP parallelized CaSSiS-BGRT approach 49

4.1 Notations 61
4.2 Statistical information on the test datasets 70
4.3 Influence of mismatches on the runtime 71

5.1 Summary of multi-stream stages to be used with DUP 85
5.2 Resulting problem sizes for the different numbers of partitions 86

6.1 CaSSiS result_array content 97
6.2 CaSSiS results_X_array content 97

B.1 CaSSiS result_array content 122
B.2 CaSSiS results_X_array content 124

Chapter 1

Introduction

1.1 Motivation

Computational methods play an essential role in the development of oligonucleotide primers and probes, the key diagnostic agents in contemporary molecular technologies. This thesis will present new approaches which extend and improve the central step in the primer and probe design process: the search and evaluation of genetic signatures. These approaches will allow the rapid sensitive and specific identification of single organisms or groups of organisms based on large sequence datasets and phylogenetic trees.

The identification of an organism is done based on its genetic material, more specifically on short matching sites called *oligonucleotide signatures* [120]. In order for the diagnostic methods to work, a signature must reliably be present within the genetic material of a target (group), and be absent in non-targets. Additionally, the signatures' thermodynamic characteristics must be within the constraints of common techniques, e.g. of primers within polymerase chain reactions (PCR) [15] or probes in nucleic acid hybridization [5, 105] based techniques.

In case of well-studied groups of organisms, sources for primers and probes (the counterparts of signatures) can be field-tested oligonucleotides from scientific publications and from curated collections (Section 1.5). Primers and probes from such sources are usually based on sequence datasets that are a few years old. This is not necessarily a disadvantage. But we currently see a vast continuing growth in the size of all kinds of public gene and genome sequence databases — an exponentially growing number of stored nucleotides (Section 1.5). The gap between the number of published oligonucleotide primers and probes and the size of the available sequence datasets widens. The specificity and sensitivity of published signatures therefore needs to be carefully reevaluated with newer datasets before applying them. In cases where the requirements of the detection method or the target group differ from the ones of publicly available primers and probes, an adaption or a redesign from scratch is necessary.

Designing signatures with computational methods and evaluating their applicability in a wet lab are both costly processes — in time and money. Signature search applications are either limited by runtime or memory constraints (or both) regarding the amount of sequence data they are able to process. Compromises must usually be made as the source material has to be

reduced to an amount that is processable on common workstations. Stripping (in most cases non-target) sequences from the dataset might have an influence on the specificity and sensitivity of the resulting signatures.

This thesis will focus on the computational process, but it thereby has also positive influence on the following evaluation in the wet lab. By vastly reducing the computation time it will allow faster evaluation cycles. By allowing to process significantly larger dataset sizes compared to other approaches [10, 12] (Section 3.3.5) it will increase the quality of resulting signature candidates. Another main goal of this thesis will be enabling the comprehensive computation of signature candidates for complete phylogenies. This will further reduce the overall time of the design process. It will enable maintainers of gene and genome databases to precompute and offer collections of signature candidates along with their up-to-date datasets. User can rely on these collections without the need of additional costly computations.

A basic requirement for signature searches is the separation of organisms[1] into target and non-target groups. In general, these group definitions are based in phylogenetic or taxonomic classifications (Section 1.4). Tools that compute phylogenies usually apply heuristics and their results therefore may differ based on the initial parameters and the used dataset revision. Taxonomic classifications were for a long time based on observable features of the organisms (e.g. their cellular structure or metabolism) and have in some cases significantly changed over the last decades. *Bergey's Manual of Systematic Bacteriology* is a good source to track these changes. Both, the computational induced uncertainties and the human factor in subdividing the organisms into groups based on the best knowledge may induce errors in the groupings. Such errors may have a great influence on the design process. A single false negative (i.e. falsely defined as non-target) sequence may sort out an otherwise good signature candidate. The approaches presented in this thesis will therefore allow comprehensive relaxed (fault-tolerant) search conditions. They will allow non-target hits within a defined range to cope with erroneous groupings.

Another very probable source for errors is the sequence material itself. Assuming a constant error rate, the exponential growth of the sequenced material results in an equally growing error rate. Additionally, the error rate largely varies with the applied sequencing technique [74]. These errors have direct influence on the signature search process and indirect influence through falsely annotated datasets, resulting in wrong groupings. The approaches presented in this thesis will deal with erroneous bases in the sequence material by allowing mismatches within a target group.

This thesis will extend and improve the overall search and evaluation of genetic signatures (Figure 1.1). The presented algorithms and implementations, called *Comprehensive and Sensitive Signature Search* (CaSSiS), will be able to comprehensively compute signatures for complete phylogenies or taxonomic groups. Unreduced up-to-date sequence datasets will be used. This will enable sequence database maintainers to provide signature candidates along with their sequence datasets. Non-target hits (false positives) as well as mismatches within a target group will be handled to deal with erroneous bases in the sequence material.

[1]In this thesis it is assumed that each organism is represented by a genetic sequence.

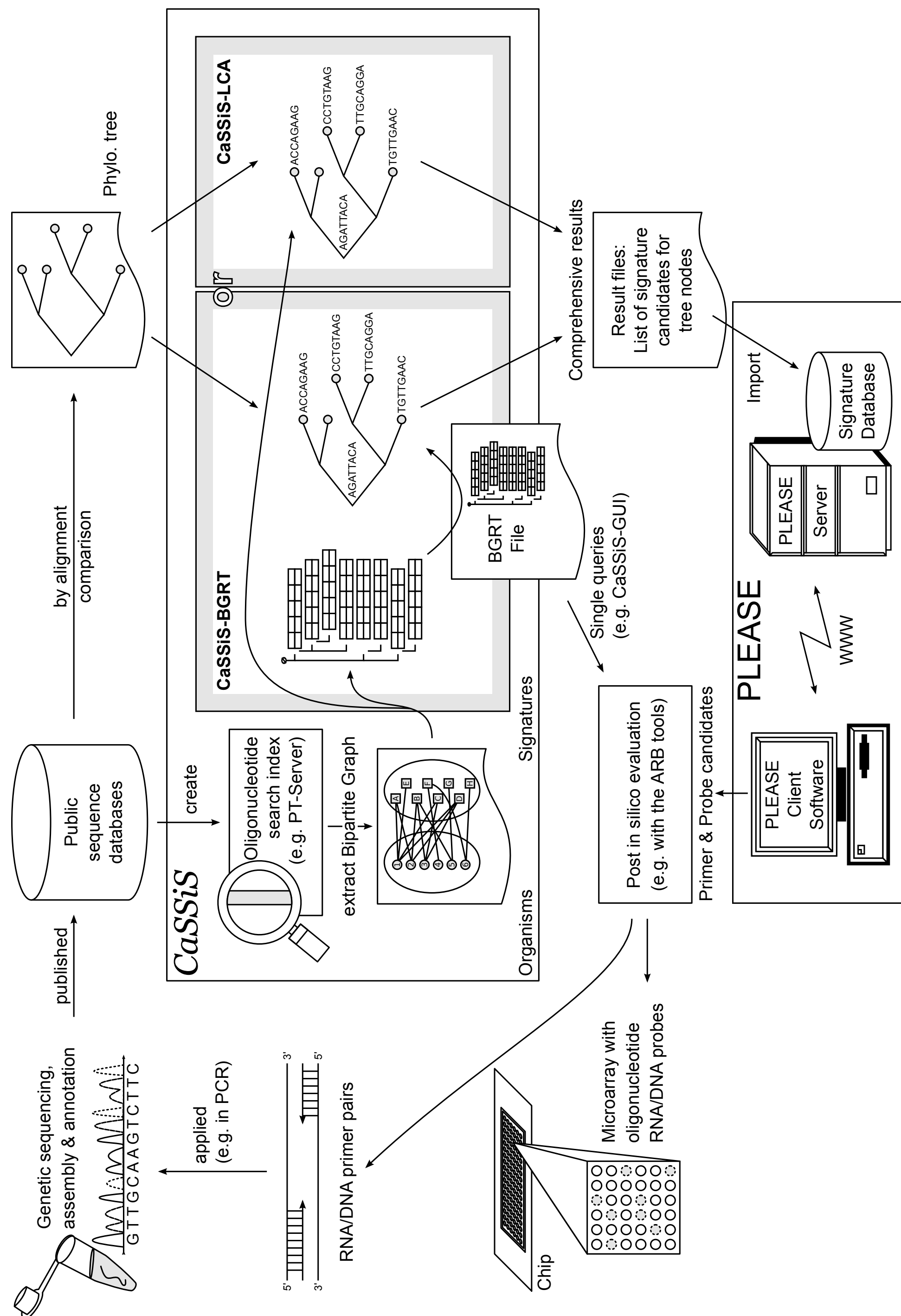


Figure 1.1: Schema of the primer and probe design pipeline in which CaSSiS is embedded. The input data for CaSSiS comes from public sequence and phylogenetic tree databases. The search index (PT-Server) is described in Chapter 2. The two algorithms CaSSiS-BGRT and CaSSiS-LCA are presented in the following Chapters 3 and 4. Not shown is our distributed approach to parallelize the search index with the DUP-System (Chapter 5). The client-server application PLEASE is presented in Chapter 6. The resulting signatures can be used as a template for new RNA/DNA primer and probes, for example to provide diagnostic microarrays.

1.2 Structure of the thesis

All chapters of this thesis are mostly self-contained. This allows reading them without having to read the previous ones. However, the approaches presented in the chapters reference the ones in preceding and subsequent chapters. This applies particularly to the Chapters 3, 4 and 5.

Being an interdisciplinary computer science work, a brief introduction to the biological background is given in the remainder of this Chapter 1. This covers the basic of nucleic acids, classification schemes and sequence collections. Chapter 2 presents related work which was used in the algorithms and the CaSSiS implementations. Chapter 3 describes the CaSSiS-BGRT algorithm, our first approach that brings together comprehensive and relaxed search methods for oligonucleotide signatures. Its design also allows creating precomputed index files to allow single queries based on freely defined groups. To gain additional speedup, a parallelized version of the BGRT traversal is explored in Section 3.4. Chapter 4 presents our second algorithm CaSSiS-LCA. It provides a direct, more memory- and runtime-efficient approach to comprehensively compute signature candidates. In Chapter 5 we evaluate a distributed signature matching approach. Distributing the search index allows the reduction of the memory consumption per partition and simultaneously a speedup. Besides providing comprehensive sets of signature candidates along with the source sequence datasets, different approaches of querying and visualizing signature datasets are evaluated in Chapter 6. Supplementary information on the used datasets and systems that were used in this thesis for testing are given in the Appendix A. The Appendix B contains a brief description ("HowTo") of the CaSSiS implementation. The open source CaSSiS library as its core development and other implementations are presented in the Appendix B. The source code as well as binary releases can be downloaded from the projects website: `http://www.lrr.in.tum.de/~cassis/`

1.3 Molecular biology and genetics

Since this is an interdisciplinary computer science work, terms are used in the following chapters that come from *molecular biology* and *genetics*. The first field covers interactions that happen on a molecular level within cells. The second field addresses the function of genes and their importance in the inheritance of characteristics. In this thesis, the biological focus lies on the components and reactions that are associated with the storage and processing of genetic information, mainly the nucleic acids RNA and DNA (Section 1.3.1). As most of the tests and measurements were computed with 16S SSU rRNA gene sequences, a brief introduction to this special nucleic acid is given in Section 1.3.2. Signature candidates are short sections on the genetic material that are more or less unique within the genetic information of a single organism or a group or organisms. They are the templates for primers and probes, which are presented in Section 1.3.3.

1.3.1 Nucleic acids

Nucleic acids are essential parts of all organisms as carriers of the genetic information. They are long macromolecules, strands, consisting of chained structural units called nucleotides. We distinguish between two types of nucleic acids, the single-stranded ribonucleic acid (RNA) and the double-stranded deoxyribonucleic acid (DNA) (Figure 1.2).

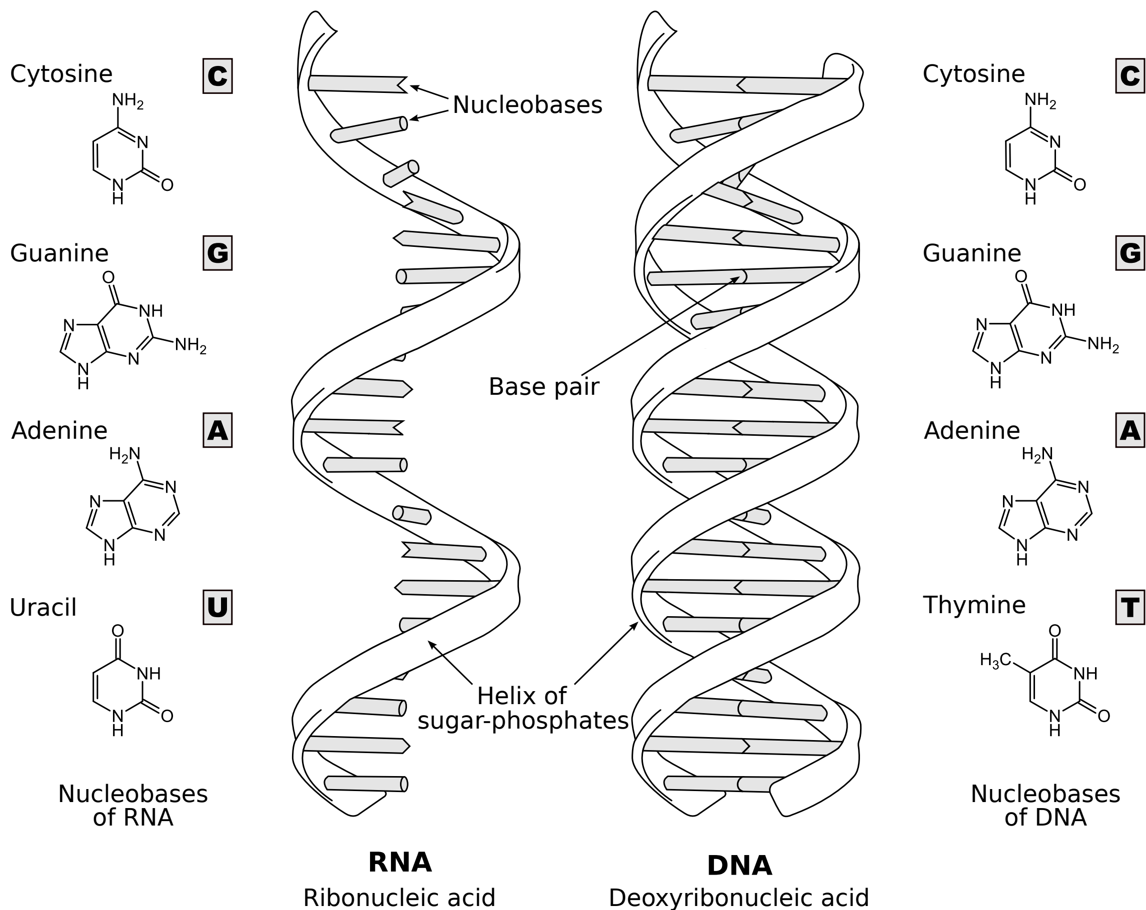


Figure 1.2: Nucleotides are composed of a five-carbon sugar (either ribose or deoxyribose), a phosphate group and a nucleobase. The sugars and phosphate groups build the backbone of ribonucleic acids. The primary nucleobases in DNA are adenine, cytosine, guanine, and thymine. In RNA, thymine is replaced by uracil. (Image adapted from: Sponk/Wikimedia/cc-by-sa)

From a computer scientist's point of view, nucleic acid strands can be seen as a stream of concatenated information units (i.e. bases) and be reduced to a sequence of characters representing these units. The *origin of replication* (ORI or oriC) is used as "base number one" when counting base positions. The bonds of the phosphate-sugar-backbone of a nucleic acid are used as an aid when describing the end-to-end orientation of a strand. The ends are defined as $3' - end$ (three prime end) and $5' - end$ (five prime end), based on which of the five numbered carbon atoms on the sugar ring is "free" for a potential binding. The reading

direction on a strand is called *upstream* towards the $5' - end$ ($3' \rightarrow 5'$) and *downstream* towards the $3' - end$ ($5' \rightarrow 3'$). In case of DNA, the two strands are called "anti-parallel": they run in opposite directions. One strand is the reverse complement of the other [19, Section 5.4]. Therefore only one of the two strains is usually given as sequence string. In most cases it will be in downstream direction as a strain is read in that direction during the process of transcription [27, Section 1.2.2.3].

In the 1970s, the International Union of Pure and Applied Chemistry (IUPAC) began to standardize the nomenclature for nucleic acids [51]. In the following years, this "IUPAC code" was extended [26, 79] to additionally include 11 ambiguity characters for every possible combination of nucleotides (Table 1.1). These ambiguity characters can either be used to describe variations among single positions on related gene sequences, or they may be used to represent positions on the sequence material that are disputable. A similar nomenclature exists for amino acids consisting of 23 characters, representing 20 amino acids[2] and 3 ambiguity characters. The algorithms described in this work are not *per se* limited to a certain type of alphabet, but external factors (project requirements and given tool sets) led to a focus on nucleic acids.

Code					Description
A	A				**A**denine
C		C			**C**ytosine
G			G		**G**uanine
T				T	**T**hymine
U				U	**U**racil
R	A		G		Pu**r**ine
Y		C		T,U	P**y**rimidine
M	A	C			A**m**ino group
K			G	T,U	**K**eto group
W	A			T,U	**W**eak hydrogen bonding interaction
S		C	G		**S**trong hydrogen bonding interaction
B		C	G	T,U	Not A (**B** follows A)
D	A		G	T,U	Not C (**D** follows C)
H	A	C		T,U	Not G (**H** follows G)
V	A	C	G		Not T (**V** follows T and U)
N	A	C	G	T,U	A**n**y base

Table 1.1: IUPAC nomenclature for nucleic acids (upper 5 rows), including ambiguity characters [26] (lower 11 rows). Although a (nucleo)base, a nucleoside (= nucleobase + sugar) and a nucleotide (= nucleoside + phosphate group) are different from a chemical point of view, all three terms can be (and are) used to describe the same genetic information units. The bases are abbreviated by their initial characters: A, C, G, T and U. Another commonly used special character is the "." dot (in rare cases a "-" dash). It represents a deleted base or a gap in a sequence.

Although not being the primary building blocks from which a cell is constructed — this

[2] There are currently 22 "proteinogenic" amino acids known, but only 20 of them can be directly encoded using base triplets [9, 52]

position is occupied by the proteins — nucleic acids are directly and indirectly involved in the construction and the metabolism of cells. Especially RNA sequences are central parts of the synthesis of proteins, which is discussed in more detail in the following Section 1.3.2. Defined subsequences, which specify proteins or functional polypeptides, are called "genes". Nucleic acid sequence information may be also arranged in higher level organization structures, most notably "chromosomes" and "plasmids". The comprehensive genetic material of an organism is called its genome.

1.3.2 Ribosomal RNA (rRNA)

An essential part of a living cell is the ribosome. Ribosomes are found in the organisms of all procaryotes (archaea and bacteria) and eucaryotes. They translate messenger RNA (mRNA) sequences into polypeptide (protein) chains, a catalytic process called protein biosynthesis. In a cell, single nucleotides are attached to short 3-base sequences called transfer RNA (tRNA). The ribosome begins reading the mRNA and allows tRNA triplets to temporarily attach to the reverse complement part on the mRNA. The peptide is connected to the others and the process continues with the next triplet. The result is a polypeptide chain that may be further processed and folded, and finally may form a protein [27, Section 1.2.3.1].

The ribosome itself is built out of subunits of ribosomal RNA (rRNA), most notably the small subunit (SSU) and large subunit (LSU). They are differentiated by their sedimentation coefficients in a centrifuge, measured in Svedberg S. Procaryotes have 70S ribosomes with a 30S SSU and a 50S LSU. Eucaryotes have 80S ribosomes with a 40S SSU and a 60S LSU [27, Section 1.2.3.1].

The rather short rRNA sequence and its vital presence in all organisms early made it an ideal candidate for sequence comparison in evolutionary biology. Especially a part of the procaryotic 30S SSU called *16S small subunit RNA* (16S rRNA) with approximately 1500 bases is used. Due to the importance of ribosomes for protein synthesis, rRNA sequences share a core structure to preserve their functionality. This is reflected by highly conserved regions on the ribosomal RNA sequences [24]. These conserved regions allow the selective enrichment and duplication of the rRNA material with PCR techniques and specific primers. Then, oligonucleotide probes can be used to distinguish between single organisms and groups. The probes rely on the more variable regions on the rRNA sequence, on signature patterns that are only present within the target group.

Both, primers and probes, are further discussed in the following Section 1.3.3. They are crucial for the success of the applied detection methods. A central part of this work is to permit and improve their design. Additionally, the differences between the rRNA sequences of a set of organisms can be used to derive phylogenetic trees, as shown in Section 1.4.

1.3.3 Oligonucleotide primers and probes

Primers are short oligonucleotide sequences that adhere to an opposite (reverse complement) nucleic acid strand. They are artificially created and designed for a particular application and

specific target sequences. The most common application of primers nowadays is the Polymerase Chain Reaction (PCR), a method invented by Kary Mullis in 1983 [78]. In 1993, he was awarded the Nobel Prize in Chemistry for his invention. PCR is used to amplify a certain range of a sequence. The amplification is done by *thermal cycling*, the repetition of three steps: First *denaturation*, where the nucleic acid strands are broken into single-stranded forms by applying heat (over 90°C). During the *annealing* step at about $50 - 60$°C, primers bind to complement sections on the single strands. In a third step called *elongation* at about $75-80$°C, the remaining single strand is completed, beginning at the primer, into a double stranded form.

In modern PCR, usually a special enzyme called "Taq polymerase" is used. This DNA polymerase was discovered in *Thermus aquaticus*, a thermophilic bacterium (it can thrive at high temperatures). The main advantage is its temperature resistance with a maximum activity around $75 - 80$°C [60]. Before this discovery allowed the stable use of thermal cycling, DNA polymerase had to be added after each cycle as it was inactivated due to the high temperature during the denaturation step. The repeated process during PCR results in multiple (in most cases) identical replicates of the original DNA sequence. Usually primer pairs, a forward-primer for the strain and a backward-primer for the opposite strain, are used.

Artificially created oligonucleotide **probes** are very similar to primers. They are used to detect the presence (or absence) of a RNA or DNA sequence by binding to them. The length of probes depends on the used technique: common "short" oligonucleotide probes have length of 20–50 nucleotides (nt), but they can reach lengths up to $1,000$ nucleotides (nt). Longer probes tend to provide a significantly better tolerance of base mismatches and thereby a better target group coverage (sensitivity). Short probes on the other hand are good at discriminating between targets and non-targets (specificity) [55, 92].

Probes are usually tagged ("labeled") with a molecular (e.g. radioactive or fluorescent) marker. A typical application is Fluorescent In Situ Hybridization (FISH) for the "*rapid identification of microorganisms in environmental and medical samples.*" [112]. In DNA microarrays, also called DNA chips, probes are attached to a surface, such as a glass slide. The slide contains microscopic dots of probes. The sample sequences are cleaned and purified. Because the probes are attached to a surface, detection is here done by labeling the target sequences instead of the probes. Targets hybridize against the fixed probes, and sequences that have not hybridized are washed away. The amount of bound DNA material can then be determined using special (fluorescence) scanners.

1.4 Taxonomic classification and phylogenetic trees

Biological classification in its modern form has its roots in the Darwinian principle of a common ancestor. Although evolutionary theories were already discussed before Charles Darwin's epoch-making publication "On the Origin of Species. . . " [28], it popularized the term "evolution" and resulted in classifying organisms on their relationship. Before, the main distinction was made based on variations of observable features like morphologic differences.

A *taxonomy* is a standardized model to classify organisms according to certain criteria.

Taxonomies are hierarchically ordered; in case of microorganisms, the most frequent occurring case in this work, it usually consists of eight subclasses (in descending order):

$$Domain \rightarrow Kingdom \rightarrow Phylum \rightarrow Class \rightarrow Order \rightarrow Family \rightarrow Genus \rightarrow Species$$

Groups of (one or more) organisms are known as taxonomic units, or taxa (singular: taxon). The most common taxonomy is separated into three domains: Archaea, Bacteria and Eucarya [119]. Two examples are given in Table 1.2.

Rank	**Black Death**	**Baker's Yeast**
Domain	Bacteria	Eucarya
Kingdom	–	Fungi
Phylum	Proteobacteria	Ascomycota
Class	Gammaproteobacteria	Saccharomycetes
Order	Enterobacteriales	Saccharomycetales
Family	Enterobacteraceae	Saccharomycetaceae
label Genus	Yersinia	Saccharomyces
Species	Y. pestis	S. cerevisiae

Table 1.2: Overview of the eight common taxonomic ranks used for classification of organisms, here with two examples: The supposed cause of Black Death [45], Yersinia pestis, and Baker's yeast.

Especially microorganisms are difficult to categorize due to their few observable features (mainly their cellular structure and metabolism) [19, Section 18.9]. In many cases, the necessary cultivation is time consuming and possible only with great effort [48]. The renaming and reorganizations of organisms and complete subclasses in the last decades demonstrate how newly achieved knowledge about the relations of organisms influences their taxonomic order. Maybe one of the best example is *Bergey's Manual of Systematic Bacteriology*, where sequencing studies began to take influence on the nomenclature that was at first only based on "classical" taxonomic groupings [19, Section 18.10].

We are now in a situation in which we have access to the sequenced genetic material of numerous (micro)organisms. Artificial scores for the evolutionary distances can be computed, based on the dissimilarities between the sequences of two or more organisms. A variety of algorithms and tools exist to create a *phylogenetic tree* with such scores, e.g. parsimony, maximum likelihood or Bayesian inference [17, Chapter 11]. In a phylogenetic tree, inner nodes represent hypothetical ancestors of the underlying organisms, also called *hypothetical taxonomic units* (HTUs). Taxonomic units (subclasses) that are mapped onto a phylogenetic tree are called *operational taxonomic units* (OTUs). The distance between organisms can be displayed by branch lengths (the longer a branch, the greater the distance to a common ancestor; see Figure 1.3).

One of the most common representatives that is used to compute phylogenetic trees is the SSU rRNA 1.3.2 gene. It es essential for the protein biosynthesis and therefore found in all organisms. Being rather "short" with of about 1500 bases and having a mixture of curated and highly variable regions make it an ideal candidate. Multiple sources exist where rRNA sequence

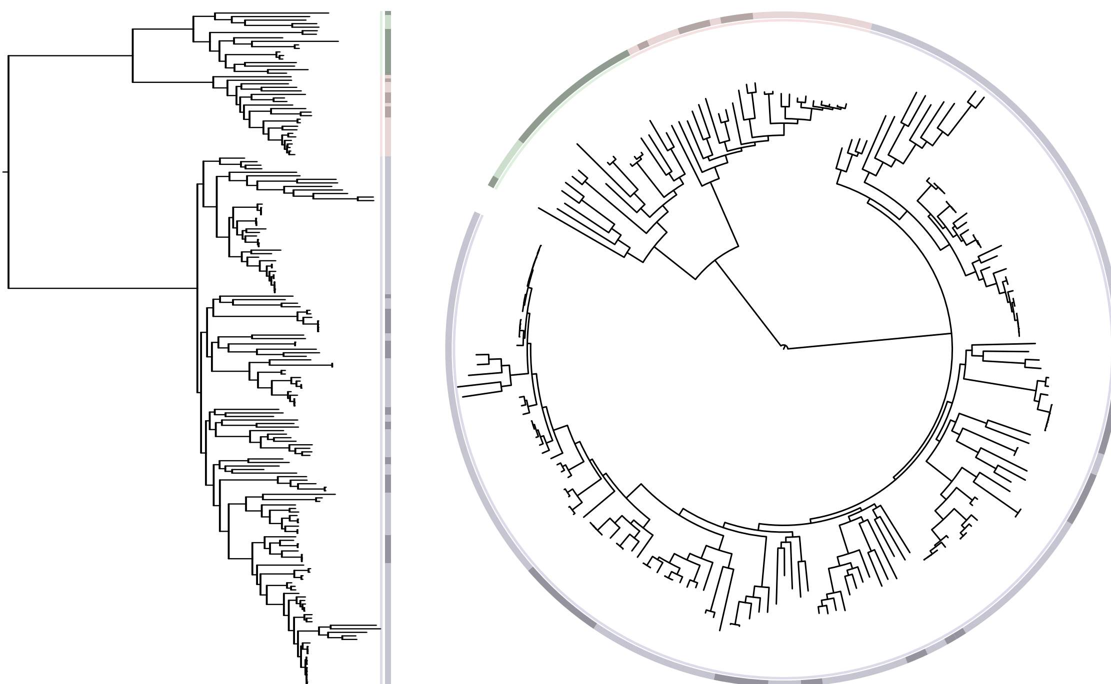


Figure 1.3: Two representations of an exemplary phylogenetic tree, generated with the Interactive Tree Of Life (iTOL) tool [63]. In the left, the tree is represented as as "dendrogram", in the right as a "circular cladogram".

datasets and corresponding phylogenetic trees can be downloaded, which will be discussed in the following chapter.

1.5 Publicly available sequence collections

In 1965 the Integrated Electronics (today better known as Intel) co-founder Gordon E. Moore formulated a 10-year prognosis of the growth of the number of components placed on an integrated circuit:

> "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years." [77, slightly abridged]

More than 40 years later this prognosis has become well known under the name "Moore's law". Although the interpretation over time slightly changed to doubling the number of transistors every 18 months, the once predicted trend still seems to continue at least for the next few years.

Similar predictions will probably soon occur for the field of genomics, as we currently see a vast continuing growth in the size of public genome databases — and lately also in the number

of sequenced organism genomes. One reason for this growth are the falling sequencing costs. Another reason is the increasing throughput of the sequencers. Time will tell if these predictions will prevail, but we can already see that the growing amount of publicly available sequence data represents a chance as well as a burden.

The growing number of genetic information needs to be carefully handled. The first (viral) genomes sequenced in the late 1970s contained about $3.5 - 5.4$ kilobases (kb) [36, 97] and at that time the genetic information was collected mostly "by hand". The first bacterial genome with a length of 1.8 megabases (Mb) was sequenced in 1995 [39]. At that time sequence datasets were already being processed with software tools. Today, typical public available DNA sequence collections contain billions of bases (Figure 1.4). A source for the ribosomal RNA gene sequences and corresponding phylogenetic trees that were used as test datasets in this thesis are the The Ribosomal Database Project (RDP-II) [71] and the non-redundant ARB SILVA database [89]. Both projects provide free access to curated datasets (Figure 1.5).

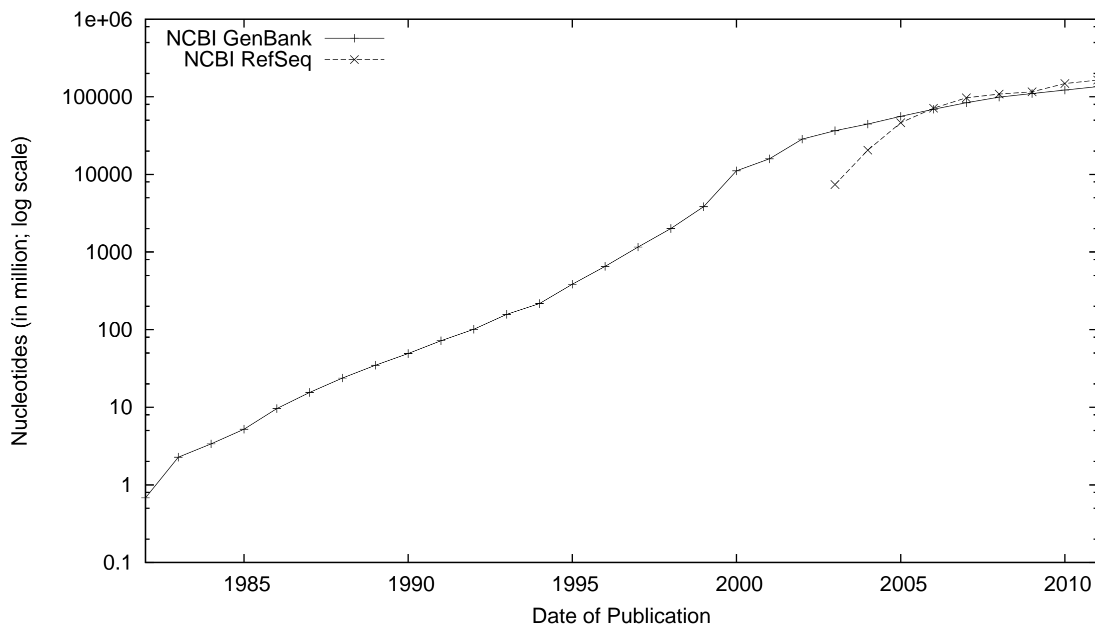


Figure 1.4: Two of the largest sequence databases are the GenBank collection of all publicly available annotated DNA sequences and the Reference Sequence (RefSeq) collection for genomic DNA, transcripts, and proteins. Both are located at the National Center for Biotechnology Information (NCBI). The RefSeq release no. 53 from May 10, 2012 contains $17,339$ organisms with $\sim$ 175.3 billion bases (nucleotides). This figure shows the exponential growth of both databases since 1982 (Genbank) and 2003 (RefSeq). Sources: `http://www.ncbi.nlm.nih.gov/RefSeq/` and `http://www.ncbi.nlm.nih.gov/genbank/`

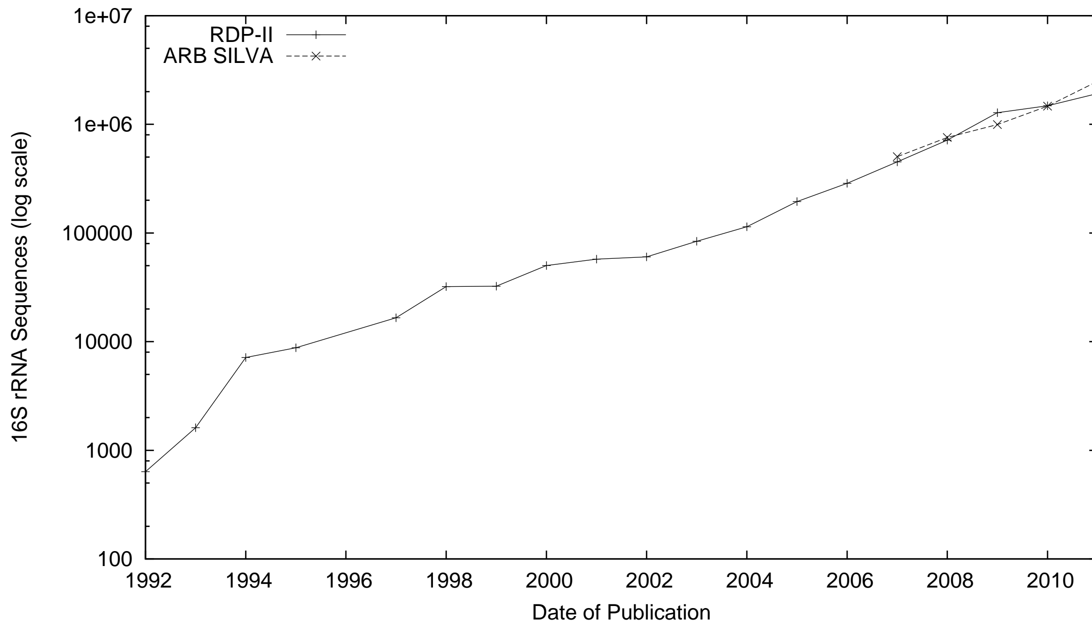


Figure 1.5: Growth rate (in 16S rRNA gene sequences per year) of the rRNA databases The Ribosomal Database Project (RDP-II) and ARB SILVA between 1992 and 2012. RDP-II currently provides free access to 2,320,464 16S rRNA sequences (Release 10, June 1, 2012). The latest ARB SILVA release *SSU Ref NR 108* from September 2011 contains 618,442 quality checked rRNA sequences and additionally provides a phylogenetic tree. The statistics were taken from the release notes of ARB SILVA and RDP-II websites. Sources: `http://www.arb-silva.de/no_cache/download/archive/` and `http://rdp.cme.msu.edu/`

1.6 Outlook

A main goal of this thesis is the comprehensive computation of signature candidates for large sequence datasets and corresponding phylogenies. Maintainers of the previously mentioned large gene and genome databases can use this to precompute and offer collections of signature candidates along with their up-to-date datasets. A crucial part of this design process is building a bipartite graph that contains the links between sequences and signatures. Efficient search indices are used to find match locations of substrings on indexed string sets. The algorithms that are presented in the following Chapters 3 and 4 are using the ARB PT-Server as search index. A brief introduction to indices and the PT-Server is given in the following Chapter 2.

Another important factor is the evaluation of the usefulness of a signature by predicting its thermodynamic characteristics. These characteristics may differ depending on what is going to be designed (primer or probe) and the biotechnological process that is going to be used. The following chapter provides background on the thermodynamic formulas that were used in the implementation of this thesis.

Chapter 2

Related Computational Methods in Bioinformatics

The search for patterns in sequence datasets quickly becomes a problem difficult to solve when applying naive search approaches without reducing the search space. Even widely applied approaches and tools (Section 3.3.5) either run into runtime constraints or lead to excessive memory consumption — or both — when processing a typical gene or genome dataset. This chapter contains necessary preparatory work in order to attenuate computational requirements for the approaches which will be presented in the following chapters.

The most obvious step is the reduction of the overall number of processed k-mer signatures. When dealing with RNA or DNA signatures of a defined length k, 4^k unique variations exist — but not all of them might be worth to be processed. In Section 3.3.4 the pre-filtering of signatures is discussed from a computational point of view. In Section 2.1 biochemical aspects are considered that may influence the search for valuable signature candidates, especially thermodynamic requirements.

Under certain conditions, a primer or probe may bind even if it does not completely match the corresponding signature site on the organism's genetic material. Unsuitable environmental conditions, on the other hand, can lead to failing bindings even if a perfect match exists *in silico*. Relaxed search methods that were applied in this work cope with such problems up to a certain degree, and they may also counteract typical errors in the datasets themselves. They are discussed in Section 2.2.3.

Efficient methods for the storage and search within sequence collections have to be applied. On the one hand signatures have to be generated from the sequences, on the other hand they have to be matched against the dataset. The implementation in this work is a way to cope with this problem.

2.1 Thermodynamic prediction and filtering

In the following subsections, thermodynamic models are discussed for their use as pre-filters. They can further reduce the signature search space by, for example, dropping signature candidates whose thermodynamic parameters are outside the boundaries defined by the selected detection method.

Whether a primer or probe binds to a signature site, or not, is (unfortunately from a computer scientist's point of view) not a binary effect. The biochemical processes that let two oligonucleotide sequences bind to each other are influenced by a number of factors: the base types and their sequential arrangement, the detection method, the conditions of the environment at which the hybridization takes place — just to name a few. All these factors have influence on the thermodynamic stability of the RNA or DNA duplex and thereby on the quality of the primer or probe.

A lot of thermodynamic models have been designed throughout the last half-century to calculate the stability and temperature-dependent behavior of RNA and DNA duplexes. They range from simple ones like the GC-content (Section 2.1.2) and the simple melting temperature predictions by Marmur or Wallace (Section 2.1.3) to complex predictions based on the ΔG Gibbs free energy (Section 2.1.4).

2.1.1 Introduction

The interaction that happens between two complementary DNA strains that form a double stranded helical structure or an RNA-RNA hybrid is called *hybridization*. Under normal conditions (e.g. 37°C) the two complementary DNA strands are only when being replicated (transcription) or when being repaired in a single stranded form. The process of forcing DNA to split up into its single stranded form by breaking the hydrogen bonds is called *denaturation* or *melting*. This process is usually heat-induced. The contrary process of RNA or DNA pairing to a complementary sequence by hydrogen bonds is called *annealing*. Single stranded RNA typically binds to complementary regions on its own strand forming secondary structures. Often, these secondary structures are important for metabolic or catalytic processes in which the RNA is directly involved.

The temperature at which annealing and denaturation take place is essential for the specificity of an oligonucleotide when binding to its target. Working at wrong temperatures can lead to incorrect bindings of primers and probes. A key parameter here is the melting temperature T_m. It is defined as the temperature at which the number of two bound oligonucleotide strands and the number of single stranded ones are in equilibrium, i.e. 50% of the oligonucleotides are forming a duplex. Annealing or hybridization is usually performed at temperatures a few degrees below that temperature, and denaturation significantly above it [95]. The length and concentration of RNA or DNA oligonucleotides are the main factors which influence the melting temperature. Besides, the salt concentration and present denaturants can also influence it.

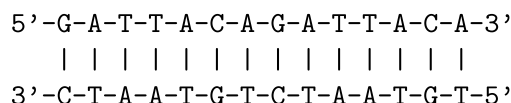


Figure 2.1: In the following sections, this DNA oligonucleotide with 14 bases will be used as a sample for the thermodynamic calculations.

2.1.2 GC-content

The guanine-cytosine content (GC-content) is the percentage of the nucleotides guanine (`G`) and cytosine (`C`) that are present in an oligonucleotide strand:

$$\%\mathrm{GC} = \frac{|\mathtt{G}| + |\mathtt{C}|}{|\mathtt{A}| + |\mathtt{T}| + |\mathtt{G}| + |\mathtt{C}|} \cdot 100\% \tag{2.1}$$

The GC-content of the sample DNA oligonucleotide in Figure 2.1 is 28.6%.

A high GC-content is an indicator for a higher melting temperature of an oligonucleotide sequence. Guanine and cytosine pairs bind by three hydrogen bonds, while adenine and thymine (uracil) pairs only bind by two hydrogen bonds (Figure 2.2). A higher number of hydrogen bonds stabilizes the oligonucleotide duplex and thereby results in a higher thermal stability [19, page 142].

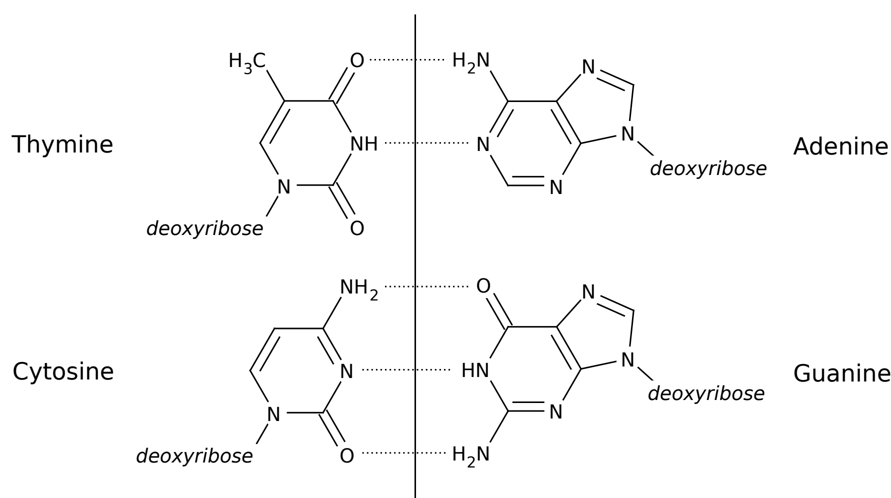


Figure 2.2: Cytosine and Guanine share three hydrogen bonds, Thymine and Adenine only two. (Not shown: Uracil also binds only with two hydrogen to Adenine.) RNA or DNA with a high GC-content tends to be more stable due to the higher number of hydrogen bonds.

2.1.3 Simple melting temperature predictions

Several simple formulas exist to predict the melting temperature of oligonucleotides. Maybe the most simple one is known as the “2+4 rule” or Wallace-rule [113]. It can only be applied

for short DNA oligonucleotide sequences for about 14–20 base pairs (at a salt concentration of 0.9M NaCl):

$$T_m = (|\texttt{A}| + [\texttt{T}]) \cdot 2°\text{C} + (|\texttt{G}| + |\texttt{C}|) \cdot 4°\text{C} \quad (2.2)$$

Guanine and Cytosine with the possibility of building three hydrogen bonds lead to a higher melting temperature; their number of occurrence is multiplied by 4°C. Adenine and Cytosine only lead to an increase of 2°C per base.

Another simple formula to compute the melting temperature for short oligonucleotide sequence lengths was defined by Marmur and Doty [73]:

$$T_m = 64.9°\text{C} + \frac{41°\text{C} \cdot (|\texttt{G}| + |\texttt{C}| - 16.4)}{(|\texttt{A}| + |\texttt{T}| + |\texttt{G}| + |\texttt{C}|)} \quad (2.3)$$

Both formulas are still widely used for a first approximate determination of the melting temperature of short oligonucleotide primers and probes. Many software tools or calculation servers [83, suppl.mat.] use a combination of both approaches for a fast prediction. For oligonucleotides up to 13 base pairs the formula according to Wallace is used and for longer oligonucleotides the formula according to Marmur and Doty.

In case of our example from Figure 2.1 with 14 bases, T_m is 28.6°C according to equation (2.3). (Equation (2.2) would lead to 36°C and would diverge even more for longer oligonucleotides.)

2.1.4 Nearest-Neighbor predictions

Instead of solely focusing on the interactions between complementary base pairs of two oligonucleotide strands, nearest-neighbor methods [18, 108, 98, 111] also take neighboring bases, i.e. the base composition, into consideration. The prediction of the melting temperature T_m for complementary oligonucleotides is based on the thermodynamic relation between the entropy S and enthalpy H. In thermodynamics, the entropy change ΔS is a measure of the randomness or uniform distribution of energy within a process, having its maximum at the equilibrium of a reaction. The change in enthalpy (or heat content) ΔH describes the energy that is emitted during a reaction. A positive enthalpy value describes an endothermic (energy absorbing) reaction, a negative value an exothermic (energy releasing) one.

The entropy ΔS and enthalpy ΔH of an oligonucleotide duplex can be calculated by summing up the respective values for the dinucleotides, the initial and terminal nucleotide, and symmetry terms [111]:

$$\Delta H° = \left(\sum_{i=1}^{n-1} \Delta H°(s_i, s_{i+1}) \right) + \Delta H°(init.\ s_1) + \Delta H°(term.\ s_n) + \Delta H°(symm.) \quad (2.4)$$

where n is the number of nucleotides in the sequence $S = 5' - (s_1, \ldots, s_n) - 3'$ and $\Delta H°(i)$ is the enthalpy of the dinucleotide at position i (from the 5'- to the 3'-end). The same formula (2.4) applies for the entropy $\Delta S°$. If the oligonucleotide is self-complementary, the symmetry correc-

tion value $\Delta H^\circ(symm.)$ in the equation is added — otherwise it is zero. Corresponding tables with the thermodynamic parameters can be taken from published data, e.g. from Sugimoto [108] or SantaLucia [98]. The latter parameters are shown in Table 2.1 on page 18.

Based on the summed up values for the entropy and enthalpy, the following equation [111] is used to calculate the melting temperature T_m for the oligonucleotide sequence

$$T_m = \frac{\Delta H^\circ}{\Delta S^\circ_{\text{corr}} + \ln(\frac{c[\text{oligo}]}{2}) \cdot 1.987 \frac{\text{cal}}{\text{K}\cdot\text{mol}}} - 273.15\text{K} \tag{2.5}$$

where c[oligo] is the concentration of the oligonucleotides (in mol/l). If the oligonucleotide is self-complementary, its concentration c[oligo] in the denominator in the equation is divided by 4 instead of 2.

The factor $\Delta S^\circ_{\text{corr}}$ is correcting the influence of the salt concentration on the entropy [111]:

$$\Delta S^\circ_{\text{corr}} = \Delta S^\circ + 0.368 \cdot n \ln(c[\text{salt}]) \tag{2.6}$$

where n is the number of nucleotides and the salt concentration c[salt], normalized to mol/l, is given as

$$c[\text{salt}] = c[\text{Na}^+] + c[\text{Mg}^{2+}] \cdot 140 \tag{2.7}$$

The enthalpy is not influenced by the salt concentration and therefore needs no correction.

In the following example, the DNA duplex from Figure 2.1 is again used to calculate the melting temperature with the nearest-neighbor method. The entropy and enthalpy changes are determined by stepping through the dinucleotides from the 5'- to the 3'-end. The respective values from the Table 2.1 are then added up (calculation was truncated):

$$\begin{aligned} \Delta H^\circ =& \Delta H^\circ(\text{init.}\mathtt{G}) + \Delta H^\circ(\mathtt{GA}) + \Delta H^\circ(\mathtt{AT}) + \Delta H^\circ(\mathtt{TT}) + \\ & \dots \\ & + \Delta H^\circ(\mathtt{TA}) + \Delta H^\circ(\mathtt{AC}) + \Delta H^\circ(\mathtt{CA}) + \Delta H^\circ(\text{term.}\mathtt{A}) \\ =& -100.2\ \frac{\text{kcal}}{\text{mol}} \end{aligned} \tag{2.8}$$

$$\begin{aligned} \Delta S^\circ =& \Delta S^\circ(\text{init.}\mathtt{G}) + \Delta S^\circ(\mathtt{GA}) + \Delta S^\circ(\mathtt{AT}) + \Delta S^\circ(\mathtt{TT}) + \\ & \dots \\ & + \Delta S^\circ(\mathtt{TA}) + \Delta S^\circ(\mathtt{AC}) + \Delta S^\circ(\mathtt{CA}) + \Delta S^\circ(\text{term.}\mathtt{A}) \\ =& -282.1\ \frac{\text{cal}}{\text{mol}\cdot\text{K}} \end{aligned} \tag{2.9}$$

Under the assumption that the concentration of the salt $c[\text{salt}] = 1$ mol/l (so no correction of the enthalpy is necessary) and the oligonucleotide concentration is $c[\text{oligo}] = 0.01\mu$mol/l, the

Dinucleotide	ΔH° in $\frac{\text{kcal}}{\text{mol}}$	ΔS° in $\frac{\text{cal}}{\text{mol}\cdot\text{K}}$	ΔG°_{37} in $\frac{\text{kcal}}{\text{mol}}$
AA	-7.9	-22.2	-1.00
AC	-8.4	-22.4	-1.44
AG	-7.8	-21.0	-1.28
AT	-7.2	-20.4	-0.88
CA	-8.5	-22.7	-1.45
CC	-8.0	-19.9	-1.84
CG	-10.6	-27.2	-2.17
CT	-7.8	-21.0	-1.28
GA	-8.2	-22.2	-1.30
GC	-10.6	-27.2	-2.17
GG	-8.0	-19.9	-1.84
GT	-8.4	-22.4	-1.44
TA	-7.2	-21.3	-0.58
TC	-8.2	-22.2	-1.30
TG	-8.5	-22.7	-1.45
TT	-7.9	-22.2	-1.00
Init./Term. A	2.3	4.1	1.03
Init./Term. C	0.1	-2.8	0.98
Init./Term. G	0.1	-2.8	0.98
Init./Term. T	2.3	4.1	1.03
Symmetry correction	0	-1.4	0.43

Table 2.1: Unified thermodynamic nearest-neighbor parameters according to SantaLucia [98]. The table contains the entropy change ΔS° and enthalpy change ΔH° at a salt concentration of 1 M NaCl. The Gibbs free energy ΔG°_{37} is given at 37°C.

melting temperature is

$$T_m = \frac{-100.2\ \frac{\text{kcal}}{\text{mol}}}{-282.1\ \frac{\text{cal}}{\text{mol}\cdot\text{K}} + \ln(\frac{0.01\frac{\mu\text{mol}}{\text{l}}}{2}) \cdot 1.987\frac{\text{cal}}{\text{K}\cdot\text{mol}}} - 273.15\text{K} = 39.9°\text{C} \tag{2.10}$$

2.1.5 Gibbs free energy ΔG°

The previously presented approaches resulted in the melting temperature T_m as main parameter for the prediction of the cross-hybridization of two oligonucleotides. A thermodynamic parameter that is used for the prediction is the Gibbs free Energy (ΔG). Its value describes how "sticky" a signature is at a certain temperature, i.e. the thermodynamic stability of a signature site and a corresponding primer or probe [98, 114].

Based on the nearest-neighbor model, the Gibbs free energy for an oligonucleotide duplex can be computed by summing up the respective values of all dinucleotides similar to Formula (2.4). ΔG values for a temperature of 37°C, published by SantaLucia [98], can be found in Table 2.1. Another more generic approach is to derive the Gibbs free energy ΔG° from the enthalpy and

entropy values with the Gibbs-Helmholtz equation

$$\Delta G^\circ_T = \Delta H^\circ - T\Delta S^\circ \tag{2.11}$$

where the temperature T is defined in Kelvin (K).

For our sample oligonucleotide from Figure 2.1, the entropy change from equation (2.9) and enthalpy change from equation (2.8) can be used to determine

$$\Delta G^\circ_{37^\circ\mathrm{C}} = -100.2\ \frac{\mathrm{kcal}}{\mathrm{mol}} - (273.15\ \mathrm{K} + 37\ \mathrm{K}) \cdot -282.1\ \frac{\mathrm{cal}}{\mathrm{mol}\cdot\mathrm{K}} = -12.71\ \frac{\mathrm{kcal}}{\mathrm{mol}} \tag{2.12}$$

in this case at 37°C.

The ΔG value is temperature dependent, as the Formula (2.11) indicates: a ΔG value of a primer at the minimum and maximum temperature of a PCR will most certainly differ. Besides that, the only relationship between the melting temperature T_m and the Gibbs free energy ΔG is their derivation of the enthalpy ΔH and entropy ΔS changes. A good explanation of the possible lack of a relation between the Gibbs free energy (ΔG) and the melting temperature (T_m) can be found in a technical report by J. Manthey [72]. He suggests using the ΔG value in combination with T_m:

> Unfortunately, there is no real relationship between ΔG and T_m where one can be used to identify the other. While it has been the standard to only use the T_m value when comparing thermodynamic equivalency between oligonucleotides, whether it is for PCR, or microarray designs, or maximizing SNP discrimination, it is suggested that the T_m value alone is also insufficient for thermodynamic comparison. The recommendation is that the two values, T_m and ΔG, together can provide much more qualitative thermodynamic understanding of a duplex or structure than either of the two values alone. Where as, the utilization of their principle components Enthalpy (ΔH) and Entropy (ΔS) can produce a quantitative thermodynamic understanding and is ideal for computational comparisons. [72, Abstract]

A factor that is important for the following Section is the Gibbs free energy ΔG value of e.g. a probe with its targets and possible non-targets. Mismatches, as described in the following Section 2.2, are used to discriminate targets from non-targets. As a consequence, the ΔG values of an oligonucleotide binding to a target sequence should differ from any non-target sequence. Although the melting temperatures of both may lie next to each other, their free energy can be used to differentiate good from bad oligonucleotide signature candidates.

2.2 Approximate Search

Searching signature sites using "paper and pencil" strategies might work for small datasets, but already a single gene will take some time depending on your skills. When confronted with longer sequences or more organisms, this method quickly fails. Search indices (Section 2.2.1) help to cope with that problem and even allow approximate string searching methods.

2.2.1 Search indices

Searching for a given sequence pattern, a signature, or querying for a non redundant set of signatures of a certain size in the complete dataset using linear methods (e.g. a sliding window over the complete dataset) is time consuming. Digital search trees [57, page 492–496], most notably *suffix tries* and *suffix trees*, are data structures that facilitate such searches. A nice overview of algorithms for the search tree construction, which are not described in detail in this work, can be found in Barsky et al. They also provide an introduction to search trees:

> A signature string $S = s_1 s_2 \ldots s_n$ is a sequence of n symbols from the alphabet $\sum$ and a terminal sentinel symbol $s_n = \$ \notin \sum$. The alphabet for RNA or DNA sequence data would be $\sum = \{A, C, G, T|U\}$. A suffix $S_i = s_i \ldots s_n, 0 \leq i \leq n$ of the signature string S is the substring that begins at position i.
>
> A *suffix trie* is a trie for all the suffixes of S. In a trie, each edge represents a character from the alphabet $\sum$. Sibling edges must represent distinct symbols; each trie node therefore has a maximum of $|\sum|$ children. Each suffix can be found in a trie by starting at the root node and following the edges that represent the continuous characters of the suffix string (Figure 2.3, left). In the worst case, the total number of nodes in the trie is quadratic in n. This happens when all paths in a trie are disjoint. In our case, the small number of characters in the alphabet ($|\sum| = 4$ for RNA or DNA sequences) should prevent the worst case.
>
> By collapsing paths that contain unary nodes into a single edge, the total number of edges and nodes can be reduced. The resulting structure is called a *suffix tree* (Figure 2.3, right). Instead of following single characters when searching a suffix, substrings that are associated with the edges are compared and concatenated. A generic suffix tree contains exactly n leaves with a degree of at least 2, and at most $n - 1$ inner nodes. In our case, the length of signature strings that are searched within a suffix tree are limited by their biochemical characteristics. Common implementations therefore reduce the search trees by pruning them at a certain depth, which was determined empirically (an example is shown in Figure 2.4).
> [13, notation and phrases adapted]

In our case two main advantages of search trees are important. First, they allow querying all unique substrings in linear time by traversing the tree. Furthermore, they allow pattern matching, i.e. searching a substring within sequence data, again in linear time depending on the substrings' length. Both features are needed when building the bipartite graph that matches sequences (organisms) with signatures. This is more deeply described in the two approaches implemented in this work: in Section 3.2.1 and in Section 4.4. The main disadvantage of search trees is their memory consumption which is significantly higher that that of the input data. A rule of thumb is a factor of 10–30 (Section 3.3.1).

Having long-term experience with the ARB Software Environment and being a co-developer has lead to the use the ARB PT-Server. It is presented in the following Section 2.2.2. But

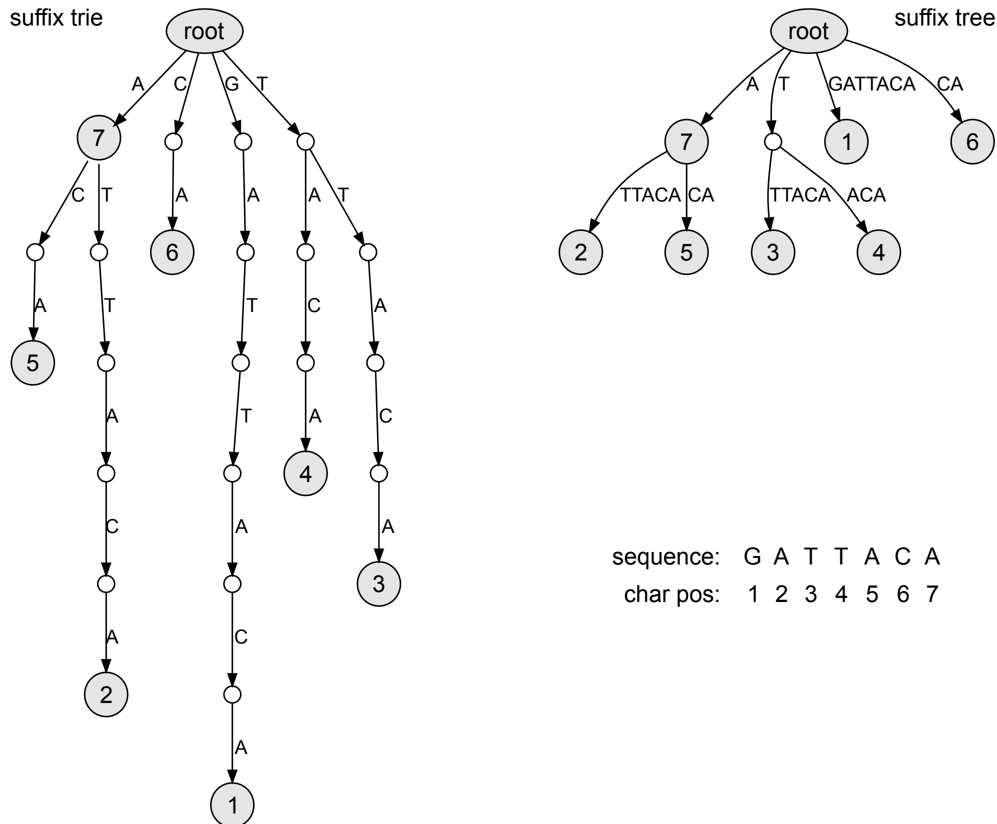


Figure 2.3: Digital search trees that facilitate the search for suffixes within sequence strings, here $S = \{\texttt{GATTACA\$}\}$. The terminal character $ is not shown in the trees. A suffix can be found through traversing the tree from the root to a terminal leave node with the starting position of the suffix in the tree. In case of a suffix trie (left), this is done by concatenating single characters from the edges. In case of a suffix tree (right), substrings at the edges are concatenated.

being a "basic tool" in computer science and of great importance in bioinformatics, many other approaches exist that provide search index features. One of the most common uses is processing "Sequencing by Oligonucleotide Ligation and Detection" (SOLiD) reads. In this case the index is used to efficiently align short sequence reads against large reference sequence datasets. The Burrows-Wheeler Alignment tool (BWA) [64] is an example for such a tool. Other approaches even rely on different types of index structures. Abouelhoda et al. has shown that the functionalities of a suffix tree can be replaced by a less memory consuming suffix array [1]. Spaced seeds may even further reduce the memory footprint and still allow relaxed string searches [21], but are tricky to use in generic approaches. So, although the underlying index structures may be generic in many cases, the implemented tools are designed and optimized for applications that do not have the same requirements as this work.

2.2.2 The ARB PT-Server and MiniPT

The *ARB Position Tree server* (PT-Server) is the central search index of ARB, a software environment for handling and analyzing rRNA data, managing protein sequences, contigs and genomes [117, 70]. It provides interfaces for signature extraction and exact and approximate string matching in nucleic acid sequence data.

The PT-Server is designed as a client-server application with a persistent search index. Requests and the corresponding answers are transmitted via TCP-IP, using an application-specific protocol. Before a PT-Server can be used for the first time, its index has to be created in memory and then be written to the hard disk. From there, the index can be loaded whenever necessary.

The ARB PT-Server implements a k-truncated suffix trie [101, similar approach] with a maximum height of $k = 20$ bases (Figure 2.4). Every edge of the suffix trie represents a base, so that the paths from the top to the bottom form all possible substrings in the dataset. Leaves contain a special array structure called *chain*. Chain entries contain sequence identifiers and absolute base positions, at which the characters from the path (from the root node to the leaf) can be found. A PT-Server specialty is the internal use of the dot character "." to indicate base positions with an unknown base. It can be compared to "N" in the IUPAC code. In the official IUPAC code a dot has a different meaning (Table 1.1). The PT-Server therefore internally works with five different branches $\{A, C, G, T|U, N\}$.

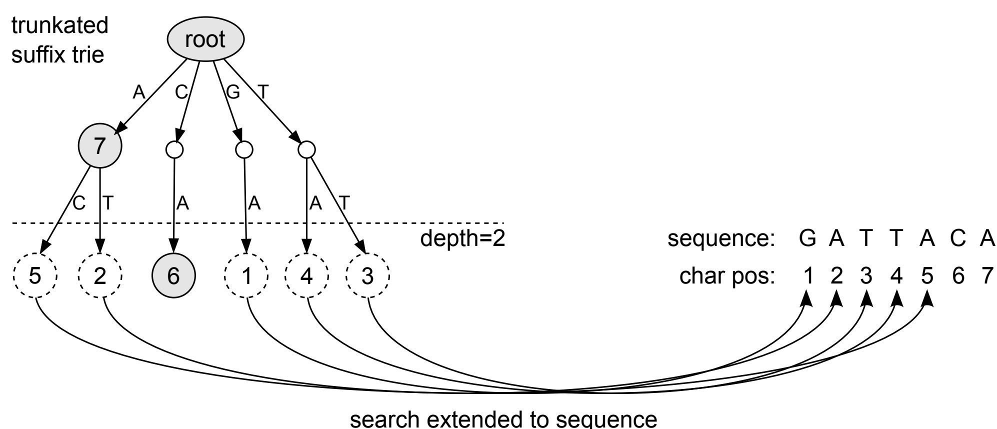


Figure 2.4: Example of a PT-Server's suffix trie structure, built from the sequence $S = \{\texttt{GATTACA\$}\}$ and truncated at a depth of 2. When the the depth limit is reached (i.e. the query string is longer than the trie's depth limit), linear search is performed in the sequence itself at the referenced character positions in the chain (leaf node).

The k-truncated suffix trie is a good compromise for applications where short oligonucleotides are being used. By default the ARB PT-Server depth is set to $k = 20$ and can (only) be changed in the source code. Empirical tests have shown that limiting the depth to 20 avoids an extreme degeneration of the suffix trie [107] and thereby allows a memory efficient fast search (Section 3.3.1). A disadvantage of this design is the way in which searches for signatures whose length exceeds k are handled. When reaching a chain (a leaf node), the search

is continued directly on sequences at the positions that are referenced in the chains. This can lead to significant performance degradation. Additionally, the sequences have to be available in memory to continue the search, if necessary. It is therefore recommended to keep the maximum search string length below the tree's depth.

An even greater disadvantage of the PT-Server is its huge dependency on main memory when building its search index. Initially, the PT-Server loads the complete source database into the main memory. Current (February 2012) sequence databases like Greengenes [29] or SILVA [89], which both contain small subunit rRNA gene sequences, already require about 4 GBytes of main memory and an end to growth is not in sight. This makes them unusable on 32 bit architectures or systems with only up to 4 GBytes of main memory.

The PT-Server estimates the required memory for building the search index by assuming a maximum of 55 bytes per base. If the memory requirements for the complete index structure exceeds the available main memory, the PT-Server splits the search index structure into 5^p partitions. Each partition represents a branch in the suffix trie. It can be processed independently. The value $1 \leq p \leq k$ is increased until the size of a partition fits into main memory.

The *MiniPT* search index is a lightweight derivate of the PT-Server, providing the same functionality. It was developed parallel to CaSSiS in order to avoid overhead in search queries. Unlike the PT-Server, which is a client-server application, MiniPT allows direct access to internal data structures. The PT-Server's communication protocol was removed. The MiniPT search index is not dependent on the ARB environment and thereby avoids a drawback of its predecessor: The PT-Server can only be constructed from an ARB database and relies on this database even after its construction. MiniPT's memory consumption is lower as it does not require the completely loaded sequence database during its build or during runtime. This is only possible due to the fact, that in this work the maximum search string length is already known when building the search index. In other cases, this would lead to a significant loss of information, as a continued search in the sequence itself would not be possible. Up to a hard defined limit, it can extend the trie truncation limit k to perform the search completely in the suffix trie. In the following chapters, the MiniPT is used as the central search index in all applications.

The PT-Server and its descendant MiniPT are not the only search indices that provide the two main features that are of importance for this thesis: providing a list of all unique substrings in linear time and allowing searching these substrings within their index structures. Another search index that was developed in the context of the ARB environment is *PTPan* (pronounced: Peter Pan). It also is a descendant of the PT-Server and was designed as an improved drop-in replacement. It therefore provides the same interfaces. Eissler et al. describes PTPan as…

> …a space-efficient indexing structure for approximate oligonucleotide string matching in nucleic acid sequence data. Based on suffix trees, it combines partitioning, truncation and a new suffix tree stream compression to deal with large amounts of aligned and unaligned data. PTPan operates efficiently in main memory and on secondary storage, balancing between memory consumption and runtime during construction and application. [33, Abstract]

PTPan provides approximate string matching based on Levenshtein “edit” distances. This allows the search for base insertions and deletions “indels” — a feature that is not provided by the PT-Server. Another advantage is its lower memory consumption during build time due to a more sophisticated partitioning strategy. PTPan, on the other hand, is significantly slower in processing queries [33].

This work and the development of PtPan were done at the same time. It was not until the beginning of 2012 that a working test version was available. It is currently (June 2012) being integrated into the testing branch of the ARB software package. For this reason, all tests in this work were performed with the at that time available ARB PT-Server and the nearly identical MiniPT.

2.2.3 Approximate string matching

Statistical analysis is necessary in order to make statements about the quality of an oligonucleotide probe. Two conditional probabilities are usually used to describe the quality: the *sensitivity* and the *specificity* of a probe [92]. Both probabilities have their roots in the statistical analysis of medical screening [2].

It is necessary to distinguish between the calculated and the real “quality” of a probe. The *in silico* quality of an oligonucleotide probe is determined by comparing its calculated target scores with the target group it was designed for (and with the non-targets). This is usually done by the software that computes and proposes the oligonucleotides as probe candidates. The real quality of an oligonucleotide probe can only be determined in a wet lab under the conditions the probe was designed for. The probe should reliably hybridize against the targets and not hybridize with possibly present non-targets. The results of the experiments in the wet lab are then compared with the results from a *gold standard* test, the best known reference that is comparable. In both cases, wet lab and *in silico* evaluation, the results are then arranged according to Table 2.2.

Outcome (wet lab / evaluation)		**True state (gold standard / target group)**		
		$\oplus$	$\ominus$	Total
Hybridized	$\oplus$	TP	FP	$TP + FP$
Not hybridized	$\ominus$	FN	TN	$FN + TN$
Total		$TP + FN$	$FP + TN$	N

Table 2.2: To simplify the statistical analysis, it is assumed that an oligonucleotide either hybridizes with a target (i.e. positive) or it does not (i.e. negative). As already shown in Section 2.1, this assumption is only as good as the underlying model. The results of N tests are ordered into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

The sensitivity (true positive rate) is the proportion of true positives (TP) that are correctly

identified to the total number of positives [14]:

$$sensitivity = P(positive|TP) = \frac{TP}{TP + FN} \tag{2.13}$$

In case of oligonucleotide probes, it shows how good a probe is in detecting a defined target group. A high sensitivity indicates a high chance that a positive result (i.e. the probe hybridizes) was caused by a target that lies within this group. This can also be used to rule out the presence of a defined group: a probe with a high sensitivity and a negative result (i.e. no hybridization) suggests that no target from the group is in the sample.

The term sensitivity is also used for the lowest concentration of a probe and their targets to obtain reproducible results: The higher the sensitivity, the lower the concentration. Thermodynamic effects have a strong influence on this sensitivity (Section 2.1). Although both cases in which the term is used are associated, the first definition will be used in the following text.

The specificity (true negative rate) is the proportion of true negatives that are correctly identified to the total number of negatives:

$$specificity = P(negative|TN) = \frac{TN}{TN + FP} \tag{2.14}$$

Ideally, a probe should have both, a high sensitivity and high specificity. In most cases, there are trade-offs between the two factors.

Chapter 3

The CaSSiS-BGRT Approach

PCR, hybridization, DNA sequencing and other important methods in molecular diagnostics rely on both sequence-specific and sequence-group-specific oligonucleotide primers and probes. Their design depends on the identification of oligonucleotide signatures in whole genome or marker gene sequences. Although genome and gene databases are generally available and regularly updated, collections of valuable signatures are rare. Even for single requests, the search for signatures becomes computationally expensive when working with large collections of target (and non-target) sequences. Moreover, with growing dataset sizes, the chance of finding exact group-matching signatures decreases, necessitating the application of relaxed search methods. The resultant substantial increase in complexity is exacerbated by the dearth of algorithms able to solve these problems efficiently.

We have developed CaSSiS-BGRT, a fast and scalable method for computing comprehensive collections of sequence- and sequence-group-specific oligonucleotide signatures from large sets of hierarchically-clustered nucleic acid sequence data. Based on the ARB Positional Tree (PT-)Server and a newly-developed BGRT data structure, CaSSiS-BGRT not only determines sequence-specific signatures and perfect group-covering signatures for every node within the cluster (i.e. target groups), but also signatures with maximal group coverage (sensitivity) within a user-defined range of non-target hits (specificity) for groups lacking a perfect common signature. An upper limit of tolerated mismatches within the target group, as well as the minimum number of mismatches with non-target sequences, can be predefined. Test runs with one of the largest phylogenetic gene sequence datasets available indicate good runtime and memory performance, and in silico spot tests have shown the usefulness of the resulting signature sequences as blueprints for group-specific oligonucleotide probes.

This Chapter is an extended version of an article [12] published in Bioinformatics, that was joint work with Christian Grothoff and Harald Meier. Section 3.4 is based on the final report for a bachelor project supervised by me: Sebastian Wiesner implemented an OpenMP-parallelized version of the CaSSiS-BGRT traversal algorithm.

3.1 Introduction

Oligonucleotide primers and probes are the key diagnostic agents in technologies that allow the rapid, sensitive and specific detection of nucleic acid signatures in samples. In fields such as medicine, food research and environmental microbiology, they are used to identify organisms with specific properties [109, 91, 23]. For applications such as microbial population analysis and molecular screening for microbial pathogens or indicators, there is the additional challenge of detecting and distinguishing organism *groups* (rather than single organisms), which can be identified in a number of ways (phylogenetically, taxonomically, etc). Designing group-specific primers and probes, however, is a significant challenge, as these primers and probes should reliably hybridize with the target sequences (i.e. have a high coverage) within the group but not interact with any non-target sequence that might be in the same sample [76, 66].

In many studies, conserved housekeeping genes or gene products such as ribosomal RNA (rRNA) are targeted [3, 102]. Considerable collections of probe or signature sequences for suitable target genes are rare. One exception is *probeBase*, which provides sequences and annotations of already published rRNA-targeted oligonucleotide probes [68]; however, many of these probes were designed in the past on the basis of small sequence data collections. Some of them would have to be reevaluated, optimized or even newly designed to take the relevant rRNA gene sequence data in comprehensive highly curated databases into account [3]. The SILVA SSU-rRNA reference database [89] contains such a curated collection of annotated nucleic acid sequence data, which is deeply hierarchically-clustered by phylogenetic relationship.

This work details a new computational method for the comprehensive search for sequence and group-specific oligonucleotide signatures (hereafter simply referred to as signatures). Our method uses the SILVA reference database to create a signature collection that could be used to provide the sequence information of binding sites and design templates for valuable phylogenetic primers and probes. Signature collections could be published alongside the generally available and regularly updated sequence databases, and in combination, both could facilitate the design of oligonucleotide primers and probes.

There are already several published approaches for searching signature or probe sequences. *PROBESEL* [53], *OligoArray* [94], *OligoWiz* [116], *YODA* [80] and *CMD/PSID* [62] all specialize in finding unique signatures for single sequences but cannot search for signatures that are specific to groups. Others, such as *PRIMROSE* [8] and ARB-ProbeDesign [70], allow searches for group-specific signatures; however, they are limited to one selected target or target group per run. Performing individual runs for all sequences or sequence groups of a large hierarchically-clustered dataset is impossible due to memory and runtime limitations. *HPD* [22] is more comprehensive and uses a bottom-up approach on a hierarchical cluster to generate both sequence- and group-specific signatures from one dataset in a single run. Unfortunately, HPD's search capability exhibits memory and runtime problems when applied to large-scale datasets for several thousand sequences and clusters [34].

More recently published tools, *ProDesign* [34] and *Insignia* [87], are capable of comprehensively identifying signatures for large collections of clustered DNA sequences and even whole

genomes. ProDesign uses a sophisticated spaced seed hashing approach to speed up its word indexing process; however, depending on the seed and the clustering used, ProDesign may lead to suboptimal results. Furthermore, for the identified group-specific signatures, ProDesign does not provide detailed information regarding coverage and specificity beyond hard-coded search constraints (more than 95% ingroup matches, less than 5% outgroup matches). Insignia relies on a large set of preprocessed genome sequences to quickly determine only those signatures that match the *entire* target sequence group [86]. As a result, potentially valuable signatures matching a subgroup may be missed. Both ProDesign and Insignia are primarily designed to handle flat clusterings and are unsuited to comprehensively process predefined deep hierarchies.

This chapter describes the specifics and implementation of *Comprehensive and Sensitive Signature Search* (CaSSiS-BGRT, Figure 3.1), a new algorithm addressing some of the limitations mentioned above. Specifically, CaSSiS-BGRT is capable of computing comprehensive sets of sequence- and group-specific signatures, even for large collections of deeply hierarchically-clustered sequences under both strict and relaxed search conditions. CaSSiS-BGRT sorts signature sequence results by degree of specificity, and all signatures guarantee the predefined Hamming distance to non-target sequences. For signatures which cover sequence groups incompletely, statistical information on the sensitivity is provided.

3.2 Material and Methods

CaSSiS-BGRT consists of three computational stages. The first stage (Section 3.2.1) is the extraction of signature candidates (candidates, because their specificity has to be further evaluated) from sequence data and their specificity evaluation; the result of the first stage is a bipartite graph relating sequences to signature candidates. The second stage (Section 3.2.2) performs hierarchical sorting of the signature candidates, resulting in a *Bipartite Graph Representation Tree (BGRT)*. The last stage (Section 3.2.3) extracts valuable signatures from the BGRT for each node in a hierarchical cluster — in our case, a phylogenetic tree. The result is a comprehensive set of signature candidates for all nodes in a phylogenetic tree (*phy*-nodes).

We will illustrate the algorithm using a running example (Figures 3.2, 3.4 and 3.5, and Table 3.1). Arabic numerals are used to refer to both the sequence entries in the analyzed data collection and the leaves representing them in the phylogenetic tree. Signatures are labeled with capital letters. Phylogenetic groups of sequences and their respective inner tree phy-nodes are specified using Roman numerals to indicate their depth in the tree, with lowercase letters used to distinguish between groups at the same depth. We will refer to matches within a target group as *ingroup hits* and non-target matches as *outgroup hits*. Additionally, because each sequence entry in our test databases represents an organism, *sequence* and *organism* are used synonymously in this manuscript.

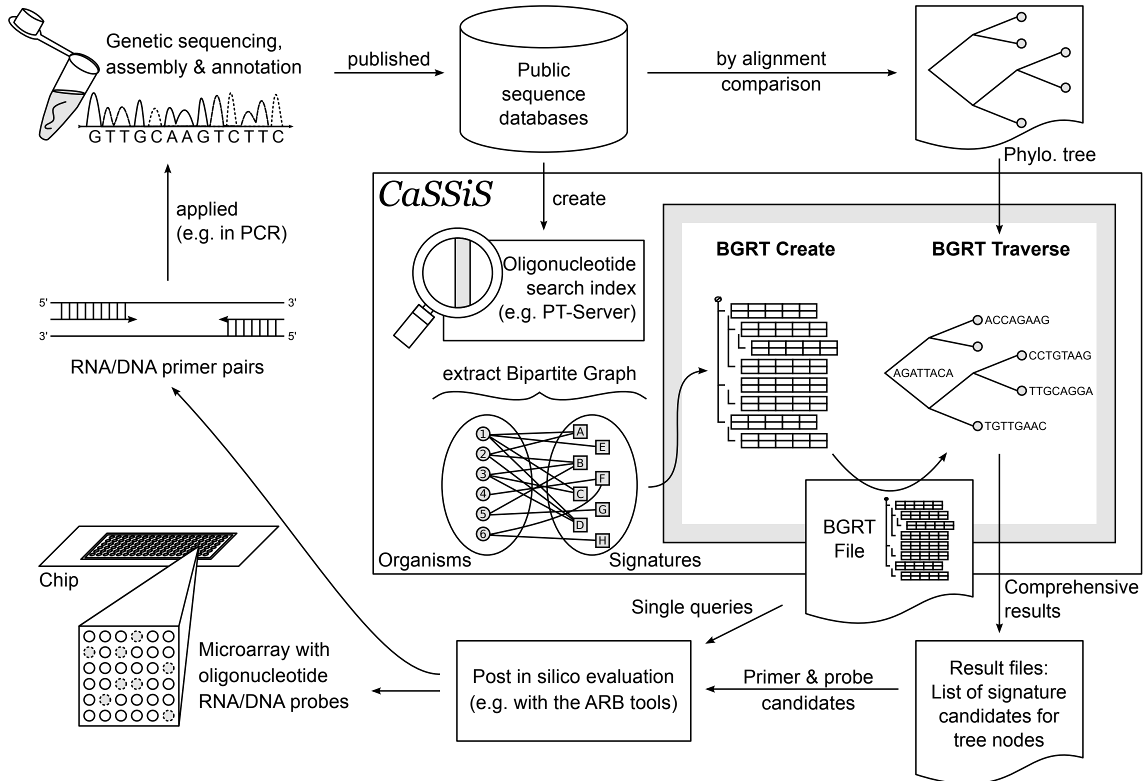


Figure 3.1: Schematic of the primer and probe design pipeline in which CaSSiS is embedded. The input data for the CaSSiS-BGRT algorithm comes from public sequence and phylogenetic tree databases. It can either be used to create comprehensive sets of signatures, which can be used as a templates for new primers and probes. Or individual queries could be used to target freely-defined groups.

3.2.1 Extraction and evaluation of signature candidates

The first stage of our algorithm generates a bipartite graph where signature candidates and sequences are unique and signature matches within sequences are represented as edges between the two sets (Figure 3.2). To build the bipartite graph in reasonable time (i.e. extract all signature candidates and their matches), the ARB Position Tree (PT-)Server [70] was used. The PT-Server supports index-based exact and inexact searches in nucleic acid sequence data using a truncated suffix tree. It returns all matches of a query sequence that meet predefined search constraints such as length, allowed Hamming distance (number of base mismatches), weighted mismatches and others.

To allow signature candidates with up to m_1 mismatches within the target group and a Hamming distance of at least $m_2 > m_1$ to the next non-target match, an upper limit of $m_2 - 1$ mismatches is used when fetching a list of matching organisms for any one signature candidate (Figure 3.3). Sequences with Hamming distance less than m_1 (I and O) are used to generate the bipartite graph. Sequences with Hamming distances between m_1 and m_2 mismatches are only counted; those totals are then added to the number of outgroup hits for the candidate that

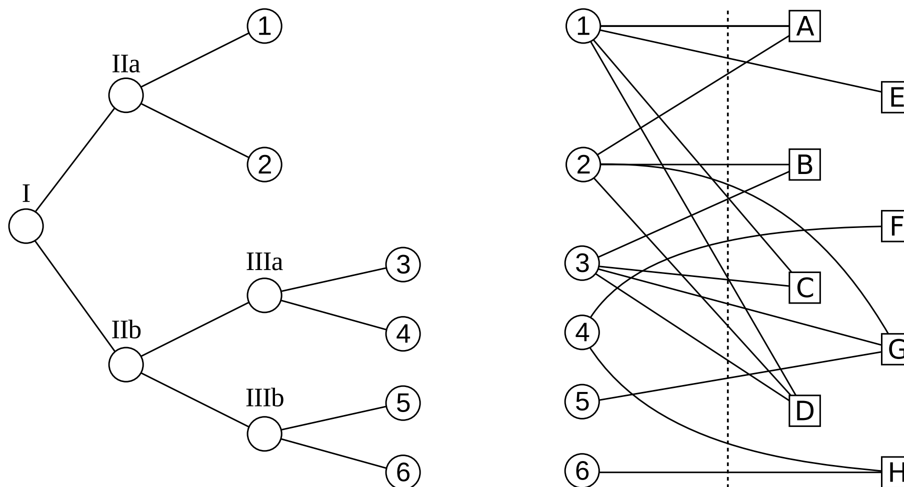


Figure 3.2: Illustration of the input data for the second and third stage of our algorithm for the running example. **Left:** the phylogenetic tree with group phy-nodes (Latin numerals) and organisms (leaves: Arabic numerals). **Right:** the bipartite graph, showing which organisms (Arabic numerals) are matched by which signature candidates (capital letters).

is computed from the bipartite graph in the next stage.

Note that currently only the Hamming distance between a signature candidate and its matched targets is used for evaluation, not the actual position and type of mismatch on the target sequences.

To restrict cross-hybridization to non-targets, CaSSiS-BGRT can be configured to check candidates for matches within the antisense strands and exclude them, at the expense of possibly producing suboptimal results. Additionally, CaSSiS-BGRT can discriminate against signatures with abnormal melting temperatures and high GC-content. But such filters must be handled with care (Section 3.3.4).

The basic melting temperature T_{m_basic} for oligonucleotides up to a length $l \leq 13$ base pairs is calculated according to the equation (2.2). For longer oligonucleotides with $l > 13$ the equation (2.3) is used. T_{m_basic} is rather inaccurate when evaluating long oligonucleotide probes, e.g. $l \geq 50$ would lead to temperatures of $100°\text{C} \geq T_{m_basic} \geq 200°\text{C}$). For short oligonucleotides (common primers and probes have lengths between $15 - 25$ nucleotides) the temperature is a good estimation.

Signature filtering can be done based on two different melting temperature calculations: the basic melting temperature T_{m_basic} (see Section 2.1.3) and a melting temperature T_{m_37} using the nearest-neighbor (base stacking) method at 37°C (see Section 2.1.4).

The implemented nearest-neighbor model (Section 2.1.4) is working with approximate parameters (especially the salt correction; [111])based on SantaLucia [98]. The entropy and enthalpy parameters in his publication were obtained around 37°C and for perfect matching duplexes. With increasing deviation from this temperature, the parameters become less accurate. For duplexes with one or more mismatches, they have to be adapted, resulting in an error of $\leq 2°\text{C}$ [111]. Additionally, the parameters were determined by measuring the thermodynamic

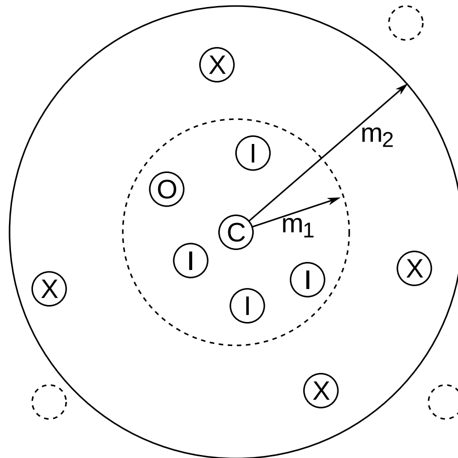


Figure 3.3: Sequences with Hamming distance less than m_2 to the candidate C are fetched. Sequences with Hamming distance up to m_1 can be ingroup (I) or outgroup (O) hits. Sequences with Hamming distances between m_1 and m_2 (X) are counted, but not added to the bipartite graph.

properties of short ($\sim$ 20 bases) oligonucleotides in solution. The parameters for probes bound to a RNA/DNA chip surfaces may therefore differ.

3.2.2 Organizing signature candidates by specificity

This stage arranges the signature candidates according to their specificities, resulting in the BGRT. Each node in the BGRT (*bgrt*-node) contains a list of organisms and a list of signatures. A signature in a descendant matches all of the organisms on the path from the root bgrt-node to the bgrt-node where the signature is located. Each signature is located at exactly one position in the BGRT, and organisms can be listed multiple times. Figure 3.4 illustrates the BGRT construction algorithm.

Note that when `2,3:B` (i.e. signature `B` matches organism `2` and `3`) is added in step 2, the construction procedure chooses not to merge with `1,2:A` because the organism with the lowest numerical ID (here `1`) is not matched by both signatures. In contrast, `1,3:C` is merged with `1,2:A` because here the organisms with the lowest numerical ID (again `1`) is matched by both signatures.

This construction of the BGRT ensures a unique construction in the case where sets partially overlap. The example illustrates the issue in Figure 3.4 in step 3. Here, there are theoretically two possible ways for inserting `1,3:C`. First, as shown in Figure 3.4, a bgrt-node with no signature for organism `1` with two sub-bgrt-nodes `2:A` and `3:C` could be created (splitting `1,2:A`). Alternatively, a bgrt-node with no signature for organism `3` with two sub-bgrt-nodes `2:B` and `1:C` could be created (splitting `2,3:B`).

As described, our algorithm always splits the bgrt-node where the numerically smallest organism ID overlaps. Consequently, the BGRT will contain deeper subtrees for organisms with low IDs. As a result, assigning organisms deep in the phylogenetic tree smaller numeric

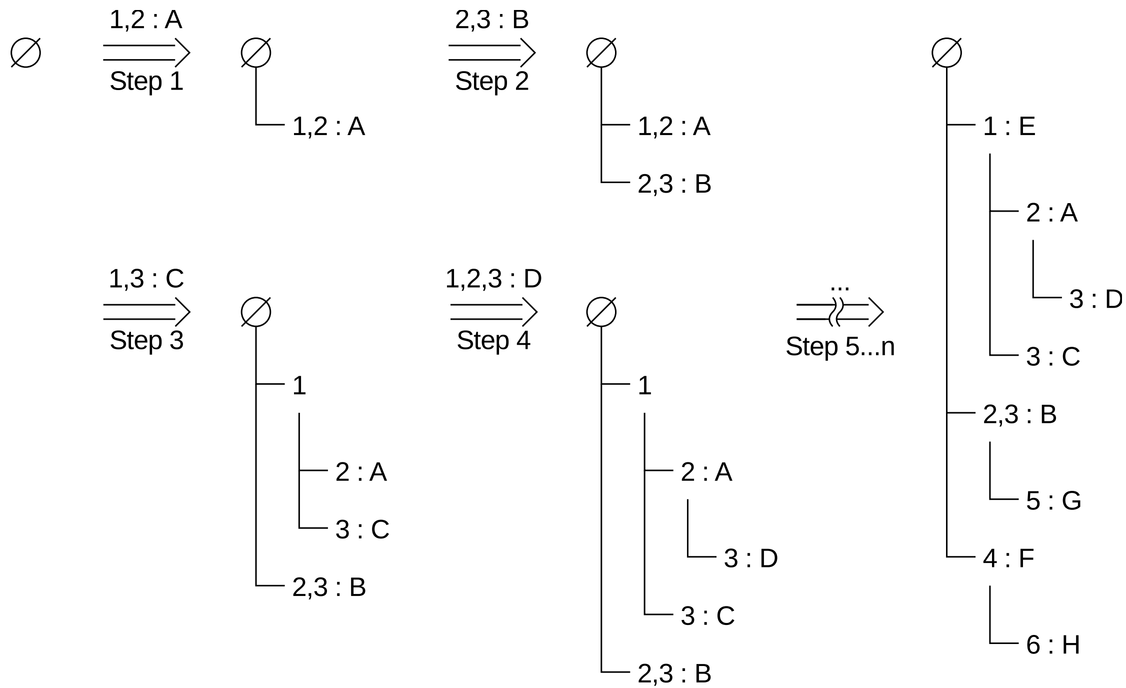


Figure 3.4: The signatures in Figure 3.2 are added to the BGRT based on the organisms they match. In each step, the algorithm inserts a signature and a numerically sorted set of organisms that match it. For insertion, the algorithm traverses the BGRT, looking for overlaps between the existing sets of organisms and the set for the signature being inserted. If the first elements of the sorted sets intersect with the current bgrt-node, the algorithm generally splits the current bgrt-node, creating new child bgrt-nodes to represent set differences. If the first elements are different, it moves on to the next sibling. If there is no other sibling, a new sibling is created.

values is likely to improve performance in stage 3. Splitting bgrt-nodes differently would have no impact on the correctness of the algorithm.

3.2.3 Determination of valuable signatures

The last stage performs a depth-first traversal of the phy-nodes of the phylogenetic tree. The processing of an organism or a group of organisms during this traversal is called a phase and we label the phase with the respective organism number (for example, “Phase 4”) or group name (for example, “Phase IIb”). In each phase, the algorithm performs a depth-first traversal of the BGRT in order to find signatures with the maximum number of ingroup hits for each entry within the range of $[0, k]$ outgroup hits. Since the organism sets are sorted numerically, the algorithm determines the number of ingroup and outgroup hits at each bgrt-node in linear time. If a signature with higher coverage (higher number of ingroup hits) for one of the entries within the same outgroup hits range for the current phase is found, the algorithm updates the result table accordingly.

The performance of the algorithm can be significantly improved by bounding the BGRT traversal. If the number of outgroup hits at a given bgrt-node is already larger than k, the

algorithm does not need to traverse the respective part of the BGRT for any of the descendants of the current phy-node: the number of outgroup hits in the BGRT subtree under the bgrt-node is guaranteed to be at least as large. Furthermore, since the algorithm traverses the phylogenetic tree in a depth-first manner, we can bound the traversal of phylogenetic subtrees by considering the best results found in the parent phase: organism groups that are parents in the phylogenetic tree contain strictly more organisms than all of their descendant nodes; hence, when compared to the best result achieved for the parent phase, the number of hits in the descendants can only be fewer (or equal) and the number of outgroup hits can only be larger (or equal).

Our algorithm tracks the best results achieved for the parent phase in an additional array associated with each bgrt-node (Figure 3.5). When traversing a bgrt-node in phase at depth d, the algorithm consults the phase result table from the parent phase with depth $d-1$ and only traverses the bgrt-node if the current best solution (for a given number of outgroup hits) is worse than the best solution of the respective BGRT subtree for the parent phase. If the algorithm decides to traverse the BGRT subtree, it stores the best solution found in the phase result table for all ancestors.

Bounding the BGRT traversal in this manner is particularly effective if the algorithm has already found a reasonably good solution for the current phase. Our simple approach for finding a good starting solution before traversing the tree is to use the best signature found in the parent phase.

Note that on average, the number of organisms the BGRT traversal algorithm will try to match in each phase is $O(1)$. Thus, the worst-case complexity of BGRT traversal is $O(nm)$, where n is the number of phy-nodes and m is the number of bgrt-nodes. However, performance is much better in practice due to the bounding method, especially given a reasonably low limit for the outgroup hit range. Storing the best solutions for the parent phases increases memory consumption from $O(m)$ to $O(md)$ where d is the depth of the phylogenetic tree. Thus, using this bounding method is a time-memory tradeoff.

outgroup	**Phase**				
#	I	IIa	1	2	IIb
0	3 (D,G)	2 (A)	1 (E)	–	2 (H)
1	–	2 (D)	1 (A,C)	1 (A,B)	2 (G)
2	–	1 (G)	1 (D)	1 (D,G)	1 (D)

outgroup	**Phase**					
#	IIIa	3	4	IIIb	5	6
0	1 (F)	–	1 (F)	–	–	–
1	1 (B,C,H)	1 (B,C)	1 (H)	1 (H)	–	1 (H)
2	1 (D,G)	1 (D)	–	1 (G)	1 (G)	–

Table 3.1: Results of the signature search for the running example. Rows are ordered by the number of outgroup hits, columns by the phase. The fields contain the number of ingroup hits and the corresponding signatures (capital letters). "–" indicates "no signature found".

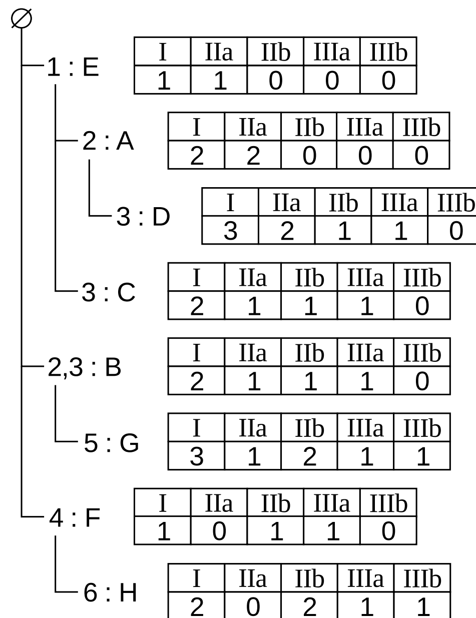


Figure 3.5: Each bgrt-node stores a table with the number of ingroup hits for the respective group phy-nodes. This table is used to bound the BGRT traversal based on the best known result for the current phase. Note that the in-memory size of the table is determined by the depth of the phylogenetic tree and not (as illustrated in print) by the total number of groups. For this example, the implementation would use the same memory cells for processing of groups IIa and IIb as well as for groups IIIa and IIIb.

The final result of this stage is a table listing for each phase and for $h \in [0, k]$ outgroup hits the signature that achieves the maximum number of ingroup hits, as shown for the running example in Table 3.1.

3.2.4 Testing conditions

We used `SSURef_102_SILVA_12_02_10_opt` (hereafter referred to as SSURef_102), a comprehensive database of small subunit rRNA sequences available in the ARB format [89], to generate our test datasets. It is the largest collection of annotated aligned SSU rRNA sequences of almost full length (more than 900 nucleotides). Furthermore, it includes a large phylogenetic tree referencing all the sequences in the dataset as Operational Taxonomy Units (OTUs) at the leaves. Inner phy-nodes correspond to groups of phylogenetically related sequences. More about the database, including information on sequence content and quality as well as sequence statistics can be found on the SILVA website (`http://www.arb-silva.de/`).

For the performance evaluation, we produced different size subsets of the SSURef_102 by applying a random sequence selection algorithm. In the phylogenetic trees, for each subset we only kept those leaves referenced by remaining sequences. The test sets range in size from 100 to 460,783 sequences, the largest test set being the complete SSURef_102 (Table 3.2).

For all test datasets, signatures with a length of 18 bases were computed using a tolerance setting of at most 10 outgroup hits. Furthermore, we used the full SSURef_102 and searched

Sequences	Nucleotides	Sequences	Nucleotides
100	152,466	10,000	15,404,216
200	306,961	20,000	30,773,749
500	762,960	50,000	76,870,889
1,000	1,540,372	100,000	153,768,356
2,000	3,081,068	200,000	289,312,540
5,000	7,693,065	300,000	433,858,110
		(*) 460,783	666,311,940

Table 3.2: Test datasets. Numbers of SSU rRNA sequences (representing organisms) and overall nucleotides within SSURef_102 (*) and subsets of it.

for 18-mer signatures for different settings ranging from 0 up to 1024 outgroup hits. We used a Hamming distance of 1 (i.e. at least one base mismatch) between target and non-target matches. Within target groups no mismatches were allowed. No melting temperature or G+C content filtering was applied.

The evaluation of the signatures, computed by CaSSiS-BGRT and from other sources, was done with the ARB ProbeMatch tool [70]. It is able to visualize the matches within a phylogenetic tree and shows the exact location of mismatches on the sequence data compared to the signature strings.

All tests were done on a workstation with 24 GB of RAM and an Intel Core i7 CPU @ 2.67 GHz (4 cores, 8 threads with hyperthreading support turned on).

3.3 Results

We present the results of runtime performance and memory consumption analyses with respect to each of the stages of CaSSiS-BGRT. Furthermore, we show data reflecting quantitative and qualitative properties of the comprehensive 18-mer signature collection CaSSiS-BGRT calculated from the full SSURef_102.

3.3.1 Performance of search index and signature candidate evaluation

The time for building the search index, as well as the overall memory consumption of the PT-Server, increased linearly with the number of nucleotide bases in the underlying sequence database. Building the full SSURef_102 ($\sim$ 660 megabases) took about 19 minutes (Figure 3.6) and consumed 26 bytes per base at its peak (about 17 GB total); the running PT-Server required just 11 bytes per nucleotide base (about 7 GB total; Figure 3.6).

Increasing the minimum Hamming distance m_2 increases the upper mismatch limit when querying the PT-Server (see Section 3.2.1). Large values for this distance parameter result in a significant increase in the overall number of edges between signature candidates and the sequences of a dataset: processing our $10,000$ sequence dataset with 2.23 million signature candidates and with a distance $m_2 = 1$ resulted in 15.2 million edges and took about 32 seconds;

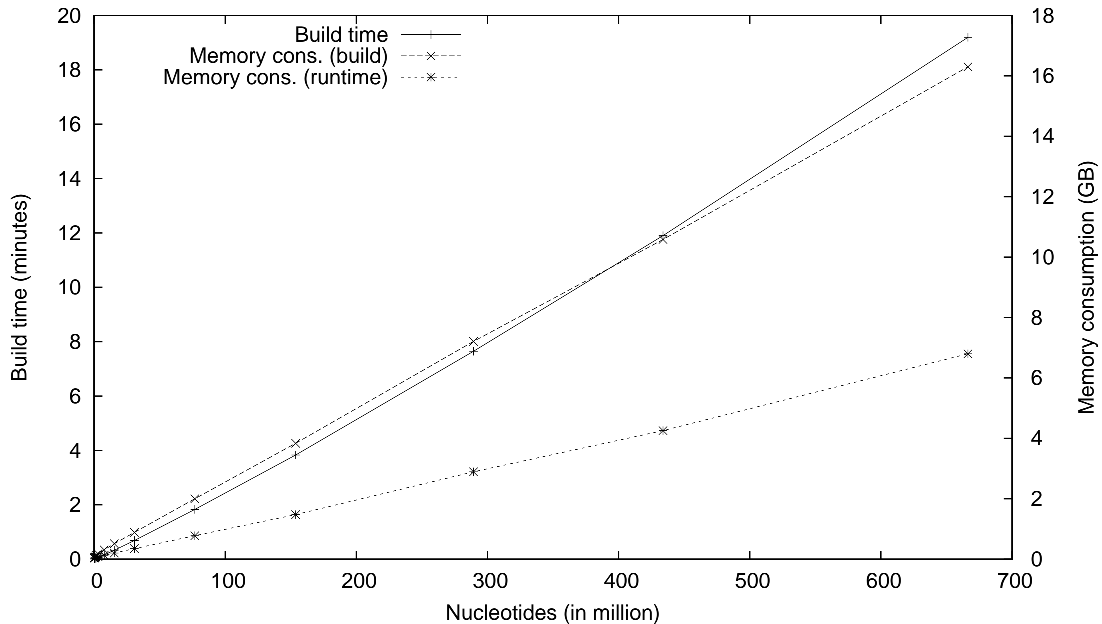


Figure 3.6: Build time and peak memory consumption of the PT-Server during the build process and at runtime in relation to the number of nucleotides.

querying the same dataset with a distance of $m_2 = 5$ led to 2.33 billion edges and increased the runtime of stage 1 to over 4 hours (detailed results given as supplementary material). While m_2 determines the runtime of the PT-Server, m_1 has an impact on the size and creation time of the BGRT as sequences between m_1 and m_2 are only counted (see Section 3.2.1).

3.3.2 Performance of the BGRT-generation

The creation of the BGRT structure based on the collected specificity information did not significantly impact the overall runtime. Its build time grew linearly in relation to the number of sequences; for the SSURef_102 database, BGRT construction took about 18 minutes. Also the number of bgrt-nodes exhibited linear growth (Figure 3.7). The BGRT computed from the SSURef_102 database resulted in 4.7 million bgrt-nodes with a tree depth of 103.

3.3.3 Performance of the BGRT-traversal

An individual search for signatures for an organism or an organism group in the BGRT is very fast. Even in the large BGRT built from the full SSURef_102, this took less than a second. But performing this search for all 921, 565 phy-nodes of SSURef_102 would have taken more than 10 days on our test system.

Bounding the BGRT traversal using the best-known current signature and bounds from the parent phase (Section 3.2.3) was shown to be an effective method for reducing the search time. We measured the proportion of the total number of BGRT branches which could be skipped during the search for signatures for each test dataset. The results (Figure 3.8) show that we

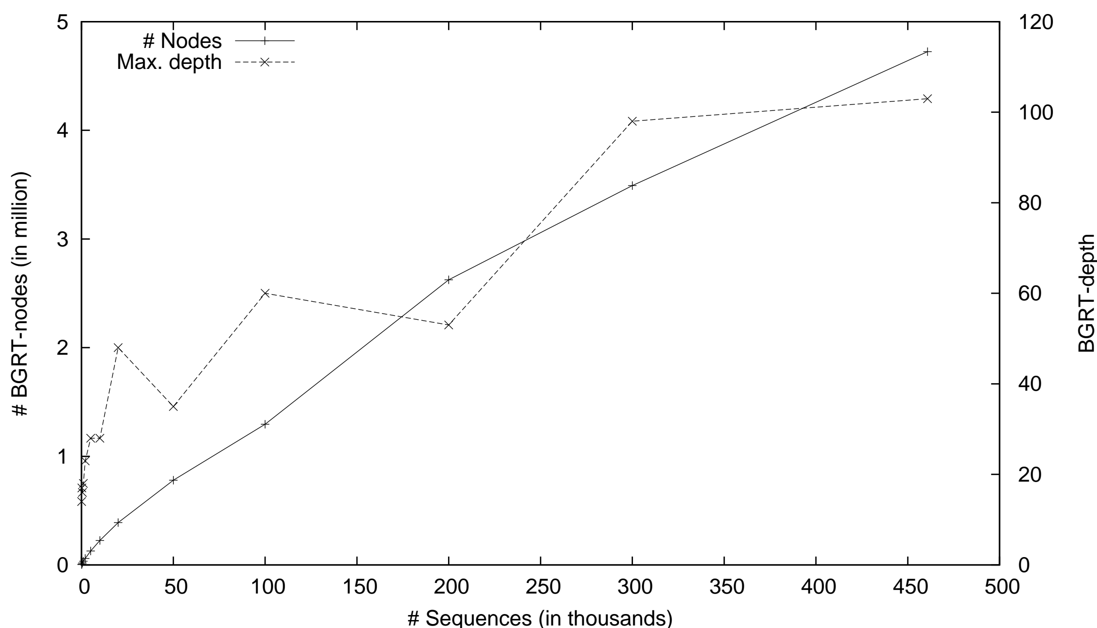


Figure 3.7: BGRT statistics for test datasets, plotting the size of the datasets against the number of bgrt-nodes in the resulting BGRT (solid line) and against the depth of the BGRT (dotted line).

were able to reduce the overall search space for datasets with more than 1000 sequence entries by about 90%. Searching for signatures for all phy-nodes in SSURef_102 with bounding took 132 hours (Figure 3.9).

The memory required to store the number of ingroup hits for each phase in the BGRT (Figure 3.5) resulted in the expected increase in memory consumption (Figure 3.9). For SSURef_102, memory consumption increased from 4.1 to 11 GB. These values include optimizations such as reducing the ingroup array size by reusing memory for different phases at the same depth.

3.3.4 Signature search space reduction

The number of signatures of a defined length found in a dataset is far lower than the theoretical possible number of signatures of that length (see Table 3.3).

Only querying the signatures found in the respective dataset is not a problem as long as strict search conditions are used. When using relaxed search conditions. i.e. allowing a mismatch distance of 1 or more within the target group, a case can be constructed where a signature candidate is valid under the defined constraints but does not appear in the sequential list of signatures that was described above. An example is given in Figure 3.10.

A comparison of the results for a test database with 64,000 sequences has shown no significant difference in the results between a run with extracted signatures (i.e. signatures, that are present in the sequences) and all generated signatures. But querying all signatures, especially under relaxed search conditions, will lead to an extreme increase in the overall runtime (Table 3.4).

The comparison was done with a signature length of 18 bases, 1 allowed mismatch within

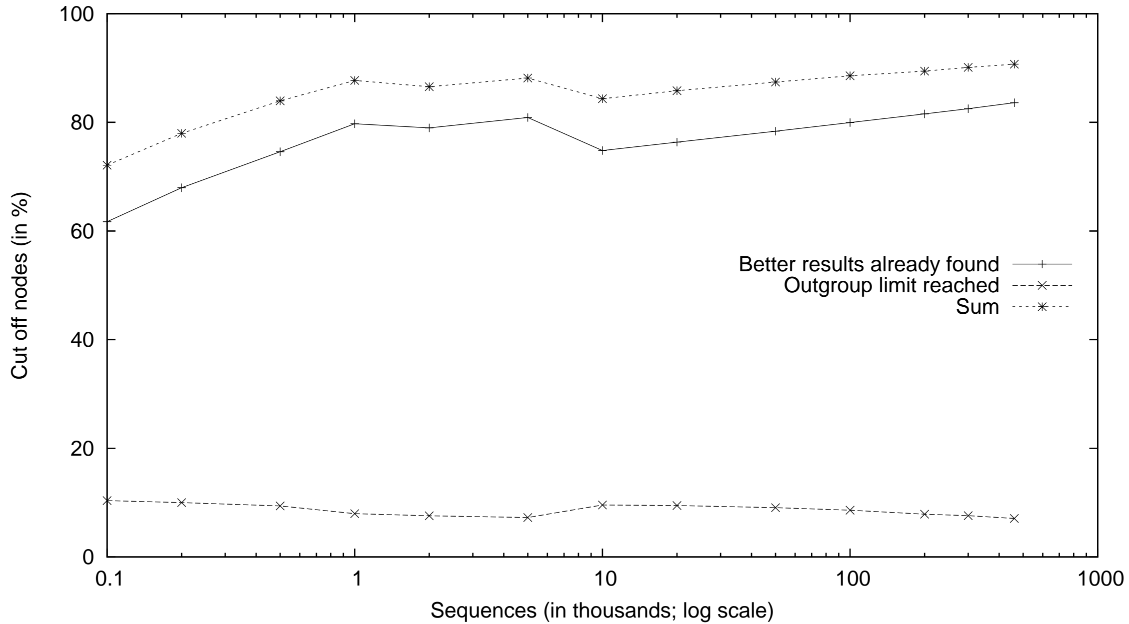


Figure 3.8: Efficiency of our bounding methods during comprehensive searches, depending on the dataset sizes (number of sequences). With growing dataset sizes, the chance of early branch cut-off in the BGRT declined. On the other hand, more cut-offs happened due to previously-found better results.

the target group and a mismatch distance of 1 to non-targets. The range from 0–10 allowed outgroup matches was processed. Both runs led to the same coverage “scores”, i.e. the same number of matched sequences for every phylogenetic tree node. They differed only in the number of signatures found — for some phylogenetic nodes more signatures were found when processing all signature candidates.

It was not tested, if the 64,000 test sequences have a certain distribution of bases that might lead to an equal score for both runs. This seems unlikely. A possible explanation can be found when directly comparing the resulting signatures for all nodes. The more organisms and thereby related signatures are covered by a group node, the higher is the chance that one of these sequences contains the signature without mismatches. The signature is thereby found in both results. All leaves had a maximum score of 1, i.e. were at least matched by one signature. Here, the chance is highest to find additional signatures when querying all signatures. In the test results, these nodes were the ones with the highest growth in the number of found signatures.

An optional reduction of the search space provided by CaSSiS is thermodynamic filtering. Signatures can be dropped if they do not lie within defined GC-content or melting temperature ranges. Although these filters appear very helpful at first sight, they should be used with caution.

The GC-content is the percentage of bases with strong hydrogen bonds (Guanine and Cytosine) present in a signature sequence (Section 2.1.2). Neither the actual base type (besides being either `G,C` or `A,T/U`) nor its position is relevant for computing this percentage value. The

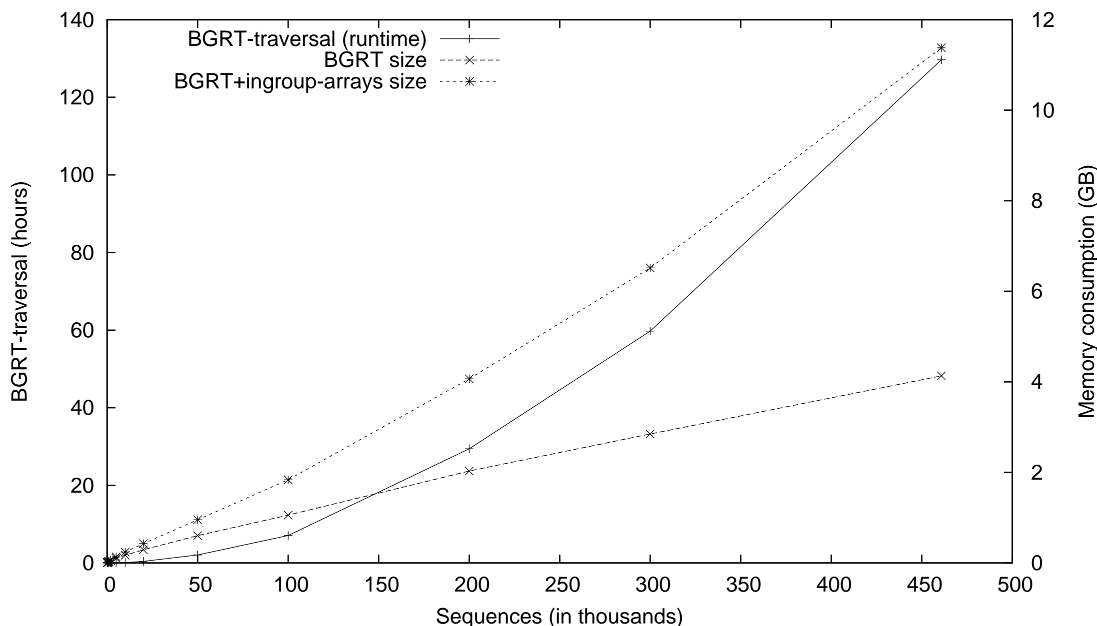


Figure 3.9: The BGRT shows linear growth in memory consumption in relation to the number of sequences. Its size, including the ingroup array (see Figure 3.5), was measured after a complete computation for all phy-nodes.

number of possible GC-content values is therefore directly influenced by the signature length l: at most $|GC| = l + 1$ different GC-content values may occur (Figure 3.11). The resulting spikes in the distribution of the GC-content make it in most cases difficult to define appropriate limits. Even a small change in one boundary may skip over a GC-content spike and thereby vastly reduce (or increase) the search space. Limits should be generously set and the distribution of the GC-content, if possible, determined before usage.

The basic melting temperature (Section 2.1.3) can be used as a filter by defining an allowed temperature range that valid signatures must meet. Not being much more sophisticated than the GC-content formula, the basic melting temperature distribution may also result in similar spikes (Figure 3.12). Determining the melting temperature with nearest neighbor (Section 2.1.4) approximations reduces this effect. An effect not shown here is the mutual influence of the GC-content and a melting temperature filter. Based on the same data with somewhat similar predictions, they may enforce each other when being applied together. If not stated otherwise, no filters were applied in the measurements presented in this work.

3.3.5 Comparison to other approaches

We used our test datasets to compare CaSSiS-BGRT with two other tools for comprehensive signature search, HPD and ProDesign (Table 3.5). We were able to process up to $1,000$ sequences with HPD, at a runtime of 99 minutes and a peak memory consumption of $1,010$ MB. Larger datasets could not be processed due to memory limitations (as a 32-bit MS Windows program, HPD is limited to 2 GB RAM). ProDesign displayed a moderate growth in memory

Length k bases	**Possible Signatures** 4^k	**Unique Signatures** (test dataset)
3	64	64
6	4,096	4,096
9	262,144	262,144
12	16,777,216	7,867,492
15	1,073,741,824	17,516,531
18	68,719,476,736	24,125,196
21	$4.40 \cdot 10^{12}$	30,334,407
30	$1.15 \cdot 10^{18}$	47,632,410
50	$1.27 \cdot 10^{30}$	80,617,100

Table 3.3: The number of possible k-mer signatures grows exponentially with their length k. For RNA and DNA signatures with the alphabet $\sum^k = \{A, C, G, T|U\}$ the growth factor is 4^k. There are fewer unique signatures present in real datasets. The third column contains the number of unique signatures extracted from a rRNA gene test dataset consisting of 512,000 rRNA genes (734,101,088 bases in total).

Evaluation of...	**signatures from dataset**	**all 4^{18} signatures**
Bipartite graph edges	90,712,819	91,116,545
Signatures in graph	7,110,160	7,474,112
Overall runtime (h:m:s)	4m 46s	59h 39m 9s

Table 3.4: Comparison of the number of edges and signatures in the resulting bipartite graphs, computed from extracted (left) and all possible signatures (right). Evaluating all signatures led only to a slight increase in the number of added signatures, but to a vast increase in the overall runtime. The dataset contained 64,000 rRNA gene sequences. The signatures had a length of 18 bases. One mismatch within the target group was allowed.

consumption, but the runtime increased dramatically with growing dataset sizes. Processing 2,000 sequences took over 7 hours and consumed 377 MB RAM at its peak. For comparison, CaSSiS-BGRT was able to process 2,000 sequences in less than 2 minutes using only 111 MB RAM.

We additionally tested Primrose and ARB ProbeDesign. In principle, both tools were able to process the more than 460,000 sequences from SSURef_102, but the two programs could only search signatures for one selected sequence or sequence group per run. Processing single randomly selected sequences without mismatches and outgroup hits took more than 5 hours with Primrose and 5 minutes with ARB ProbeDesign (the creation time of the index excluded). Analyzing all 460,000 sequences with these tools would thus take 262 years or 133 months respectively. (Recall that CaSSiS-BGRT took just 132 hours to find primers for all sequences and sequence groups for this data set (see Section 3.3.3).)

Searches for sequence groups were not conducted with Primrose due to the unexpected long runtime on single sequences. Furthermore, since it is using the NCBI taxonomy and is not capable of processing the phylogenetic tree from SSURef_102, a more thorough comparison of

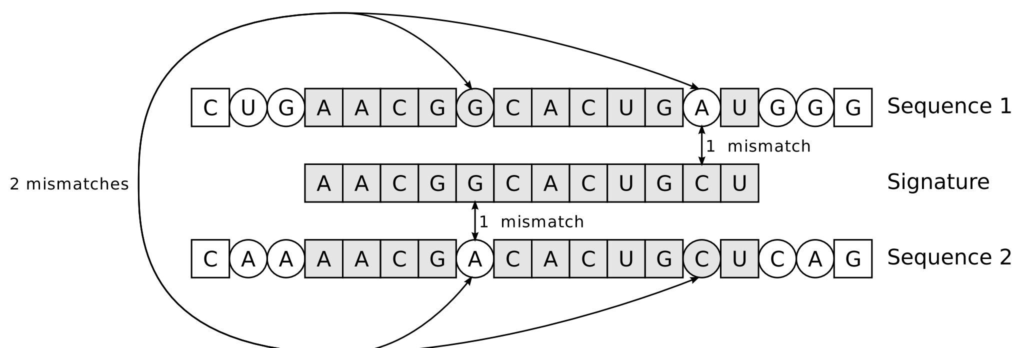


Figure 3.10: Under certain constraints signatures exist, that match within the defined limits but cannot be derived from the sequence data. In this example two sequences with a length of 18 bases are given. A signature with a length of 12 bases is searched and up to one mismatch is allowed. Every subsequence of Sequence 1 with a length of 12 has at least two mismatches in Sequence 2 (displayed as circles), and vice versa. It is still possible to create an appropriate signature with only one mismatch to each of the sequences (matches highlighted gray).

CaSSiS-BGRT with Primrose regarding group specific signatures is out of the scope of this work.

ARB ProbeDesign processed sequence groups in 25 seconds to 70 minutes per group (see supplementary material). Based on the shortest runtime measured for every node in the phylogenetic tree, a comprehensive computation using ARB ProbeDesign would take almost 270 days — excluding the initial configuration and final summarization and evaluation for every query.

By using appropriately adapted settings, ARB ProbeDesign was able to deliver results comparable to those of CaSSiS-BGRT. Applying settings that were too strict or too lax represented a trade-off between computational costs and unsatisfactory results, often leading to a re-run. CaSSiS-BGRT avoids these issues through a more sophisticated preparation of the results.

3.3.6 Evaluation of the computed signature collection

The probability of finding an 18-mer that matches only a single organism in SSURef_102 is 55%. Signatures that match all organisms in a particular group and have no outgroup hits were found for only 14% of all groups. Allowing a small number of outgroup hits led to a noticeable increase in the number of phy-nodes with complete coverage. By computing signatures up to two outgroup hits for the SSURef_102 dataset, the percentage grew from 14% to 23%. For single sequences, the percentage increased from 55% to 71%. Figure 3.13 shows the number of sequences and sequence groups completely covered by signatures against the number of outgroup hits. For higher numbers of allowed outgroup hits, the curves flatten out.

In addition to quantitative aspects, the qualitative usefulness of the signature collection computed for SSURef_102 has been examined by spot tests. For selected target groups, we compared computed signatures with relevant entries in probeBase [68] or the literature concerning availability, coverage and specificity.

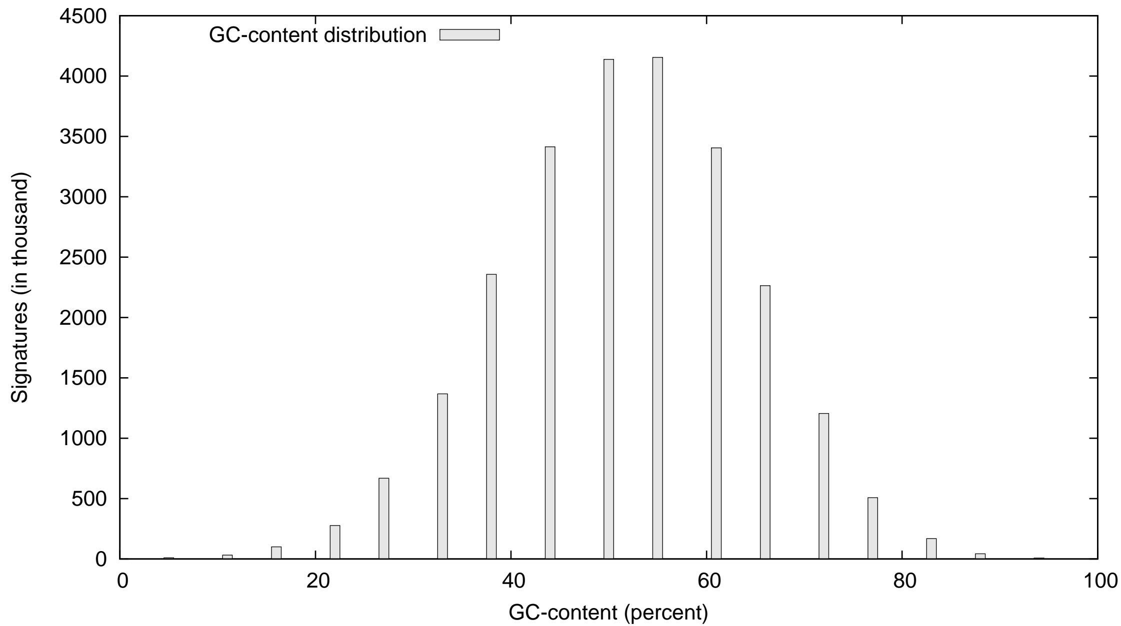
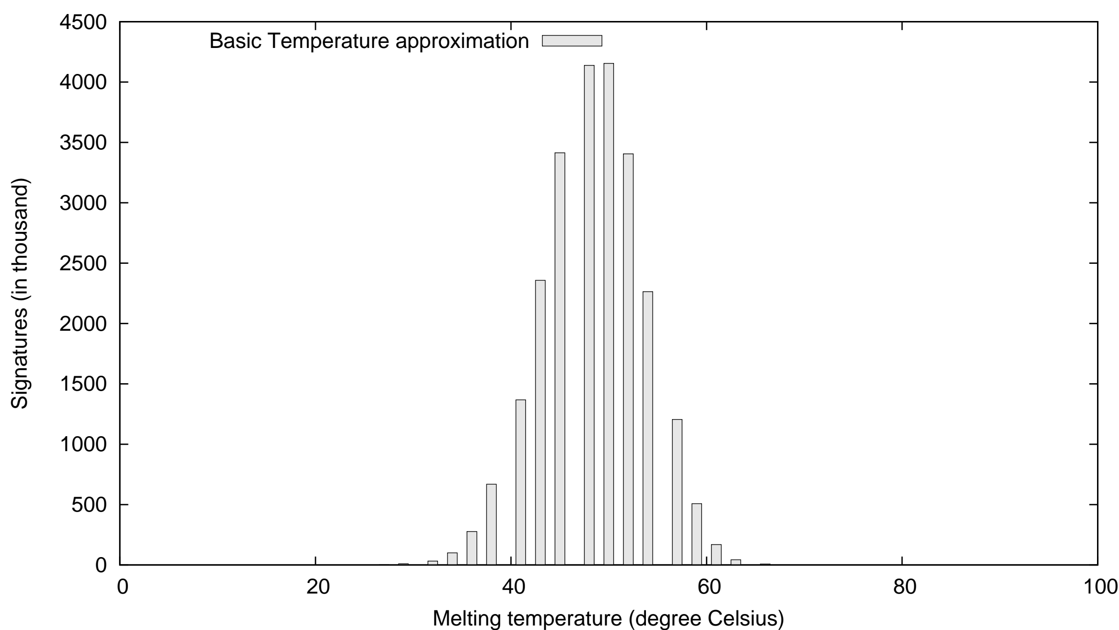


Figure 3.11: GC-content distribution for 24, 125, 196 signatures extracted from a test dataset with 512, 000 SSU rRNA sequences. The signatures had a length of 18 bases, resulting in 19 different possible GC-content percentages.

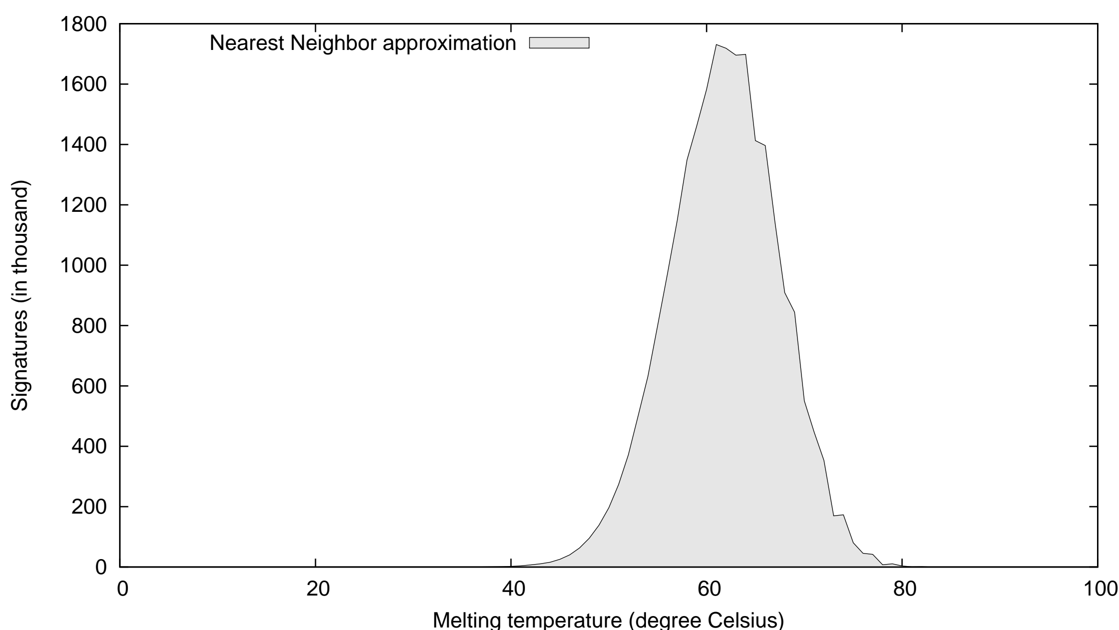
The target sequence of the SSU-targeted probe EUB338, which is used worldwide with different hybridization technologies for the detection of members of the domain "Bacteria" [4, 3], matches 355, 790 sequences in SSURef_102 and three outgroup sequences from other domains. The signature with the highest ingroup coverage found by CaSSiS-BGRT also has three outgroup matches, but matches 356,185 ingroup sequences; significantly, its sequence is almost identical to the EUB338 target sequence except that it is shifted to the SSU rRNAs 3-prime-end by one position. We cannot state exactly which signature is the better one to be targeted in practice. We can state, however, that this 18-mer signature found by CaSSiS-BGRT is highly valuable, since it is almost identical to one of the most frequently targeted signatures cited. This finding was possible because of CaSSiS-BGRT' advantageous capacity to search with relaxed specificity constraints.

The probe VP403, a new 20-mer targeting the 16S rRNA of Verrucomicrobium, most Prostecobacter spp., and uncultured relatives specifically, has been recently successfully applied for Fluorescence in situ Hybridization by [7]. CaSSiS-BGRT also found the complementary signature for this bacterial group, namely the three possible 18-mer substrings of the 20-mer signature sequence targeted by VP403 (N.B.: in this study we conducted CaSSiS-BGRT searches for 18-mer signatures only). All three signatures show the same coverage and specificity properties *in silico*.

Another signature we evaluated was an 18-mer computed for the group Deinococcaceae_Deinococcus. It hits *in silico* 199 out of the 238 ingroup sequences. In regards to



(a) Basic melting temperature approximation



(b) Nearest Neighbor approximation

Figure 3.12: Melting temperature distribution for 24, 125, 196 signatures (with a length of 18 bases) extracted from a test dataset with 512, 000 SSU rRNA sequences. CaSSiS provides a "basic" melting temperature computation (Section 2.1.3) and one based on "Nearest Neighbor" predictions (Section 2.1.4). They may significantly differ, as indicated in this figure, and their applicability differs based on the later applied detection method the signature length.

rRNA	**CaSSiS-BGRT**		**PT-Server**	**HPD**		**ProDesign**	
Sequences	runtime	memory	memory	runtime	memory	runtime	memory
100	9	12	18	40	45	1	162
200	17	16	25	178	122	3	164
500	31	31	48	1.226	361	36	166
1.000	56	49	85	5.985	1.010	928	203
1.500	79	66	97	–	–(1)	4.679	277
2.000	111	85	109	–	–	26.159	377
5.000	302	175	169	–	–	–(2)	–
10.000	873	311	262	–	–	–	–

Table 3.5: Comparison of the runtime (wall time, in seconds) and the peak memory consumption (in MB) of CaSSiS-BGRT, HPD, and ProDesign. (1) HPD: 2 GB Memory limit reached (32-bit program; no more memory available). (2) ProDesign: The measurement was aborted due to extreme runtime (in step 3, reclustering). The memory consumption of CaSSiS-BGRT and the PT-Server were measured separately. (They are two separate processes). The peak memory consumption of the PT-Server was measured after stage 1, as it is only needed in this stage. The peak memory consumption of CaSSiS-BGRT was measured after stage 3. Test system: Lenovo Thinkpad X200s; Intel Core2 Duo CPU L9400 @ 1.86GHz; 8 GB RAM; Windows 7 Professional 64-bit; Ubuntu 10.10 64-bit

the group coverage, this signature is clearly superior to the only signature published so far for this group [118], which matches just 37 ingroup sequences. Furthermore, we could rapidly find 7 new signatures with a coverage of more than 95% for the group Coprothermobacter. We have no information about the significance of this group of bacteria, but no signature for this group has been published to date.

These results indicate that the comprehensive 18-mer collection computed from SSURef_102 by CaSSiS-BGRT does include sequences which could be valuable diagnostic targets. Using CaSSiS-BGRT, we found signatures which have been already successfully targeted in the wet lab. We identified signatures which have ingroup coverages superior to previously published signatures, and we found potentially valuable signatures for bacterial groups for which no SSU rRNA-targeted signature has been published so far. More details are presented in the supplementary material in Section A.2.

3.4 Parallelizing the BGRT traversal with OpenMP

The CaSSiS-BGRT approach was in its initial implementation designed as a single process, consisting of three subsequent stages (Section 3.2). The first two extract valuable signature candidates and build the BGRT data structure. The third stage traverses the BGRT structure and searches the best possible signatures for every node of a phylogenetic tree. Consequently, the first two stages were combined as BGRT “create” and the latter defined as BGRT “process”. This also allowed storing the BGRT data structures into files, and thereby their sharing and independent processing.

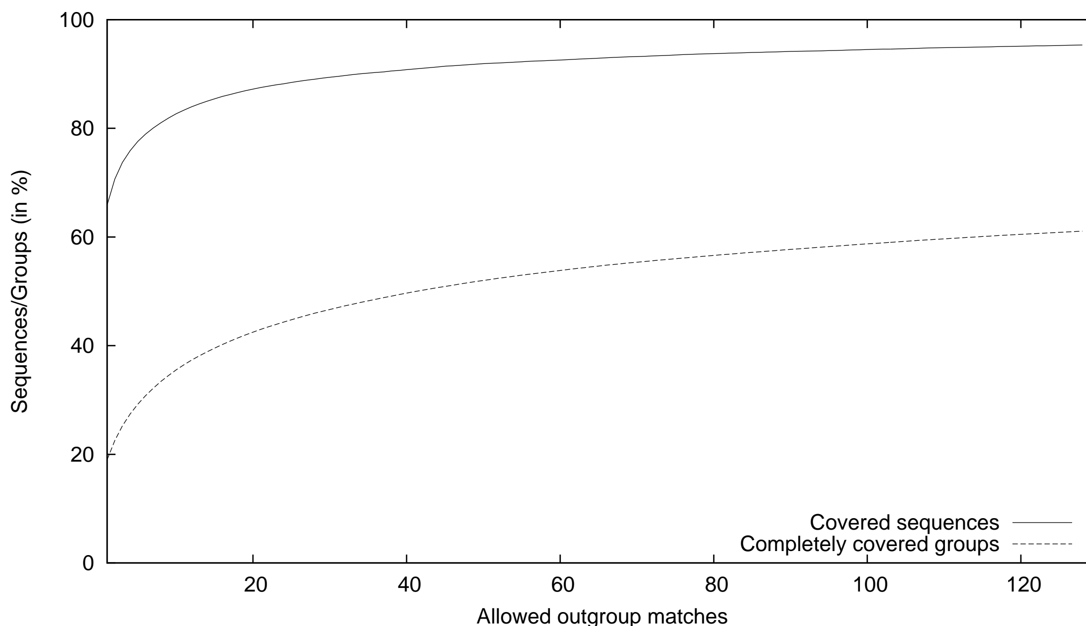


Figure 3.13: Percentage of sequences and sequence groups in SSURef_102 which are completely covered by at least one signature. Searches are performed under relaxed specificity conditions (disjoint values; stepwise allowance of 0 to 128 outgroup matches).

When comparing the runtime of BGRT "create" (Section 3.3.2) with BGRT "process" (Section 3.3.3), the runtime and memory consumption of the latter grows faster with increasing dataset sizes. The parallelization of the BGRT traversal, being the bottleneck, was therefore further evaluated in a student's project. This section presents aggregated extracts from the final report of the bachelor project "OpenMP Parallelization of CaSSiS-BGRT" by Sebastian Wiesner, that was supervised by me. It documents the modifications that were necessary to parallelize the CaSSiS-BGRT traversal with OpenMP and the achieved runtime gains.

3.4.1 OpenMP Implementation

CaSSiS-BGRT contains multiple data structures, most notably the phylogenetic tree, the BGRT structure, and the result array (Section 3.2). The phylogenetic tree and the BGRT are only read in the third stage of the algorithm, the BGRT traversal, and therefore needed no modification to be thread-safe. However, each new best result is immediately stored in a result array. The array therefore must be thread-safe to prevent race conditions, lost updates and other typical problems that might occur when parallelizing an algorithm.

Two different kinds of strategies could be used to avoid these problems. The first would be simply changing the result array into a thread local resource. Each thread independently fills the result array and in the end the results are merged. An obvious drawback of this approach could be a reduced number of cutoffs and thereby a decreased effectivity of the bounding during BGRT traversal. On the other side it completely avoids any kind of locking and allows to perform cutoffs truly parallel. To be effective it is assumed, that each thread still gathers

enough cutoff data and thus the reduction of the cutoff rate is negligible compared to the gain of unhindered parallelization.

A second approach would be adding locking mechanisms to the result array and thereby allowing exclusive access for one thread at a time. The result array would remain shared and all results would be available at the end without the need to merge them.

The actual implementation of a parallelized BGRT traversal was preceded by tests to check the level and quality of OpenMP support provided by the GNU C++ compiler. The most notable one was the attempt to use the `task` directive [82]. Measurements revealed, that tasks greatly slowed down the overall runtime. Apparently, the OpenMP implementation used by the GNU C++ compiler was unable to automatically determine a sensible threshold for task-based parallelism. New tasks were generated on each single recursive call, straightly following the directives, so that the task management overhead by far exceeded any possible gain from parallelization. Given these observations, tasks were dismissed.

Classic work-sharing directives from OpenMP 2.0 were used to parallelize the BGRT traversal. Unlike tasks, these directives are unable to parallelize any kind of irregular problems, e.g. recursions or unbounded loops. During BGRT traversal, a bounded loop (Procedure traverseBgrTree) is used to iterate over all top-level nodes in a BGRT tree. The heavy computational work is performed in a recursive function (Procedure traverseBgrTreeRecursion) which is called for each of these nodes. The rest (outside the loop) deals with the relatively cheap traversal of the phylogenetic tree. It was chosen to distribute the computational work among the processing cores of the system by simply parallelizing the first bounded loop.

OpenMP provides the `for` directive [82] to parallelize such bounded loops. Procedure traverseBgrTree shows how this directive was applied in the CaSSiS-BGRT sources. Leaving locking aside, this little change was sufficient to parallelize the computational work by letting OpenMP split the loop into somehow even parts, and execute each of these parts in a separate thread. The exact strategy how the work is distributed among the threads, and the scheduling strategy, are implementation-defined. We did not influence the standard scheduling strategies [82] that OpenMP provides.

Input: BGRTNode *node*
1 . . . process current BGRT node . . .
2 **foreach** *childnode of node* **do**
3 traverseBgrTreerecursion(*childnode*);
4 **end**

Procedure traverseBgrTreeRecursion(BGRTNode *node*) recursively traverses BGRT nodes.

3.4.2 Benchmark setup and results

The influence of the OpenMP parallelized BGRT traversal was measured on a 64 bit Ubuntu 11.10 system equipped with 36 GB of main memory. Its two 6-core Nehalem (X5670 Westmere) processors, fixed at 2.93 GHz, allowed with activated HyperThreading up to 24 concurrent threads. For the test, five databases of increasing size (8,000, 16,000, 32,000, 64,000

```
Input: BGRTree tree
1 #pragma omp parallel
2 {
3 #pragma omp for
4 foreach 1st_level_node of tree do
5     traverseBgrTreeRecursion(1st_level_node);
6 end
7 }
```

Procedure traverseBgrTree(...) initial bounded loop iterates over all top level nodes in the BGRT.

and 128,000 randomly selected rRNA gene sequences) were derived from the ARB SILVA SSURef_104 dataset. Each database was processed with different parallelization settings. First, OpenMP was completely disabled to obtain a serial reference measurement. Then the datasets were processed with 1, 2, 4, 6 and further in binary steps up to 24 OpenMP threads.

The runtimes in Table 3.6 and the resulting speedup in Figure 3.14 show, that the overhead of the single threaded OpenMP parallelization compared to the serial version of the BGRT traversal is negligible. The speedup grew sublinear with the number of threads. With 24 threads, the traversal was up to about 4.5 times faster. Within our test range, the benchmarks did not reveal a point, at which the speedup stagnates or even declines. This should not imply that no such point exists. Additionally, a certain overhead due to the management of the threads via OpenMP is noticeable. Note, that the BGRT construction was not affected by parallelization, and its runtime was identical for same dataset sizes. The higher speedup of the dataset with 32,000 sequences is probably due to a favorable distribution of the randomly selected sequences.

3.5 Discussion

The CaSSiS-BGRT algorithm enables fast and comprehensive search for sequence- and sequence group-specific signatures in large hierarchically-clustered sequence datasets with modest memory requirements. Adding CaSSiS-BGRT to their workflow (Figure 3.15) could be of interest for maintainers of hierarchically-clustered databases, such as SILVA, RDP, or Greengenes [89, 25, 29]. Many of these maintainers provide online tools for matching oligonucleotide sequence strings against their data collections, such as *Probe Match* on the RDP website or *Probe* by Greengenes. However, neither of them provide tools for signature search, nor do they offer a collection of signature sequence candidates for their data — two features CaSSiS-BGRT is able to provide. CaSSiS-BGRT could also be beneficial for users who want to process collections from projects like FunGene at the Michigan State University (`http://fungene.cme.msu.edu/`), which maintains more than 40 aligned collections of homologous gene sequences. An appropriate clustering, usually a phylogenetic tree, could be computed from the available aligned sequences using third party tools like FastTree or RaxML [88, 106]. By using precomputed BGRT-files (which contain the BGRT structure), single queries based on freely defined group definitions

Threads	**Dataset size (rRNA sequences)**				
#	8,000	16,000	32,000	64,000	128,000
serial	6.39 (5.94)	22.70 (21.78)	96.12 (94.36)	242.32 (238.61)	897.05 (890.21)
1	6.57 (6.14)	22.36 (21.53)	77.00 (75.38)	255.17 (251.93)	–
2	5.97 (5.54)	19.31 (18.44)	66.35 (64.72)	207.73 (204.28)	–
4	4.44 (4.00)	15.22 (14.39)	51.14 (49.37)	164.22 (160.88)	–
6	3.87 (3.45)	12.96 (12.12)	43.45 (41.66)	139.30 (135.91)	–
8	3.36 (2.93)	11.05 (10.21)	37.77 (36.11)	121.93 (118.73)	–
10	3.04 (2.60)	10.08 (9.21)	35.37 (33.70)	113.15 (109.84)	–
12	2.78 (2.36)	9.27 (8.45)	30.49 (28.85)	99.17 (95.84)	328.23 (321.86)
14	2.59 (2.16)	8.64 (7.81)	28.12 (26.48)	94.60 (91.34)	301.02 (294.69)
16	2.45 (2.03)	7.96 (7.14)	27.16 (25.52)	88.63 (85.45)	283.28 (277.03)
18	2.35 (1.94)	7.66 (6.83)	25.02 (23.42)	84.97 (81.81)	269.58 (263.32)
20	2.29 (1.87)	7.36 (6.55)	24.37 (22.77)	81.17 (77.93)	258.42 (252.13)
22	2.31 (1.87)	7.30 (6.47)	23.79 (22.17)	77.68 (74.37)	246.55 (239.88)
24	2.22 (1.79)	6.89 (6.06)	22.82 (21.22)	75.67 (72.45)	238.92 (232.58)

Table 3.6: Overall runtime of the OpenMP parallelized CaSSiS-BGRT approach in minutes. The sole runtime of the BGRT traversal step is shown in brackets. The sequence datasets were computed without allowing mismatches ($m_1 = 0$) and a distance of one ($m_2 = 1$). Results within a range from 0–10 outgroup hits were computed. Some of the measurements with 128,000 sequences could not be done due to the lack of computation time.

can be processed.

The computation of valuable signatures for every group within a large phylogenetic tree can quickly lead to excessive runtime. Other tools we have tested were either unable to process current dataset sizes or they would have needed an extremely long time to do so (Section 3.3.5). CaSSiS-BGRT copes with this by relying on the BGRT structure for the storage of the relation between signatures and sequences (Section 3.2.2). Its combination with a phylogenetic tree allows CaSSiS-BGRT to avoid expensive computations to determine clusters and to instead focus on finding signatures. This allows CaSSiS-BGRT to process the SILVA SSURef_102 dataset with more than 460,000 sequences. The group hierarchy of the phylogenetic tree is critical for the bounding method; given only flat clusterings, the 90% reduction in the BGRT-traversal using our bounding method (Section 3.3.3) would not be applicable. Additionally, only processing signatures that are present in a dataset is in most cases sufficient (Section 3.3.4). Processing all theoretically possible numbers of signatures does not lead to a significant increase in the quality of the results, especially when evaluating signatures for large target groups.

The BGRT only contains the relation between signatures and the organisms they match (Section 3.2.2). The information where exactly a signature matched a sequence as well as the position(s) of possible mismatches are not stored within the BGRT. Adding this information directly to the BGRT would lead to a huge increase in its memory consumption, without causing significant benefits. Through this data reduction, a fast signature search even for freely defined groups of organism identifiers is possible. By making the BGRT storable, multiple different

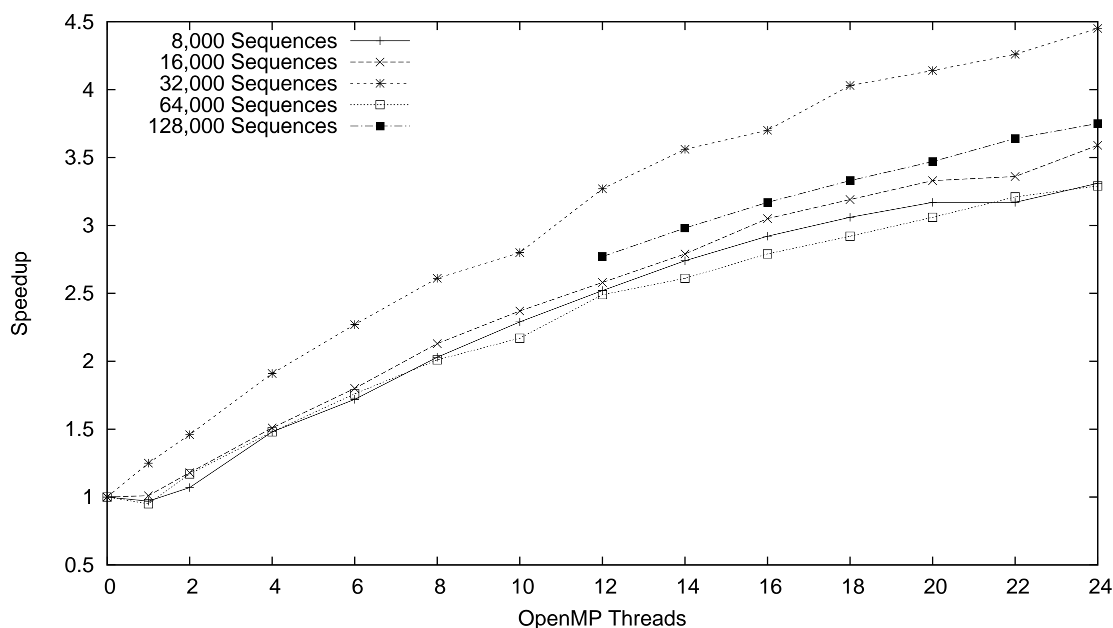


Figure 3.14: The speedup of the parallelized BGRT traversal step grows sublinear with the number of threads. For each database size, it is calculated by dividing the traversal time for each measured number of threads with the corresponding serial measurement (in the graph displayed as 0 OpenMP threads).

clusterings based on the same set of sequences could be evaluated in stage 3 without the necessity of repeating the previous two stages. The "loss" on information could be compensated with a post-processing stage of the results. A fourth stage could be added to CaSSiS-BGRT to gain this information. This stage would be cheap in runtime and memory consumption, compared to the BGRT creation and traversal stages.

Processing large phylogenetic sequence datasets means processing fuzzy data for two reasons: probable errors in the sequences, and errors in the clusterings. Although SILVA, the source of our test datasets, is a maintained secondary database containing high quality sequences and annotations from public databases [89], the occurrence of erroneous information, e.g. sequence errors, cannot be ruled out. Naturally, sequence errors influence the results of a CaSSiS-BGRT calculation negatively, in particular with regards to the signatures selected for single organisms. We are aware of regions with sequence errors that are selected as organism-specific signatures where "uniqueness" has been induced by the error itself. This problem is not solved by any extant tool, including CaSSiS-BGRT. In order to minimize such erroneously selected signatures, probabilities for the occurrence of highly individual signatures within certain gene regions — e.g. based on conservation profiles — would have to be taken into account for SSURef_102. Such a method is computationally expensive and could lead to the exclusion of valuable signatures as well. However, the main application of CaSSiS-BGRT is searching for group-specific signatures. Here, the effect of erroneous sequences is not dramatic, since CaSSiS-BGRT selects signatures

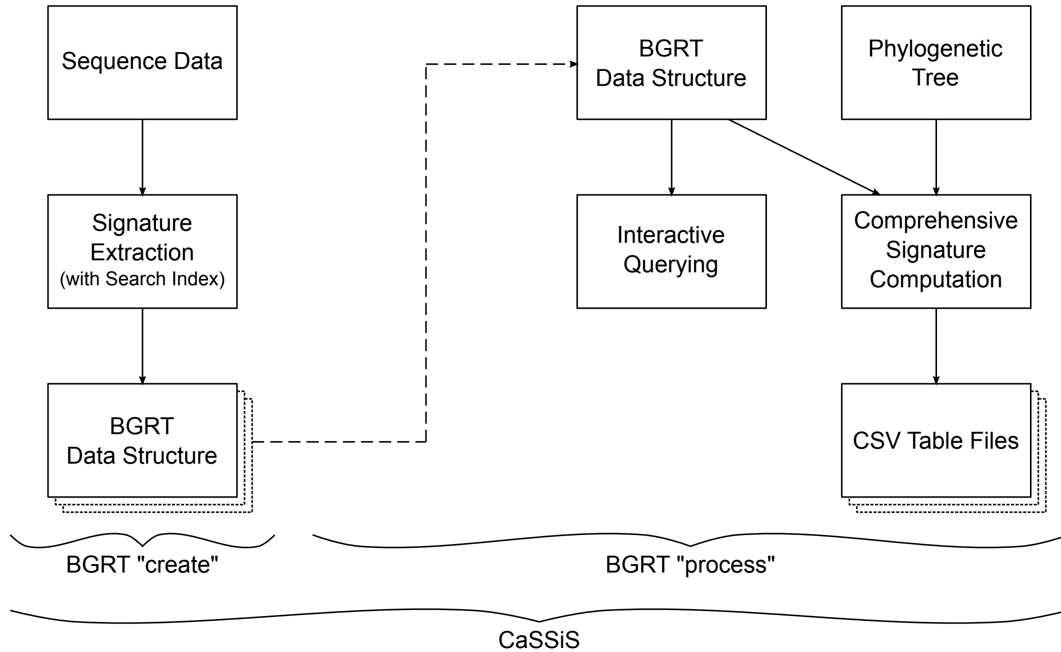


Figure 3.15: CaSSiS-BGRT enables the rapid computation of comprehensive sets of valuable sequence- and sequence-group-specific signatures. In a first step, the BGRT data structure is created and stored. In combination with a phylogenetic tree, the BGRT structure can then be used for comprehensive computations of signature candidates. End users who are primarily interested in few defined target groups, but do not want to give up a large amount of background data, can use these pre-computed BGRT structures to generate signature candidates for freely defined groupings of sequences.

with the highest possible group coverages. This approach reduces the chance of selecting an erroneous signature significantly; the probability of a signature being erroneous decreases with the increasing number of group sequences in which it occurs. Some other systems [22, 34] also have this capability, but exhibit significant limitations in runtime and memory performance (Section 3.3.5).

Furthermore, since several heuristics have to be applied for reconstructing large phylogenies, an error-free tree cannot be guaranteed. As a result, the phylogenetic trees could suffer from misplaced organisms. If CaSSiS-BGRT was to only consider signatures that match within a target group, this would lead to suboptimal results for large datasets; for example, perfect signatures were only found for 14% of all groups for SSURef_102. By allowing outgroup matches when searching for signatures, *false negatives* (e.g. misarranged OTUs in a phylogenetic tree) can be found with CaSSiS-BGRT. The only other approach that tries to cope with such uncertainties is ProDesign. ProDesign uses reclustering [34] to find more perfect group signatures. This program, however, is not applicable with a large dataset such as SSURef_102 due to its computational complexity (Section 3.3.5 and supplementary material).

Aside from mitigating negative effects introduced by inexact input data, the ability to find signatures with outgroup hits can help to determine further valuable diagnostic sites. Such sig-

natures become applicable when the co-occurrence of target and cross-reacting non-target DNA within the samples examined can be ruled out [3] or when negative probes sensing specifically for the presence of the non-target DNA are additionally applied [75].

Using the ARB PT-Server for high-throughput matching of the signature candidates in stage 1, inexact searches according to a predefined mismatch limit can be performed. CaSSiS-BGRT is able to enforce a defined minimum Hamming distance to outgroup sequences (m_2) as well as an upper limit for mismatches to the target sequences (m_1; see Section 3.2.1). This feature could be valuable for finding signatures that provide advanced sensitivity and specificity properties under non-standard conditions. These could include degenerated signature sequences or signatures which cover some ingroup sequences with a small number of weak mismatches and exhibit large Hamming distances to outgroup sequences simultaneously. The ARB PT-Server also supports *weighted mismatches*, a more sophisticated distance measurement that also considers the position of a mismatch. Its applicability has been shown for hybridization approaches [121], and it allows experienced users a usability prediction without requiring a post-evaluation in the wet lab.

We are not aware of any other comprehensive approach that allows the definition of a maximum Hamming distance to targets or a minimum distance to non-targets. Due to hard-coded search constraints [34] ProDesign is unable to guarantee any of these two. ARB-ProbeDesign [70] can at least be configured to guarantee a distance of one mismatch to the outgroup. Insignia first calculates short signatures that have at least one mismatch with all outgroup signatures. It then concatenates these signatures if they overlap within the targets [86]. However, Insignia's method does not guarantee a defined minimum distance to non-target sequences for signatures of a particular length.

By providing a comprehensive signature collection for hierarchically-clustered sequence data, CaSSiS-BGRT could support so-called multiple probe approaches. Here, multiple oligonucleotide probes, targeting signatures with overlapping specificities, e.g. different taxonomic levels, are used in order to increase the overall specificity. Signature sets with nested specificities are often used for bacterial diagnostics or biodiversity studies. They allow the detection and classification of previously unknown bacteria: specific signatures for known organisms or organism groups are not detected, but a signature of a superior taxon is [99, 69, 100]. Additionally, collections of oligonucleotide signatures of different lengths (e.g 15-25) can significantly improve the signature supply for primer and probe design and can be created by combining the results from multiple CaSSiS-BGRT-runs.

CaSSiS-BGRT creates result files containing target names (taxonomic group/organism names, if available) or node IDs, the group size, the coverage, and the respective signatures for each node in the phylogenetic tree. Each file contains the signatures with a particular number of outgroup matches featuring the highest in-group coverage. Oligonucleotide probes or primers, derived from signature sequences found by CaSSiS-BGRT in SSURef_102, could be worth a trial in wet laboratory experiments. This is indicated by spot tests in which signatures found by CaSSiS-BGRT have already been successfully targeted in earlier FISH studies [4, 7]. For optimizing each probe *in silico* as far as possible according to application-dependent re-

quirements, however, additional information, such as names of inexact outgroup matches, their exact Hamming distance to the signature string, as well as mismatch types and positions, would be helpful for downstream users. Such information is currently not provided with the CaSSiS-BGRT result files, although it could be determined in a future version. This information can be easily retrieved by simple string matching against the SSURef_102, using either the online tool *probecheck*, or, if the computational hardware resources are sufficient, applying the tool ARB ProbeMatch (both approaches rely on the ARB PT-Server [66, 70]).

Although CaSSiS-BGRT performs quite well on the large test dataset analyzed, we evaluated further strategies to enhance its performance, especially in regard to memory consumption. For our tests, the SSURef_102 dataset was computed on a workstation with 24 GB of RAM. Clearly, larger genomic datasets could exceed the capacity of individual machines. Additionally, the signature extraction during the first two stages depends on an (external) search index, the PT-Server, which is only able to sequentially process requests. For the construction of the BGRT, a distributed approach was therefore evaluated which is presented and discussed in Chapter 5. With this approach, we were able to accelerate the matching of signatures with the PT-Server up to five-fold using parallel and distributed computing. Furthermore, by partitioning the dataset, we were able to reduce the memory consumption per phy-node inversely proportional to the number of partitions for the first stage (the second stage is not performance critical; see Section 3.3.2).

In order to improve the critical third stage, the impact of multicore processing on its runtime was evaluated. An OpenMP parallelized version of the BGRT traversal was implemented as a student's work (Section 3.4). The results, the achieved four-fold speedups on 24 cores, were disappointing. The probable causes are multiple locks that were used to synchronize the threads during the merge of the results for each phy-node. In retrospect, a better solution could have been to give up the used "global" bounding method where cutoffs are shared throughout all threads. Using thread-local bounds could still have lead to a certain number of cutoffs. Locking mechanisms could have been prevented with thread-local result tables (instead of one shared) which are merged after all results were computed. Both approaches were not implemented because the expected increase in memory consumption was considered to be too high. As Figure 3.9 shows is the memory footprint of a BGRT linear to the number of sequences it was computed from, but the additional information for the cutoffs and the result entries leads to an exponential growth of the memory consumption. This factor would probably multiply with the number of threads.

Another compromise could be the partitioning of the BGRT structure. Partitioning the BGRT is likely to have a negative impact on the efficacy of the bounding method as well, but with less overall impact in a distributed approach like in Chapter 5. This would enable both parallel processing on different bgrt-nodes within a computer cluster as well as reduction of per-node memory consumption.

But a far more important reason why the parallelization was not pursued was the development of a more efficient approach which is presented in the following Chapter 4. The CaSSiS-BGRT algorithm, presented in this chapter, has a worst-case complexity of $O(|M| \cdot |V|)$ time

where M is the size of the BGRT and and $|V|$ is the set of all organisms. The implemented bounded search, a time-memory trade-off, requires $O(d \cdot |M|)$ memory where d is the depth of the phylogenetic tree. The new algorithm, called *CaSSiS-LCA*, allows to solve the same combinatorial problem in almost linear time (empirically observed). Without the need for a BGRT lookup structure, its overall memory consumption is vastly reduced to the memory footprint of search index and the result entries in the phylogenetic tree nodes.

Besides optimization and parallelization, replacing the ARB PT-Server by a faster approximate search method could further accelerate the performance. Especially when searching for signatures with guaranteed Hamming distances to the outgroup (or to within-group sequences) of more than one base, stage 1 becomes the runtime-critical computational step (see Section 3.3.1 and supplementary material). Periodic spaced seed based search methods showed promising results when applied to mapping high throughput reads to the human genome [21]. However, their suitability for usage in CaSSiS-BGRT was not examined in this work. Data handling, search efficiency for matches with more than 3 mismatches, and performance for short oligonucleotide searches resulting in huge match lists would have to be taken into account.

Chapter 4

The CaSSiS-LCA Approach

This chapter presents a new algorithm for finding oligonucleotide signatures that are specific and sensitive for organisms or groups of organisms in large-scale sequence datasets. We assume that the organisms have been organized in a hierarchy, for example a phylogenetic tree. The resulting signatures, binding sites for primers and probes, match the maximum possible number of organisms in the target group while having at most k matches outside of the target group.

The key step in the algorithm is the use of the Lowest Common Ancestor (LCA) to search the organism hierarchy; this allows to solve the combinatorial problem in almost linear time (empirically observed). The presented algorithm improves performance by several orders of magnitude in terms of both memory consumption and runtime when compared to the best-known previous algorithms while giving identical, exact solutions.

This chapter gives a formal description of the algorithm, discusses details of our publicly available implementation and presents the results from our performance evaluation.

This Chapter is an extended version of an article [10] published in the ACM Journal of Experimental Algorithmics. It is a collaboration with Prof. Mikhail J. Atallah and Christian Grothoff and describes the CaSSiS-LCA algorithm.

4.1 Introduction

Molecular diagnostic techniques, which are applying polymerase chain reaction (PCR) [15] or RNA/DNA hybridization [5, 105], are becoming a standard in various fields of life sciences and medicine. They rely on oligonucleotide primers and probes, short (15 to 25 bases) subsequences of DNA or RNA. These subsequences bind to longer sequences from a biological sample to start the desired biochemical reaction. Binding sites on the samples, the complement of the primers or probes, are called *oligonucleotide signatures* (hereafter simply referred to as signatures). To interact only with sequences from the target organism, they must be specific to the target organism or group of organisms.

A source for such primers and probes can be curated databases like probeBase [68] or, in many cases, the design “by hand.” Given the size of modern sequence datasets, software tools are necessary to design new or re-evaluate already published candidates before testing in the

wet lab. In this chapter we present an efficient algorithm for the comprehensive *in silico* search for good signature candidates (Figure 4.1). Maintainers of gene and genome databases, such as SILVA, RDP, or Greengenes [89, 25, 29], could use it to precompute and offer a collection of signature candidates along with their datasets, but it also is useful for end-users with custom sequence collections based on projects like FunGene[1].

For applications in this domain, our algorithm assumes that the organisms are hierarchically clustered. Clusterings could be based on any kind of classification where inner nodes represent related groups. An example is a phylogenetic tree: Inner nodes represent derived evolutionary relationships between groups of organisms, and the individual organisms correspond to the leaves. Phylogenetic trees, if not supplied with sequence datasets, can be computed using third-party tools such as FastTree or RAxML [88, 106].

The other main input is a bipartite graph (Figure 4.1, center) that relates signatures to matched sequences (organisms). A relation means that a signature is present in a sequence. Such a bipartite graph is easily computed using existing search index tools that construct suffix trees over sequence data. Signatures are extracted by traversing the tree until a certain (length) constraint is met, and references to the sequences can be found in the underlying nodes.

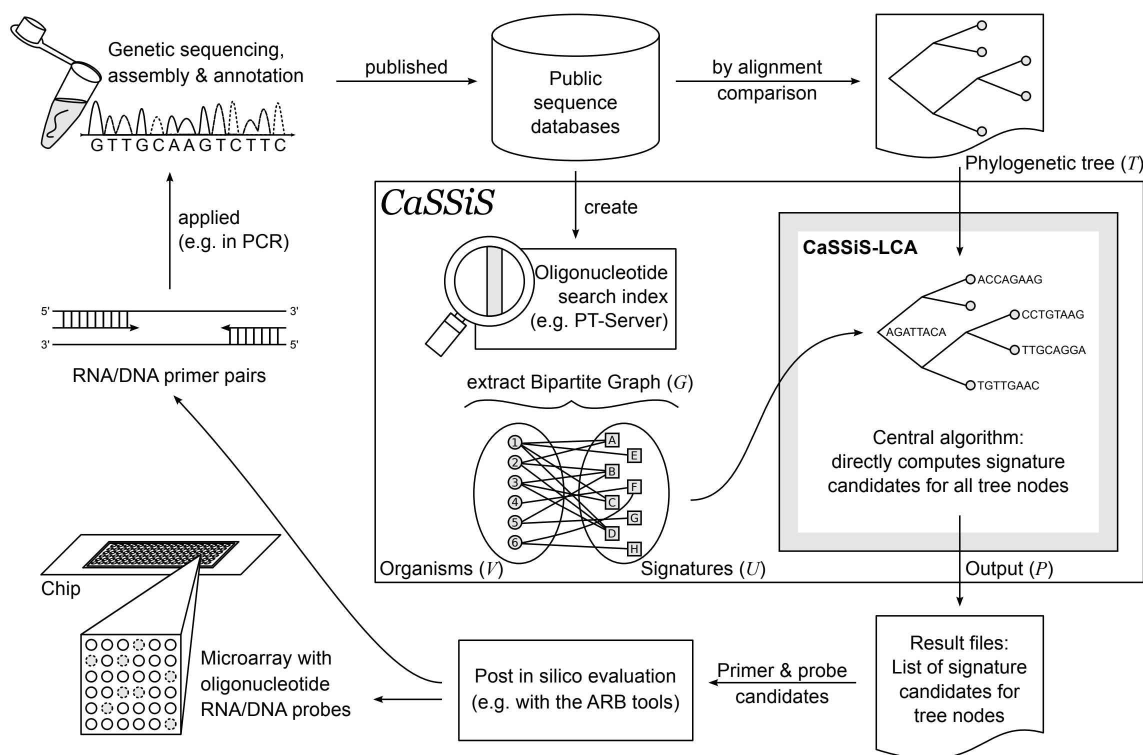


Figure 4.1: Schematic of the primer and probe design pipeline in which CaSSiS is embedded. The input data for the new algorithm CaSSiS-LCA comes from public sequence and phylogenetic tree databases. The resulting signatures can be used as a template for new RNA/DNA primers and probes (e.g. to provide diagnostic microarrays).

[1] http://fungene.cme.msu.edu/

It should be noted that depending on the target group and the available sequence data, there may not be a perfect match; a perfect match would be a signature that matches all target sequences and has no matches outside of the target group (no false-positives). Thus, we are interested in algorithms that support relaxed search conditions; the algorithm should minimize the number of target sequences that are not matched (false-negatives) while allowing for at most k false-positives (where k is typically a small number). In practice, inaccuracies in the available sequence data and additional constraints (such as melting point restrictions) complicate the situation further. However, many of these issues have been addressed in the prior chapter 3. In this chapter, we concentrate on the central algorithm for the search for good signature candidates, hereinafter referred to as CaSSiS-LCA.

We will use the following formalism to present and discuss our algorithm. Let $G = (U, V, E)$ be a bipartite graph where edges $(u, v) \in E$ represent that a signature $u \in U$ matches an organism $v \in V$. Furthermore, let T be a tree with leaves in V (e.g. T might represent a phylogenetic tree). Furthermore, let $D(t)$ be the set of all descendants of $t \in T$ and $P(t)$ the result set of signatures for t. Then, this chapter presents an $O(k|U| \log |V| + k|V| + |E|)$ time algorithm (detailed analysis in Section 4.3.3) which determines for all elements $t \in T$ those element(s) $u \in U$ that maximize the number of edges $(u, v) \in E$ with $v \in D(t)$ while not having more than k edges $(u, v') \in E$ with $v \notin D(t)$. For each t, the resulting signatures are stored in $P(t)$. Note that in practice $|E|$ is several orders of magnitude larger than $k|U| \log |V|$; thus the runtime is practically linear in the size of the input.

Following the notation from the previous Chapter 3, we will call edges $(u, v) \in E$ with $v \in D(t)$ *ingroup* matches for group t and edges $(u, v') \in E$ with $v \notin D(t)$ *outgroup* matches for group t. The bound k is the maximum number of outgroup matches that can be tolerated. If the resulting signature u is used for diagnostics, outgroup matches would result in false-positive tests. The goal of the algorithm is to maximize the number of ingroup matches. If there exist organisms $v \in D(t)$ where $(u, v) \notin E$, this would result in false-negative tests when using signature u to test for group t. In other words, the presented algorithm finds for each group of organisms $t \in T$ all of those signatures $u \in U$ that have less than k false-positives and minimize false-negatives.

The remainder of this chapter is structured as follows. We review related work in Section 4.2. Our algorithm is presented in Section 4.3. Details about our implementation and its performance are given in Section 4.4.

4.2 Related Work

Currently, the use of primers and probes is widespread in the field of medical diagnostics. In previous work, we have shown that computational methods can find typical representatives that are applied in this field [12]. An example is *EUB338*, a domain-specific probe used for the detection of organisms classified as “bacteria” [3]. Our algorithm not only found the signature corresponding to EUB338, but it also presented a signature with a higher coverage (its position was shifted by one base compared to EUB338) [12].

The *in silico* search for oligonucleotides is provided by various tools. Most of them are specialized in either primer design [32, 85, 35, 93] for PCR applications, or the design of probes [22, 34] that could be applied in DNA microarrays. However, more generic approaches also exist that try to identify signature sites [70, 62].

For various reasons, we found these tools unsuitable for the comprehensive computation of hierarchically clustered sequence datasets. They are typically limited to processing datasets of a few thousand gene sequences or a few genomes due to excessive memory or time requirements [12]. In most cases, oligonucleotides are computed only for one predefined set of targets and nontargets per run. Another limit is the lack of relaxed nonheuristic search methods, which gain importance when processing large datasets [12].

Prior to this work, the only algorithms known to us capable of doing a comprehensive nonheuristic signature computation based on large hierarchically linked gene and genome sequence datasets were Insignia [87] and the first CaSSiS implementation, hereinafter referred to as CaSSiS-BGRT [12].

4.2.1 Insignia

Insignia is a Web application developed and maintained by the Center for Bioinformatics and Computational Biology at the University of Maryland. It currently (May 2012) contains 13,928 genomic organism sequences (11,274 viruses/phages and 2,653 nonviruses).

Insignia consists of two pipelines. The first "match pipeline" is used to pre-compute "match cover" arrays M for every pair of organisms. For example, for $v_1, v_2 \in V$ and $v_1 \neq v_2$, the match cover array $M(v_1, v_2)$ contains the positions and lengths of all sequence regions of v_1 that are also present at one or more positions on v_2. To find common regions for an organism pair, Insignia uses MUMmer [59] to build a suffix-tree-based search index. Signatures of a defined length matching v_1 and v_2 are extracted, merged if their positions on v_1 overlap, and added as regions to the match cover.

A match cover consists of integers, that is, position and length pairs on a reference sequence. The number of pairs in a match cover is bound by the sequence length l. To process ≈80 billion nucleotides from NCBI RefSeq genome database[2], the first pipeline had to be distributed across a 192-node cluster [86]. The authors did not provide information about the actual runtime of the algorithm. The memory consumption for the match cover M for 300 organisms is reported to be only ≈2GB [87].

After precomputing the match cover, the second "signature pipeline" is triggered over Insignia's web interface. The signature pipeline computes regions (i.e., one or more overlapping signatures) shared between a user-defined group of target organisms $V' \subseteq V$ which must also be absent in the background $V'' = V \setminus V'$ [87]. One target organism $v' \in V'$ is designated as the reference organism and used to visualize the result.

The signature pipeline consists of three steps. In a first step, an intersection $I_{v'} := \bigcap_{v_t \in V'} M(v', v_t)$ of the match cover structures from v' with the other target organisms from

[2] http://www.ncbi.nlm.nih.gov/RefSeq/

V' is created. It contains only sequence regions on v' that are shared by all targets V'. In a second step, a union $U_{v'} := \bigcup_{v_b \in V''} M(v', v_b)$ of the match cover structures from v' with all background organisms V'' is created. $U_{v'}$ thus contains all regions on v' that match one or more organisms from the background V''. In the last step, the regions from $I_{v'}$ are compared to the ones in $U_{v'}$ to find possible signature candidates. Valid signatures for the targets V' have to completely lie within a region on $I_{v'}$ and must not entirely lie within a region on $U_{v'}$.

All operations in the signature pipeline have (practically) linear time complexity in the size of the match cover [87]. The match cover intersection $I_{v'}$ for a target group V' (and a reference organism v') has a time complexity of $O(|I_{v'}| \log |V'|)$. The $\log |V'|$ factor can be treated as a constant due to the bounded number of genomes [87]. A single query takes, on average, one minute to process [87].

The final output of Insignia is a list of regions consisting of overlapping k-mer signatures. Insignia was primarily designed to process single queries (single targets or groups of target sequences) and not for handling deep hierarchies. In contrast to the work presented in this chapter, Insignia is only able to report k-mer signatures perfectly matching the whole target group without allowing nontarget matches. However, for many reasonable groups of organisms such perfect signatures often simply do not exist. In our test datasets, only 55% of the organisms and 14% of all groups were perfectly covered [12, Section 3.5] by one or more signatures. Furthermore, Insignia can also not be used to find signatures with small mismatches to the target sequences (which is useful to tolerate sequencing errors) or to enforce larger Hamming distances to nontarget (background) organisms. CaSSiS and the improvements over CaSSiS that are presented in this chapter address these shortcomings.

4.2.2 CaSSiS-BGRT

The CaSSiS-BGRT [12] algorithm and the CaSSiS-LCA algorithm presented in this chapter use the same input sources and provide the same outputs. Specifically, both approaches use the ARB PT-Server [70] to construct a bipartite graph that matches signature candidates to organisms. ARB first generates all possible signatures of the specified length and then matches them (using a suffix trie) against the sequences of the organisms. The PT-Server supports approximate matching, for example, to compensate for sequencing errors in the database. The algorithms then process the resulting data stream and generate a map P, which contains a set of promising signature candidates for each $t \in T$.

The two approaches differ in the central algorithm, which searches and evaluates signature candidates. Given a phylogenetic tree and the bipartite graph, CaSSiS-BGRT uses a new data structure, the bipartite graph representation tree (BGRT), to process more than 460,000 gene sequences (660M nucleotides, matched without mismatches or outgroup hits) in about 132 hours on an Intel Core i7 with 24GB of system memory [12]. The algorithm employed by CaSSiS has worst-case complexity $O(|M| \cdot |V|)$ time, where M is the size of the BGRT (which is in turn bounded by $|U|$, the number of edges in the bipartite graph) and $|V|$ is the set of all organisms. CaSSiS-BGRT uses a time-memory trade-off to implement a bounded search to significantly

reduce the execution time in practice; however, as a result, CaSSiS-BGRT requires $O(d \cdot |M|)$ memory where d is the depth of the (phylogenetic) tree T.

While 132 hours may seem sufficient to find signatures for all sequences and sequence groups of interest, SSURef 102 only contains long (>900nt) annotated aligned SSU rRNA sequences and not full genomes. CaSSiS-BGRT cannot be expected to process contemporary data sets containing full genomes, as memory consumption is linear in the number of nucleotides and 700-million nucleotides already require about 16GB of RAM. For comparison, a human genome has 3.3-billion nucleotides, and ideally a signature search should to consider all available sequence data for all organisms. The CaSSiS-LCA algorithm presented in this chapter significantly outperforms CaSSiS-BGRT both in terms of memory and time complexity and is thus able to process full genomes.

4.3 The CaSSiS-LCA Algorithm

In this section, we present our new algorithm CaSSiS-LCA. We build up to the full-featured algorithm in three steps to introduce each of the key ideas separately and to properly highlight how the algorithm handles the different cases.

The bipartite graph $G = (U, V, E)$ has the key property that in practice we can expect there to be a relatively small number of organisms in V (hundreds of thousands) and many more signatures U (billions) and even more edges (Figure 4.2). Thus, it is impractical to load E (or even U) into main memory at any given time. Existing tools that generate signature candidates and match them against organisms can efficiently create E in the form of a data stream, giving all of the tuples $(u, v) \in E$ for a given $u \in U$ in a single contiguous block in the overall stream. The basic philosophy of our algorithm is thus to do stream processing [11] over a stream that represents the bipartite graph. Each round of the algorithm is given a $u \in U$ and the set $S_u \subseteq V$ of all organisms $v \in S_u$ that match signature candidate u. Our stream processing algorithm must then decide to keep u in a preliminary result set or discard u for good.

Prior to the main algorithms, we always perform some basic precomputations. First, we number the organisms sequentially in the tree T from left to right; thus, (without loss of generality) $v \in \mathbb{N}$. Second, we precompute the sparse tables necessary for computing lowest common ancestors in T. This precomputation can be done in $O(|V|)$ time [16, 37, 40]. Henceforth, we can compute the lowest common ancestor (LCA) of v and v' (denoted by $LCA(v, v')$) for $v, v' \in T$ in $O(1)$ time.

We will present the new algorithm in three consecutive steps. The simplest algorithm, presented in Section 4.3.1, is only considering perfect matches. A more relaxed algorithm that allows partial matches is given in Section 4.3.2. The actual algorithm that additionally allows outgroup matches follows in Section 4.3.3. A list of notations used in the three algorithms is shown in Table 4.1.

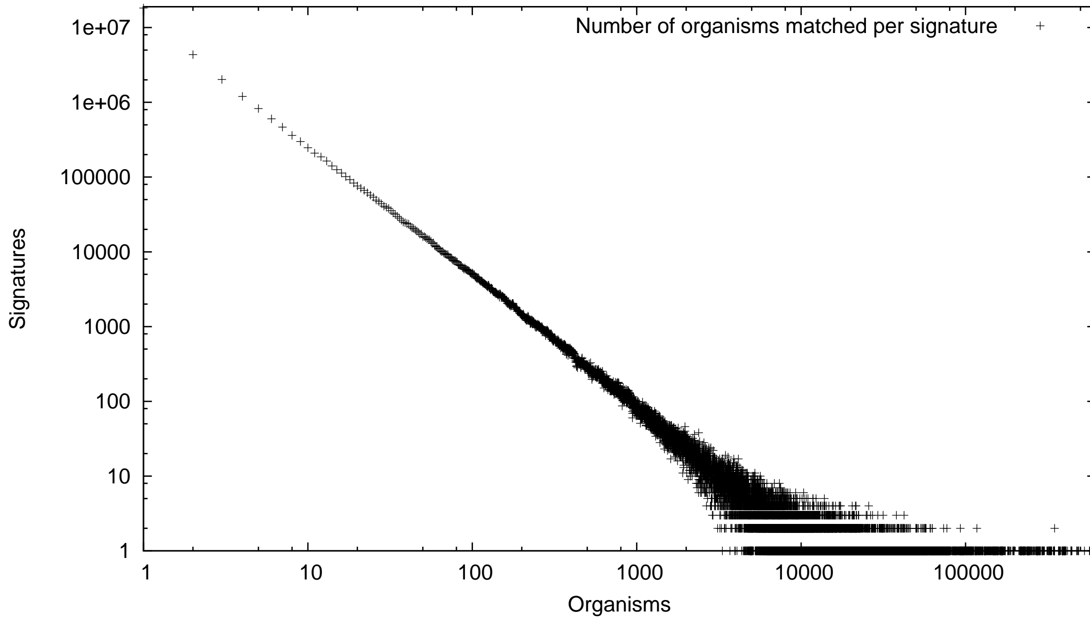


Figure 4.2: This figure shows the number of signatures that reference a certain number of organisms. It is based on the bipartite graph of the complete SSURef 108 dataset (618, 442 organisms, 31, 976, 771 signatures). Of the signatures, 57% match a single organism and only 11% match 10 or more organisms. Both axis use a logarithmic scale.

$C(t)$	Child nodes of node $t \in T$
$D(t)$	Set of all descendants of $t \in T$
E	Edges $(u \in U, v \in V)$ in the bipartite graph
G	Bipartite graph (U, V, E)
$\text{indexof}_{min}(e, S)$	Returns the index of element $e \in S$ in the array S, if $e \notin S$, the index e would have had if e was in S minus one is returned
$\text{indexof}_{max}(e, S)$	Returns the index of element $e \in S$ in the array S, if $e \notin S$, the index e would have had if e was in S is returned
$LCA(v, v')$	Lowest Common Ancestor of $v \in V$ and $v' \in V$
$P(t)$	Result set for tree node $t \in T$
$P(t, k)$	Result set for tree node $t \in T$ and $k \in \mathbb{N}$ outgroup hits
S_u	Set of sequences $S_u \subset V$ that are matched by a signature u
sort S	Returns a sorted array with the elements from S
T	(Phylogenetic) Tree
U	Signature set
V	Sequence set

Table 4.1: Notations that are used in the following three algorithms.

4.3.1 Perfect Match Algorithm

We begin our exposition with an algorithm for the simple case of finding a probe that provides perfect group coverage, that is, we are only interested in finding probes that match all sequences

in the target group and have no outgroup hits.

Figure 4.3 illustrates the two key cases the perfect match algorithm considers. First, the common case that the organisms matched by the signature do not correspond perfectly to any group. Second, the desired case that the signature corresponds exactly to a particular group, which LCA can identify in $O(1)$ time.

Algorithm: Perfect Matching
Input: $G(U, V, E)$, T, $LCA(v, v')$, $|D(t)|$
Output: $P : T \to \mathcal{P}(U)$

```
1  for u ∈ U do
2      P(u) ← ∅;
3  end
4  for u ∈ U do
5      S_u ← sort {v_u | (u, v_u) ∈ E};
6      v_u^min ← min S_u ;
7      v_u^max ← max S_u ;
8      û ← LCA(v_u^min, v_u^max);
9      if |S_u| = |D(û)| then
10         P(û) ← P(û) ∪ {u};
11     end
12 end
```

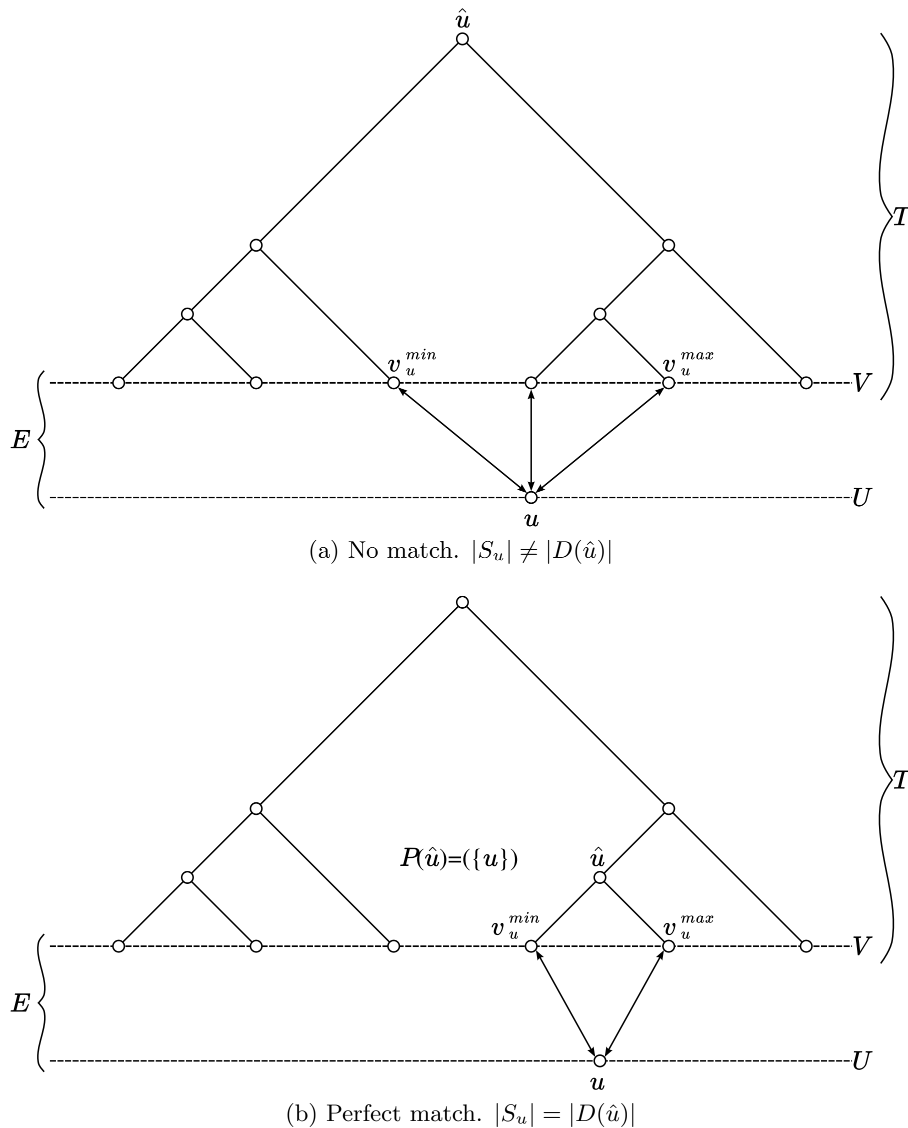
Algorithm 1: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to signature that perfectly cover all organisms in $D(t)$ (with no false-positives).

Algorithm 1 can then be used to find perfect matches for each organism or group of organisms (i.e. relation P). The key idea here is to use LCA to quickly determine the only $t \in T$ that might be matched perfectly by a given signature u, and then to use arithmetic to determine if u matches all descendants $D(t)$. For this algorithm, we need to precompute the number of descendants $|D(t)|$ for all $t \in T$ (which is trivial to do in $O(|V|)$ time). For determining the LCA $\hat{u}$, the minimum and maximum organism identifiers v_u^{min} and v_u^{max} are fetched and used. This can be done in $O(1)$ time in the sorted list S_u. Although sorting might be exaggerated in order to find the minimum and maximum identifiers, it was used for consistency to the third (implemented) algorithm where it becomes mandatory.

The complexity of the computation given in Algorithm 1 is $O(|U| + |E|)$ ($|E|$ from sorting arrays of integers with a total of $|E|$ entries in linear time, $|U|$ from processing each signature u). Together with the precomputation, the overall complexity of this first algorithm is thus $O(|V| + |E| + |U|)$ time.

4.3.2 Partial Group Coverage

The second version of the algorithm will now relax the constraint that the group coverage must be perfect. Instead, we will permit that some organisms in the group are not covered by the signature. The goal of the algorithm is to find those signatures that provide the maximum group



(a) No match. $|S_u| \neq |D(\hat{u})|$

(b) Perfect match. $|S_u| = |D(\hat{u})|$

Figure 4.3: Illustration of Algorithm 1. In 4.3a, the set size $|S_u| = 3$ is smaller than $|D(\hat{u})| = 6$, so the candidate u is not a perfect probe for any sequence or sequence group; 4.3b shows a probe that would be a perfect match ($|S_u| = 2 = |D(\hat{u})|$).

coverage (minimizing false-negatives) while not allowing any outgroup hits (no false-positives).

Let $r \in T$ be the root of the tree T and let $C(t) \subset T$ denote the children of $t \in T$. Algorithm 2 can then be used to find those signatures that maximize group coverage (with no outgroup hits) in $O(|U| + |E|)$ time. The result is stored in a map P which maps each $t \in T$ to a pair consisting of the set of signatures $\mathcal{U} \subseteq U$ and the number of organisms matched in the target group by all $u \in \mathcal{U}$.

A key step in Algorithm 2 is the propagation of results up the tree in Procedure PropagateUp. This step exploits the fact that the parent p' of a node $p \in T$ always represents a superset of organisms, and thus a probe that has no outgroup hits for p will also have no outgroup hits

Algorithm: Partial Group Coverage
Input: $G(U, V, E)$, T, $LCA(v, v')$
Output: $P : T \to (\mathcal{P}(U), \mathbb{N})$

1 **for** $u \in U$ **do**
2     $P(u) = (\emptyset, 0)$;
3 **end**
4 **for** $u \in U$ **do**
5     $S_u \leftarrow$ sort $\{v_u | (u, v_u) \in E\}$;
6     $v_u^{min} \leftarrow$ min S_u ;
7     $v_u^{max} \leftarrow$ max S_u ;
8     $\hat{u} \leftarrow LCA(v_u^{min}, v_u^{max})$;
9     $(\mathcal{U}, n') \leftarrow P(\hat{u})$;
10     $n \leftarrow |S_u|$;
11   **if** $n > n'$ **then**
12         $P(\hat{u}) \leftarrow (\{u\}, n)$;
13   **end**
14   **if** $n = n'$ **then**
15         $P(\hat{u}) \leftarrow (\mathcal{U} \cup \{u\}, n')$;
16   **end**
17 **end**
18 PropagateUp (T, r, P);

Algorithm 2: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to the set of signatures that provide maximum coverage of the organisms in $D(t)$ (with no false-positives). The algorithm runs in $O(|U| + |V| + |E|)$ time, as we can use bucket sort to sort in $O(|E|)$ time and Procedure PropagateUp adds $O(|V|)$ time.

for p'. A suitable probe candidate for p can therefore also be a good candidate for p'. Thus, Procedure PropagateUp is needed to ensure that, probes that provide partial group coverage are found. Note that for partial group coverage, the computation of $|D(t)|$ is no longer required.

Figure 4.4 illustrates the key steps in Algorithm 2. Given a signature u, the algorithm first determines v_u^{min} and v_u^{max}, then determines $\hat{u}$ using LCA, associates the signature u with $|S| = 2$ ingroup hits with $P(\hat{u})$, and finally (assuming there were no other, better signatures found in U) propagates the signature u to the ancestors of $\hat{u}$.

4.3.3 Allowing at most k Outgroup Hits

Finally, we present the complete CaSSiS-LCA algorithm that computes for each $t \in T$ those signatures $u \in U$ that maximize the number of matched target organisms $v \in D(t)$, while not matching more than k organisms $v' \notin D(T)$. More precisely, our algorithm computes for each $i \in \{0, \ldots, k\}$ and each $t \in T$ the set of signatures $u \in U$ that have exactly i outgroup hits and the maximum number of ingroup hits.

Before the main algorithm, we precompute for each $t \in T$ its leftmost and rightmost leaf in the subtree rooted at t, which we will refer to as the border $B(t) = (v_t^{min}, v_t^{max})$ (where $v_t^{min} := min(D(t))$ and $v_t^{max} := max(D(t))$). It is trivial to do this precomputation in $O(|V|)$

Input: T, p, $P : T \to (\mathcal{P}(U), \mathbb{N})$
Output: Updated $P : T \to (\mathcal{P}(U), \mathbb{N})$

```
1  foreach c ∈ C(p) do
2      PropagateUp (T, c, P);
3      if p ≠ r then
4          p' ← parent (p);
5          (𝒰, n) ← P(p);
6          (𝒰', n') ← P(p');
7          if n > n' then
8              P(p') ← (𝒰, n);
9          end
10         if n = n' then
11             P(p') ← (𝒰 ∪ 𝒰', n');
12         end
13     end
14 end
```

Procedure PropagateUp(T, p, P): Helper function to propagate good matches up the tree using depth first traversal (in $O(|V|)$ time).

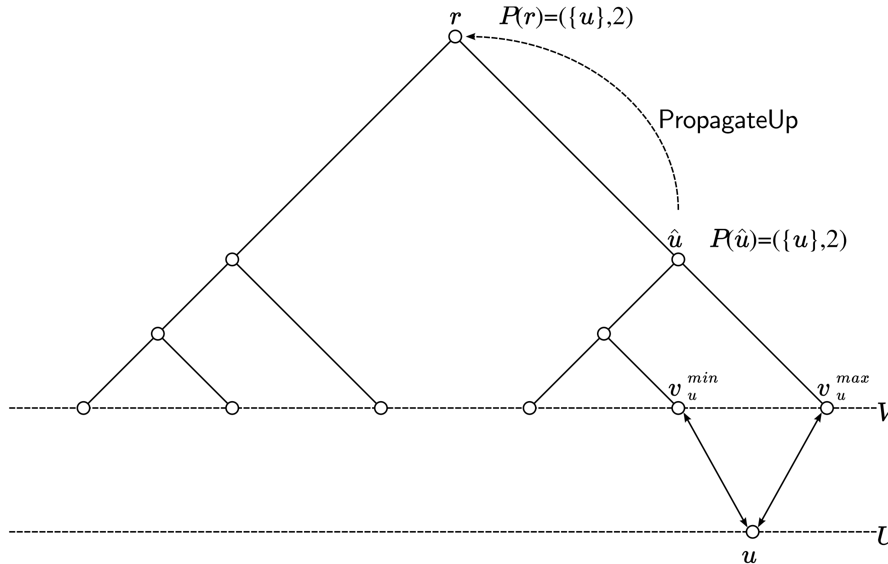


Figure 4.4: Illustration of Algorithm 2. The coverage of the probe u is $|S_u| = 2$. It is added at the node $\hat{u}$. Note that the PropagateUp step happens once (for each $t \in T$) at the end of the algorithm and not after each $u \in U$.

time.

Given this, Algorithm 3 computes P, a mapping from pairs $(T, \{0, \ldots, k\})$ (representing a target organism or target group of organisms and a number of outgroup hits) to a pair $(\mathcal{P}(U), \mathbb{N})$ consisting of a set of signatures and the number of organisms matched by those signatures in the target group. The Procedure PropagateUpWithK corresponds to the procedure PropagateUp,

Algorithm: Partial Group Coverage with Outgroup Hits
Input: $G(U, V, E)$, T, $LCA(v, v')$, k
Output: $P : (T, \{0, \ldots, k\}) \rightarrow (\mathcal{P}(U), \mathbb{N})$

```
1  for u ∈ U do
2      for o ∈ {0, ..., k} do
3          P(u, o) = (∅, 0);
4      end
5  end
6  for u ∈ U do
7      S_u ← sort {v_u | (u, v_u) ∈ E};
8      v_u^min ← min S_u ;
9      v_u^max ← max S_u ;
10     û ← LCA(v_u^min, v_u^max);
11     (𝒰, n') ← P(û, 0);
12     n ← |S_u|;
13     if n > n' then
14         P(û, 0) ← ({u}, n);
15     end
16     if n = n' then
17         P(û, 0) ← (𝒰 ∪ {u}, n');
18     end
19     PropagateDown (T, û, P, S_u, k, u) ;
20 end
21 PropagateUpWithK (T, r, P, k);
```

Algorithm 3: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to the set of signatures that provide maximum coverage of the organisms in $D(t)$ (with no false-positives). The algorithm runs in $O(k|U| \log |V| + k|V| + |E|)$ time. The algorithm requires $O(|V|)$ space for the LCA data structure and can process the bipartite graph $(u, v_u) \in E$ in a streaming fashion; thus, there is no need to store the entire bipartite graph in memory. There are $O(k|V|)$ result entries in the output P. As written, the union operation in line 17 creates the theoretical possibility of $O(k \cdot |V| \cdot |U|)$ space (if every signature is a perfect signature for some k and some sequence). If only a best match (instead of all best matches) is desired, one can replace $(\mathcal{U} \cup \{\sqcap\})$ with $\{u\}$ to improve space consumption to $O(k \cdot |V|)$. In either case, the algorithm requires space linear to the size of the output.

differing only in propagating $k + 1$ matches up.

Algorithm 3 requires another helper Procedure PropagateDown, which propagates signatures down the tree T. At the LCA node, a signature will only have ingroup matches and it has the highest sensitivity, but the signature could also be valuable candidate for its children, which in their case results in outgroup matches. Propagating signatures downwards requires recalculating the number of ingroup and outgroup hits at each step. Given that at most k outgroup hits are allowed, the propagation stops after a total of at most $O(k)$ downward steps.[3]

At each step, the procedure uses the border B and a binary search to quickly determine

[3] Without loss of generality, we can assume that T is a binary tree; thus, $|C(p)| \leq 2$ can be assumed and the first iteration over $|C(p)|$ children of p is also in $O(k)$ time.

```
    Input: T, p, P : T → (P(U), N), k
    Output: Updated P : T → (P(U), N)
 1  foreach c ∈ C(p) do
 2      foreach o ∈ {0, ..., k} do
 3          PropagateUpWithK (T, c, P, k);
 4          if p ≠ r then
 5              p′ ← parent (p);
 6              (U, n) ← P(p, o);
 7              (U′, n′) ← P(p′, o);
 8              if n > n′ then
 9                  P(p′, o) ← (U, n);
10              end
11              if n = n′ then
12                  P(p′, o) ← (U ∪ U′, n′);
13              end
14          end
15      end
16  end
```

Procedure PropagateUpWithK(T, p, P, k) Helper function to propagate good matches up the tree using depth-first traversal (in $O(k|V|)$ time). This procedure closely corresponds to Procedure PropagateUp, except that we need to propagate the best signatures up $k+1$ times.

the number of outgroup and ingroup hits for the new target group. Let $\text{indexof}_{min}(e, S)$ be a function that returns the index of element $e \in S$ in the zero-indexed, sorted array S. If $e \notin S$, the index of the element left of the position e would have had in S should be returned (we never use $\text{indexof}_{min}(e, S)$ on elements e that have no smaller element in S). Similarly, let $\text{indexof}_{max}(e, S)$ be a function that returns the index of element $e \in S$ in the zero-indexed, sorted array S, and if $e \notin S$ returns the index of the element right of the position, e would have had in S (again, we never use $\text{indexof}_{max}(e, S)$ on elements e that have no larger element in S). Figure 4.6 illustrates how indexof is used to quickly determine the number of outgroup hits o. As S is sorted, both indexof computations can be done in $O(\log |S|)$ time using binary search. Procedure PropagateDown then lists the steps necessary to propagate signatures down the tree.

Figure 4.5 illustrates the downward propagation for $k = 1$. Going toward the left, the propagation immediately terminates as $o = 2 > k$. Propagating toward x, the propagation first updates the result set for $o = 1$ outgroup hit and then terminates on the next level as the number of outgroup hits again rises to $o = 2 > k$.

4.4 Implementation and Results

All experiments were performed on an Intel Core i7-920 (2.67GHz) Debian GNU/Linux system with 16GB of main memory. We evaluated the performance of our algorithm using the sequence

Input: T, p, $P : T \to (\mathcal{P}(U), \mathbb{N})$, S, k, u, indexof(e, S)
Output: Updated $P : T \to (\mathcal{P}(U), \mathbb{N})$

```
1  foreach c ∈ C(p) do
2      (v_c^min, v_c^max) ← B(c);
3      i_c^min ← indexof_min(v_c^min, S) ;
4      i_c^max ← indexof_max(v_c^max, S) ;
5      o ← i_c^min + |S| − i_c^max − 1;
6      n ← |S| − o;
7      if o ≤ k then
8          (U′, n′) ← P(c, o);
9          if n > n′ then
10             P(c, o) ← ({u}, n);
11         end
12         if n = n′ then
13             P(c, o) ← (U′ ∪ {u}, n′);
14         end
15         PropagateDown (T, c, P, S, k, u);
16     end
17 end
```

Procedure PropagateDown(T, p, P, S, k, u) Helper function to propagate good matches down the tree, bounded by k. The recursion is bounded to at most $O(k)$ calls, thus the complexity of this procedure is $O(k \log |S|)$ time and thus $O(k \log |V|)$ as $S \subseteq V$.

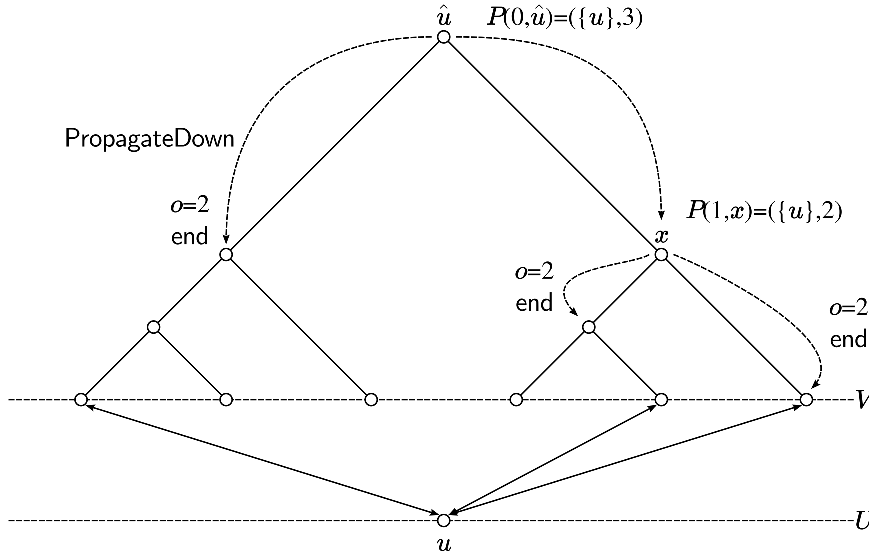


Figure 4.5: Illustration of Algorithm 3. For the example, we use an outgroup limit of $k := 1$. At $\hat{u}$ (and its parent nodes) no outgroup matches are possible, i.e. $P(0, \hat{u}) \leftarrow (\mathcal{U}, 3)$. The match is then propagated towards the child nodes. At node x, the match is added with 1 outgroup match, i.e. $P(1, x) \leftarrow (\mathcal{U}, 2)$. In the other child nodes, the outgroup limit k is exceeded.

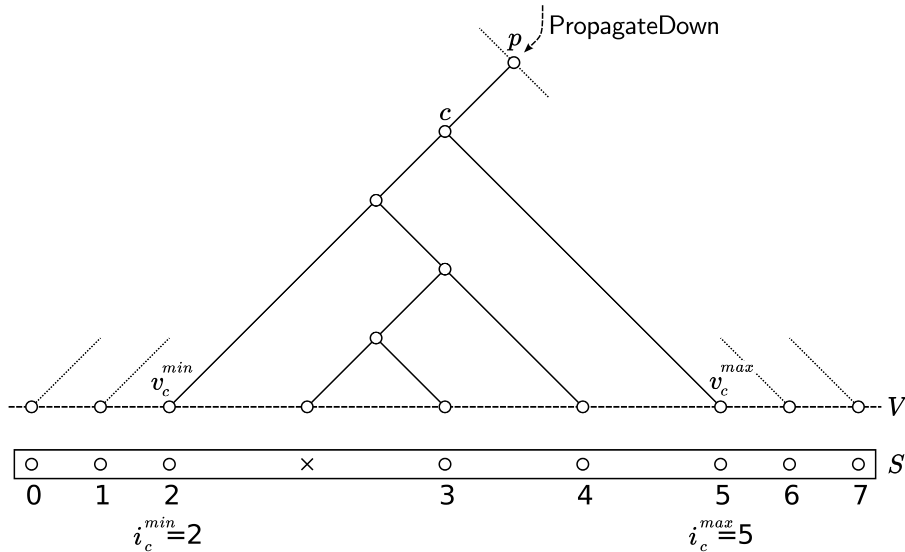


Figure 4.6: Detailed illustration of one iteration in the foreach loop of the PropagateDown procedure. The left- and rightmost descendants (v_c^{min} and v_c^{max}) of the node c are read. Then, the indexof procedures are used to fetch their positions in the match set S: $i_c^{min} = 2$ and $i_c^{max} = 5$. The number of outgroup matches at c is $o = i_c^{min} + |S| - i_c^{max} - 1 = 2 + 8 - 5 - 1 = 4$, the number of matches is, therefore, $n = |S| - o = 4$

data and phylogenetic tree from the SILVA SSURef 108 dataset[4]. It should be noted that the output of the new algorithm is identical to the output from CaSSiS-BGRT (Chapter 3.3.6).

The inputs to our implementation are a (binary) phylogenetic tree in the Newick format [81] and MultiFasta[5] formatted 16S rRNA gene sequence datasets. Each sequence represents an organism. For our experiments, we used a modified ARB PT-Server to generate the bipartite graph that maps signature candidates to organisms. The modifications allowed direct access to the result sets without double parsing (once in the PT-Server and again in CaSSiS). Also, the memory management was adapted to reduce the memory consumption.

The PT-Server allows the definition of a Hamming distance when matching a signature against the sequences. Our implementation utilizes this to allow a certain Hamming distance m_1 for matches within the target group as well as enforcing a minimum Hamming distance $m_2 > m_1$ to sequences outside of the target group. The latter is implemented by adding the number of organisms with a distance between m_1 and m_2 to the initial number of outgroup hits for the probe. This strategy was also used and discussed in [12]. In both approaches, using the same Hamming distance values result in identical bipartite graphs.

For the precomputation, we used the canonical approach of reducing the LCA problem into a range minimum query (RMQ) problem [40]. We used the *Sparse Table (ST)* algorithm described by Bender and Farach-Colton [16] to preprocess with a complexity of $O(n \log n)$ time and to achieve $O(1)$ time for the RMQ queries during the main phase of the algorithm. While solutions

[4] `http://www.arb-silva.de/documentation/background/release-108/`

[5] `http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml`

for preprocessing the LCA look-up with linear runtime and memory complexity exist [16, 37, 38], we did not implement those as the $O(n \log n)$ processing is not the bottleneck (costing less than 1% of the total execution time) and the constant factors for the main phase of the algorithm would be higher for those other schemes. However, for the complexity analysis, we assume that the best-known linear algorithm for LCA could be used and thus the LCA precomputation will take $O(|V|)$ time and space.

We created test datasets of increasing sizes by randomly selecting sequences from the original SILVA SSURef 108 dataset. We reduced the phylogenetic trees to only include leaves which reference the selected sequences. The test datasets range from $16,000$ to $512,000$ sequences (Table 4.2, upper part).

To evaluate the processing of genome sequences with the CaSSiS-LCA approach, Wolfgang Ludwig provided us an unpublished dataset consisting of procaryotic (bacterial) genome sequences of varying completeness. Additionally, he provided a phylogenetic tree referencing these sequences at its leaves and containing group definitions at its inner nodes. We extracted three test datasets and corresponding trees with 100, 200 and 400 complete genome sequences (Table 4.2, lower part).

Sequences $\|V\|$	**Bigraph Edges $\|E\|$**	**Signatures $\|U\|$**	**Nucleotides**
16,000	22,675,424	3,035,608	22,961,088
32,000	45,332,247	4,654,334	45,882,367
64,000	90,712,819	7,110,160	91,766,991
128,000	181,546,054	10,767,681	183,567,793
256,000	363,195,618	16,219,346	367,064,051
512,000	726,690,069	24,125,196	734,101,089
(*) 618,442	882,460,379	31,976,771	891,481,250
100 genomes	340,765,921	302,033,422	348,881,663
200 genomes	663,933,797	573,445,204	680,490,674
400 genomes	1,328,208,091	1,039,549,521	1,360,775,974

Table 4.2: The upper part of the table contains test datasets derived from the SSURef 108 (*) dataset. Sequences represent rRNA genes of organisms. For each dataset, the total number of nucleotides and unique exact-matching 18 mer signatures are shown. The bigraph edges represent matches between 18 mer signatures and sequences. The lower part of the table contains the statistical information for the test datasets consisting of 100, 200 and 400 complete bacterial genome sequences.

As expected, the main phase of the algorithm spends most of its time performing the downward propagation. However, execution time is dominated by far by the queries to the ARB PT-Server (Figure 4.7), especially for larger Hamming distance values (m_1 and m_2; Table 4.3). It should be noted that it would be trivial to run multiple instances of the ARB PT-Server in a cluster [11] (memory per node permitting). Distributing the queries across multiple search indices and aggregating the results afterwards would further decrease the runtime of our approach.

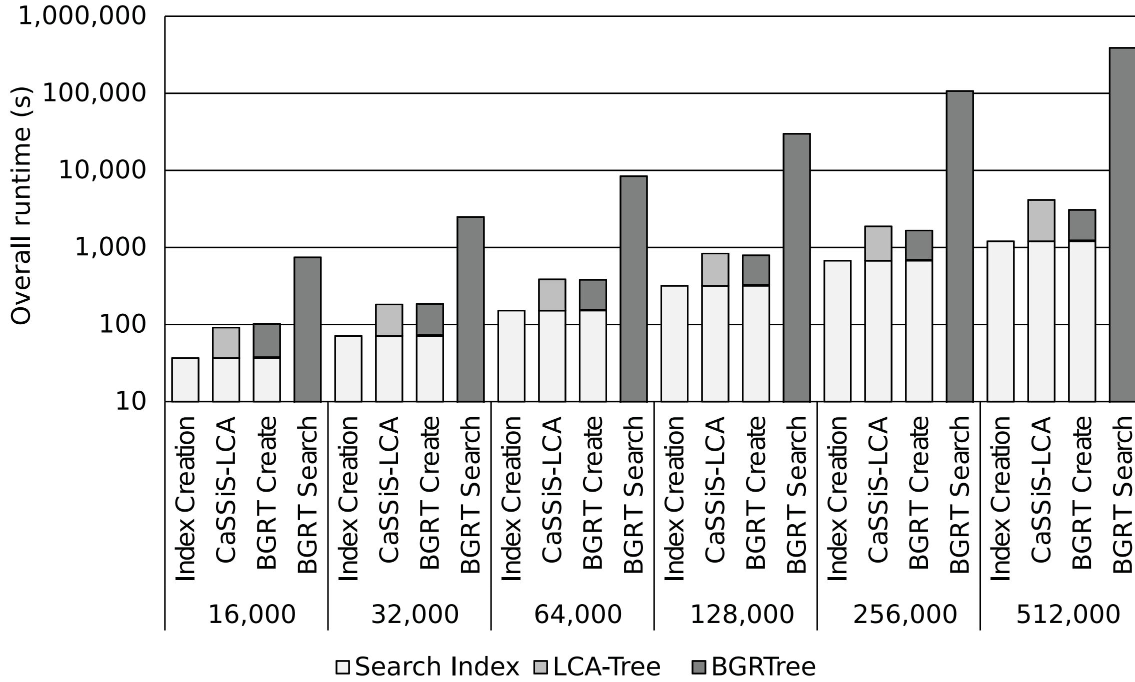


Figure 4.7: Overall runtime of the CaSSiS-BGRT and CaSSiS-LCA implementations for growing dataset sizes (16,000 to 512,000 sequences) with $m_1 = 0$, $m_2 = 1$ and $k = 10$. CaSSiS-BGRT was split in its two computing stages BGRT Create and BGRT Search. The creation of the BGRT and the CaSSiS-LCA approach runtimes include building the search index. The measured runtimes grow linear with the dataset sizes.

Distances m_1	m_2	**Runtime** (seconds)	**Bigraph Edges** $\|E\|$	**Signatures** $\|U\|$
0	1	54	22,675,424	3,035,608
0	2	188	3,669,142	1,932,039
0	3	1,167	1,746,651	1,113,679
0	4	5,292	752,918	534,838
1	2	259	369,546,279	3,035,608
1	3	1,233	4,399,485	1,229,004
1	4	5,378	1,315,221	566,610
2	3	1,488	1,386,742,230	3,035,608
2	4	5,628	3,402,659	646,788
3	4	6,203	3,349,697,538	3,035,608

Table 4.3: Allowing mismatches within the target group by increasing the Hamming distance m_1 increases the number of edges $|E|$ in the bipartite graph. The number of signatures $|U|$ decreases with growing Hamming distances m_2 to nontargets. The number of sequences, here $|V| = 16,000$, stays constant. The total runtime is mainly influenced by the time needed to process the search index queries and, therefore, increases with growing mismatch distances (m_2).

Both algorithms, CaSSiS-BGRT and CaSSiS-LCA, begin with the computation of the search index. Afterward, the search index is loaded into memory for further processing. Up to this point, runtime and memory consumption for both algorithms are identical. Afterward, either a BGRT is created and in a later step processed (CaSSiS-BGRT), or the results are directly inserted at the appropriate node in the phylogenetic tree (CaSSiS-LCA). Figure 4.8 illustrates the growth in memory requirement of all major steps for different input sizes. With growing dataset sizes, the signature search based on the BGRT becomes the most memory consuming step.

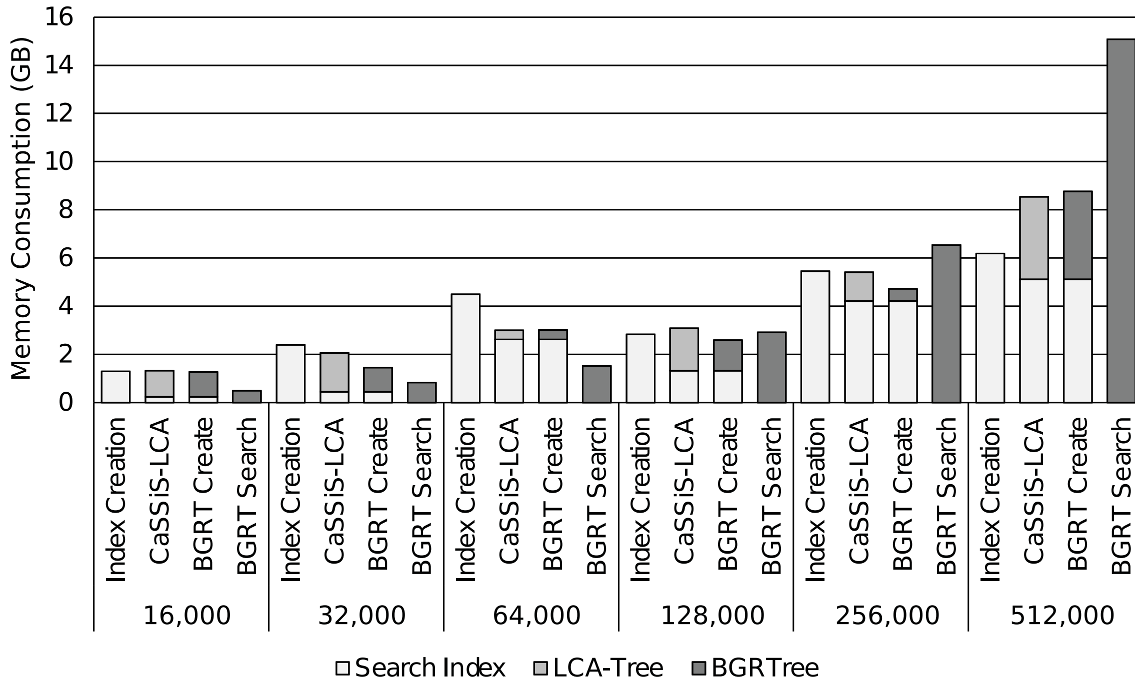


Figure 4.8: Comparison of the memory consumption of CaSSiS-BGRT and CaSSiS-LCA for growing dataset sizes (16,000 to 512,000 sequences) computed with $m_1 = 0$, $m_2 = 1$, and $k = 10$. Results for the complete SSURef 108 dataset are not shown as the BGRT search step exceeded the available main memory on our test system (20.1GB; the CaSSiS-LCA implementation only required 10.8GB). Note that the CaSSiS-BGRT approach was split into its two main steps, BGRT Create and BGRT Search. The memory consumption of the search index after its computation is identical for CaSSiS-LCA and BGRT Create. For the BGRT search/traversal, the search index is not needed anymore. Traversing the BGRT consumes far more memory than the LCA approach, although the same phylogenetic tree structure is used to store the results (in the nodes).

We used the largest test dataset to provide a detailed comparison of the two approaches (Figure 4.9). Due to the identical search index computation and result processing in both implementations, the most significant difference is the runtime of the search for promising signatures at the BGRT- and the LCA-Tree-traversal steps.

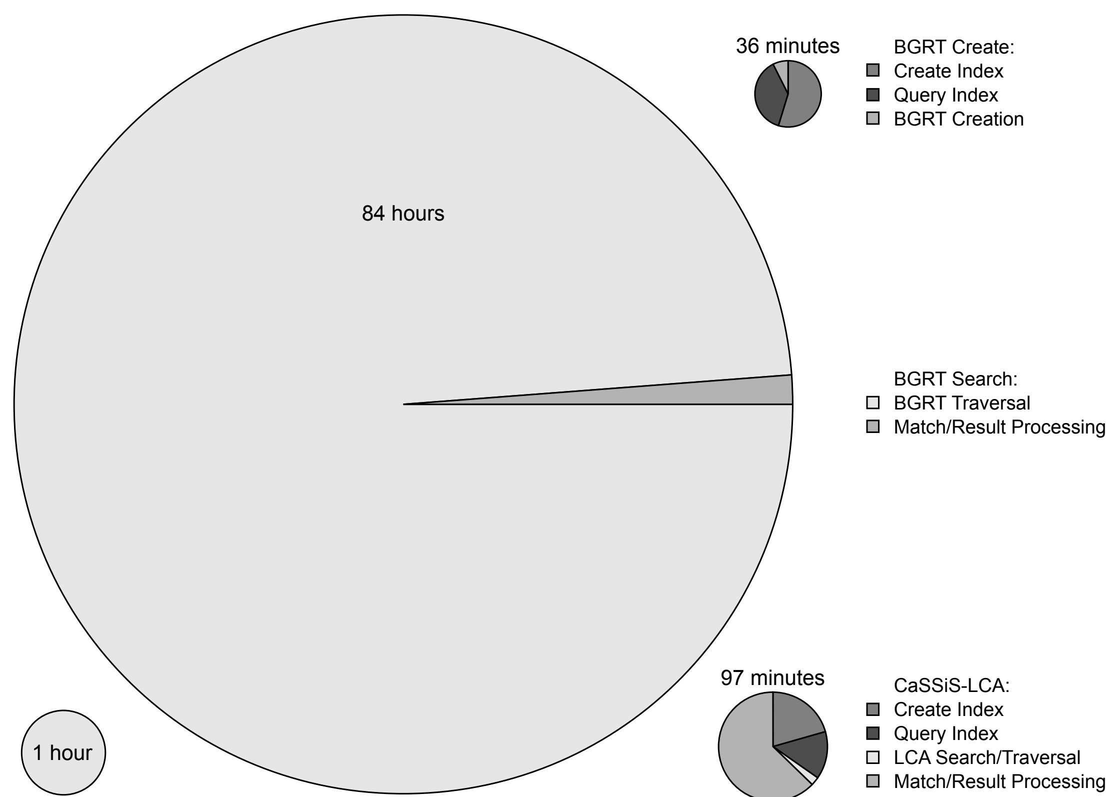


Figure 4.9: Detailed comparison of the runtime of CaSSiS-BGRT and CaSSiS-LCA algorithm for the dataset with 512, 000 sequences. CaSSiS-BGRT was split in its two computing stages BGRT Create and BGRT Search. Steps with a runtime below one minute (e.g. loading the BGRT) are not shown. The runtime is represented by area. The most notable difference is the reduction of the 83 hours of the BGRT Traversal step to 148 seconds during LCA Search/Traversal".

4.5 Discussion

When compared to the CaSSiS-BGRT approach from the previous Chapter 3, CaSSiS-LCA offers a vastly reduced memory footprint and a significantly faster runtime. Both approaches share the same advantages and disadvantages when it comes to the selection of signature candidates based on relaxed search conditions and the application of filters. These were already discussed in Section 3.5. A major difference from a user perspective is that CaSSiS-LCA does not rely on a BGRT lookup structure. During computation, signature candidates are directly added to the respective nodes in the phylogenetic tree. As a result, it only allows the comprehensive computation and is not capable of processing single freely defined queries — an advantage of a the BGRT structure, which can be stored as a file and loaded when needed. CaSSiS-LCA is therefore directed primarily to the maintainers of hierarchically-clustered databases, such as SILVA, RDP, or Greengenes [89, 25, 29]. However, it should be pointed out that both approaches lead to the same results when used with the same configuration.

With CaSSiS-LCA, we were not only able to process gene datasets but also ones containing complete bacterial genomes.

In all our tests, one single identifier was used per sequence, i.e. the terms “sequence” and “organism” were used interchangeably. Also applying this definition when processing complete genomes is possible, but might lead to several disadvantages. Compared to gene datasets, the longer genome sequences obviously increase the chance of matches. From a computer science perspective they are considered equally legitimate. And of course all matches also need to be considered when a resulting primer and probe is tested in the wet lab. But sometimes it is desirable to have matches within a certain range on the sequence, e.g. a certain gene. By using per-sequence identifiers, promising signatures that match at similar gene locations within a target group might get rejected due to outgroup matches at completely different sequence locations (e.g. different genes). And vice versa, the match positions of a signature with a high target group coverage might be completely scattered over the complete genomes.

A biological problem that is currently not covered when processing genomic datasets is the possibility of multiple occurrences of functionally identical genes at different sequence positions. CaSSiS only stores relations between sequences and signatures, e.g. “a signature X matches a sequence A”. The information at which locations matches occur are not stored.

A solution to both mentioned problems could be an extension to the identifier scheme used throughout CaSSiS: “Composite Primary Keys” (also called “Compound Keys”) could be used. These keys could consist of 2 or more sequence attributes, i.e. names or accession numbers[6]. When querying for signatures for a target group, adding a specific gene identifier would narrow down the region where the signature may lie. Matches on different locations would count to the outgroup, thus allowing a more fine-grained search. Such a search can already be achieved through a modification of the identifiers that are fed to CaSSiS in the input sequence datasets. Additionally, the group definition must as well be adapted to contain the specific sub-identifiers, e.g. a certain gene for a defined group of genomes. But as future work, this feature could be added to CaSSiS itself.

4.6 Conclusion

With CaSSiS-LCA, we have presented a runtime and memory efficient solution for the the *in silico* search for promising signature candidates even under relaxed search conditions. Combined with modern algorithms for genome-scale pattern matching this approach will allow the computation of specific and sensitive signatures.

Our tests with genome datasets have shown that the selection and configuration of the used search index is very important in order to even be able to process such large datasets. The search indices' memory consumption is added to the consumption of CaSSiS during runtime — and both should fit into the main memory. The CaSSiS approaches are based on the ARB PT-Server or rather its descendant MiniPT. Its memory requirement grows linearly with the number of

[6] Accession numbers are unique identifiers that are used to refer to single nucleic acid sequences, proteins, and even complete genomes.

nucleotides it is indexing (Figure 3.9), which is already quite effectively. But a future step could be switching to an even more efficient search index for pattern matching, such as PTPan [33] or SeqAn [30]. PTPan stores its index on secondary storage (e.g. a hard drive). It also supports Levenshtein distances (the PT-Server currently only supports Hamming distances), which might lead to better results during relaxed searches. SeqAn provides suffix array search structures, which are considered more memory efficient than suffix tree structures [1]. Both approaches are slower in answering queries when compared to the ARB PT-Server [33], a factor which should at least be considered.

Another approach is partitioning the search index in a cluster. By this, we predict that CaSSiS-LCA should be able to process genome data of virtually arbitrary size. It would mostly be limited by the size of the clustering that is used to store the resulting signature candidates. The following Chapter 5 presents first promising results.

Chapter 5

Distributed Signature Matching

Finding optimal oligonucleotide signatures that are both specific and sensitive for organisms or groups of organisms in large-scale sequence datasets is a very computationally expensive task. In the last two Chapters 3 and 4 the approaches CaSSiS-BGRT and CaSSiS-LCA were presented, that tackle this problem from the algorithmic side. We have accelerated the signature gathering and evaluation step and largely reduced its memory consumption.

Looking at the entire CaSSiS implementation, there still remains a barely altered component: the search index. This index is necessary to compute the mapping between the signatures and matched sequences. It has huge influence on the overall runtime and, depending on the search settings, may also have a very high memory consumption.

We present in this chapter our experiences with parallelizing and distributing the ARB PT-Server, the search index for RNA/DNA sequence databases that was used in this work. The case study shows that it is possible to rapidly parallelize and distribute an existing complex legacy bioinformatics application and obtain significant speedups.

For this purpose we have used the DUP System, a simple framework for parallel stream processing. The DUP System enables developers to compose applications from stages written in almost any programming language and to run distributed streaming applications across all POSIX-compatible platforms. Parallel applications written with the DUP System do not suffer from many of the problems that exist in traditional parallel languages. The DUP System includes a range of simple stages that serve as general-purpose building blocks for larger applications.

This Chapter is an extended version of a LNCS "Network and Parallel Computing" article [11]. It was joint work with Tilo Eißler, Nathan Evans, Prof. Chris GauthierDickey, Christian Grothoff, Krista Grothoff, Jeff Keene, Harald Meier, Craig Ritzdorf, and Prof. Matthew J. Rutherford.

5.1 Introduction

Exact and inexact string searching in gene sequence databases plays a central role in molecular biology and bioinformatics. Many applications require string searches, such as searching for gene

sequence relatives and mining for PCR-primers or DNA-probes in DNA sequences [80, 65, 53]; both of these applications are important in the process of developing molecular diagnostic assays for pathogenic bacteria or viruses based upon specific DNA amplification and detection.

In the ARB software package, a suffix-tree–based search index, called the PT-Server, is the central data structure used by applications for fast sequence string matching [70]. A PT-Server instance is built once from the sequence entries of a gene sequence database of interest and is stored permanently on disk.

In order to perform efficient searches, the PT-Server is loaded into main memory in its entirety. Its memory requirement grows linearly with the number of nucleotides it is indexing (Figure 3.9). If the entire index structure cannot fit into the available main memory (the PT-Server requires $\sim$ 26 bytes of memory per sequence base at peak; Section 3.3.1), the structure cannot be efficiently searched.

In addition to memory consumption, the runtime performance of the search can be quite computationally intensive. An individual *exact* string search — in practice, short sequence strings of length 15–25 base pairs are searched for — is quick (3–15 milliseconds). However, the execution time can become significant when millions of *approximate* searches are performed during certain bioinformatic analyses, such as probe design.

In the near future, the number of published DNA sequences will explode due to the availability of new high-throughput sequencing technology [103]. As a result, current sequential analysis methods will be unable to process the available data within reasonable amounts of time. Furthermore, rewriting more than half-a-million lines of legacy C and C++ code of the high-performance ARB software package is prohibitively expensive. The goal of this case study was to see how readily the existing ARB PT-Server could be distributed and parallelized. Specifically, we were interested in parallelization in order to reduce execution time and in distribution in order to reduce per-system memory consumption.

We have used the DUP System[1], a language system which facilitates productive parallel programming for stream processing on POSIX platforms. It is introduced in Section 5.2. It is not the goal of the DUP System to provide ultimate performance; we are instead willing to sacrifice *some* performance gain for significant benefits in terms of programmer productivity. By providing useful and intuitive abstractions, the DUP System enables programmers without experience in parallel programming or networking to develop correct parallel and distributed applications and obtain speedups from parallelization.

The key idea behind the DUP System is the multi-stream pipeline programming paradigm and the separation of multi-stream pipeline specification and execution from the language(s) used for the main computation. Multi-stream pipelines are a generalization of UNIX pipelines. However, unlike UNIX pipelines, which are composed of processes which read from at most one input stream and write to a single output stream (and possibly an error stream), multi-stream pipelines are composed of processes that can read from any number of input streams and write to any number of output streams. In the remainder of this document, we will use the term "stage" for individual processes in a multi-stream pipeline. Note that UNIX users — even those with

[1] The DUP System is available at `http://dupsystem.org/`

only rudimentary programming experience — can usually write correct UNIX pipelines which are actually parallel programs. By generalizing UNIX pipelines to multi-stream pipelines, we eliminate the main restriction of the UNIX pipeline paradigm — namely, the inherently linear data flow.

In order to support the developer in the use of multi-stream pipelines, the DUP System includes a simple coordination language which, similar to syntactic constructs in the UNIX shell, allows the user to specify how various stages should be connected with streams. The DUP runtime then sets up the streams and starts the various stages. Key benefits of the DUP System include:

1. Stages in a multi-stream pipeline can run in parallel and on different cores;
2. Stages can be implemented, compiled and tested individually using an appropriate language and compiler for the given problem and architecture;
3. Stages only communicate using streams; streams are a great match for networking applications and for modern processors doing sequential work;
4. If communication between stages is limited to streams, there is no possibility of data races and other issues that plague developers of parallel systems;
5. While the DUP System supports arbitrary data-flow graphs, the possibility of deadlocks can be eliminated by only using acyclic data-flow graphs;
6. Applications built using multi-stream pipelines can themselves be composed into a larger multi-stream pipeline, making it easy for programmers to express hierarchical parallelism

In Section 5.3.2 we present experimental results from a case study involving the ARB PT-Server in combination with the DUP System. The case study shows that it is possible to rapidly parallelize and distribute an existing complex legacy bioinformatics application and obtain significant speedups using DUP.

5.2 The DUP System

The fundamental goal of multi-stream pipelines is to allow processes to read from multiple input streams and write to multiple output streams, all of which may be connected to produce the desired data-flow graph. This generalization of linear UNIX pipelines can be implemented using traditional UNIX APIs,[2] especially the `dup2` system call. Where a typical UNIX shell command invocation only connects `stdin`, `stdout` and `stderr`, the DUP System establishes additional I/O streams before starting a stage. Using this method, traditional UNIX filters (such as `grep`) can be used as stages in the DUP System without modification. New stages can be implemented in any language environment that supports POSIX-like input-output operations (specifically,

[2]The APIs needed are supported by all platforms conforming to the POSIX standard, including BSD, GNU/Linux, OS X, and z/OS.

reading and writing to a file). Since `dup2` also works with TCP sockets, the DUP System furthermore generalizes multi-stream pipelines to distributed multi-stream pipelines.

5.2.1 Related Work

The closest work to the DUP System presented in this chapter are multi-stream pipelines in CMS [49]. CMS multi-stream pipelines provide a simple mini-language for the specification of virtually arbitrary data-flow graphs connecting stages from a large set of pre-defined tools or arbitrary user-supplied applications. The main difference between CMS and the DUP System (which uses parallel execution of stages) is that CMS pipelines are exclusively record-oriented and implemented through co-routines using deterministic and non-preemptive scheduling with zero-copy data transfer between stages. CMS pipelines were designed for efficient execution in a memory-constrained, single-tasking operating system with record-oriented files. In contrast, DUP is designed for modern applications that might not use record-oriented I/O and need to run in parallel and on many different platforms.

Another close relative to the DUP System are Kahn Process Networks (KPNs) [54]. A major difference between DUP and KPNs is that buffers between stages in DUP are bounded, which is necessary given that unbounded buffers cannot really be implemented and that in general determining a bound on the necessary size of buffers (called channels in KPN terminology) is undecidable [84]. Note that the UNIX command `buffer` can be used to create buffers of arbitrary size between stages in DUP. Another major difference with KPNs is that DUP does not require individual processes to be deterministic. Non-determinism on the process level voids some of the theoretical guarantees of KPNs; however, it also enables programmers to be much more flexible in their implementations. While DUP allows non-determinism, DUP programmers explicitly choose non-deterministic stages in specific places; as a result, non-determinism in DUP is less pervasive and easier to reason about compared to languages offering parallel execution with shared memory.

Where CMS pipelines focus on the ability to glue small, reusable programs into larger applications, the programming language community has extended various general-purpose languages and language systems with support for pipelines. Existing proposals for stream-processing languages have focused either on highly-efficient implementation (for example, for the data exchange between stages [42]) or on enhancing the abstractions given to programmers to specify the pipeline and other means of communication between stages [110]. The main drawback of all of these designs is that they force programmers to learn a complex programming language and rewrite existing code to fit the requirements of the particular language system. The need to follow a particular paradigm is particularly strong for real-time and reactive systems [104, 61]. Furthermore, especially when targeting heterogeneous multi-core systems, quality implementations of the particular language must be provided for each architecture. In contrast, the DUP language implementation is highly portable (relying exclusively on canonical POSIX system calls) and allows developers to implement stages in any language.

On the systems side, related research has focused on maximizing performance of streaming

applications. For example, StreamFlex [104] eliminates copying between filters and minimizes memory management overheads using types. Other research has focused on how filters should be mapped to cores [58] or how to manage data queues between cores [42]. While the communication overheads of DUP applications can likely be improved, this could not be achieved without compromising on some of the major productivity features of the DUP System (such as language neutrality and platform independence).

In terms of language design and runtime, the closest language to the DUP Assembly language is Spade [47] which is used to write programs for InfoSphere Streams, IBM's distributed stream processing system [6]. The main differences between Spade and the DUP Assembly language is that Spade requires developers to specify the format of the data stream using types and has built-in computational operators. Spade also restricts developers of filters to C++; this is largely because the InfoSphere runtime supports migrating of stages between systems for load-balancing and can also fuse multiple stages for execution in a single address space for performance. Dryad [50] is another distributed stream processing system similar to Spade in that it also restricts developers to developing filters in C++. Dryad's scheduler and fault-tolerance provisions further require all filters to be deterministic and graphs to be free of cycles, making it impossible to write stages such as `faninany` or `holmerge` in Dryad. In comparison to both Spade and Dryad, the DUP System provides a simpler language with a much more lightweight and portable runtime system. DUP also does not require the programmer to specify a specific stream format, which enables the development of much more generic stages. Specifically, the Spade type system cannot be used to properly type stream-format agnostic filters such as `cat` or `fanout`. Finally, DUP is publicly available whereas both Spade and Dryad are proprietary.

DUP is a coordination language [41] following in the footsteps of Linda [20]: the DUP System is used to coordinate computational blocks described in other languages. The main difference between DUP and Linda is that in DUP, the developer specifies the data flow between the components explicitly, whereas in Linda, the Linda implementation needs to match tuples published in the tuplespace against tuples published by other components. The matching of tuples in the Linda system enables Linda to execute in a highly dynamic environment where processes joining and leaving the system are easily managed. However, the matching and distribution of tuples also causes significant performance issues for tuplespace implementations [115]. As a result, Linda implementations are not suitable for distributed stream processing with significant amounts of data.

5.2.2 The DUP Assembly Language

The DUP Assembly language allows developers to specify precisely how to connect stages and where those stages should be run. Figure 5.1 lists the DUP Assembly code for a distributed "Hello World" example program.

In essence, the DUP language allows developers to specify a directed graph using an adjacency list representation and IO redirection syntax similar to that of well-known UNIX shells [90]. The nodes in the directed graph are the stages initiated by DUP. A DUP pro-

```
s @10.0.0.1[0<in.txt,1|g1:0,3|g2:0]$ fanout;
g1@10.0.0.1[1|in:0]                $ grep Hello;
g2@10.0.0.2[1|in:3]                $ grep World;
in@10.0.0.2[1>out.txt]             $ faninany;
```

Figure 5.1: DUP specification. `in.txt` is passed to `fanout` (“`0<in.txt`”) which copies the stream to all outputs; in this case output 1 to stream 0 ($\equiv$ `stdin`) at `g1` (“`1|g1:0`”) and output 3 to stream 0 at `g2` (“`3|g2:0`”). `g1` and `g2` run `grep`, the outputs (1 $\equiv$ `stdout`) flowing into stage `in` as streams 0 and 3 respectively. `in` merges those streams and writes the output into `out.txt`. The resulting data flow is illustrated in Figure 5.3.

gram consists of a list of statements, each of which corresponds to one such node. Statements start with a label that is used to reference the respective stage in the specification of other stages. The keyword `DUP` is used to reference streams associated with the controlling `dup` command in the case that the `dup` command itself is used as a stage.

```
<PROGRAM>   ::= <STAGE>*
<STAGE>     ::= <LABEL> '@' <ADDRESS> '[' <EDGELIST> ']' '$' <COMMAND> ';'
<EDGELIST> ::= <EDGE> (',' <EDGE>)*
<EDGE>      ::= <INTEGER> <OP> <NODE>
<NODE>      ::= <REMOTEPR> | <UNIX_PATH>
<REMOTEPR> ::= <LABEL> ':' <INTEGER>
<OP>        ::= '|' | '<' | '>' | '>>'
```

Figure 5.2: Simplified grammar for the DUP Assembly language. Note that we do not expect programmers to need to develop applications by *directly* using this language in the future; this language is the “assembly” language supported by the DUP runtime system. Higher-level languages running on top of DUP that facilitate (static) process scheduling and aspect oriented programming are under development.

The label is followed by a hostname specifying on which system the stage will be run. A helper process, `dupd`, will be started on the specified host, listen on a port to establish network connections and eventually supervise stages run there. The address is followed by a comma-separated list of edges representing primarily the outgoing streams for this stage. Input streams are only explicitly specified in the case of input from files or the controlling `dup` command. Inputs from other stages are not specified because they can be inferred from the respective entry of the producing stage. DUP supports four different ways to create streams for a stage:

Read An input file edge consists of an integer, the “<” operator and a path to the file to be used as input. The integer is the file descriptor from which the stage will read the input stream. `dupd` is responsible for opening the input stream and validating that the file exists and is readable.

Write An output file edge for writing consists of an integer, the “>” operator and a path to the file to be overwritten or created. The integer is the file descriptor to which this stage

will write. dupd checks that the specified path can be used for writing.

Append An output file edge for appending consists of an integer, the ">>" operator and a path to the file. The integer is the file descriptor to which this stage will write.

Pipe Non-file output edges consist of an integer, the "|" operator, a stage label, the ":" character and another integer. The first integer specifies the file descriptor to which this stage will write. The label specifies the process on the other end of the pipe or TCP stream and the second integer is the file descriptor from which the other stage will read. If an edge list contains a label that is not defined elsewhere in the configuration file then the program file is considered malformed and rejected by dup.

The final component of a complete stage statement is the command (with arguments) that is used to start the process. Figure 5.2 contains a formal grammar for the DUP language. The grammar omits I/O redirection from/to the controlling dup command for clarity.

5.2.3 DUP System Architecture

The dup client interprets the mini-language from Section 5.2.2 which specifies how the various stages for the application should be connected. dup then connects to hosts running ssh servers and starts dupd helper processes which then receive control information via the SSH tunnel. The control information specifies the binary names and arguments for the stages as well as how to establish TCP streams and UNIX pipes to connect the stages with each other.

Figure 5.3 illustrates how the components of the system work together.

The primary interaction between dup and the dupds involves four key steps [44]:

1. dup starts the dupds and transmits session information. This includes all of the information related to processes that are supposed to be run on the respective dupd.

2. When a stage is configured to transmit messages to a stage initiated by another dupd, the dupd responsible for the data-producing stage establishes a TCP connection to the other dupd and transmits a header specifying which stage and file descriptor it will connect to the stream. If dup is used as a filter, it too opens similar additional TCP streams with the respective dupds. The main difference here is that dup also initiates TCP connections for streams where dup will ultimately end up receiving data from a stage.

3. Once a dupd has confirmed that all required TCP streams have been established, that all required files could be opened, and that the binaries for the stages exist and are executable, it transmits a "*ready*" message to the controlling dup process (using the connection on which the session information was initially received).

4. Once all dupds are ready, dup sends a "*go*" message to all dupds. The dupds then start the processes for the session.

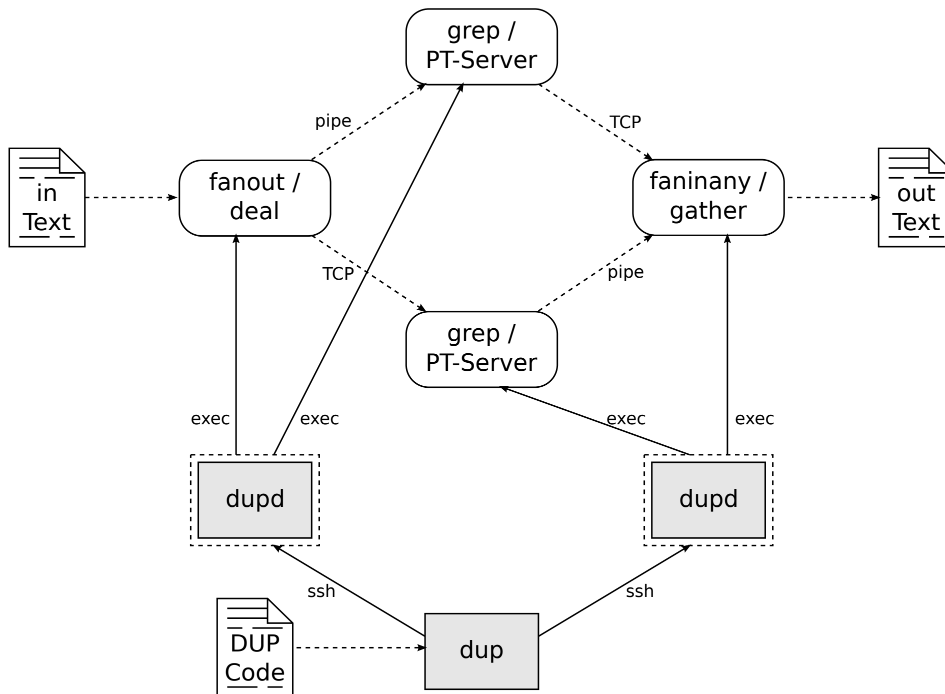


Figure 5.3: Overview for one possible configuration of the DUP System. Dashed lines show application data flow. Solid lines correspond to actions by DUP. Data-flow graph for DUP Assembly corresponding to the illustration are given in Figure 5.1.

5.2.4 Generic DUP Stages

Taking inspiration from stages available in CMS [46, 49], the DUP System includes a set of fundamental multi-stream stages. UNIX already provides a large number of filters that can be used to quickly write non-trivial applications with a linear pipeline. Examples of traditional UNIX filters include `grep` [43], `awk` [31], `sed` [31], `tr`, `cat`, `wc`, `gzip`, `tee`, `head`, `tail`, `uniq`, `buffer` and many more [90].

While these standard tools can all be used in the DUP System, none of them support multiple input or output streams. In order to facilitate the development of multi-stream applications with DUP, we provide a set of primitive stages for processing multiple streams. Some of the stages currently included with the DUP System are summarized in Table 5.1. Many of the stages listed in Table 5.1 are inspired by the CMS multi-stream pipeline implementation [49]. Naturally, we expect application developers to write additional application-specific stages.

5.2.5 DUP Programming Philosophy

In order to avoid the common data consistency issues often found in parallel programming systems, stages and filters for DUP should not perform any updates to shared storage outside of the memory of the individual process. While the DUP System has no way to enforce this property, updates to files or databases could easily cause problems; if stages were allowed

Stage	Description	I/O Streams in	I/O Streams out
`fanout`	Replicate input n times	1	n
`faninany`	Merge inputs, any order	n	1
`gather`	Merge inputs, round-robin (waits for input)	n	1
`holmerge`	Forward input from stream that has sent the most data so far, discard data from other streams until they catch up	n	1
`deal`	Split input round robin to output(s), or per control stream	2	n
`mgrep`	Like `grep`, except non-matching lines output to secondary stream	1	2
`lookup`	Read keys from stream 3; tokens to match keys from stream 0; write matched tokens to 1, unmatched to 4 and unmatched keys to 5	2	3
`gate`	forward 1st input to 1st output until 2nd input ready	2	1

Table 5.1: Summary of general-purpose multi-stream stages to be used with DUP in addition to traditional UNIX filters. Most of the filters above can either operate line-by-line in the style of UNIX filters or using a user-specified record length.

to update storage, changes in the order of execution could easily result in unexpected non-determinism. This might be particularly problematic when network latency and stage scheduling cause non-deterministic runs in a larger system that replicates parts of the computation (e.g., in order to improve fault-tolerance).

For applications that require parallel access to shared mutable state, the DUP System can still be used to parallelize (and possibly distribute) those parts that lend themselves naturally to stream processing. Other parts of the code should then be designed to communicate with the DUP parts of the application through streams.

We specifically expect stages developed for the DUP System to be written in many different languages. This will be necessary so that the application can take advantage of the specialized resources available in heterogeneous multi-core or HPC systems. Existing models for application development on these systems often force the programmer to use a particular language (or small set of languages) for the entire application. For example, in a recent study of optimization techniques for CUDA code [96], twelve benchmark programs were modified by porting critical sections to the CUDA model. On average, these programs were only 14% CUDA-specific, yet the presence of CUDA sections limits the choice of languages and compilers for the entire program. The implications are clear: the use of a monolithic software architecture for programs designed to operate efficiently on high-performance hardware will severely restrict choices of development teams and possibly prevent them from selecting the most appropriate programming language and tool-chain for each part of a computation. Using the DUP System, developers will be able

to compose larger applications from stages written in the most appropriate language available.

Another important use-case for DUP is the parallel and distributed execution of legacy code. In contrast to other new languages for parallel programming, which all too often advocate for large-scale (and often manual) program translation efforts, the DUP philosophy calls for writing thin wrappers around legacy code to obtain a streaming API. As we experienced in our case study, it is typically easy to adapt legacy applications to consume inputs from streams and to produce outputs as streams.

5.3 Material and Methods

5.3.1 Testing conditions

The study used 16 compute nodes of the Infiniband Cluster in the Faculty of Informatics at the Technische Universität München [56]. Each node was equipped with an AMD Opteron 850 2.4 GHz processor with 8 GB of memory, and the nodes were connected using a 4x Infiniband network. The SILVA database (SSURef_91_SILVA_18_07_07_opt.arb) [89], which stores sequences of small subunit ribosomal ribonucleic acids and consists of 196,890 sequence entries (with 289,563,473 bases), was used for preparing test database sets and respective PT-Servers. We divided the original database into 1, 2, 4, 8, and 16 partitions, and a random sampling algorithm was used for composing the partitioned database sets (within each database analysis set, each partition is about the same size). The PT-Servers used in this study were created from these partitions. Table 5.2 characterizes the resulting partitions and PT-Servers.

# Part.	**# Sequences**	**# MBases**	**Memory (MB)**	
			part.	total
1	196,890	289.6	1,430	1,430
2	98,445	144.7	745	1,489
4	49,222	72.4	402	1,609
8	24,611	36.2	231	1,849
16	12,305	18.1	145	2,327

Table 5.2: Resulting problem sizes for the different numbers of partitions. This table lists the average number of sequences and bases for the PT-Server within each partition and the resulting memory consumption for each PT-Server as well as the total memory consumption for all partitions.

For the queries, we selected 800 inverse sequence strings of rRNA-targeted oligonucleotide probe sequences of length 15–20 from probeBase, a database of published probe sequences [68]. Each retrieved sequence string has matches in the SILVA database and the respective PT-Server instance. Applying these real world query sequence strings ensured that every search request required non-trivial computation and communication. We generated four sets of inverse sequence strings (400 strings each) by random string distribution of the original dataset from probeBase,

and every test run was performed with these four datasets. The presented performance values are the means of the four individually recorded runs.

5.3.2 Adapting the PT-Server for DUP

In the ARB software package, `arb_probe` is a program which performs, per execution, one string search using the PT-Server when a search string and accompanying search parameters are specified (these are passed as command line arguments). For DUP, `arb_probe` had to be modified to read the parameters and the search string as a single line from `stdin` and pass one result set per line to `stdout`. It took one developer (who had experience with ARB but not DUP or distributed systems) about three hours to create the modified version `arb_probe_dup` and another two hours to compile DUP on the Infiniband Cluster, write adequate DUP scripts and perform the first run-time test. Debugging, testing, optimization and gathering of benchmark results for the entire case study was done in less than two weeks.

All searches were conducted using the program `arb_probe_dup` with similar parameters: `id 1 mcmpl 1 mmis 3 mseq ACGTACGT`. The first parameter (`id 1`) set the PT-Server ID; the second activated the reverse complement sequence (`mcmpl 1`). For each dataset and approach, the third parameter was used to perform an exact search (`mmis 0`) in order to find matches identical with the search string and an approximate search (`mmis 3`) in order to find all identical strings and all similar ones with maximum distance of three characters to the search string. The last parameter indicated the match sequence.

Figure 5.4 shows the DUP assembly code for the replicated run with two servers. Here, identical PT-Servers are used with the goal of optimizing execution time. Figure 5.6 shows the equivalent DUP assembly code for the partitioned setting. In this case, since each PT-Server only contains a subset of the overall database, all requests are broadcast to all PT-Servers using `fanout`.

```
snd@opt1[0<queries.txt,1|pt1:0,3|pt2:0] $ deal;
pt1@opt1[1|rcv:0]                       $ arb_probe_dup;
pt2@opt2[1|rcv:3]                       $ arb_probe_dup;
rcv@opt2[1>results.txt]                 $ faninany;
```

Figure 5.4: DUP specification for the replicated configuration that uses identical ARB PT-Servers. The queries are simply distributed round-robin over the (in this case two) available PT-Servers and the results collected as they arrive. Note that the order of individual results is not relevant for the correct function of CaSSiS.

5.4 Results

As shown in Table 5.2, partitioning the original database into n partitions results in almost proportional reductions in per-node memory consumption: doubling the number of partitions means almost halving the memory consumption per PT-Server partition. In practice we expect

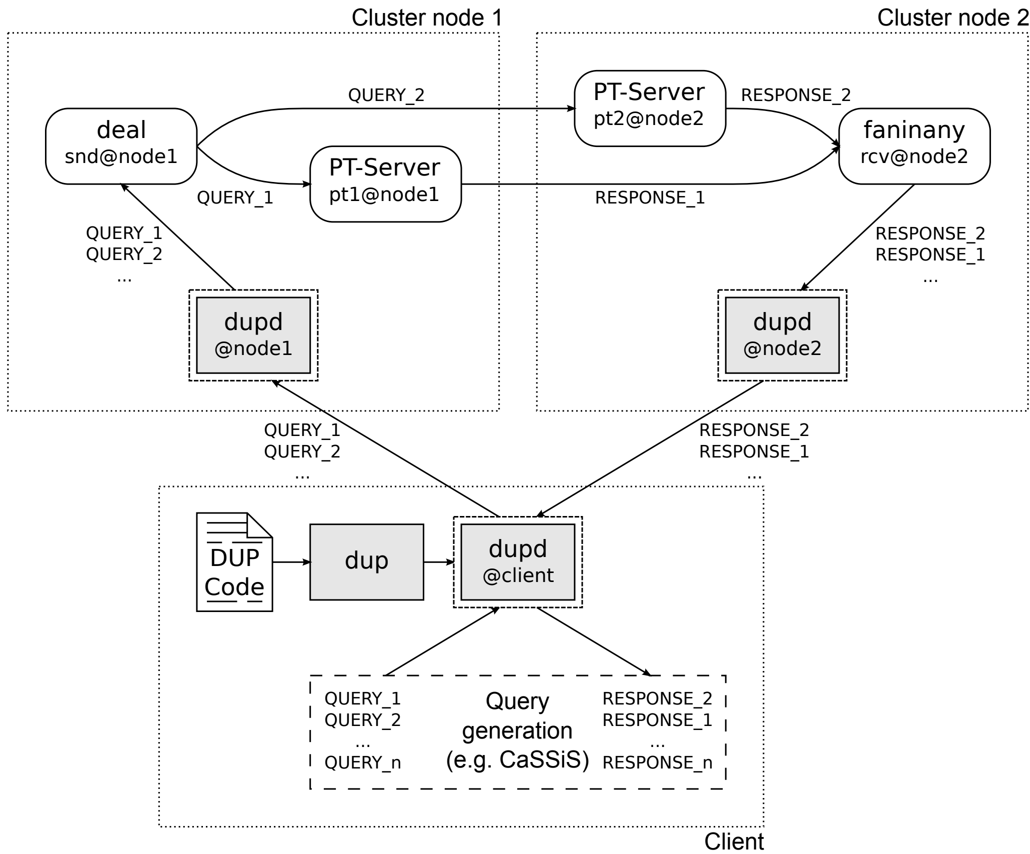


Figure 5.5: PT-Server querying using replicated datasets. Data-flow graph of the code from Figure 5.4. Note: Predefined lists of queries were used in our test, and the results were collected in text form for further evaluation.

significantly larger databases to be partitioned, resulting in partition sizes close to the size of the main memory of the HPC node responsible for the partition.

Figure 5.8 summarizes the speedup we obtained using n PT-Server replicas (each processing a fraction of the queries). This answers the question as to how much performance could be gained by distributing the queries over n identical (replicated) PT-servers, each containing the full database. Compared with a local version (direct communication between a PT-Server and arb_probe_dup) we have measured a speedup of 5.84 for 16 compute nodes.

The available bandwidth of the compute cluster using TCP is about 107 MB/s, which is close to the 85 MB/s consumed on average by the collector node for 16 compute nodes. For this run, the average CPU utilization of the 16 compute nodes is about 34% and the master node uses about 56%. The legacy ARB code produces rather verbose output, which explains why this benchmark is IO-bound on our cluster when run with more than approximately 8 compute nodes. Converting the human-readable output to a compact binary format would likely result in a significant performance improvement; however, the purpose of this study was to evaluate possible speedups for legacy code without significant changes to the existing infrastructure and

```
snd@opt1[0<queries.txt,1|pt1:0,3|pt2:0] $ fanout;
pt1@opt1[1|rcv:0]                       $ arb_probe_dup;
pt2@opt2[1|rcv:3]                       $ arb_probe_dup;
rcv@opt2[1>results.txt]                 $ gather;
```

Figure 5.6: DUP specification for the partitioned configuration where each ARB PT-Server only contains a slice of the database. In this example two partitions are used. The queries are broadcast to the available PT-Servers and the results collected in round-robin order. This ensures, that results for the same query arrive in one batch.

changing the message format of the ARB framework would be a significant change.

The overall runtime for querying a partitioned PT-Server with one sequence string set (400 requests) was in a range of 2 seconds (16 partitions) to 8.25 seconds (one partition) for exact searches, and 16 seconds (16 partitions) to 73 seconds (one partition) for approximate searches. For the replicated PT-Servers, execution time for exact searches ranged from approximately 8.3 seconds on one node to 1.5 seconds on 16 nodes. The approximate search (up to three mismatches) ranged from 72 seconds on one node to 13 seconds on 16 nodes. In an additional test run with the replicated servers, we increased the number of requests (to 2000) by repeating the string set to increase the measured time and reduce possible side effects. The execution time ranged from 140.91 seconds (one node) to 27.09 seconds (16 nodes) for exact searches, and 1479.60 seconds (one node) and 222.26 (16 nodes) for the approximate search.

5.5 Conclusion

The significant challenges with writing efficient parallel high-performance code are numerous and well-documented. The DUP System presented in this chapter addresses some of these issues using multi-stream pipelines as a powerful and flexible abstraction around which an overall computation can be broken into independent stages, each developed in the language best suited for the stage, and each compiled or executed by the most effective tools available. Our experience so far makes us confident that DUP can be used to quickly implement parallel programs, to obtain significant performance gains, and to experiment with various dataflow graph configurations with different load-distribution characteristics.

The speedups achieved in our tests are by themselves clearly not sensational; however, the ratio of speedup to development time is. Programmer productivity is key here, especially since researchers in bioinformatics are rarely also experts in distributed systems. Furthermore, the improvements in performance and memory consumption are significant and have direct practical value for molecular biologists and bioinformaticians: aside from the acceleration of sequence string searches by a factor 3.5 to 5.8, this approach also offers biologists the possibility to search very large databases using the ARB PT-Server without having to access special architectures with extreme extensions to main memory. Adapting existing DUP dataflows to new topologies to improve the overall performance due to changed requirements should easily be done by

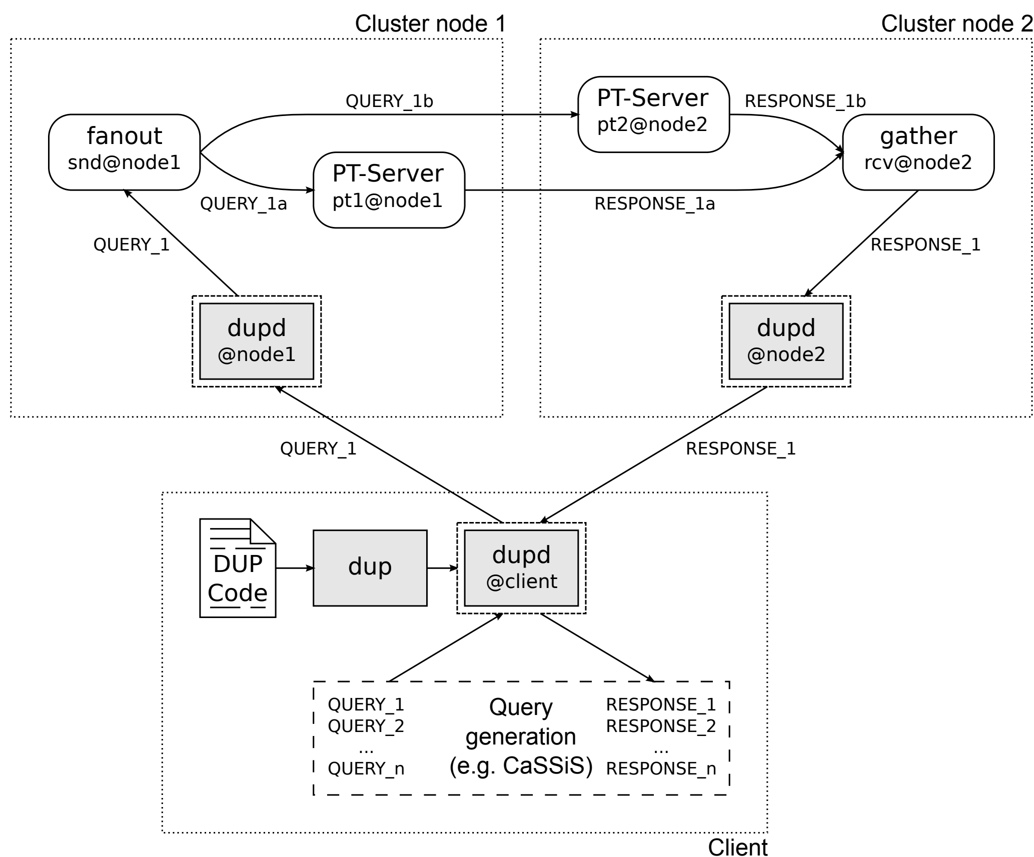


Figure 5.7: PT-Server querying using dataset partitions. Data-flow graph of the code from Figure 5.6. Note: Predefined lists of queries were used in our test, and the results were collected in text form for further evaluation.

adapting the configuration files and installing the respective software.

In the future, DUP could be used to drive large-scale genome database analyses. Depending on the problem size, we expect to use DUP to combine partitioning and replication in one system. For example, it would be easy to create n replicas of m partitions in order to improve throughput while also reducing the memory consumption of the PT-Servers. The memory requirements of CaSSiS, which still would run on a the node with the most memory, were already reduced in the previous Chapter 4 Finally, assuming additional performance is desired, the ARB data format could be changed to be less verbose and thereby avoid bandwidth limitations.

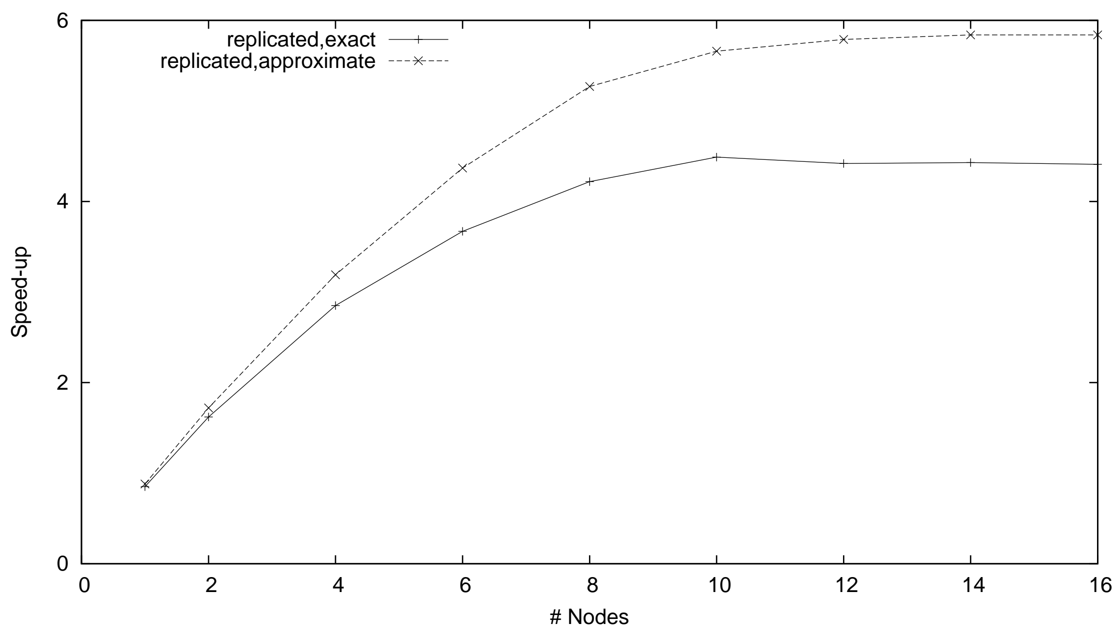


Figure 5.8: Speedup of sequence string matches for the replicated PT-Server. The plot shows the average speedup over five runs for an exact search and an approximate search (with up to three mismatches).

Chapter 6

Web-based Signature Evaluation with PLEASE

In this chapter we describe the *Probe Library Query and Evaluation System* (PLEASE), a multiple component client-server application. PLEASE provides a structured access to signature sequences that were precomputed with CaSSiS, and it allows the comprehensive evaluation of their specificities and sensitivities. Organisms matched by a signature sequence are displayed as difference alignments including hit values. Additionally, signatures and matched organisms are highlighted in a graphical representation of a phylogenetic tree. This tree structure also serves to select phylogenetic groups to be evaluated.

The PLEASE software application was derived from the the ARB Probe Library, developed by Tina Lai, Lothar Richter, and Ralf Westram in 2003. In the context of students' work which was supervised by me, the application was modified and new features added: For his thesis, the apprentice (Auszubildender) Peter Klimpt extended PLEASE with a remote probe match functionality in 2008 (Section 6.2.2). Tianxiang Lu did a complete refactoring of the server architecture and the client-server communication (Section 6.2.1) in 2011 for his diploma thesis. A prototype implementation of PLEASE was tested on a server at the TU München.

6.1 Introduction

Oligonucleotide probes targeted to corresponding signature sites on ribosomal ribonucleic acids (rRNA) are widely applied in molecular microbial diagnostics [23, 67]. The application of probes in hybridization techniques such as FISH and diagnostic microarrays [3] enable a cultivation independent detection and identification of microorganisms.

The amount of sequence data in publicly available rRNA databases is currently growing at very strong rate (Figure 1.5 shows the growth rate for the RDP-II and ARB SILVA databases). Despite the vast improvements achieved in the previous chapters, processing up-to-date datasets with CaSSiS can still turn the *in silico* search for signature candidates into a computationally intensive task. This is especially true when applying very relaxed search conditions and using

older computer hardware requirements for the ARB software environment [70] in combination with ARB SILVA databases are a 64bit system with minimum 4GB RAM [1]. Processing current ARB SILVA rRNA datasets requires more than 4GB of main memory to at least be able to use the ARB PT-Server [33]. Setting up database servers specifically for probe candidates could help to overcome this limitation. Online applications could access these databases and thereby allow easy retrieval of probe candidates and provide additional quality evaluation.

probeBase [68] is an online database for curated oligonucleotide probes. It provides 2,662 probes (status as of September 2012) targeted to either small or large subunit rRNA genes. It additionally provides information on specificity and experimental usage conditions. This information can be cross-checked with probeCheck [66], an online probe match tool based on the ARB probe match functionality. However, this service may not be satisfactory in every case for a wet lab biologist. A proper evaluation of the specificity of the candidates by using the functions provided in probeBase and probeCheck (comparing probe match lists, re-formatting the results) by hand can become very time consuming. Some of the probes were designed several years ago by analyzing smaller datasets. They may not be suitable due to an inappropriate specificity as outlined by in silico cross-matching in current large datasets [3]. For the majority of phylogenetic or taxonomic prokaryotic groups, probe candidates are not available in probeBase [68].

The following sections present a more comprehensive and interactive web-based approach, the Probe Library Query and Evaluation System (PLEASE). This client-server application allows online retrieval and fine evaluation of signatures, the binding sites of oligonucleotide probes on rRNA gene sequences. The PLEASE server provides access to multiple signature sets which were precomputed with CaSSiS. From a Java client application, signature candidates can be accessed via navigation through a phylogenetic tree. Due to the use of a static database for the signatures, client requests are (on average) answered within seconds and the memory footprint of the client application is kept low.

6.2 Implementation

The PLEASE software application was derived from the the ARB Probe Library (ARB-PL), a client-server application with similar functionality. But the original version was clearly limited when it came to the quantity of the processed data and the provided functionality. Several design changes were necessary during the transition from the old ARB-PL design (Figure 6.1) to the new client-server structure (Figure 6.2) of PLEASE.

PLEASE consists of two client-server components: The “Probe Library” for accessing signature candidates and the “Probe Match” for their additional evaluation. The “PLEASE Client” acts as the main software application which is downloaded and run by the user. It combines the client applications of both components under a common graphical user interface.

The client application depends on an installed Java Runtime Environment 1.6 (or higher). It has been implemented as a Java Web Start application. PLEASE can therefore in principle be run from a browser windows on every Internet-connected operating system.

[1] `http://www.ribocon.com/home/arb/arbuntu/`

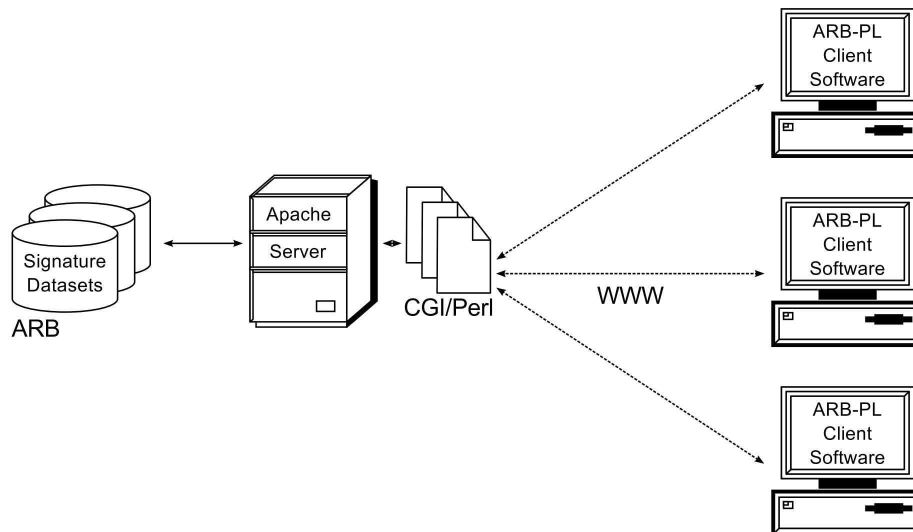


Figure 6.1: The ARB-PL client-server structure used an Apache web server and CGI scripts for handling client requests. An ARB database was used as the database back-end. Perl scripts were used to process the requests and generate database queries.

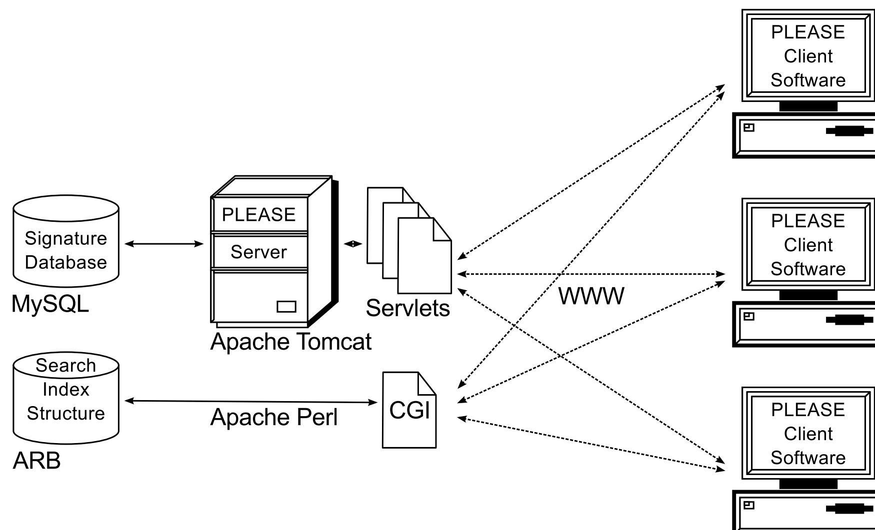


Figure 6.2: The PLEASE client/server structure was completely redesigned in order to overcome limitations of the old approach. The ARB database was replaced by a MySQL database back-end. Apache Tomcat is used as web-server and servlet-container. The CGI-scripts are replaced by Java servlets to handle client requests.

The server installation additionally requires Java Swing and Apache Tomcat. A MySQL database server is necessary as database back-end. Although the server component of PLEASE is not depending on native GNU/Linux services or applications, the current version was developed and tested only under GNU/Linux. An Intel Quad Core I7 920 system with Ubuntu

Server 10.04.2 LTS as operating system and 16GB of main memory was used for this purpose.

The source dataset for our tests was the SILVA SSURef_102 database. The original database contains 460,783 SSU rRNA gene sequences (about 666 million nucleotides). In order to advance the finding of phylogenetic oligonucleotide probes of high quality in terms of specificity and sensitivity, we reduced database and phylogenetic tree by removing sequence that were too short ($\leq 1,200$ nucleotides) or entries of insufficient quality.

6.2.1 The PLEASE Probe Library

The first step was the transition from the ARB database structure to a SQL-based database management system. The switch from pure ARB datasets to the precalculated results of CaSSiS as source for the signatures has offered this opportunity. ARB uses a hierarchical in-memory database structure. It allows fast read access to entries in the database, at the expense of memory usage: large datasets can lead to excessive memory consumption. But its biggest drawback is the lack of JOINS, which makes it necessary to manually cache intermediate results. This lack results in multiple accesses and subsequent hardcoded merges by PLEASE, instead of optimized merges within the database management system. It was therefore decided to switch to MySQL[2]. We compared the two freely available MySQL engines MyISAM and InnoDB. The latter supports transaction based queries. MyISAM, on the other hand, is able to handle "read" operations faster than InnoDB. With only one import run every few weeks but numerous `SELECT` operations and no need for transactions, the MyISAM engine was chosen as the PLEASE database back-end. The following requirements were the decisive factors for us:

- Only few `INSERT` operations during the import of precalculated signature sets (i.e. not "write-intensive"). After the import of new datasets no write operations are performed on the database.
- No `UPDATE` operations necessary. Again, the database entries are all read-only after the initial import process. Possible updates are performed by switching the complete database.
- Queries need not be transaction-based, as there are no larger blocks of multiple queries during the runtime.
- Frequent `SELECT` operations (i.e. "read-intensive") when requests by a PLEASE client have to be processed.
- Server-enforced data integrity checking is not necessary, as the database is read-only after the import process. During import, integrity checks are manually performed by the importer.
- Foreign keys are externally defined during the import process. Their automatic management by the database management system after the import process is not necessary.

[2] `http://www.mysql.com/`

- To facilitate the generation of foreign keys during the import process, `AUTO_INCREMENT` is a necessary feature.

The PLEASE Probe Library stores the signature candidate database(s) including associated data. It also contains phylogenetic trees based on SSU-rRNA sequences. Terminal nodes (leaves) of the trees reference to the respective sequence-specific signature candidates, whereas internal nodes point to signatures specific for groups of sequences defined by the node.

The signature datasets are imported from the result files that are created by either the CaSSiS-BGRT or CaSSiS-LCA approach: The CSV files `result_array.csv` (Table 6.1) and the associated `results_X.csv` (Table 6.2; `X` represents the number of outgroup hits). Additionally, the phylogenetic tree that was used to compute during the CaSSiS computations is needed.

Group		# outgroup hits										
ID	**Size**	0	1	2	3	4	5	6	7	8	9	10
Bacteria	74	69	22	0	1	0	0	0	0	0	0	0
Proteobacteria	31	24	29	11	25	29	25	11	23	12	26	14
Eukarya	25	25	3	0	0	0	0	0	0	0	0	0
Actinobacteridae	18	18	18	18	17	17	18	11	17	18	17	11
Firmicutes	15	13	13	11	11	12	10	13	13	13	12	4
Bacilli	12	12	12	12	9	10	11	10	10	10	12	11
Metazoa	12	11	10	9	9	9	11	10	9	10	11	12
Gammaproteobacteria_1	11	11	11	10	10	8	9	9	10	10	9	9
Alphaproteobacteria	10	9	8	8	6	7	9	9	6	6	9	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 6.1: The CaSSiS `result_array.csv` file contains for every node the number of ingroup matches, that occurred over the entire range of allowed outgroup matches. For example in the group 'Proteobacteria' at least one signature was found, that matches 29 of the 31 sequences with one outgroup hit.

ID	Size	Ingroup	Signatures...		
Bacteria	74	69	UCCUACGGGAGGCAGCAG	CUCCUACGGGAGGCAGCA	...
Proteobacteria	31	24	CGCCCGUCACACCAUGGG	GCCCGUCACACCAUGGGA	...
Eukarya	25	25	GGGCAAGUCUGGUGCCAG	GGCAAGUCUGGUGCCAGC	...
Actinobacteridae	18	18	GCCGGGGUCAACUCGGAG	GGGGUCAACUCGGAGGAA	...
Firmicutes	15	13	CAGCAGUAGGGAAUCUUC	GCAGCAGUAGGGAAUCUU	...
Bacilli	12	12	GGCAGCAGUAGGGAAUCU	AGGCAGCAGUAGGGAAUC	
Metazoa	12	11	GGUAGUGACGAAAAAUAA	GUAGUGACGAAAAAUAAC	...
Gammaproteobacteria_1	11	11	AUGAAUUGACGGGGGCCC	UGAAUUGACGGGGGCCCG	
Alphaproteobacteria	10	9	AGAGGUGAAAUUCGUAGA	AGGUGAAAUUCGUAGAUA	...
⋮	⋮	⋮	⋮	⋮	

Table 6.2: A separate result file exists for every number of outgroup hits. In the example above, an excerpt from the file `results_0.csv`, all signatures match without outgroup hits.

Although only run once, the method that is used to import the datasets from the CSV files has great influence on the overall runtime — directly on the runtime of the import process, and later indirectly through its influence on the database scheme. Although a single `INSERT` operation may seem fast, using them serially led to large runtime issues. In our tests, even the

import of a small CaSSiS result set (computed from a few thousand sequences) generated tens of millions import operations. The whole import process took multiple days.

A second way to import data into MySQL tables is the `LOAD DATA` operation. This feature was designed to import bulk data from external CSV files. The MySQL process requires read access rights to the file and its folder. Additionally, the structure of the destination table must correspond with the file structure. According to the official reference manual[3], the `LOAD DATA` operation can speedup the import process about 20-fold compared to multiple INSERT operations. One prerequisite for the maximum import speed is an empty target table or, at least, the deletion of the indices in advance. In our case, the import tool reads the CaSSiS result files at a defined path and imports them into temporary tables. Then, the content of the temporary tables is imported into the appropriate destination tables. A table index is (re)built once after the import, instead of constant updates with each insert.

Much more important than the time for importing datasets is the answer time of database queries when processing PLEASE client requests. Two concepts, using indices and the partitioning of datasets, can be used to prevent costly searches through complete database tables. The idea behind partitioning is splitting up datasets (tables) into smaller chunks or logic units to distribute the load when processing queries. A further step is the distribution of complete databases onto different physical nodes of a cluster. This concept was partly used in the original Probe Library, where the signatures were split into different database files based on their length. In PLEASE, only indices were used — a feature the ARB database is lacking. Indices reduce the necessary search costs with the tables and are used in combination with primary and foreign keys. These keys are usually integers which are easy to sort and compare. Their main disadvantage, high runtime costs when inserting or updating data, do not apply for our given model where the database is seldomly updated (e.g. every few weeks).

The old ARB-PL system relied on a single pre-computed ARB database with a proprietary hierarchical scheme. With the switch to a SQL-based database management system, this scheme was exchanged with an entity-relationship model. The dependencies between the individual data sets is shown in Figure 6.3. Five entities were defined to represent the datasets as tables:

tree A set of CaSSiS result files (i.e. CSV tables) always is computed from a phylogenetic tree. Groups and single sequences in the sets resemble inner nodes and leaves in the tree. References to the tree (not the tree itself) is therefore stored as an entity in the database. On request, the tree file can be looked up, transferred to the PLEASE client and displayed there. Four attributes *id, name, file_path, outgroup* are defined.

node Every node in the tree again is an entity, with a "readable" identifier (e.g. the group or organism name). It contains the four attributes *TreeId, ID, Name, Size.*

match Each node can be matched by one or more signatures. These match entities contain four attributes *NodeId, ID, outgroup, ingoup.* The number of outgroup hits is defined by

[3] `http://dev.mysql.com/doc/refman/5.1/de/insert-speed.html`

the CaSSiS result file, e.g. "results_1.csv" contains signatures with one outgroup hit. The number of ingroup hits is given for each node in the result file.

match_to_signature A key design feature to reduce the amount of redundant signature entries is an entity which links unique signature entities to matches. This entity contains two foreign keys as attributes, *MatchId* and *SignatureId*.

signature Signature entities contain the five attributes *ID, signature, length, mismatch, distance*. The length is stored to speed up search queries which only apply for a for a certain signature length. The attributes mismatch and distance are closely linked to a signature match and could have also be stored in the match_to_signature entity. As a design decision they were stored in the signature entry to to facilitate the processing of certain queries. A certain amount of redundancy (identical signature strings with different distances) was taken into account.

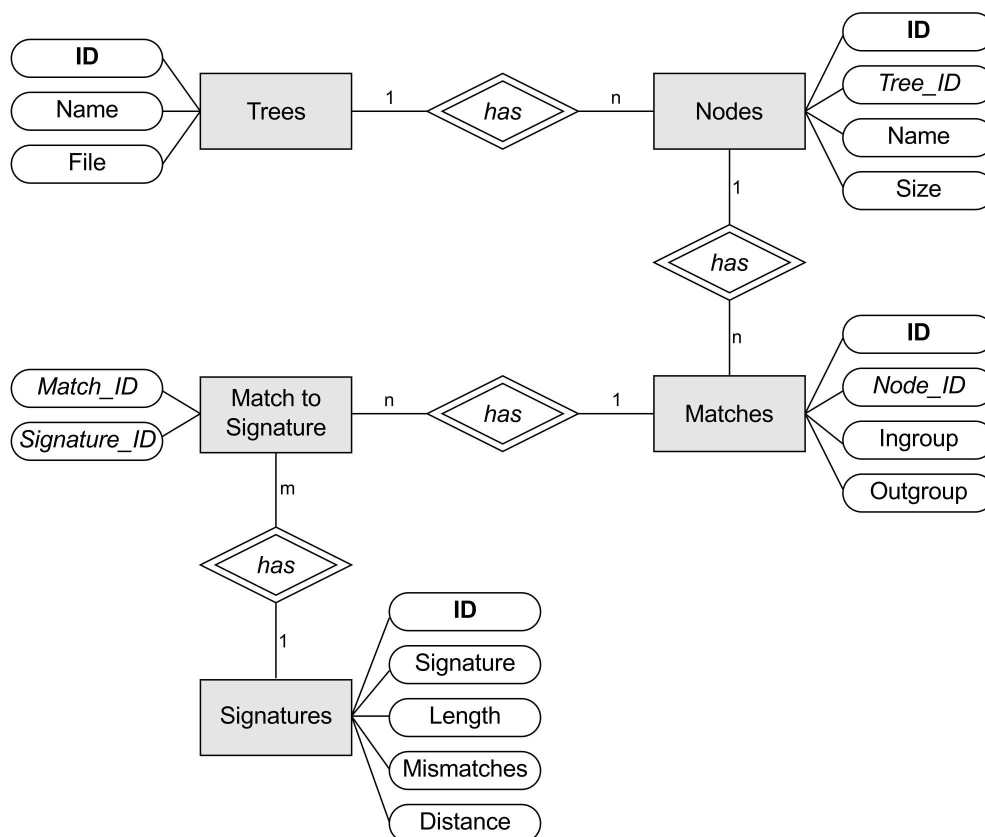


Figure 6.3: Entity-Relationship-Model of the PLEASE database.

The Probe Library Server is a combination of six Java servlets which are running on an Apache Tomcat server. The client directly addresses its queries to these servlets. The servlets themselves rely on two Java back-end classes which process and forward client requests to the

MySQL database via a JDBC[4] driver.

The servlets provide the phylogenetic trees and detailed information about nodes and lists of the related probe sequences. By retrieving the server and tree revision numbers, clients may notify users about updates and automatically download newer tree files. At the moment only one tree file is accessible. Future releases will provide access to multiple trees, one for each PT-Server offered by the Probe Evaluation Server:

- They process “hello” calls by the client to fetch initial version and configuration information from the server, e.g. the names (IDs) of available phylogenetic trees.
- They submit phylogenetic tree files (in an adapted NEWICK format [81]) to the client.
- They submit the maximum number of outgroup hits for a phylogenetic tree and associated result sets.
- They look-up and return all names (IDs) that are associated with a certain phylogenetic tree node. The ID is submitted by a client.
- They are used to query all signatures for a phylogenetic tree node that match the submitted constraints.

6.2.2 The PLEASE Probe Match

The PLEASE Probe Match server consists of a CGI interface script running on an Apache Webserver. The CGI script, written in Perl, receives and processes probe match requests and returns the results in XML-formatted form. All query parameters, sent to the server by the client, are stored in the result header. For every result entry, identification attributes similar to the ARB probe match tool (e.g. name, full name, accession number) as well as information about the type and position of possible mismatches (e.g. e.coli position, mismatch value and base) are included. This also allows the subsequent association of responses, as well as a session-free processing of requests up to a certain degree. By returning result matches in XML format, an interface is provided which may also be used by other applications beside the PLEASE Probe Match Client. All query information and associated results are combined in one file.

The CGI Perlscript requires a running ARB PT-Server, the search index, to process the probe matches. It is able to handle multiple search indices and provides a function to transmit a list of available server identifiers to clients. A small tool “arb_probe” is used to handle the PT-Server queries. It allows submitting query settings via command line parameters, and possible matches are returned via output stream.

The PT-Servers have to be pre-computed; they should correspond to the provided signature datasets and the phylogenetic trees. In our tests, we have used datasets of growing sizes that were derived from he SILVA SSURef 102 dataset. Currently, the list available servers has to be hardcoded directly in a configuration section of the Perl script.

[4]Java Database Connectivity, a data access technology / API.

6.2.3 The PLEASE Client

The Probe Library Query and Evaluation System Client (short: PLEASE Client) is a platform independent Java application which is easily distributed using the Java Web Start technology. Downloading, running the client and its installation process, if wanted, is initiated just by clicking on a link on our web page. The PLEASE Client requires a Java Runtime Environment 1.6, at least 1 GB RAM and access to the Internet to launch server requests. Consistent to the interfaces provided by the PLEASE server (Figure 6.2), the PLEASE Client as well consists of two modules: it combines the client functionalities of the "PLEASE Probe Library" with the "PLEASE Probe Match" in one graphical user interface (GUI). Easy access to the signature candidates is possible through a browsable phylogenetic probe tree. The retrieved oligonucleotide signatures can be matched against sequence datasets. The results of probe matches can be listed and interactively displayed in the phylogenetic tree.

The "Probe Library Client" (Figure 6.4) communicates via servlet API with the Java servlets on the Apache Tomcat server. When started, it first retrieves a list of the available phylogenetic trees from the server and loads a defined default tree with its node specific references. Other trees can be selected by the user, downloaded, and displayed. Tree datasets are stored and submitted in an adapted form of the NEWICK tree format [81]. The PLEASE client supports switching between multiple phylogenetic tree files and displays them as separate tabs. Although several different calculations can be made starting from a single tree file and sequence dataset, each of these calculations are referenced with their own (possibly redundant) tree datasets. This small redundancy was tolerated to facilitate the interface design.

The main function of the Client is to allow the user to interactively browse the phylogenetic tree and select groups or leaves (= sequences) of interest. Selections are sent to the server and node-specific information is returned and displayed: signatures including coverage information and thermodynamic information is displayed in a list. The specificity of a selected signature candidate, i.e. matched branches and sequences/leaves, can be highlighted in the currently displayed tree. Optionally, the signature list including their parameters can be stored.

The "PLEASE Probe Match Client" (Figure 6.5) is an online nucleic acid sequence evaluation tool for short oligonucleotide sequences. Currently, oligonucleotides with lengths from 15–20 nucleotides (nt) are supported, as this is a common length in many typical hybridization techniques. It provides an intuitive graphical user interface: The User pastes his DNA/RNA sequence as the search string in $5' \rightarrow 3'$ direction and has the opportunity to influence the match with additional parameters. We have tried to match those of the ARB Probe Match functionality [70]. Two options should be emphasized here: The option "Search Database" determines the database to be searched. The database need not have to match the one that was used for tree calculation. And the option "Use Weighted Mismatch" determines a value for the relative strength of the bonding of the duplex between the signature and the respective matched sequence. This value includes empirical knowledge from the behavior of fluorescent oligonucleotide probes for the detection of whole bacterial cells using FISH [5].

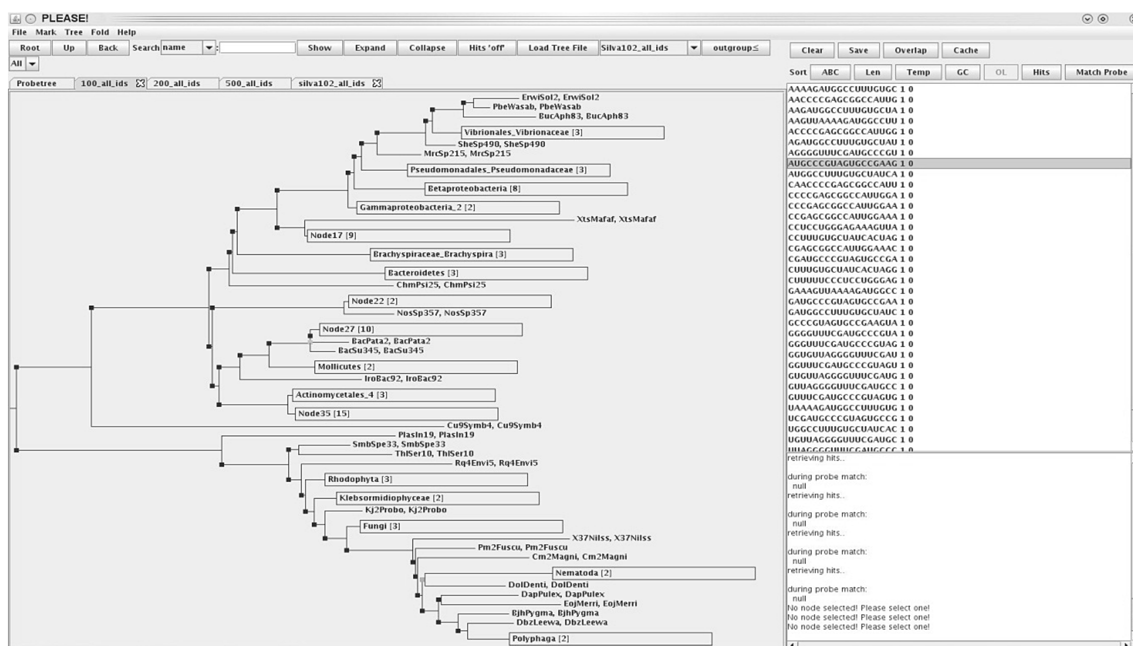


Figure 6.4: Screenshot of the PLEASE Probe Library user interface. The phylogenetic tree (left area) is for navigation. Inner nodes in the tree represent groups. Sequences (organisms) are positioned at the leaves. By selecting a node, a list of signature candidates is fetched (if available) and presented in the upper right area. Single signatures can be selected and more detailed information (e.g. matches and thermodynamic parameters) are presented in the lower right area. Right clicking on a signature opens the Probe Match UI (Figure 6.5). Switching between multiple datasets is down through tabs.

6.3 Results and Discussion

The *Probe Library Query and Evaluation System* (PLEASE) is a versatile client-server application for the retrieval and evaluation of signature candidates. These candidates can be the starting point for the design of oligonucleotide primers and probes.

The Probe Library (Section 6.2.1) was completely ported onto a modern web server and database architecture. Unlike the the old ARB Probe Library, PLEASE is now able to handle huge signature datasets that were computed from sequence databases and corresponding phylogenetic tree with more than $100,000$ entries.

Signature candidates can be directly accessed by clicking onto a phylogenetic tree node in the client user interface. If available, a list of candidates including scores and thermodynamic parameters is displayed. The user interface allows text-searches for defined group or leave node names (e.g. Eukarya or Firmicutes). Without direct access, traversing the phylogenetic tree may result in excessive "clicking" until the requested group or leave node is reached. The binary tree structure of the phylogenetic trees results in a lot of unnamed inner nodes without a group definition. It was proposed to hide or at least automatically expand and collapse such unnamed inner nodes. Due to the lack of time, this feature was not yet implemented.

PLEASE! ProbeMatch

Probe: ACGTACGTACGT — Save Results

PT-Server: Probe Tree (ref.) — Use weighted mismatches — Clear Entries

Mismatches: 1 — Check reverse complements — Check Probe

ACGTACGT... (MM1,PT1) ×

Full Name	Acc	Mis	Nmis	Wmis	Ecoli	Rev	ACGTACGTACGT
Rhabditis rainai	AF083008	1	0	0.2	591	0	AUCUGAGUC-========g===-UCGUCUUUU
Bacteriovorax sp. NF2	EF092442	1	0	1.0	180	0	AUACCGCAU-=========a==-AGAAAGGAU
Bacteriovorax sp. NF3	EF092443	1	0	1.0	180	0	AUACCGCAU-=========a==-AGAAAGGAU
Planococcus sp. KRPC10y	DQ375559	1	0	1.1	24	0	.........-===========C-UGGCGGCGU
uncultured bacterium	AY532553	1	0	1.1	121	0	GGUAGUAAC-=====G======-ACUCCCAAG
uncultured bacterium	AY532578	1	0	1.1	121	0	GGUAGUAAC-=====G======-ACUCCCAAG
uncultured bacterium	EU135137	1	0	1.1	1037	0	ACCGGUGGA-=========G==-GUUGCAUGG
Streptomyces sp. MAR3...	DQ448740	1	0	1.1	1439	0	CUCACACGG-U===========-CGUUGCAAC
uncultured bacterium	AB252433	1	0	1.5	121	0	GUCAGUAAC-======U=====-ACCGCUAAC
uncultured bacterium	AY540495	1	0	1.5	452	0	GGAAGAGAA-==U=========-GUUACGUAA
uncultured bacterium	DQ080179	1	0	1.5	452	0	GGAAGAGAA-==U=========-GUUACGUAA
unidentified bacterium	AY345540	1	0	1.5	1461	0	GUAGGCAGU-======C=====-GUAGGGUCC

1 - 12 (12) mis < 1 >

Figure 6.5: Screenshot of the PLEASE Probe Match user interface. This function can be used to fetch specificity information not only for signature candidates but also freely defined signature strings. Results (matched sequences) are displayed in a table, including exact information about possible mismatches.

By using CGI for queries and returning datasets as XML datasets, a universal interface is provided which may also be used by other applications beside the PLEASE Match Client. In the future, the PLEASE Probe Match Server might be modified to directly return a result dataset as Comma Separated Value (CSV). At the moment, this feature is (only) provided by exporting the results from the PLEASE Evaluation Client.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Over the last years changes on websites that provide biological data sets could be noticed. The maintainers switched from being mere providers of a single service to offering packages of multiple services which are interconnected. Most of these new services on the one hand arose from users' need to evaluate the data sets because of their increasing sizes — the available computer power could not keep up with the amount of available data that needed to be processed. It is therefore advantageous to give (limited) access to functionalities running powerful central computing systems and, if applicable, precalculate the provided data. On the other hand, it offers advantages for the users to have a continuous work flow, a direct connection between the data selection and the data evaluation.

A good example for this change is the ARB SILVA website [89], maintained by the Max Planck Institute for Marine Microbiology in Bremen. At first, it only provided a curated data set of aligned rRNA sequences and a corresponding phylogenetic tree. In 2008, a year after their launch, they presented a Web aligner which allowed users to upload and align their own sequences to the existing alignment in their dataset. In 2010, a taxonomy browser and search functionality was added to improve the selection and download of sequence sets.

The same effect, only from the opposite direction, could be observed at *probeBase* [68]. This website at the University of Vienna manages since 2003 a curated database with published oligonucleotide probes, mainly for rRNA gene sequences. A new service was added in 2007 which allows testing the specificity of the probes against ARB SILVA sequence data sets. This service, *probeCheck* [66], is based on the ARB PT-Server and thereby very similar to the probe match functionality offered by PLEASE (Section 6.2.2). Both were developed at the same time.

Being the ARB software environment [70] developers, our team at the TU München is in contact with both groups and this thesis was partly inspired by this fruitful collaboration. The thesis details new computational methods for the comprehensive and relaxed search for sequence- and group-specific oligonucleotide signatures in whole genome or marker gene sequence datasets: the Comprehensive and Sensitive Signature Search (CaSSiS, Figure 7.1). This functionality is of interest for both groups, but it might also be useful for others not mentioned

here. As with other services (e.g. the aligner), the computation of signatures based on large sequence datasets is for most users too expensive. The ARB SILVA website could provide pre-calculated sets of signatures along with the sequence data sets in their taxonomic browser. The user-sided computation of signatures would no longer be necessary. CaSSiS bridges the gap between ARB SILVA and the other services probeBase and probeCheck. Published probe candidates that were tested in the wet lab could be provided along with the signatures computed with CaSSiS. A prototypical approach, which was intended to test such a unifying service, was presented in this thesis with PLEASE (Chapter 6).

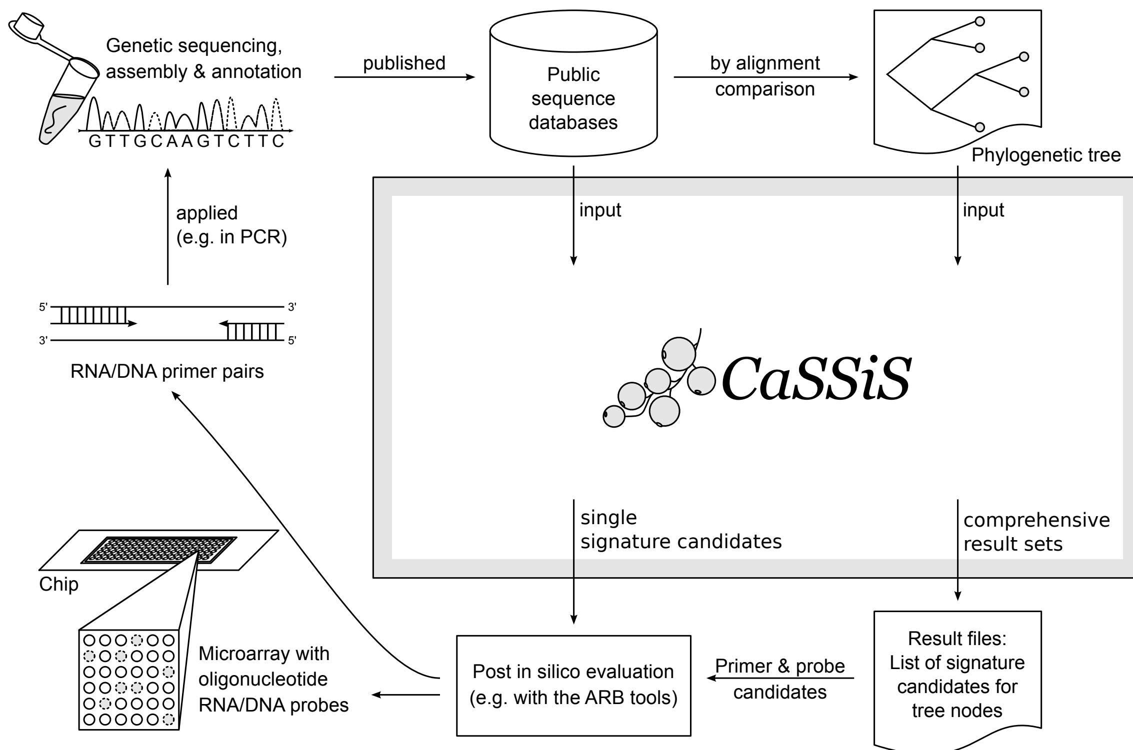


Figure 7.1: Schema of the CaSSiS and PLEASE primer and probe design pipeline.

7.2 Future Work

Due to the focus on rRNA sequence datasets in the project this thesis was developed for, the current CaSSiS implementation is limited to nucleic acid sequence data. Future approaches could extend CaSSiS to also process other types of sequence strings. The biggest obstacle is the underlying search index which has to support the relevant sequence type. CaSSiS itself internally handles signatures as mere strings and does not interpret them. Thus, at least the code basis of CaSSiS could easily be modified to support other types of sequence data as well.

Both solutions presented in this work, CaSSiS-BGRT and the CaSSiS-LCA, are based on the ARB PT-Server[1]. Especially when searching under very relaxed conditions (by allowing

[1]Just to be exact: CaSSiS uses MiniPT, a descendant of the PT-Server with identical features, but a lower

multiple mismatches) or when guaranteeing a high Hamming distance, querying the PT-Server becomes the runtime-critical computational step. This has been adressed by distributing the index in Chapter 5. But replacing the index by a faster approximate search index could further accelerate the overall performance. An abstract search index interface was created to facilitate this. Two promising approaches for this future task are PtPan [33] and SeqAn [30].

For each node within a tree, CaSSiS only returns the signature(s) with the highest coverage. CaSSiS allows the definition of a specificity range, but a sensitivity range can not yet be defined. The reason for this was a design decision at the beginning of this thesis, as allowing outgroup hits ranges was considered to be more important.

Especially in RNA/DNA microarray applications, combinations of probes are used to increase the coverage or the identification rate of single organisms or groups. CaSSiS supports this by allowing the design of signature candidates for complete phylogenies, i.e. inner group nodes at different hierarchical levels can be combined. A line for future work could be the search for combinations of multiple probes to better cover all organisms in a target group, as for some groups of organisms probes with perfect coverage simply do not exist.

memory footprint. MiniPT allows direct access to the internal data structures, a feature not available in the PT-Server.

Appendix

Appendix A

Supplementary Material

A.1 Test systems

Multiple test systems were used between 2009 and 2012 to perform runtime and memory consumption measurements of our CaSSiS implementations. The following list shows the characteristics of the individual test systems, ordered by the first appearance (section) in which they were used for measurements.

thinkpad	
Processor	Intel® Core™2 Duo CPU L9400 @ 1.86GHz (Montevina)
Cores	2 cores
Cache	6 MB L2 cache
RAM	8 GB DDR3-RAM
OS	GNU/Linux (Ubuntu 10.10, 64 bit), MS Windows 7 (64 bit)
Used in ...	Section 3.3.5

nehalem1	
Processor	Intel® Xeon® CPU X5670 @ 2.93GHz (Westmere)
Cores	2x 6 cores (24 threads with HyperThreading turned on)
Cache	12 MB L3 cache
RAM	36 GB DDR3-RAM
OS	GNU/Linux (Ubuntu 11.10, 64 bit)
Used in ...	Section 3.4

atbode223 + atbode230	
Processor	Intel® Core^{TM} i7 CPU 920 @ 2.67 GHz (Bloomfield)
Cores	4 cores (8 threads with HyperThreading turned on)
Cache	1 MB L2 cache, 8 MB L3 cache
RAM	24 GB DDR3-RAM
OS	GNU/Linux (Ubuntu 10.04.2 LTS, 64 bit), later GNU/Linux (Ubuntu 11.10, 64 bit)
Used in ...	Section 4.4; Section 6.2

infiniband (opteron nodes)	
Cluster	32 AMD Opteron 4-way nodes 4x SDR Infiniband interconnect
Processor	AMD Opteron^{TM} Processor 850 @ 2.4 GHz (SledgeHammer) 4 processors per node
Cores	1 core
Cache	1 MB L2 cache
RAM	8 GB DDR-RAM
OS	GNU/Linux (Ubuntu 8.04.2 LTS, 64 bit)
Used in ...	Section 5.3.1

A.2 Signature evaluation

This section contains in revised form the analysis of the signatures that were computed with the CaSSiS-BGRT approach (Chapter 3). It is part of the supplementary material from the joint publication [12] in Bioinformatics. The analysis was done by Harald Meier, whom I thank for allowing me to add it to my thesis.

A.2.1 Computing the signature dataset

We computed a comprehensive oligonucleotide signature collection from the SILVA SSURef_102 rRNA database[1]. This database contains 460, 783 SSU rRNA sequence entries, each including annotation information. It is assumed, that each sequence represents an unique organism. The database also contains a phylogenetic tree with the same number of leafs, each referring to the an organism. Internal nodes with identifiers in the tree refer to groups of organisms. The comprehensive collection of signature sequences has been calculated using CaSSiS-BGRT under the following constraints:

- Find all sequence specific 18-mer signatures.
- For every phylogenetic group defined by an internal tree node, find all 18-mer signatures with full group coverage (which appear in every sequence of the group).

[1] `http://www.arb-silva.de/download/arb-files/`

- For phylogenetic sequence groups without full coverage, find those signatures with the highest coverage.
- Find signatures for a range of different specificity levels (0–10 allowed outgroup matches).
- A signature should not have a mismatch to ingroup sequences.
- A signature should have at least one mismatch to an outgroup sequences.
- A signature should not occur twice in a sequence entry or its inverse sequence.

A.2.2 Comparison with relevant published signatures

For selected target groups, we compared signatures that were computed with the CaSSiS-BGRT approach with relevant entries in probeBase [68], a curated oligonucleotide probe database with online access. We also checked the available literature concerning availability, coverage and specificity.

One signature proposed by CaSSiS-BGRT is already available through probeBase. Another evaluated signature even exhibits a better coverage of the target group than previously published ones. For groups without any available published signatures, such as Coprothermobacter, we found new promising signatures. Furthermore, a valuable signature was identified which could only be found under relaxed search conditions. For each signature sequence below, the following information is specified:

Signature Source:

- Signature-Sequence (target name)
- # of ingroup hits / total # of ingroup members
- # of outgroup hits
- # of mismatches with next similar outgroup sequence

CaSSiS found group specific signatures which exhibit a better coverage and the same or better specificity than signatures published previously for the same target. One probe/signature — *pB-00011* (probeBase Accession no.) [118] — was found in probeBase/literature specifying the genus Deinococcus as target group. This is a 35 mer, which matches only 37 out of the 238 sequences of the group Deinococcus perfectly. Therefor only 5 out of 42 described species of the genus Deinococcus are covered.

probeBase pB-00011:

- `GTACGTTGGCTAAGCGCAGGATGCTGTGCTTGGCG` (Genus Deinococcus)
- 37 of ingroup hits / total 238 of ingroup members

- 0 of outgroup hits
- 2 of mismatches with next similar outgroup sequence

CaSSiS identified a signature which covers 85% of the 238 sequences of the group "Deinococcus", and 31 out of the 42 described Deinococcus species. The signature sequence introduced above has one strong central mismatch to the next outgroup sequence, denoted as "an uncultured Acidobacterium", and two central mismatches ore more to other outgroup sequences. For the group Deinococcaceae/Deinococcus CaSSiS clearly finds signatures superior to already published ones in both coverage and specificity.

CaSSiS signature:

- `UGGACAGAAGGUGACGCU` (Genus Deinococcus)
- 199 of ingroup hits / total 238 of ingroup members
- 0 of outgroup hits
- 1 of mismatches with next similar outgroup sequence

CaSSiS found (new) signatures for phylogenetic or taxonomic groups, for which no group specific probe has been published so far. A group of 368 closely related 16S rRNA sequences which have a common internal node in the phylogenetic tree specified "Coprothermobacter" has been selected. CaSSiS found several signatures at different specificity levels for this group. The signature sequence with the best group coverage found by CaSSiS is listed below:

Signature Source:

- `UACCCCAGUAGAAAGGGA` (Group Coprothermobacter)
- 340 of ingroup hits / total 368 of ingroup members
- 1 of outgroup hits
- 1 of mismatches with next similar outgroup sequence

These signatures match more than 90% of the sequences of the Coprothermobacter group, but match additionally one outgroup sequences. Signature candidates without outgroup matches cover only about 15% ingroup sequences. This example shall demonstrate that in the signature collection calculated by CaSSiS signatures with reasonable coverage can be retrieved for sequence groups, for which to the best of our knowledge an oligonucleotide signature or probe sequence has not been published so far.

CaSSiS' ability to perform additional searches under relaxed conditions makes the identification of highly valuable oligonucleotide signatures possible. *EUB338* (probeBase Acc. no. *pB-00159*) [4] is a widely used oligonucleotide probe, which targets a signature present in more than 90% of available bacterial 16S rRNA sequences. Even in 2008, Amann and colleagues described this probe in Nature Reviews Microbiology as highly specific, since it did not match any sequence outside of the Domain Bacteria (i.e. had no outgroup match).

CaSSiS found a signature sequence, which is — compared with the *EUB338*-target — shifted by one position to the 3'-end. It covers even more group sequences than the already published *EUB338*-target sequence. Surprisingly it is listed in the result table as "signature with the highest group coverage and three outgroup matches". In the database the matched outgroup sequences are specified as eukaryotic entries. Matching the original target sequence of *EUB338* against the whole SSURef_102_SILVA database, we received about 400 less group hits, but surprisingly the same three (unproblematic) outgroup matches within the domain Eukarya.

CaSSiS:

- **Signature source**
- `CUCCUACGGGAGGCAGCA` (Domain Bacteria)
- 356,188 of ingroup hits / total 391,167 of ingroup members
- 3 of outgroup hits
- 1 of mismatches with next similar outgroup sequence

probeBase: pB-00159 (EUB338):

- **Signature source**
- `ACUCCUACGGGAGGCAGC` (Domain Bacteria)
- 355,789 of ingroup hits / total 391,167 of ingroup members
- 3 of outgroup hits
- 1 of mismatches with next similar outgroup sequence

The probe *VP403*, published in 2010 by Arnds and colleagues [7], was just recently submitted to probeBase (accession number *pB-02645*). It is a 20-mer targeting the 16S rRNA of Verrucomicrobium, most Prostecobacter spp., and uncultured relatives (In SSU_Ref102 equivalent with the group Verrucomicrobiaceae_2, a subgroup of the family Verrucomicrobiaceae). The authors show in their publication that the probe works well for whole cell detection by fluorescence in situ hybridization [7]. CaSSiS found for the same group (Verrucomicrobiaceae_2)

three 18-mers, which are the three possible substrings of the 20-mer signature sequence targeted by *VP403*. All three have the same coverage and specificity (Note: we conducted CaSSiS test runs for 18-mers only). This example also indicates, that CaSSiS finds signatures, which work in wet lab applications (hybridize with complementary oligonucleotide probes, shown for FISH [7].

CaSSiS:

- `GUGGAGGAUAAGGUCUUC` / `CGUGGAGGAUAAGGUCUU` / `UGGAGGAUAAGGUCUUCG` (NODE185359: Verrucomicrobium, Prostecobacter spp., and relatives)
- 71 of ingroup hits / total 88 of ingroup members
- 1 of outgroup hits
- 1 of mismatches with next similar outgroup sequence

probeBase: pB-02645 (VP403):

- `CGUGGAGGAUAAGGUCUUCG` (Verrucomicrobium, Prostecobacter spp., and relatives)
- 71 of ingroup hits / total 88 of ingroup members
- 1 of outgroup hit
- 1 of mismatches with next similar outgroup sequence

In summary, the spot-tests performed to illuminate the quality of the 18-mer-signature collection confirmed the assumption, that CaSSiS performs well in finding valuable sequence and group specific signatures, even in large hierarchically clustered datasets. CaSSiS finds signatures which are comparable or identical to already published useful signatures, which have been shown to work in the wet lab by at least one hybridization technique (see *EUB338*, *VP403*).

For some groups, for which signatures are already published, CaSSiS finds signatures of higher in silico coverage and specificity (see Deinococcus). In addition CaSSiS found an in silico promising new signature sequences, for so far untargeted groups of microorganisms (see Coprothermobacter). Some highly valuable probes were found because of CaSSiS special ability to search under relaxed conditions, too.

Appendix B

The CaSSiS software package

During the last four years of the development of CaSSiS, its tools and functions have gone through numerous changes. To keep track of these changes (Section B.6) and to have a central hub for the source code and binary releases, a Git repository and a project homepage hosted by the Lehrstuhl für Rechnertechnik und Rechnerorganisation / Parallelrechnerarchitektur (TU München) was created. This chapter is an overview of the tools and their functionalities that are provided by the CaSSiS project. Parts of this chapter are also presented as HowTos and supplementary documents on the CaSSiS homepage.

Figure B.1: Homepage of the CaSSiS project: `http://cassis.in.tum.de/` (Screenshot; September 2012)

B.1 Availability

The CaSSiS homepage provides multiple download options for CaSSiS. the latest release is available as source code as well as a GNU/Linux (64 bit Ubuntu) binary package. Older releases (most of previous the stable and maintenance releases) are available a 32 bit and 64 bit binary packages. The following two sections detail the content of the (latest) binary release package and how to build CaSSiS from source. The latter might be important on Linux distributions where the binary releases may have different library dependencies.

B.1.1 Building CaSSiS from source

The CaSSiS project relies on the `cmake`[1] build system. CMake allows the platform- and compiler-independent generation of build files. In case of CaSSiS, large parts of the code are compatible with the GNU project C and C++ compilers and Microsoft Visual C++. Due to the search index (a core functionality) with its highly adapted code, a complete platform independence was not yet archived. This may change in future releases by the use of other search index approaches.

Please make sure to have at least CMake version 2-6 installed. In CaSSiS, CMake is configured to prevent builds in the same directory as the source code was extracted. In such a case, the build will be halted with an error message.

The following example shows how a build could look like on a GNU/Linux system. A "Makefile" is generated during the process that allows building all necessary CaSSiS components. The output messages were omitted in the example and the users home directory `/home/$USER/cassis` was assumed as the destination of our build. Please adapt all paths according to your needs.

It is assumed, that the package containing the CaSSiS source code was already downloaded to a temporary location, here `/tmp`. An intermediate build folder `/tmp/build` will be created and used. If no further destination is defined, the installation will be done in the respective folders under `/usr`. The parameter `-DCMAKE_INSTALL_PREFIX` can be used to define an alternative installation destination. This is done in the example (set to `/home/$USER/cassis`):

```
~$ cd /tmp/
/tmp$ tar xvfj CaSSiS-0.5.1-src.tar.bz2
/tmp$ mkdir build
/tmp$ cd build/
/tmp/build$ cmake ../CaSSiS-0.5.1-src -DCMAKE_INSTALL_PREFIX=/home/$USER/cassis
/tmp/build$ make install
/tmp/build$ cd /home/$USER/cassis/
~/cassis$ mv lib/* bin/* .
~/cassis$ rm -r bin lib
```

[1] `http://www.cmake.org/`

The command line tools (Section B.2) of CaSSiS only depends on standard GNU C/C++ libraries. When built on an Ubuntu 12.04 64 bit system, CaSSiS is linked against (sample output):

```
~/cassis$ ldd cassis
  linux-vdso.so.1 =>  (0x0000...)
  libCaSSiS.so.0 => ./libCaSSiS.so.0 (0x0000...)
  libminipt.so => ./libminipt.so (0x0000...)
  libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x0000...)
  libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x0000...)
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x0000...)
  libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x0000...)
  /lib64/ld-linux-x86-64.so.2 (0x0000...)
```

B.1.2 Binary packages

There are two versions of archives, one for 32 bit Linux architectures and one for 64 bit architectures. CaSSiS was built with Ubuntu 11.10 (Natty Narwhal). The CaSSiS binary tar archive contains the following files:

```
cassis               The CaSSiS command line tool.
cassis-gui           A graphical user interface for CaSSiS. (buggy)
libCaSSiS.so         (symbolic link)
libCaSSiS.so.0       (symbolic link)
libCaSSiS.so.0.4.0   The CaSSiS Library v0.4.0
libminipt.so

arb\_license.txt      The ARB license (for libARBDB.so and libminipt.so)
gpl.txt              The GNU General Public License v3
lgpl.txt             The GNU Lesser General Public License v3
readme.txt           Help file.
```

B.2 CaSSiS command line interface

The CaSSiS command line interface provides the parameter `-all` which, if enabled, generates and queries all possible signature strings of a defined length. By default, only signatures extracted from the respective datasets will be used. Under strict search conditions (i.e. no allowed mismatches between signatures and the target sequences), the results will be the same. Querying all signatures especially under relaxed search conditions will lead to an extreme runtime overhead.

There are two versions of archives, one for 32 bit Linux architectures and one for 64 bit architectures. CaSSiS was built with Ubuntu 11.10 (Natty Narwhal). The CaSSiS binary tar archive contains the following files:

```
cassis              The CaSSiS command line tool.
cassis-gui          A graphical user interface for CaSSiS. (buggy)
libCaSSiS.so        (symbolic link)
libCaSSiS.so.0      (symbolic link)
libCaSSiS.so.0.4.0  The CaSSiS Library v0.4.0
libminipt.so

arb\_license.txt       The ARB license (for libARBDB.so and libminipt.so)
gpl.txt               The GNU General Public License v3
lgpl.txt              The GNU Lesser General Public License v3
readme.txt            Help file.

USING CASSIS:

Comment: Please make sure to point 'LD_LIBRARY_PATH' to the correct directory,
if necessary. (export LD_LIBRARY_PATH="/path/to/cassis/lib)

CaSSiS usage: cassis {1pass|create|process|info} [options]

cassis 1pass
Mandatory: -seq [... -seq] -tree
Optional:  -all -dist -gc -idx -len -mis -og -out -rc -temp -wm
Comment:   '1pass' uses the faster CaSSiS-LCA algorithm.

cassis create
Mandatory: -bgrt -seq [... -seq]
Optional:  -all -dist -gc -idx -len -mis -rc -temp -wm

cassis process
Mandatory: -bgrt -tree|-list
Optional:  -og -out

cassis info
Mandatory: -bgrt

Options (alphabetical):
-all              Evaluate all 4^len possible signatures.
                  (Not recommended, may take forever... Default: off)
-bgrt <filename>  BGRT file path and name.
-dist <number>    Minimal mismatch distance between a signature candidate
                  and non-targets. Must be higher than "-mis <number>".
```

```
                  (Default: 1.0 mismatches)
-gc <min>-<max>   Only allow signatures within a defined G+C content range.
                  (Default: 0 -- 100 percent)
-idx <name>       Defines the used search index:
                      minipt = "MiniPt Search Index" (Default)
-len {<len>|<min>-<max>}
                  Length of the evaluated oligonucleotides. Either a
                  fixed length or a range. (Default: 18 bases)
                  Lengths must be between 10 and 25 bases.
-list <filename>  Instead of a phylogenetic tree, a list with comma separated
                  identifiers can be used to define groups the should be
                  queried. Each line in the list defines one group.
                  The output format is set to 'sigfile'.
                  (Comment: Only available in 'cassis process'.)
-mis <number>     Number of allowed mismatches within the target group.
                  (Default: 0.0 mismatches)
-og <limit>       Number of outgroup hits up to which group signatures are
                  computed. (Default: 0)
-out <format>     Defines the output format.
                      classic  = "Classic CSV format" (Default)
                      detailed = "Detailed CSV format"
                      sigfile  = "Signature file for each group/leaf"
-rc               Drop signatures, if their reverse complement matches
                  sequences not matched by the signature itself.
                  (Default: off)
-seq <filename>   MultiFasta file as sequence data source Multiple sequence
                  sources can be defined.
-temp <min>-<max> Only allow signatures with a melting temperature within
                  the defined range. (Default: -273 -- 273 degree Celsius)
-tree <filename>  Signature candidates will be computed for every defined
                  (i.e. named) node within a binary tree. Accepts a binary
                  Newick tree file as source.
-v                Verbose output
-wm               Enable "weighted mismatch" values. (Default: off)

Caution: Combining the "-gc" and "-temp" filters can cause unwanted side
        effects because they influence each other.
```

B.3 The CaSSiS result files

CaSSiS was designed to compute comprehensive signature sets based on large sequence databases. To be able to further process/parse the result files of CaSSiS, the simple but flexible CSV format was chosen. Smaller files can be read/edited with spreadsheet software (Excel, OpenOffice, ...). Larger files can still be easily parsed with an appropriate editor or small scripts. When directly run without further parameters, CaSSiS creates two different types of result files in the CSV format in the current directory: 'result_array.csv' and 'results_xxx.csv' where 'xxx' is defined by the outgroup hits range.

result_array.csv: This file gives an overview of the results. The rows represent the range from 0 to max. outgroup hits. Columns represent the nodes of the hierarchical cluster. All information within this file is also included in the following ones.

Group		**# outgroup hits**					
ID	**size**	0	1	2	3	4	5
Bacteria	74	69	22	0	1	0	0
Proteobacteria	31	24	29	11	25	29	25
Eukarya	25	25	3	0	0	0	0
Actinobacteridae	18	18	18	18	17	17	18
Firmicutes	15	13	13	11	11	12	10
Bacilli	12	12	12	12	9	10	11
Metazoa	12	11	10	9	9	9	11
Gammaproteobacteria_1	11	11	11	10	10	8	9
Alphaproteobacteria	10	9	8	8	6	7	9
Betaproteobacteria	8	8	8	8	8	8	8
Arthropoda	6	6	6	6	6	6	6
Burkholderiales	6	5	6	6	6	6	6
Lactobacillales	6	5	5	6	5	6	6
Streptomycineae_Streptomycetaceae	6	6	6	6	6	6	6
Streptomyces	4	2	4	4	4	4	4
Actinomycetales_4	3	3	3	3	3	3	3
Actinomycetales_Micrococcineae_1	3	3	3	2	3	3	3
Bacteroidetes	3	3	2	2	3	2	2
Brachyspiraceae_Brachyspira	3	3	3	3	3	3	3
Corynebacterineae	3	3	3	3	3	3	3
Cyanobacteria	3	3	3	3	2	2	3
Fungi	3	3	3	2	3	3	3
Microbacteriaceae	3	3	3	3	3	3	3
Pseudomonadales_Pseudomonadaceae	3	3	3	3	3	3	3
Rhizobiales_1	3	3	3	3	3	3	3
Rhodophyta	3	3	3	3	2	3	3
Streptococcus	3	3	3	3	3	3	3
Vibrionales_Vibrionaceae	3	3	3	3	3	3	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table B.1: The CaSSiS `result_array.csv` file contains for every node the number of ingroup matches, that occurred over the entire range of allowed outgroup matches. For example in the group 'Proteobacteria' at least one signature was found, that matches 29 of the 31 sequences with one outgroup hit.

results_xxx.csv: This file contains the actual results of CaSSiS, separated by their number of outgroup hits 'xxx'. The first column contains the node identifier within the hierarchical cluster. The second and third column show the number of organisms within the group/node (for leaves/organisms: 1) and the number of actually hit organisms by the signatures for this node. The appropriate signatures follow subsequently.

B.4 CaSSiS graphical user interface

As a prototypical implementation, an intuitive graphical interface based on the CaSSiS library was created (Figure B.2). It was designed to allow single freely defined queries. The CaSSiS UI allows group selection within a loaded phylogenetic tree or its definition as a list of identifiers. By defining a range of non-target hits, users may influence the specificity. Single requests are usually processed within a second. The result is a list of signatures (Figure B.3) with maximum coverage (sensitivity) for each entry within the range of allowed non-target matches and their thermodynamic characteristics.

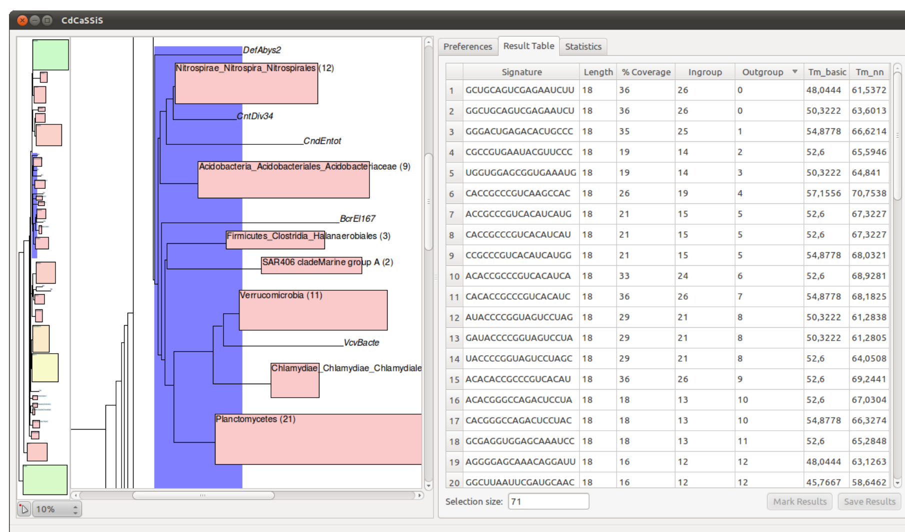


Figure B.2: Screenshot: The CaSSiS user interface. Left area: group selection within a phylogenetic tree. Right area: tabs with signature candidates for a selected phylogenetic group. Not visible: Preferences and statistical data.

B.5 CaSSiS bgrt2Graphviz

The tool `bgrt2graphviz` is a small helper that allows the conversion of BGR-Tree structures into Graphviz (`.gv`) files. These files can then be further processed into Scalable Vector Graphics (SVG) or other graphic formats. The Figures B.4 and B.5 show two different representation of a BGRT tree built from 10 rRNA sequences.

```
Usage: bgrt2Graphviz <bgrt-file> <Graphviz-file>
```

ID	Size	Ingroup	Signatures...		
Bacteria	74	69	UCCUACGGGAGGCAGCAG	CUCCUACGGGAGGCAGCA	...
Proteobacteria	31	24	CGCCCGUCACACCAUGGG	GCCCGUCACACCAUGGGA	...
Eukarya	25	25	GGGCAAGUCUGGUGCCAG	GGCAAGUCUGGUGCCAGC	...
Actinobacteridae	18	18	GCCGGGGUCAACUCGGAG	GGGGUCAACUCGGAGGAA	...
Firmicutes	15	13	CAGCAGUAGGGAAUCUUC	GCAGCAGUAGGGAAUCUU	...
Bacilli	12	12	GGCAGCAGUAGGGAAUCU	AGGCAGCAGUAGGGAAUC	
Metazoa	12	11	GGUAGUGACGAAAAAUAA	GUAGUGACGAAAAAUAAC	...
Gammaproteobacteria_1	11	11	AUGAAUUGACGGGGGCCC	UGAAUUGACGGGGGCCCG	
Alphaproteobacteria	10	9	AGAGGUGAAAUUCGUAGA	AGGUGAAAUUCGUAGAUA	...
Betaproteobacteria	8	8	AGCAGUGAAAUGCGUAGA	AGCAGUGGGGAAUUUUGG	...
Arthropoda	6	6	AACACGGGAAACCUCACC	AAGAACGAAAGUUAGAGG	...
Burkholderiales	6	5	ACGCUCAUGCACGAAAGC	ACUGACGCUCAUGCACGA	...
Lactobacillales	6	5	AAUGCGUAGAUAUAUGGA	ACGCUGAGGCUCGAAAGC	...
Streptomycineae_Streptomycetaceae	6	6	AGAUCAGCAUUGCUGCGG	AGCAUGUGGCUUAAUUCG	...
Streptomyces	4	2	ACAAUGAGCUGCGAUGCC	CAAUGAGCUGCGAUGCCG	...
Actinomycetales_4	3	3	AAAGUCGGCAACACCCGA	AAGUCGAGCGGAAAGGCC	...
Actinomycetales_Micrococcineae_1	3	3	AAACGUUGGGCACUAGGU	AACGUUGGGCACUAGGUG	...
Bacteroidetes	3	3	AAUUCGAUGAUACGCGAG	AGCGGUGAAAUGCAUAGA	...
Brachyspiraceae_Brachyspira	3	3	AAAAAAGUUGCCUCAGUU	AAAAACCUUACCUGGGUU	...
Corynebacterineae	3	3	AACACGUGGGUGAUCUGC	AAGGCGGGUCUCUGGGCA	...
Cyanobacteria	3	3	AAGCGGUGGAGUAUGUGG	AAGGAGGAAUCGCUAGUA	...
Fungi	3	3	CUAGAGCUAAUACAUGCU		
Microbacteriaceae	3	3	AAGGCAGAUCUCUGGGCC	ACCGAUGGCGAAGGCAGA	...
Pseudomonadales_Pseudomonadaceae	3	3	AAAGCACUUUAAGUUGGG	AACCCUUGUCCUUAGUUA	...
Rhizobiales_1	3	3	AACCCCGGAACUGCCUUU	AACUGCCUUUGAUACUGG	...
Rhodophyta	3	3	AAUACGUCCCUGCCCUUU	AAUUGACGGAAGGGCAUC	...
Streptococcus	3	3	AAAGCGGCUCUCUGGUCU	AAAGCUCUGUUGUAAGAG	...
Vibrionales_Vibrionaceae	3	3	AAAGCCCGGGGCUCAACC	AACCCUUAUCCUUGUUUG	...
⋮	⋮	⋮	⋮	⋮	

Table B.2: A separate result file exists for every number of outgroup hits. In the example above, an excerpt from the file `results_0.csv`, all signatures match without outgroup hits.

	A	B	C	D	E	F	G
1	Index	Species/Group ID	Size	Ingroup	Signatures...		
2	2	Bacteria	278862	248618	CCUACGGGAGGCAGCAGU		
3	582330	Eukarya	34451	31030	GUGGUGCAUGGCCGUUCU		
4	582359	Opisthokonta	20523	4862	GCGCGCUACACUGAAGGA		
5	582362	Metazoa	12246	4862	GCGCGCUACACUGAAGGA		
6	571775	Crenarchaeota	5192	4361	UGGUGUCAGCCGCCGCGG		
7	557725	Archaea	12303	4246	CGGUGCCAGCCGCCGCGG		
8	557731	Euryarchaeota	7013	4133	ACCGGUGCCAGCCGCCGC		
9	624663	Archaeplastida	4324	3958	AGAACGAAAGUUGGGGGC		
10	473823	Actinobacteria_Actinobacteria	26798	3717	GCCGGUGGCCCAACCCCU		
11	16	Proteobacteria	98192	3668	AGAGUAUGGUAGAGGGUG		
12	473832	Actinobacteridae	22838	3542	GCGACAUUCCACGUCGUC		
13	411362	Firmicutes_Bacilli	22697	3166	GGGUCAUUGGAAACUGGA		
14	582378	Arthropoda	6088	3104	CGGGACUCAUCCGAGGCC		
15	607212	Fungi	8092	2614	GCUCAAGCCGAUGGAAGU		
16	34	Gammaproteobacteria_1	34345	2595	GGGGGUAGAAUUUCAGGU		
17	473835	Actinomycetales_1	11758	2544	GAGUUCGGUAGGGGAGAU		
18	68729	Betaproteobacteria	18283	2454	UGGCAGAGGGGGGUAGAA		
19	473849	Corynebacterineae	5449	2051	GGCGAUACGGGCAUAACU		
20	122908	Alphaproteobacteria	26902	1562	AGAAUUUCUAGUGUAGAG		
21	411388	Lactobacillales	8390	1494	AUCUUACCAGAAAGGGAC		
22	406458	Anaerococcus_1	1374	1201	UCAAAAAGCCUGUCCCAG	CUCAAAAAGCCUGUCCCA	
23	68743	Burkholderiales	13135	1018	GGCGAAAGCUUUGCUAAU	AGGCGAAAGCUUUGCUAA	AACGAGGCGAAAGCUUUG
24	607217	Ascomycota	5115	922	CCGUUCGGCACCUUACGA	CCCGUUCGGCACCUUACG	
25	607220	Ascomycota	5052	922	CCGUUCGGCACCUUACGA	CCCGUUCGGCACCUUACG	

Figure B.3: Comprehensive signature computation result (CSV dataset, 0 mismatches, 1 non-target hit)

```
Sample session:
  $ ./bgrt2graphviz tree.bgrt output.gv
  $ twopi -Tsvg output.gv -o output.gv.svg
```

Graphviz[2] is a graph visualization software licensed under the "Eclipse Public License".

B.6 Changelog

This is a summary of the changes and updates during the development of CaSSiS in reverse chronological order. CaSSiS was managed with a Git repository. Its creation and the first commit is dated back to the 21.November 2009. The changelog was derived from the commit messages and combined where applicable.

CaSSiS 0.5.0 (Released 6.August 2012; Stable release)

- Added `-list` parameter to allow processing without phylogenetic trees.
- Updated ARB compatibility.
- Disabled warnings when processing huge sequence datasets.
- Fixed mismatch parameter settings.
- Increased MiniPT tree depth (and thereby the signature length limit) to 25.
- Fixed a bug in the melting temperature formula.
- Added a test tool for thermodynamic parameters.

[2] `http://www.graphviz.org/`

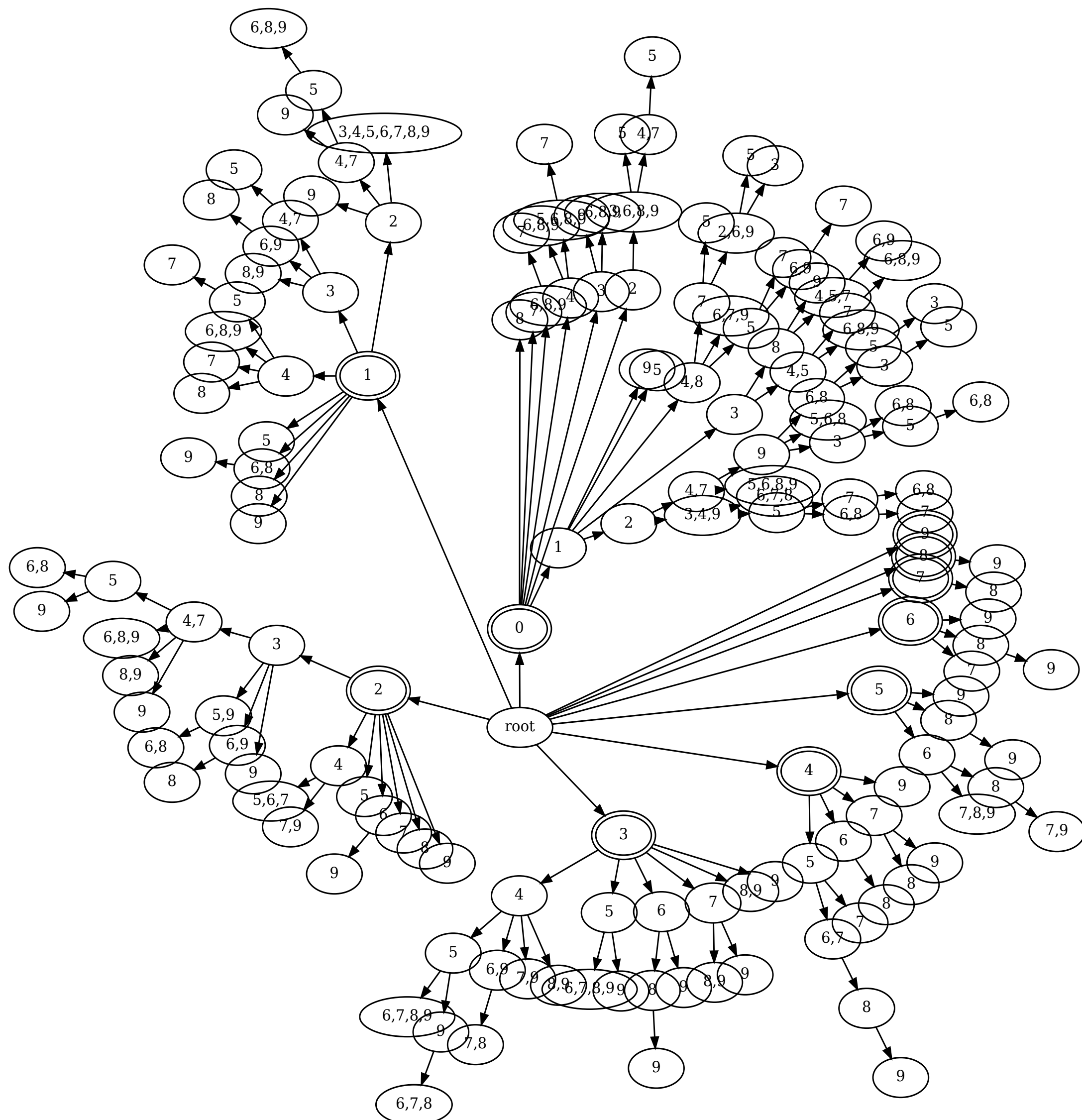


Figure B.4: Rooted circular visualization of a BGRT built from 10 sequences. First level nodes are represented by double circles.

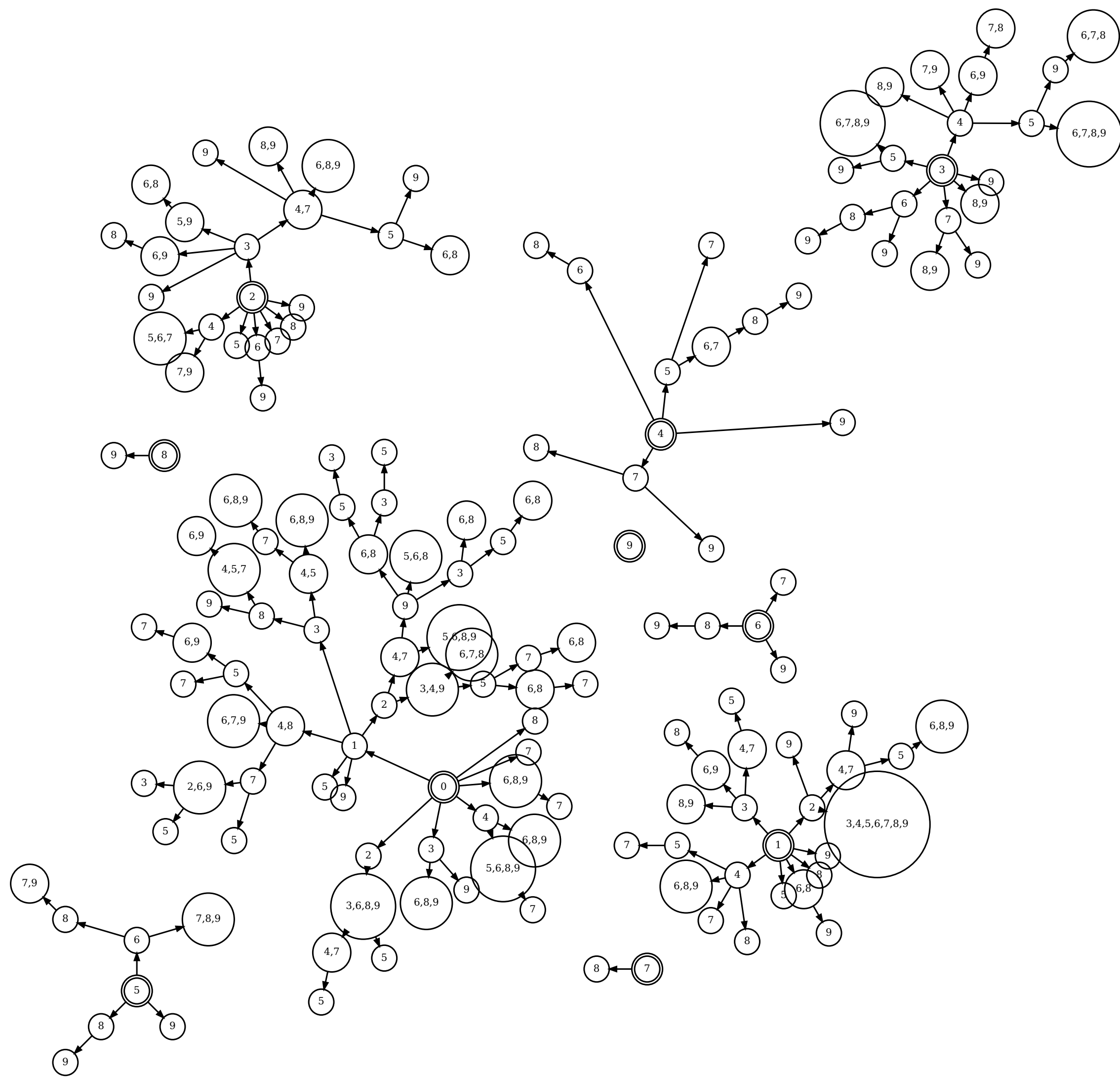


Figure B.5: Unrooted visualization of a BGRT built from 10 sequences. First level nodes are represented by double circles.

- Minor changes in the graphviz visualization tool.
- Added (untested) DUP support.
- Added stubs for the index server.
- Major refactoring.
- Added another form of detailed output: Signature files.
- Minor changes in cmake build chain.
- Unofficial pThread support.
- Various fixes for the (untested) pThreads version.
- Added a second output format: Detailed CSV
- Updated various copyright headers and comments.
- Updated (better) error messages instead of mere assertions.
- Various bugfixes that address the formatting of the input file.

CaSSiS 0.4.0 (Released 14.January 2012; Stable release)

- CaSSiS is now under the LPGL v3 license.
- Added `bgrt2graphviz` tool for testing purposes.
- Updated/fixed the untested pThreads support.
- Fixed error handling.
- Removed obsolete comments.
- Code clean-up.
- Various changes in the ARB Pt-Server interface.
- Moved MiniPT into its own namespace. Avoids collisions due to global (legacy) variables.
- Added new parameter: check all signatures (all 4^{length} possibilities).
- Disabled CaSSiS tree load when creating a BGRT.
- Reverted the (too high) memory estimation when building a MiniPT index.
- Updated the output of statistical information.
- Removed erroneous gene memory consumption calc. from the MiniPT index.

- Added numbering to the snapshots (debug version only).
- Fixed a bug in the CMakeslist.
- Redesigned the memory and runtime dump.
- Refactoring: Merged the BGRT create and 1pass process in CaSSiS.
- Updated the ARB interface (compatibility).
- Added a (unified) CaSSiSTree class and removed obsolete PhyloTree class.
- Removed warnings when building the MiniPt Index.
- Updated the parameter parser.
- Various minor fixes. Fixed a dependency problem.
- Renamed binary project to: `CaSSiS-CLI` (command line interface tool).
- Added CaSSiS-tool as a unified binary for the BGRT and LCA approach.
- Added IndexInterface base class as an interface for search indices.
- Added ARB database support to CaSSiS-LCA. Untested.
- Major refactoring of the CaSSiS library.

CaSSiS 0.3.3 (Internally released 24.October 2011; Maintenance release)

- Added the MiniPT search index (derived from the ARB PT-Server).
- Removed obsolete code.
- Removed some warnings (at compile time).
- Do not handle warnings as errors by default (at compile time).
- Added placeholders for the SeqAn and the PtPan search indices.
- Updated/fixed the Newick tree importer. Also updated the internal tree structure.
- Various updates that should keep the PT-Server compatible with the ARB version.
- Removed some of the debug code.
- Added a sparse table to the CaSSiS-LCA. This allows LCA-searches in O(1).
- Removed unnecessary code.
- Applied some fixes that should reduce the memory consumption.

- Added log messages during runtime. Disabled by default.
- Various modifications that are necessary to support “outgroup matches” in CaSSiS-LCA.
- Added a RMQ implementation to CaSSiS-LCA.
- Removed obsolete code from the PT-Server.
- Refactoring: “species” to “sequence”.
- Added `make package` by adding CPack.
- Various updates/fixes to the pthreads support. Untested and disabled by default.
- Disabled a PT-Server warning that resulted in gigabytes of output.
- Fixed a double free when computing signatures of different lengths.
- Added the MiniPT tool for PT-Index debugging.
- Added basic Base4Set merge support. Untested.
- Added VarInt support to the BGRT tools.
- Removed redundant code.

CaSSiS 0.3.2 (Released 12.August 2011; Stable release)

- Fixed parameter handling. Fixed help in CaSSiS.
- Disabled various unimplemented functions.
- Optimized the BGRT layout.
- Added “Creme de CaSSiS” (CdCaSSiS), a user interface version of CaSSiS.
- Removed the outdated unit tests. (Not visible to the end user.)
- Removed the outdated statistics code.
- Removed various warnings during compilation.
- Various small improvements. (Too small to be mentioned in detail.)
- Removed obsolete ARBHOME environment variable dependency.
- Reduced the code complexity a bit.
- Small update that might speed up the PT-Server a little bit.
- Removed proprietary memory management from the PT-Server.

- Fixed three memory leaks (in the PT-Server and the ARB-DB code).
- Moved the more generic BGRT generation code parts from the ARBDB into CaSSiS.
- Added ARB files to the repository. Created a project-specific ARBDB library.
- Directly added the PT-Server (former ARB PT-Server) to CaSSiS. This code does no longer match the original ARB PT-Server code.
- Various changes due to strict gcc settings.
- Added date/version numbers to the cmake files and the source code.
- Updated the documentation.
- Switched to templates for internal types.

CaSSiS 0.3.1 (Released 27.June 2011; Maintenance release)

- Fixed minor dependency problems in the shell scripts.

CaSSiS 0.3.0 (Internally released 9.June 2011; Stable release)

- Added/updated the documentation.
- Fixed various memory leaks and minor bugs.
- Added I/O functionality for `Base4Set` and `Base4_t`. Added Base4 type functionality.
- Added the Base4 support throughout the BGRT library, resulting in less memory consumption.
- Various changes to reduce the memory consumption of a loaded BGRT.
- Implemented a new (slightly faster) traversal approach.
- Updated the CaSSiS PT-Server code due to ARB source code updates. Updated the ARB PT-Server dependencies.
- Clean-up does not remove computed BGRT-files any longer.
- Adapted the batch job script according to the new parameters.
- Split the CaSSiS functionalities into three different functions: create / traverse / info
- Updated the parameter parser function.
- Set a dummy `ARBHOME` environment variable to satisfy the PT-Server.
- Added branch lengths to the phylogenetic tree.

- Moved thermodynamic functionalities into the BGRT library.
- Added single group search to the BGRT library.
- BGRT file checksum added.
- Updated phylogenetic tree reduction.
- Removed progress bar from phylogenetic tree reader.
- Disabled dump stats by default.
- Refactored the CSV result dump.
- Updated the BGRT filename generation.
- Renamed all referenced probe(s) to signature(s)
- Updated the BGRT library (separate base types)
- Updated the output. BGRT header information is now dumped when loading a BGRT.
- Code moved from C to C++.
- Major changes in the handling of the BGRT file. BGRT structures can now be stored/loaded.

CaSSiS 0.2.0 (Released 7.March 2011; Stable release)

- Updated the error handling within CaSSiS.
- Updated/fixed an error in the mismatch calculation.
- Fixed the mismatch parameter parsing.
- Support of weighted mismatches (ARB PT-Server functionality).
- Various preparations for the usage of thermodynamics (G+C, melting temp, ...) during the BGRT build process.
- Updated the PT-Server+BGRT to count outgroup matches between m_1 and m_2.
- Code clean-up. Updated to the newest ARB code base. Code split into three libraries (`libCORE.so`, `libARBDB.so` and `libptserver.so`).
- Removed obsolete output (status messages) in the release build.
- Updated the batch job script: Removed the PT-Server name parameter because it is obsolete.
- Minor fix to correctly sync. the ids between CaSSiS and the PT-Server.

- Fixed a bug when using mismatch distances > 1.
- Added an ARB PT-Server wrapper. No need for a PT-Server binary any longer.
- The BGRT functionality is combined in a dynamic library: `libbgrt.so`
- Fixed ID handling for phylogenetic trees that only contain a subset of the sequences within the BGRT.
- Swapped the processing order of the BGRT and phylogenetic tree.
- Memory leak fix.

CaSSiS 0.1.1 (Released 22.October 2010; Maintenance release)

- Added debugging information to the BGRT traversal.
- Applied two hotfixes for the cut-off algorithm. (Fixed data loss.)
- Minor improvement in the BGRT traversal.

CaSSiS 0.1.0 (Released 10.September 2010; Stable release)

- Warning: The 0.1.0 release of CaSSiS contains a flaw that leads to the loss of promising signatures.
- First official release.

B.7 License for CaSSiS

GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. (`http://fsf.org/`) Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.
As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.
“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.
A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.
The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.
The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.
You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.
If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:
a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.
The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:
a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.
You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and

reverse engineering for debugging such modifications, if you also do each of the following:
a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
d) Do one of the following:
0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.
You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:
a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.
The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library as you received it specifies

that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.
If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

GNU General Public License

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. (`http://fsf.org/`) Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble
The GNU General Public License is a free, copyleft license for software and other kinds of works.
The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program — to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.
To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
For the developers' and authors' protection, the GPL clearly explains that there is no warranty

for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.
Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.
Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.
The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.
"This License" refers to version 3 of the GNU General Public License.
"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.
"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.
To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.
A "covered work" means either the unmodified Program or a work based on the Program.
To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.
To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.
An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties

are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.
The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.
A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.
The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.
The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.
The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.
All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.
You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in

conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.
Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.
No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.
When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.
You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.
You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.
You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:
a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices;

however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.
A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.
You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:
a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.
A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written

to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.
Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:
a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.
All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.
If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.
Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.
You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.
An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.
A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.
A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do

not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contra-

dict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.
If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.
THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, RE-

PAIR OR CORRECTION.

16. Limitation of Liability.
IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

B.8 License for MiniPT

COPYRIGHTS
The ARB software and documentation are not in the public domain. External programs distributed together with ARB are copyrighted by and are the property of their respective authors unless otherwise stated. All other copyrights are owned by Lehrstuhl für Mikrobiologie, TU München.

USAGE LICENSE
You have the right to use this version of ARB for free. Please read as well the attached copyright notices below whether you may or may not install this package. Since many of the included programs is free software and nobody is allowed to sell that software you may safely assume ARB will never become a commercial product.

REDISTRIBUTION LICENSE
This release of the ARB program and documentation may not be sold or incorporated into a commercial product, in whole or in part, without the expressed written consent of the Technische Universität München and of its supervisors Ralf Westram or Wolfgang Ludwig.
All interested parties may redistribute and modify ARB as long as all copies are accompanied by this license information and all copyright notices remain intact. Parties redistributing ARB must do so on a non-profit basis, charging only for cost of media or distribution.

If you modify parts of ARB and redistribute these changes the 'Lehrstuhl für Mikrobiologie' of the TU München gains the right to incorporate these changes into ARB and to redistribute them with future versions of ARB.

DEBIAN DISTRIBUTION
Hereby anybody is granted the right to build debian-packets of the ARB software package (`http://www.arb-home.de/`) and publish them on debian mirrors (or any other way of debian-distribution). This includes any debian derivatives like ubuntu. The ARB developers may (but most likely wont ever) revoke this granting. If really done so, it'll only affect ARB versions released after such a revocation.

DISCLAIMER
THE TU MÜNCHEN AND THE VARIOUS AUTHORS OF ARB GIVE NO WARRANTIES, EXPRESSED OR IMPLIED FOR THE SOFTWARE AND DOCUMENTATION PROVIDED, INCLUDING, BUT NOT LIMITED TO WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE. User understands the software is a research tool for which no warranties as to capabilities or accuracy are made, and user accepts the software "as is." User assumes the entire risk as to the results and performance of the software and documentation. The above parties cannot be held liable for any direct, indirect, consequential or incidental damages with respect to any claim by user or any third party on account of, or arising from the use of software and associated materials. This disclaimer covers both the ARB core applications and all external programs used by ARB.

Bibliography

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53 – 86, 2004. The 9th International Symposium on String Processing and Information Retrieval.

[2] D. G. Altman and J. M. Bland. Statistics Notes: Diagnostic tests 1: sensitivity and specificity. *Br. Med. J.*, 308(6943):1552+, June 1994.

[3] R. Amann and B. M. Fuchs. Single-cell identification in microbial communities by improved fluorescence in situ hybridization techniques. *Nat. Rev. Microbiol.*, 6(5):339–348, 2008.

[4] R. I. Amann, B. J. Binder, R. J. Olson, S. W. Chisholm, R. Devereux, and D. A. Stahl. Combination of 16S rRNA-targeted oligonucleotide probes with flow cytometry for analyzing mixed microbial populations. *Appl. Environ. Microbiol.*, 56:1919–1925, Jun 1990.

[5] R. I. Amann, W. Ludwig, and K. H. Schleifer. Phylogenetic identification and in situ detection of individual microbial cells without cultivation. *Microbiol Rev.*, 59(1):143–169, March 1995.

[6] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, P. Selo, Y. Park, and C. Venkatramani. Spc: A distributed, scalable platform for data mining. In *Workshop on Data Mining Standards, Services and Platforms (DM-SPP)*, 2006.

[7] J. Arnds, K. Knittel, U. Buck, M. Winkel, and R. Amann. Development of a 16S rRNA-targeted probe set for Verrucomicrobia and its application for fluorescence in situ hybridization in a humic lake. *Syst. Appl. Microbiol.*, 33:139–148, Apr 2010.

[8] K. E. Ashelford, A. J. Weightman, and J. C. Fry. PRIMROSE: a computer program for generating and estimating the phylogenetic range of 16S rRNA oligonucleotide probes and primers in conjunction with the RDP-II database. *Nucleic Acids Res.*, 30(15):3481–3489, August 2002.

[9] J. F. Atkins and R. Gesteland. Biochemistry. The 22nd amino acid. *Science*, 296(5572):1409–1410, May 2002.

[10] K. C. Bader, M. J. Atallah, and C. Grothoff. Efficient relaxed search in hierarchically clustered sequence datasets. *J. Exp. Algorithmics*, 17(1):1.4:1.1–1.4:1.18, July 2012.

[11] K. C. Bader, T. Eißler, N. Evans, C. GauthierDickey, C. Grothoff, K. Grothoff, J. Keene, H. Meier, C. Ritzdorf, and M. J. Rutherford. Distributed stream processing with DUP. In *Network and Parallel Computing*, volume 6289 of *Lecture Notes in Computer Science*, pages 232–246. Springer Berlin / Heidelberg, 2010.

[12] K. C. Bader, C. Grothoff, and H. Meier. Comprehensive and relaxed search for oligonucleotide signatures in hierarchically clustered sequence datasets. *Bioinformatics*, 27:1546–1554, Jun 2011.

[13] M. Barsky, U. Stege, and A. Thomo. A survey of practical algorithms for suffix tree construction in external memory. *Software: Practice and Experience*, 40(11):965–988, 2010.

[14] F. Barth, P. Mühlbauer, F. Nikol, and K. Wörle. *Mathematische Formeln und Definitionen.* Bayerischer Schulbuch-Verlag Lindauer, München, 1976.

[15] J. M. S. Bartlett and D. Stirling. A short history of the polymerase chain reaction. In J. M. Bartlett and D. Stirling, editors, *PCR Protocols*, volume 226 of *Methods in Molecular Biology*, pages 3–6. Humana Press, 2003. 10.1385/1-59259-384-4:3.

[16] M. Bender and M. Farach-Colton. The lca problem revisited. In G. Gonnet and A. Viola, editors, *LATIN 2000: Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer Berlin / Heidelberg, 2000.

[17] H.-J. Böckenhauer and D. Bongartz. *Algorithmische Grundlagen der Bioinformatik: Modelle, Methoden und Komplexität.* Teubner, Stuttgart u.a., 2003.

[18] K. J. Breslauer, R. Frank, H. Blocker, and L. A. Marky. Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci. U.S.A.*, 83:3746–3750, Jun 1986.

[19] T. D. Brock and M. T. Madigan. *Biology of Microorganisms.* Prentice Hall, Englewood Cliffs, N.J, 1994.

[20] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[21] Y. Chen, T. Souaiaia, and T. Chen. PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. *Bioinformatics*, 25:2514–2521, Oct 2009.

[22] W. H. Chung, S. K. Rhee, X. F. Wan, J. W. Bae, Z. X. Quan, and Y. H. Park. Design of long oligonucleotide probes for functional gene detection in a microbial community. *Bioinformatics*, 21:4092–4100, Nov 2005.

[23] J. E. Clarridge. Impact of 16S rRNA gene sequence analysis for identification of bacteria on clinical microbiology and infectious diseases. *Clin. Microbiol. Rev.*, 17(4):840–862, Oct 2004.

[24] T. Coenye and P. Vandamme. Intragenomic heterogeneity between multiple 16S ribosomal RNA operons in sequenced bacterial genomes. *FEMS Microbiol. Lett.*, 228(1):45–49, Nov 2003.

[25] J. R. Cole, Q. Wang, E. Cardenas, J. Fish, B. Chai, R. J. Farris, A. S. Kulam-Syed-Mohideen, D. M. McGarrell, T. Marsh, G. M. Garrity, and J. M. Tiedje. The Ribosomal Database Project: improved alignments and new tools for rRNA analysis. *Nucleic Acids Res.*, 37:D141–145, Jan 2009.

[26] A. Cornish-Bowden. Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic Acids Res.*, 13:3021–3030, May 1985.

[27] G. Czihak, H. Langer, and H. Ziegler, editors. *Biologie: Ein Lehrbuch*. Springer-Verlag, Berlin New York, 1990.

[28] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. J. Murray, sixth edition, June 1872.

[29] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl. Environ. Micorbiol.*, 72(7):5069–5072, July 2006.

[30] A. Doring, D. Weese, T. Rausch, and K. Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 9(1):11, 2008.

[31] D. Dougherty. *Sed and AWK*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1991.

[32] J. Duitama, D. M. Kumar, E. Hemphill, M. Khan, I. I. Mandoiu, and C. E. Nelson. PrimerHunter: a primer design tool for PCR-based virus subtype identification. *Nucleic Acids Res.*, 37:2483–2492, May 2009.

[33] T. Eißler, C. P. Hodges, and H. Meier. PTPan — overcoming memory limitations in oligonucleotide string matching for primer/probe design. *Bioinformatics*, 27(20):2797–2805, 2011.

[34] S. Feng and E. Tillier. A fast and flexible approach to oligonucleotide probe design for genomes and gene families. *Bioinformatics*, 23:1195–1202, May 2007.

[35] G. F. Ficetola, E. Coissac, S. Zundel, T. Riaz, W. Shehzad, J. Bessiere, P. Taberlet, and F. Pompanon. An in silico approach for the evaluation of DNA barcodes. *BMC Genomics*, 11:434, 2010.

[36] W. Fiers, R. Contreras, F. Duerinck, G. Haegeman, D. Iserentant, J. Merregaert, W. Min Jou, F. Molemans, A. Raeymaekers, A. Van den Berghe, G. Volckaert, and M. Ysebaert. Complete nucleotide sequence of bacteriophage MS2 RNA: primary and secondary structure of the replicase gene. *Nature*, 260(5551):500–507, Apr 1976.

[37] J. Fischer and V. Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In M. Lewenstein and G. Valiente, editors, *Combinatorial Pattern Matching*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer Berlin / Heidelberg, 2006.

[38] J. Fischer and V. Heun. A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In B. Chen, M. Paterson, and G. Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer Berlin / Heidelberg, 2007.

[39] R. D. Fleischmann, M. D. Adams, O. White, R. A. Clayton, E. F. Kirkness, A. R. Kerlavage, C. J. Bult, J. F. Tomb, B. A. Dougherty, and J. M. Merrick. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, 269(5223):496–512, Jul 1995.

[40] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 135–143, New York, NY, USA, 1984. ACM.

[41] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[42] J. Giacomoni, T. Moseley, and M. Vachharajani. Fastforward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 43–52, New York, NY, USA, 2008. ACM.

[43] E. Goebelbecker. Using grep: Moving from DOS? discover the power of this linux utility. *Linux J.*, 18, Oct. 1995.

[44] C. Grothoff and J. Keene. The DUP protocol specification v2.0. Technical report, The DUP Project, 2010.

[45] S. Haensch, R. Bianucci, M. Signoli, M. Rajerison, M. Schultz, S. Kacki, M. Vermunt, D. A. Weston, D. Hurst, M. Achtman, E. Carniel, and B. Bramanti. Distinct clones of yersinia pestis caused the black death. *PLoS Pathog*, 6(10):e1001134, 10 2010.

[46] J. P. Hartmann. *CMS Pipelines Explained.* IBM Denmark, `http://vm.marist.edu/~pipeline/`, Sep 2007.

[47] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, R. Soule, and Kun-Lung-Wu. Spade language specification. Technical report, IBM Research, March 2009.

[48] P. Hugenholtz, B. M. Goebel, and N. R. Pace. Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *J. Bacteriol.*, 180:4765–4774, Sep 1998.

[49] IBM. *CMS Pipelines User's Guide.* IBM Corp., `http://publibz.boulder.ibm.com/epubs/pdf/hcsh1b10.pdf`, version 5 release 2 edition, Dec 2005.

[50] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *European Conference on Computer Systems (EuroSys)*, pages 59–72, Lisabon, Portugal, March 2007.

[51] IUPAC-IUB Commission on Biochemical Nomenclature (CBN). Abbreviations and symbols for nucleic acids, polynucleotides and their constituents. Recommendations 1970. *Biochem. J.*, 120:449–454, Dec 1970.

[52] IUPAC-IUB Commission on Biochemical Nomenclature (CBN) and Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Newsletter 1999. *European Journal of Biochemistry*, 264(2):607–609, 1999.

[53] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics*, 18(10):1340–1349, October 2002.

[54] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, New York, NY, 1974.

[55] M. D. Kane, T. A. Jatkoe, C. R. Stumpf, J. Lu, J. D. Thomas, and S. J. Madore. Assessment of the sensitivity and specificity of oligonucleotide (50mer) microarrays. *Nucleic Acids Res.*, 28(22):4552–4557, Nov 2000.

[56] T. Klug. Hardware of the InfiniBand Cluster. `http://www.lrr.in.tum.de/Par/arch/infiniband/ClusterHW/cluster.html`, 2008.

[57] D. Knuth. *The Art of Computer Programming.* Addison-Wesley, Boston, 1998.

[58] M. Kudlur and S. Mahlke. Orchestrating the execution of stream programs on multicore platforms. In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, pages 114–124, New York, NY, USA, 2008. ACM.

[59] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biol.*, 5:R12, 2004.

[60] F. C. Lawyer, S. Stoffel, R. K. Saiki, S. Y. Chang, P. A. Landre, R. D. Abramson, and D. H. Gelfand. High-level expression, purification, and enzymatic characterization of full-length Thermus aquaticus DNA polymerase and a truncated form deficient in 5' to 3' exonuclease activity. *PCR Methods Appl.*, 2:275–287, May 1993.

[61] E. A. Lee. Ptolemy project. `http://ptolemy.eecs.berkeley.edu/`, 2008.

[62] H. P. Lee, T.-F. Sheu, and C. Y. Tang. A parallel and incremental algorithm for efficient unique signature discovery on dna databases. *BMC Bioinformatics*, 11:132, 2010.

[63] I. Letunic and P. Bork. Interactive Tree Of Life v2: online annotation and display of phylogenetic trees made easy. *Nucleic Acids Res.*, 39:W475–478, Jul 2011.

[64] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25:1754–1760, Jul 2009.

[65] C. Linhart and R. Shamir. The degenerate primer design problem. *Bioinformatics*, 18 Suppl 1:S172–181, 2002.

[66] A. Loy, R. Arnold, P. Tischler, T. Rattei, M. Wagner, and M. Horn. probeCheck–a central resource for evaluating oligonucleotide probe coverage and specificity. *Environ. Microbiol.*, 10:2894–2898, Oct 2008.

[67] A. Loy and L. Bodrossy. Highly parallel microbial diagnostics using oligonucleotide microarrays. *Clinica Chimica Acta*, 363(1–2):106 – 119, 2006.

[68] A. Loy, F. Maixner, M. Wagner, and M. Horn. probeBase–an online resource for rRNA-targeted oligonucleotide probes: new features 2007. *Nucleic Acids Res.*, 35(Database issue):D800–804, Jan 2007.

[69] S. Lücker, D. Steger, K. U. Kjeldsen, B. J. MacGregor, M. Wagner, and A. Loy. Improved 16S rRNA-targeted probe set for analysis of sulfate-reducing bacteria by fluorescence in situ hybridization. *J. Microbiol. Methods*, 69:523–528, Jun 2007.

[70] W. Ludwig, O. Strunk, R. Westram, L. Richter, H. Meier, Yadhukumar, A. Buchner, T. Lai, S. Steppi, G. Jobb, W. Förster, I. Brettske, S. Gerber, A. W. Ginhart, O. Gross, S. Grumann, S. Hermann, R. Jost, A. König, T. Liss, R. Lüssmann, M. May, B. Nonhoff, B. Reichel, R. Strehlow, A. Stamatakis, N. Stuckmann, A. Vilbig, M. Lenke, T. Ludwig, A. Bode, and K. H. Schleifer. ARB: a software environment for sequence data. *Nucleic Acids Res.*, 32(4):1363–1371, 2004.

[71] B. L. Maidak, J. R. Cole, T. G. Lilburn, C. T. Parker, P. R. Saxman, R. J. Farris, G. M. Garrity, G. J. Olsen, T. M. Schmidt, and J. M. Tiedje. The RDP-II (Ribosomal Database Project). *Nucleic Acids Res.*, 29(1):173–174, Jan 2001.

[72] J. A. Manthey. mFold, Delta G, and Melting Temperature: What Does it Mean? Technical report, Integrated DNA Technologies, Bioinformatics Group, Integrated DNA Technologies Inc., 1710 Commercial Park, Coralville, Iowa 5224, USA, 2005.

[73] J. Marmur and P. Doty. Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. *J. Mol. Biol.*, 5:109–118, Jul 1962.

[74] S. J. McIlroy, D. Tillett, S. Petrovski, and R. J. Seviour. Non-target sites with single nucleotide insertions or deletions are frequently found in 16s rrna sequences and can lead to false positives in fluorescence in situ hybridization (fish). *Environmental Microbiology*, 13(1):33–47, 2011.

[75] H. Meier, A. Krause, M. Kräutner, and A. Bode. Development and implementation of a parallel algorithm for the fast design of oligonucleotide probe sets for diagnostic DNA microarrays. *Concurr. Comput.: Pract. Exper.*, 16(9):873–893, 2004.

[76] M. Mitsuhashi, A. Cooper, M. Ogura, T. Shinagawa, K. Yano, and T. Hosokawa. Oligonucleotide probe design–a new approach. *Nature*, 367:759–761, Feb 1994.

[77] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, Apr. 1965.

[78] K. B. Mullis. The unusual origin of the polymerase chain reaction. *Scientific American*, 262(4), Apr. 1990.

[79] Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Nomenclature of electron-transfer proteins. Recommendations 1989. *J. Biol. Chem.*, 267:665–677, Jan 1992.

[80] E. K. Nordberg. YODA: selecting signature oligonucleotides. *Bioinformatics*, 21(8):1365–1370, April 2005.

[81] G. Olsen. The "Newick's 8:45" Tree Format Standard. `http://evolution.genetics.washington.edu/phylip/newick_doc.html`, August 1990.

[82] OpenMP Architecture Review Board. OpenMP Application – Program Interface, 2008. `http://www.openmp.org/mp-documents/spec30.pdf`.

[83] A. Panjkovich and F. Melo. Comparison of different melting temperature calculation methods for short DNA sequences. *Bioinformatics*, 21:711–722, Mar 2005.

[84] T. M. Parks. *Bounded Scheduling of Process Networks.* PhD thesis, University of California, Berkeley, December 1995.

[85] P. Peterlongo, J. Nicolas, D. Lavenier, R. Vorc'h, and J. Querellou. c-gamma:comparative genome analysis of molecular markers. In V. Kadirkamanathan, G. Sanguinetti, M. Girolami, M. Niranjan, and J. Noirel, editors, *Pattern Recognition in Bioinformatics*, volume 5780 of *Lecture Notes in Computer Science*, pages 255–269. Springer Berlin / Heidelberg, 2009.

[86] A. M. Phillippy, K. Ayanbule, N. J. Edwards, and S. L. Salzberg. Insignia: a DNA signature search web server for diagnostic assay development. *Nucleic Acids Res.*, 37:W229–234, Jul 2009.

[87] A. M. Phillippy, J. A. Mason, K. Ayanbule, D. D. Sommer, E. Taviani, A. Huq, R. R. Colwell, I. T. Knight, and S. L. Salzberg. Comprehensive DNA signature discovery and validation. *PLoS Comput. Biol.*, 3:e98, May 2007.

[88] M. N. Price, P. S. Dehal, and A. P. Arkin. Fasttree 2 – approximately maximum-likelihood trees for large alignments. *PLoS ONE*, 5(3):e9490, 03 2010.

[89] E. Pruesse, C. Quast, K. Knittel, B. M. Fuchs, W. Ludwig, J. Peplies, and F. O. Glöckner. SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Res.*, 35(21):7188–7196, December 2007.

[90] E. Quigley. *UNIX Shells*. Prentice Hall, 4 edition, September 2004.

[91] D. Raoult, P. E. Fournier, and M. Drancourt. What does the future hold for clinical microbiology? *Nat. Rev. Microbiol.*, 2:151–159, Feb 2004.

[92] A. Relogio, C. Schwager, A. Richter, W. Ansorge, and J. Valcarcel. Optimization of oligonucleotide-based DNA microarrays. *Nucleic Acids Res.*, 30:e51, Jun 2002.

[93] T. Riaz, W. Shehzad, A. Viari, F. Pompanon, P. Taberlet, and E. Coissac. ecoPrimers: inference of new DNA barcode markers from whole genome sequence analysis. *Nucleic Acids Res.*, 39:e145, Nov 2011.

[94] J.-M. Rouillard, M. Zuker, and E. Gulari. OligoArray 2.0: design of oligonucleotide probes for DNA microarrays using a thermodynamic approach. *Nucleic Acids Res.*, 31(12):3057–3062, June 2003.

[95] W. Rychlik, W. J. Spencer, and R. E. Rhoads. Optimization of the annealing temperature for DNA amplification in vitro. *Nucleic Acids Res.*, 18(21):6409–6412, Nov 1990.

[96] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. mei W. Hwu. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82, New York, NY, USA, 2008. ACM.

[97] F. Sanger, G. M. Air, B. G. Barrell, N. L. Brown, A. R. Coulson, C. A. Fiddes, C. A. Hutchison, P. M. Slocombe, and M. Smith. Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*, 265(5596):687–695, Feb 1977.

[98] J. SantaLucia. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. U.S.A.*, 95:1460–1465, Feb 1998.

[99] K. Schleifer and R. Amann. Nucleic acid probes and their application in environmental microbiology. In G. Garrity, editor, *Bergey's Manual of Systematic Bacteriology*, volume 6289, pages 67–82. Springer, New York, 2001.

[100] S. Schönmann, A. Loy, C. Wimmersberger, J. Sobek, C. Aquino, P. Vandamme, B. Frey, H. Rehrauer, and L. Eberl. 16S rRNA gene-based phylogenetic microarray for simultaneous identification of members of the genus Burkholderia. *Environ. Microbiol.*, 11:779–800, Apr 2009.

[101] M. H. Schulz, S. Bauer, and P. N. Robinson. The generalised k-truncated suffix tree for time- and space-efficient searches in multiple dna or protein sequences. *Int. J. Bioinformatics Res. Appl.*, 4:81–95, February 2008.

[102] M. Severgnini, P. Cremonesi, C. Consolandi, G. Caredda, G. De Bellis, and B. Castiglioni. ORMA: a tool for identification of species-specific variations in 16S rRNA gene and oligonucleotides design. *Nucleic Acids Res.*, 37:e109, Sep 2009.

[103] J. Shendure and H. Ji. Next-generation DNA sequencing. *Nat. Biotechnol.*, 26:1135–1145, 2008.

[104] J. H. Spring, J. Privat, R. Guerraoui, and J. Vitek. Streamflex: high-throughput stream programming in java. *SIGPLAN Not.*, 42(10):211–228, 2007.

[105] D. A. Stahl, B. Flesher, H. R. Mansfield, and L. Montgomery. Use of phylogenetically based hybridization probes for studies of ruminal microbial ecology. *Appl. Environ. Microbiol.*, 54:1079–1084, May 1988.

[106] A. Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22:2688–2690, Nov 2006.

[107] O. Strunk. *ARB: Entwicklung eines Programmsystems zur Erfassung, Verwaltung und Auswertung von Nuklein- und Aminosäuresequenzen.* PhD thesis, TU München, 1993.

[108] N. Sugimoto, S. Nakano, M. Yoneyama, and K. Honda. Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes. *Nucleic Acids Res.*, 24:4501–4505, Nov 1996.

[109] F. C. Tenover. Rapid detection and identification of bacterial pathogens using novel molecular technologies: infection control and beyond. *Clin. Infect. Dis.*, 44:418–423, Feb 2007.

[110] W. Thies, M. Karczmarek, and S. P. Amarasinghe. Streamit: A language for streaming applications. In *CC '02: Proceedings of the 11th International Conference on Compiler Construction*, pages 179–196, London, UK, 2002. Springer-Verlag.

[111] N. von Ahsen, M. Oellerich, V. W. Armstrong, and E. Schutz. Application of a thermodynamic nearest-neighbor model to estimate nucleic acid stability and optimize probe design: prediction of melting points of multiple mutations of apolipoprotein B-3500 and factor V with a hybridization probe genotyping assay on the LightCycler. *Clin. Chem.*, 45:2094–2101, Dec 1999.

[112] M. Wagner, M. Horn, and H. Daims. Fluorescence in situ hybridisation for the identification and characterisation of prokaryotes. *Current Opinion in Microbiology*, 6(3):302 – 309, 2003.

[113] R. B. Wallace, J. Shaffer, R. F. Murphy, J. Bonner, T. Hirose, and K. Itakura. Hybridization of synthetic oligodeoxyribonucleotides to phi chi 174 DNA: the effect of single base pair mismatch. *Nucleic Acids Res.*, 6:3543–3557, Aug 1979.

[114] S. Weckx, E. Carlon, L. De Vuyst, and P. Van Hummelen. Thermodynamic behavior of short oligonucleotides in microarray hybridizations can be described using gibbs free energy in a nearest-neighbor model. *The Journal of Physical Chemistry B*, 111(48):13583–13590, 2007.

[115] G. C. Wells. *A Programmable Matching Engine for Application Development in Linda.* PhD thesis, University of Bristol, 2001.

[116] R. Wernersson and H. B. Nielsen. OligoWiz 2.0–integrating sequence feature annotation into the design of microarray probes. *Nucleic Acids Res.*, 33:W611–615, Jul 2005.

[117] R. Westram, K. C. Bader, E. Prüsse, Y. Kumar, H. Meier, F. O. Glöckner, and W. Ludwig. ARB: A software environment for sequence data. In F. J. de Bruijn, editor, *Handbook of Molecular Microbial Ecology I: Metagenomics and Complementary Approaches*, chapter 46, pages 399–406. John Wiley & Sons, Hoboken, New Jersey, 2011.

[118] M. G. Wise, J. V. McArthur, and L. J. Shimkets. 16S rRNA gene probes for Deinococcus species. *Syst. Appl. Microbiol.*, 19:365–369, 1996.

[119] C. R. Woese, O. Kandler, and M. L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proc. Natl. Acad. Sci. U.S.A.*, 87:4576–4579, Jun 1990.

[120] C. R. Woese, J. Maniloff, and L. B. Zablen. Phylogenetic analysis of the mycoplasmas. *Proc. Natl. Acad. Sci. U.S.A.*, 77(1):494–498, Jan 1980.

[121] L. S. Yilmaz, L. I. Bergsven, and D. R. Noguera. Systematic evaluation of single mismatch stability predictors for fluorescence in situ hybridization. *Environmental Microbiology*, 10(10):2872–2885, 2008.