Fachhochschule Weihenstephan
University of Applied Sciences

Diplomarbeit

Entwicklung von Werkzeugen zur taxonomischen Analyse von Proteinhomologen und deren Anwendung im Anammox-Metagenomics Projekt

Autor: Dominik Lindner
Datum: 23. Oktober 2006

Betreuer:
Prof. Dr. Bernhard Haubold (FH Weihenstephan)
Dr. Thomas Rattei (TU München, Lehrstuhl für genomorientierte Bioinformatik)

Eidesstattliche Erklärung

Gemäß §23 Abs. 6 der Prüfungsordnung

Ich erkläre hiermit an Eides statt, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt wurde.

Es wurden keine als die angegebenen Quellen oder Hilfsmittel benutzt. Wörtliche und sinngemäße Zitate sind als solche gekennzeichneet.

Freising, den 23. Oktober 2006

Abstract

Im Rahmen der Diplomarbeit wurde eine Softwarekomponente zur Visualisierung von taxonomischen Bäumen implementiert. Die Softwarekomponente soll im SIMAP-Webinterface [1] eingesetzt werden. SIMAP (Similarity Matrix of Proteins) ist eine Protein-Alignment-Datenbank zur schnellen Suche nach Proteinhomologen. Im SIMAP-Webinterface soll die Softwarekomponente dazu dienen, dem Benutzer die taxonomische Verteilung von Proteinhomologen anzuzeigen, und ihm eine Navigationsmöglichkeit bieten, um in der meist großen Anzahl von Proteinhomologen eine gezielte Auswahl treffen zu können. Als Zeichenalgorithmus wurde der ER-Algorithmus [2] implementiert. Die Implementierung unterstützt sowohl horizontale, als auch vertikale Bäume, und bietet Möglichkeiten zur grafischen Anpassung des Baumes (verschiedene Knotenverbindungstypen, Definition von Abständen, usw.). Für das zu verwendende Grafikframework gab es mehrere Alternativen (Graphics-API, SVGGraphics2D, AWT/Swing, usw.), wobei ich mich aufgrund der guten Anpassbarkeit für die Verwendung von AWT/Swing entschied. Der Zeichenalgorithmus wurde in Form eines LayoutManagers implementiert, wodurch er allgemein für das Problem der Baumdarstellung verwendet werden kann. Für den speziellen Fall 'Taxonomiebaum für das SIMAP-Webinterface' wurde ein Enterprise Java Bean [3] implementiert, welches unter Verwendung dieses LayoutManagers entsprechende Methoden zur Visualisierung von und Navigation in taxonomischen Bäumen bereitstellt, und in die SIMAP Softwarearchitektur integriert werden kann.
Darüberhinaus habe ich untersucht, inwieweit es möglich ist, mit Hilfe der Proteinhomologen in SIMAP und der entsprechenden taxonomischen Information Community-Genome am Beispiel des Anammox Community-Genomes [4] taxonomischen analysieren zu können. Durch die Analyse des Community-Genomes hinsichtlich Open Reading Frames (ORFs) konnten potentielle Proteine vorhergesagt werden, und mittels FASTA [5] die entsprechenden homologen Proteine in SIMAP gefunden werden. In einem ersten Schritt habe ich versucht, taxonomische Information über das Community-Genom durch Betrachtung der ähnlichsten Proteinhomologe zu gewinnen. Da diese Methode aber nur einen Bruchteil der vorhandenen Information nutzt (nur die ähnlichsten Proteinhomologen), und somit nicht zu besonders aussagekräftigen Ergebnissen führt, habe ich in einem zweiten Schritt versucht, durch Berechnung von Distanzvektoren aus der Homologieinformation, und deren Vergleich mit einem phylogenetischen Referenzbaum (Peer Bork's Baum des Lebens [6]) bessere Ergebnisse zu erzielen. Der zweite Ansatz ('Referenzbaum-Methode') ist tatsächlich erfolgsversprechender, obgleich sich Probleme zeigten, die hauptsächlich in der Diskrepanz zwischen taxonomischen und phylogenetischen Bäumen begründet liegen. Die Referenzbaum-Methode bietet Ansätze zur Verbesserung, und könnte zu einem Verfahren zur schnellen, automatischen Analyse von Community-Genomen weiterentwickelt werden.

Inhaltsverzeichnis

1. Aufgabenstellung **6**

2. Einleitung **7**
2.1. Taxonomische Information in Genom- und Proteindatenbanken 7
2.1.1. Genom-/Proteindatenbanken 7
2.1.2. NCBI Taxonomie-Datenbank 8
2.1.3. Phylogenetische Bäume 10
2.2. SIMAP (Similarity Matrix of Proteins) 12
2.2.1. Einleitung 12
2.2.2. Zahlen und Fakten 14
2.3. Taxonomische Analyse von Metagenomics-Daten 14
2.3.1. Metagenomics 14
2.3.2. Methoden 15
2.3.3. Anammox-Community Genom 16
2.4. Software, APIs, Spezifikationen 17
2.4.1. Java Enterprise Edition/Enterprise Computing 17
2.4.2. Grafikformate/-frameworks 21

3. Taxonomie zur Visualisierung und Benutzerführung **25**
3.1. Analyse 25
3.1.1. Zielsetzungen 25
3.1.2. Grafikformate 26
3.1.3. Grafikframeworks 28
3.1.4. Software-Architektur 29
3.2. Zeichenalgorithmus 30
3.3. Implementierung 32
3.3.1. 'Tree-Drawing' Komponente 32
3.3.2. Taxonomiebaum 33
3.3.3. Enterprise Java Bean 34
3.4. Ergebnis 35
3.5. Ausblick 35

4. Taxonomische Analyse des Anammox-Community Genomes **36**
4.1. Vorbereitung 36
4.2. Genvorhersage 39
4.3. Homologe in der SIMAP-Datenbank 40
4.4. Taxonomische Analyse 40
4.4.1. 'Beste Homologen'-Methode 41
4.4.2. Referenzbaum-Methode 42
4.5. Ergebnis 48
4.6. Ausblick 51

5. Zusammenfassung **52**

Literaturverzeichnis **52**

Abkürzungsverzeichnis **55**

Stichwortverzeichnis **57**

Tabellenverzeichnis **60**

Abbildungsverzeichnis **60**

Appendix **61**

A. Einleitung **62**
A.1. SIMAP 62
A.2. Grafikformate und -frameworks 63
A.3. Peer Bork's 'Tree of Life' 64
A.4. Universelle, nicht-HGT COGs 65

B. Taxonomie zur Visualisierung und Benutzerführung **66**
B.1. Tree-Layout 66

C. Taxonomische Analyse des Anammox-Community Genomes **67**
C.1. 'Beste Homologe'-Analyse 67
C.2. Referenzbaum-Methode 74

D. Quellcode **77**

1. Aufgabenstellung

Das National Centre for Biotechnology Information, USA (NCBI) hat mit seiner Taxonomie-Datenbank die Möglichkeit geschaffen, Taxa (Arten, Gattungen, usw.) eine eindeutige ID zuzuordnen und diese in einen taxonomischen Kontext einzuordnen [7]. Die Taxonomie ist baumartig aufgebaut und ordnet Arten Gattungen zu, Gattungen zu Ordnungen, usw. Die Verbindung dieser Informationen mit den Daten einer Protein-Homologie-Datenbank wie SIMAP [1] eröffnet die Möglichkeit zur taxonomischen Untersuchung von Proteinhomologen.
Bei der Suche nach Homologen gibt das SIMAP-Webinterface eine Liste von Proteinen mit den gängigen Daten wie Bitscore, Evalue, usw. zurück. Obwohl die Information vorhanden ist, bleibt der taxomische Zusammenhang dabei verborgen. Für die taxonomische Analyse von Proteinhomologen über das SIMAP-Webinterface ist eine grafische Darstellung der taxonomischen Information notwendig, die dem Benutzer verdeutlicht, wie sich homologe Proteine taxonomisch verteilen. Diese grafische Darstellung bietet sich zudem als Navigationsinstrument für den Benutzer an, um in der möglichen Vielzahl gefundener Homologen navigieren zu können. Im Rahmen der Diplomarbeit soll eine solche Softwarekomponente implementiert werden.
Darüberhinaus soll eine Methode entwickelt werden, mit Hilfe von Proteinhomologen und taxonomischer Information Community-Genome analysieren zu können. Ein Community-Genom enthält Genome mehrerer Organismen. Darunter können sich sowohl Genome, die bereits sequenziert wurden, als auch noch unbekannte Genome befinden. Mittels Genvorhersage können die potentiellen Proteine, für welche die verschiedenen Genome kodieren, ermittelt werden. Durch taxonomische Analyse der in SIMAP gefundenen homologen Proteine, könnte man u. U. feststellen, welche Genome sich in dem Community-Genom befinden, bzw. im Fall unbekannter Genome, wie sie taxonomisch einzuordnen wären. Diese Möglichkeit soll im zweiten Teil der Diplomarbeit untersucht werden.

2. Einleitung

2.1. Taxonomische Information in Genom- und Proteindatenbanken

2.1.1. Genom-/Proteindatenbanken

Aktuell sind die Sequenzen von ungefähr 6 Millionen Proteinen bekannt. Diese Sequenzen sind in mehreren Datenbanken abgelegt, mit IDs versehen, annotiert und mit weitereren primären und sekundären Informationen verknüpft (z. B. Literatur, Proteinfeatures, usw.) [8]. Eine wichtige, primäre Information ist Taxonomie. Eine minimale taxonomische Information, die in jeder Datenbank zur Verfügung stehen sollte, ist die Angabe, aus welcher Spezies eine bestimmte Proteinsequenz stammt. Mit ausführlicheren taxonomischen Informationen ließen sich aber noch weitere interessante Fragestellungen untersuchen, z. B. 'Zu welcher Gattung, Familie, usw. gehört diese Spezies, in der dieses bestimmte Protein vorkommt?'. Die Verknüpfung mit Homologiedaten eröffnet schließlich noch weitere Möglichkeiten, wie z. B. die Frage 'In welchen Spezies, Gattungen, usw. kommen ähnliche Proteine vor?' Aus der Verknüpfung Proteine - Homologie - Taxonomie kann man also weitere sekundäre Information gewinnen.
Wie bereits erwähnt gibt es zahlreiche Proteindatenbanken. Einige davon werde ich im folgenden exemplarisch auf das Vorhandensein dieser taxonomischen Klassifizierung untersuchen.
Als Beispiel-Anwendungsfall wähle ich das Enzym 'Acyl-CoA dehydrogenase' mit der Uniprot-ID 'Q89VR3', zu dem ich nach taxonomischen Informationen suchen will.

- Uniprot/TrEMBL (`http://www.ebi.ac.uk/trembl/`): Information zur Spezies, aus der das Protein stammt ist in Form von Name und Taxonomie-ID vorhanden. Weitere taxonomische Information ist ebenfalls verfügbar: Es wird genannt, zu welcher Gattung, Ordnung und Phylum diese Spezies gehört. Es gibt auch eine Homologie-Funktion in Form eines Linkouts zu UniRef, welche die Anzeige von Protein-Clusters mit min. 50, 75 und 100% Identität ermöglicht.

- UniRef (`http://www.ebi.ac.uk/uniref/`): UniRef zeigt eine Liste von Proteinen, gehörend zu einem Protein-Cluster mit min. 50, 75 oder 100% Identität mit einem gegeben Protein an. Zu jedem Protein wird der Speziesname inkl. Taxonomie-ID genannt. Weitergehende taxonomische Information (Gattung, Ordnung, usw.) ist nicht vorhanden.

- Interpro (`http://www.ebi.ac.uk/interpro/`): Interpro stellt grafisch dar, in welchen Taxa eine gegebene Proteindomain vorkommt. Die Übersicht ist aber statisch (d.h. nicht navigierbar) und enthält nur wenige Taxa. Eine Navigation durch die Taxonomie ist nur über Textlinks möglich.

- Prosite (`http://www.expasy.org/prosite/`): Prosite enthält keine taxonomischen Informationen. Diese ist nur verfügbar über einen Linkout zu entsprechenden Uniprot-Einträgen.

- PDB (`http://www.rcsb.org/pdb/`): PDB enthält ebenfalls keine taxonomischen Informationen. Eine Verknüpfung von Proteinstrukturen mit der entsprechenden taxonomischen Information ist aber durch PDBeast (`http://130.14.29.110/Structure/PDBEAST/pdbeast.shtml`) verfügbar.
- PFAM (`http://www.sanger.ac.uk/Software/Pfam/`): Zu einer gegeben Proteindomain sind keine taxonomischen Informationen verfügbar. Es gibt aber eine Taxonomie-Suche, welche z. B. zum Auffinden von Proteindomains in unterschiedlichen Taxa u. ä. dient.
- NCBI (`http://www.ncbi.nlm.nih.gov/`): Die NCBI-Proteindatenbank (National Center for Biotechnology Information, USA)verknüpft jedes Protein mit der Spezies (Name und Taxonomie-ID). Zur Spezies sind ebenfalls taxonomische Informationen in Form von Text vorhanden. Wie bei Uniprot werden Gattung, Ordnung und Phylum genannt zu welcher die betreffende Spezies gehört. Weiterhin sind diese Angaben per Link mit dem NCBI-Taxonomiebrowser verknüpft. Der Taxonomiebrowser ermöglicht die Navigation durch den taxonomischen Baum über Textlinks.

Datenbank	**Angabe d. Spezies**	**Gattung, Ordnung, ...**	**Grafisch**	**Navigierbar**
Uniprot/TrEMBL	×	×	-	-
UniRef	×	-	-	-
Interpro	×	×	teilweise	Text
Prosite	Linkout	Linkout	-	-
PDB	extern	extern	-	-
NCBI	×	×	-	Text

Tabelle 2.1.: Überblick: Taxonomische Information in Proteindatenbanken
× : Information vorhanden
- : Information nicht vorhanden
Linkout: Link auf eine andere Webseite
extern: Information vorhanden, aber nicht direkt (z. B. über Linkout) erreichbar
Taxonomische Information ist in vielen Datenbanken verfügbar, aber sie ist kaum in die Webinterfaces integriert.

Fazit Bei vielen Datenbanken sind taxonomische Informationen vorhanden, meistens als Angabe der taxonomischen Zugehörigkeit der Spezies zu der ein entsprechendes Protein gehört. Information zur Verteilung von Proteinen über verschiedene Taxa ist aber kaum vorhanden. Eine Ausnahme stellt Interpro dar, wo die Verteilung auch rudimentär grafisch dargestellt wird, aber nicht grafisch navigierbar ist. Allen Datenbanken, die taxonomische Informationen anbieten, ist gemein, dass sie als gemeinsame Referenz die NCBI Taxonomie verwenden.

2.1.2. NCBI Taxonomie-Datenbank

NCBI hat eine Datenbank entwickelt, um Spezies taxonomisch zuordnen zu können. Die Datenbank stellt keine Primärquelle von taxonomischen Informationen dar, sondern wurde aus mehreren Datenquellen [7] entwickelt. Dazu wurden taxonomische, phylogenetische und andere Daten, sowie externe Taxonomieexperten zu Rate gezogen, um einen einheitlichen taxonomischen Baum zu entwickeln.

Die NCBI Taxonomie besteht im wesentlichen aus zwei Relationen (Tabellen), 'node' und 'name' (Abbildung 2.1). 'node' enthält alle Knoten, wobei jeder Knoten eine ID, eine Parent-ID und einen Rang hat. Über die Parent-ID kann der Baum konstruiert werden. Der Rang gibt an, um welche Art Knoten es sich handelt (Spezies, Gattung, Ordnung, usw.). In der Relation 'name' sind die entsprechenden Namen der Knoten zu finden. Ein Knoten kann durchaus mehrere Namen haben, d. h. es gibt verschiedene Klassen von Namen: Wissenschaftliche Namen, gängige Namen, Synonyme, usw. Darüberhinaus sind noch weitere Informationen mit der NCBI Taxonomie verknüpft, z. B. genetische Codes, Literatur, usw.

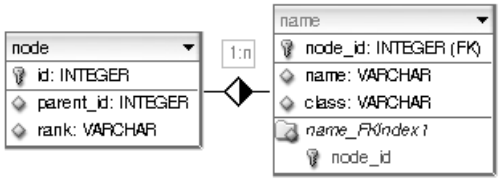

Abbildung 2.1.: Die zwei wichtigsten Tabellen der NCBI-Taxonomie
Die Tabellen 'node' und 'name' enthalten die wesentlichen Informationen, um den NCBI Taxonomie-Baum konstruieren zu können.

Beim NCBI Taxonomiebaum handelt es sich um einen gewurzelten Baum, der im Gegensatz zu den meisten phylogenetischen Bäumen nicht binär ist. Über die Parent-ID, die jeder Knoten besitzt wird seine Struktur festgelegt (Abbildung 2.2).

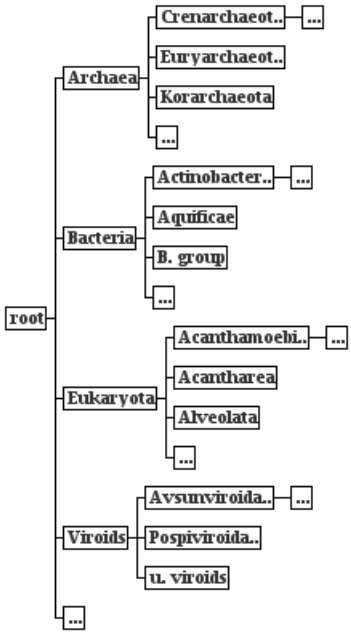

Abbildung 2.2.: Struktur des NCBI Taxonomie-Baums
Die Abbildung zeigt die obersten Ebenen des NCBI Taxonomie-Baums.

2.1.3. Phylogenetische Bäume

Was sind phylogenetische Bäume?

Mit Hilfe phylogenetischer Bäume versucht man Verwandtschaftsverhältnisse und die Evolution von Organismen, bzw. Proteinen/Genen zu untersuchen [9]. Phylogenetische Bäume basieren oft auf multiplen Sequenzalignments. Will man einen phylogenetischen Baum mit verschiedenen Spezies erstellen, muß zunächst ein zu alignierendes Protein ausgewählt werden, welches in allen Spezies vorkommt. Die Auswahl dieses Proteins hat einen entscheidenden Einfluß auf die Qualität des resultierenden Baumes. Es sind vor allem folgende Punkte zu beachten:

- Alle zu untersuchenden Spezies besitzen das Protein
- Die Spezies weisen Unterscheide in dem Protein auf (sonst: Distanz = 0)
- Das entsprechende Gen wurde nicht horizontal zwischen den Spezies transferiert (sonst: Unterschätzung von Distanzen und fehlerhafte Topologie)

Ein phylogenetischer Baum stellt Distanzen zwischen den jeweiligen Spezies dar, bzw. im Umkehrschluß, wie ähnlich sich die Spezies sind. Deshalb werden zunächst all-against-all Alignments berechnet, um jeweils die Distanz eines Knotens (Spezies) zu allen anderen zu ermitteln. Für die Berechnung der Distanz gibts es verschiedene Formeln. Eine übliche Methode zur Distanzberechnung bei Proteinsequenzen ist z. B. die Berechnung aus dem Alignment-Score zu Selfscore[1] Verhältnis [10]:

$$\text{Distanz } d = -ln(\frac{\text{Score}}{\text{SelfScore}}) \tag{2.1}$$

Anhand der einzelnen Distanzen werden die Spezies nun Clustern zugeordnet. Auch hierfür gibt es verschiedene Verfahren. Eine bekannte, relativ einfache Clustering-Methode ist beispielsweise 'Unweighted Pair Group Method with Arithmatic Mean' (UPGMA) [11]. Bei diesem Verfahren wird die Distanzmatrix mehrmals durchlaufen, wobei jeweils die beiden ähnlichsten Knoten (geringste Distanz) zu einem neuen Knoten, dessen Distanzen aus den Mittelwerten der beiden Knotendistanzen bebildet werden, zusammengefaßt. Der Clustering-Prozess ist beendet sobald alle Knoten auf diese Weise einem Cluster zugeordnet wurden. Das Ergebnis ist ein binärer Baum (Abbildung 2.3).

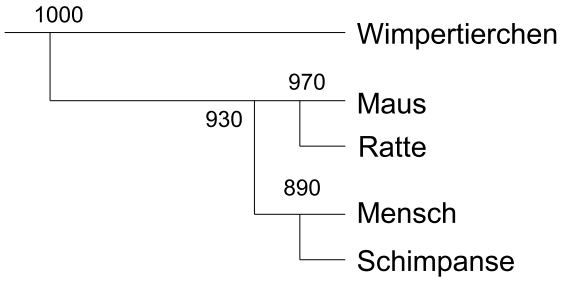

Abbildung 2.3.: Beispiel: Fiktiver phylogenetischer Baum mit Angabe der Bootstrap-Unterstützung

[1] Selfscore: Alignment einer Sequenz mit sich selbst; höchstmöglicher Alignment-Score

Ein phylogenetischer Baum ist eine Schätzung. Man versucht, mit Hilfe gegebener Daten (Sequenzen) Parameter zu schätzen. Die Parameter in diesem Fall sind die einzelnen Cluster. Um die Qualität einer solchen Schätzung beurteilen zu können, gibt man üblicherweise ein Vertrauensinterval an. Dieses gibt an, in welchem Ausmaß sich der Parameter ändern kann, wenn man die Schätzung mit anderen Daten aus derselben Grundmenge wiederholt. Das Vertrauensinterval kann man mit statistischen Methoden ermitteln, sofern die Verteilung der Daten bekannt ist. Im Falle der Schätzung eines phylogenetischen Baumes ist diese Verteilung allerdings nicht bekannt, außerdem stehen keine weiteren Daten zur Verfügung.
Deshalb bedient man sich der sog. Bootstrapping-Methode [12, 13]. Aus den Sequenzen wird ein multiples Alignment gebildet. Dieses multiple Alignment stellt eine $m \times n$ Matrix dar, mit m Zeilen (Sequenzen) und n Spalten (Länge der Sequenzen). Aus dieser Matrix werden nun neue Datensets generiert, indem n-mal zufällig (mit Zurücklegen) Spalten aus der Matrix entnommen und zu einem neuen Set zusammengestellt werden. Damit wird anschließend ein neuer phylogenetischer Baum berechnet.
Auf diese Weise werden mehrere Bäume erstellt und jeweils das Auftreten der Cluster des Ursprungsbaumes gezählt. Im Ursprungsbaum kann man nun als Vertrauenskriterium bei jedem Cluster die Bootstrapunterstützung angeben, d. h. in wievielen Fällen dieses Cluster auch im Bootstrappingprozess vorgekommen ist (bei dem fiktiven Baum in Abbildung 2.3, hätte der Cluster Ratte-Maus beispielweise eine Bootstrapunterstützung von 97%).

Phylogenetische contra taxonomische Bäume

Im Gegensatz zur rein phylogenetischen Klassifikation bezieht die Taxonomie weitere Informationen ein. Bereits im 18. Jahrhundert stellte Carl von Linné fest, dass sich Spezies hierarchisch ordnen lassen, obwohl er ein Verfechter der Unveränderlichkeit der Arten war. Die Gruppierung von Spezies, Zuordnung zu Gattungen, usw. geschah nach morphologischen Gesichtspunkten. Veröffentlicht hat er sein taxonomisches System in der Systema Naturae [14], welche er wiederholt aktualisierte. Mit diesem Werk legte er den Grundstein für die moderne Taxonomie.
Heute werden allerdings neben den rein morphologischen Merkmalen auch die phylogenetischen Merkmale berücksichtigt. Nichtsdestotrotz sind phylogenetische Bäume keine taxonomischen Bäume. Die Unterschiede zeigen sich bereits an der Oberfläche. Aufgrund der Clustering-Methoden sind phylogenetische Bäume meist binäre Bäume. Aus taxonomischer Sicht ergibt eine solche binäre Struktur allerdings meist keinen Sinn. Davon abgesehen kann ein phylogenetischer Baum alleine auch keinen taxonomischen Anspruch erfüllen, da die Methoden fehlerbehaftet sind. Statt des ganzen Organismus untersucht man mit der phylogenetischen Methode oft nur ein oder einige wenige Proteine. Bei der Auswahl dieser Proteine kann es bereits zu Verfälschungen kommen (siehe Kapitel 2.1.3). Proteine, die von komplexen evolutionären Ereignissen betroffen sind (z. B. horizontaler Gentransfer, Duplikation, Deletion), führen zu fehlerhaften Ergebnissen, wenn man sie zur Berechnung von Phylogenie verwendet [15]. Ebenso ist die Annahme einer konstanten evolutionären Geschwindigkeit, wie sie z. B. der UPGMA Clustering-Algorithmus voraussetzt, nicht korrekt, da man weiß, daß sich Gene in unterschiedlichen Organismen nicht exakt mit der gleichen Geschwindigkeit entwickeln [16]. Es gibt aber auch andere phylogenetische Verfahren auf der Ebene ganzer Genome, bei welchen der Einfluss einzelner Proteine, die von komplexen evolutionären Ereignissen, wie z. B. horizontalen Gentransfer betroffen sind, begrenzt ist [17].
Auch wenn die Phylogenie alleine keinen taxonomischen Anspruch erfüllen kann, so ist sie doch eine wichtige Datengrundlage für die Taxonomie [18]. Allerdings muß man sich der Unterschiede bewußt sein, wenn man mit beiden Bereichen arbeitet.

Beispiel: Peer Bork's Tree of Life

Peer Bork stellte im Mai 2006 eine Methode zur automatischen Erstellung eines phylogenetischen Baumes des Lebens vor [6]. Dabei dienen bestimmte Proteine als Grundlage für die Berechnung des Baumes. Wie bereits in Kapitel 2.1.3 erwähnt, ist nicht jedes Protein für die Berechnung einer Phylogenie geeignet. Um geeignete Proteine zur Berechnung der Phylogenie zu finden, untersuchte die Arbeitsgruppe um Peer Bork alle 'Clusters of Orthologous Groups' (COGs) der NCBI COG-Datenbank. COGs sind Gruppen homologer Proteine, die in mehreren Organismen vorkommen, und dort jeweils die gleiche Aufgabe erfüllen.
Per Definition umfaßt ein COG mehrere Organismen, keineswegs jedoch alle. Deswegen wurde nach COGs gesucht, die universell sind, d. h. möglichst in allen Spezies vorkommen. Dabei handelt es sich meist um COGs, die in allen Organismen primäre, für die Lebensfähigkeit essentielle Aufgaben erfüllen, z. B. ribosomale Proteine.
Eine weitere wichtige Voraussetzung für die Eignung als Datengrundlage für phylogenetische Bäume ist, dass die Gene der betreffenden Proteine nicht horizontal transferiert wurden. Im Gegensatz zum vertikalen Gentransfer (Vererbung), werden beim horizontalen Gentransfer Gene innerhalb einer Generation und auch über die Artengrenze hinweg übertragen. Das heißt, dass ein solches Protein nahezu identisch in nicht verwandten Arten vorkommen kann. Deshalb würde ein phylogenetischer Baum, der solche Proteine als Datengrundlage verwendet, Verwandschaften vortäuschen, die in Wirklichkeit überhaupt nicht gegeben sind. Aus diesem Grund wurden die COG-Proteine hinsichtlich horizontalen Gentransfers untersucht, und COGs, die horizontal gentransferierte Proteine beinhalten, ausgeschlossen.
Nach dieser Selektion sind 31 universelle COGs übrig geblieben (siehe Tabelle A.2, Seite 65), die als Datengrundlage für phylogenetische Bäume geeignet erschienen. Mit Hilfe dieser COGs wurde dann ein phylogenetischer Baum erstellt (siehe Appendix A.2, Seite 64). Peer Bork's Baum des Lebens (im nachfolgenden kurz 'Bork-Baum' genannt) umfaßt 191 Spezies, und ist - wie üblich - ein binärer Baum. Der Baum ist sowohl in bewurzelter, als auch in unbewurzelter Form verfügbar.

2.2. SIMAP (Similarity Matrix of Proteins)

2.2.1. Einleitung

Aufgrund des Zusammenhangs 'ähnliche Sequenz' ⇒ 'ähnliche Struktur' ⇒ 'ähnliche Funktion' sind Sequenzvergleiche (engl. Alignments) bis heute ein essentielles Mittel zur Genom- und Proteomanalyse [19]. Zwei Sequenzen lassen sich relativ schnell alignieren. Bewegt man sich aber in den Dimensionen 'Sequenz gegen komplettes Genom' oder sogar 'Sequenz gegen mehrere Genome' können diese Berechnungen auf einem einzelnen Rechner oft nicht mehr in sinnvoller Zeit (Sekunden oder wenige Minuten) durchgeführt werden. Um schnelle Suchen nach ähnlichen Sequenzen in einer großen Ausgangsmenge zu ermöglichen, ist eine Vorberechnung der Alignments notwendig, was zudem den positiven Effekt hat, dass wiederholte Berechnungen der gleichen Alignments verhindert werden.

SIMAP [1] ist eine solche Datenbank. Sie enthält die Ähnlichkeitsmatrix von aktuell etwa 6 Millionen Proteinsequenzen. Von jeder neu hinzukommenden Proteinsequenz werden die potentiellen Homologen mittels FASTA [5] ermittelt und anschließend die exakten Alignments mit dem Smith-Waterman-Algorithmus [20] berechnet und in der Datenbank ablegt. Diese Berechnungen werden vom einem Rechencluster, und seit Anfang 2006 auch verteilt über das Internet

mittels eines Boinc-Clients ausgeführt. Auf diese Weise kann jeder PC-Besitzer Rechenzeit für das SIMAP Projekt zur Verfügung stellen (ähnlich Seti@Home, das inzwischen auch auf die Boinc-Platform umgestiegen ist).
Die Datenbank enthält zu jedem Proteinpaar die Informationen 'Sequenz IDs', 'Smith-Waterman-Score', 'Identiy-Score', 'Similarity-Score', 'Overlap Size', sowie die Start- und Endkoordinaten des Alignments. Diese Alignment-Informationen sind in Form von Dateien, welche in Verzeichnissen strukturiert sind, abgelegt. Ein Teil der Sequenz-ID dient dabei als Schlüssel und kodiert den Verzeichnispfad, der andere Teil bildet den Dateinamen. Das heißt, für jede Sequenz gibt es eine Datei, die die Homologie-Informationen zu allen anderen Sequenzen der Datenbank enthält. Diese Datenstruktur stellte sich im Laufe der Entwicklung der SIMAP-Datenbank als die performanteste und v. a. am besten skalierbare Lösung heraus.
Neben der Lokalisation der Homologiedaten, dient die Sequenz-ID generell als eindeutiger Schlüssel, über den auch weitere Informationen zu den Proteinen eingeholt werden, z. B. taxonomische Informationen, Proteinfeatures, usw. Der Zugriff auf die Datenbank erfolgt normalerweise nicht direkt über das Dateisystem, sondern über eine Logikschicht, bestehend aus Enterprise Java Beans (Abbildung 2.4). Die Prsentationsschicht bietet dem externen Benutzer auf unterschiedliche Arten Zugang zur SIMAP-Datenbank. Es besteht sowohl ein webbasierter Zugang (Webinterface) über das Content Management System (CMS) OpenCMS, als auch über Webservices. Webservices erlauben anderen Entwicklern, eigene Applikationen mit der SIMAP-Datenbank zu verbinden. Für interne Applikationen besteht außerdem die Möglichkeit, die Beans der Logikschicht direkt anzusprechen.

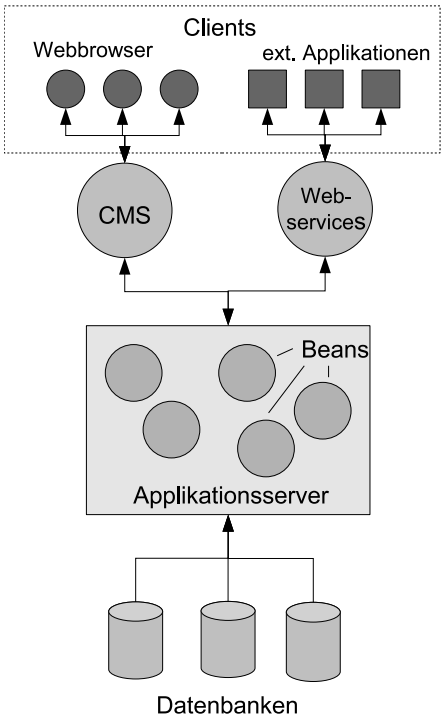

Abbildung 2.4.: SIMAP Architektur
3-Schichten-Architektur, bestehend aus Datenschicht (Datenbank), Logikschicht (Applikationsserver) und Präsentationsschicht (CMS, Webservices).

Das SIMAP-Webinterface (`http://mips.gsf.de/simap/`, Screenshot siehe Abb. A.1, Seite 62) ermöglicht die Volltextsuche nach Proteinen und die Suche nach Homologen zu diesen Proteinen. Die Homologensuche läßt sich mit verschiedenen Parametern konfigurieren, z. B. maximale Anzahl von angezeigten Homologen, maximaler EValue, Suche nur in bestimmten Datenbanken/Taxa. Das Ergebnis einer solchen Suche ist eine Liste von Proteinen mit Angabe der Beschreibung, verschiedenen Alignmentergebnissen (Score, EValue, usw.), falls gewünscht das Alignment selbst, usw.
Da diese Ergebnisliste aber u. U. hunderte oder tausende von Homologen enthalten kann, wäre es für den Benutzer hilfreich, wenn er eine zusätzliche Navigationshilfe, wie z. B. einen grafischen taxonomischen Baum zur Verfügung hätte. Mit dem taxonomischen Baum könnte sich der Benutzer auch schnell einen Überblick verschaffen, wie sich die Homologen taxonomisch verteilen. Die Implementierung einer solchen Softwarekomponente ist Teil dieser Diplomarbeit.

2.2.2. Zahlen und Fakten

Datenbanken	559
Proteine	21.131.070
Sequenzen	6.115.287
Aminosäuren	2.042.396.417
FASTA Hits	81.946.834.951
Speicherbedarf	ca. 1 TB

Tabelle 2.2.: Umfang der SIMAP Datenbank
(Stand: Sept. 2006, aktuelle Statistik:
`http://mips.gsf.de/genre/proj/simap/release.html`)

2.3. Taxonomische Analyse von Metagenomics-Daten

2.3.1. Metagenomics

Unter Metagenomics (auch Environmental oder Community Genomics genannt) [21] versteht man die Analyse von Genomen, die aus Umweltproben stammen (im Gegensatz zu Laborkulturen). Tatsächlich können viele Bakterien nicht isoliert im Labor kultiviert werden, da sie komplexe Umweltbedingungen und/oder Gemeinschaften (verschiedene Arten, die denselben Lebensraum teilen) benötigen, um existieren zu können. Das Genom eines solchen Bakteriums zielgerichtet zu sequenzieren, ist Ziel der Metagenomik, das erstmals mit dem 2006 sequenzierten Anammox-Community Genom erreicht wurde [4]. Um die Genome sequenzieren zu können, bedient man sich normalerweise der Shotgun-Methode [22] (Abbildung 2.5). Dabei wird die DNA mit Restriktionsenzymen zerkleinert. Die daraus entstandenen Bruchstücke werden sequenziert und die Sequenzen mit Hilfe von Overlap-Alignments zu sog. Contigs zusammengesetzt. Das Ergebnis einer solchen Sequenzierung ist eine Vielzahl von Contigs, die zu verschiedenen Genomen gehören. Man versucht dann, die Contigs des Zielorganismus zu identifizieren und zu einem möglichst vollständigem Genom zusammenzusetzen.
Um einen ersten Überblick über die bakterielle Gemeinschaft zu bekommen, bedient man sich häufig der 16S rRNA Methode. Diese kurzen ribosomalen RNAs ($\sim$ 1,5 kb) kommen in allen bakteriellen Spezies vor und eignen sich gut, um einzelne Spezies identifizieren oder zumindest voneinander unterscheiden und phylogenetische Analysen betreiben zu können [23]. Diese Methode kommt v. a. in Diversitätsuntersuchungen, z. B. in der mikrobiellen Ökologie zum Einsatz, wird aber auch im Umfeld von Metagenomics-Projekten eingesetzt.

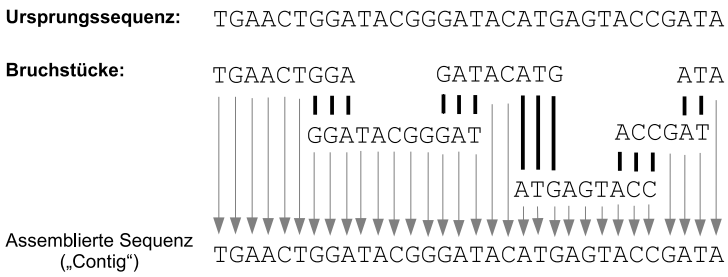

Abbildung 2.5.: Assemblierung von Shotgun-Sequenzen
Die DNA wird mit Restriktionsenzymen zerkleinert, die Bruchstücke sequenziert und deren Sequenz anschließend mit Hilfe von Overlap-Alignments wieder zusammengesetzt.

2.3.2. Methoden

Um die Community-DNA in einzelne Genome auftrennen zu können (sog. 'Binning'), bzw. bestenfalls sogar einer bestimmten Spezies zuordnen zu können, gibt es verschiedene bioinformatische Methoden [24].

Clustering

Ein Contig, bzw. die ORFs[2] auf einem Contig haben verschiedene charakteristische Eigenschaften. Zum Beispiel:

- GC-Gehalt: Anteil von Guanin und Cytosin in der DNA-Sequenz
- Codon-Usage: Spezies unterscheiden sich darin, welche Codons sie bevorzugt für die Codierung bestimmter Aminosäuren verwenden.
- Tetra-, Penta-, ... Nukleotidfrequenzen

Diese Eigenschaften kann man nun für die einzelnen Contigs bestimmen, und anschließend einen Clustering-Algorithmus anwenden, um die Contigs zueinander zuordnen, bzw. von anderen unterscheiden zu können.

Phylogenie

Eine weitere Methode zur Analyse eines Community-Genomes ist die phylogenetische Methode. Das heißt, man sucht nach ähnlichen Sequenzen und berechnet dafür phylogenetische Bäume. Auf diese Weise lassen sich die einzelnen Spezies und deren Verwandtschaftsverhältnisse im Community-Genom finden [25]. Da die einzelnen phylogenetischen Bäume jedoch noch mittels Bootstrapping verifiziert werden müssen (siehe Kapitel 2.1.3, Seite 10) und man es mit einer großen Anzahl zu untersuchender Sequenzen zu tun hat, ist diese Verfahren sehr aufwendig.

[2] ORF: 'Open Reading Frame'; Ein nicht durch Stop-Codons unterbrochene DNA-Sequenz; notwendige aber nicht hinreichende Bedindung dafür, dass eine DNA-Sequenz für ein Protein codiert

Referenzbaum-Methode

Diese Methode soll im Rahmen der Diplomarbeit untersucht werden (siehe Kapitel 4, Seite 36). Die Fragestellung lautet, inwieweit es möglich ist, ein Community-Genom mit Hilfe eines Referenzbaumes taxonomisch zu analysieren.
Voraussetzungen der Methode sind ein Referenzbaum und eine Proteindatenbank, die Marker-Proteine möglichst vieler Spezies des Referenzbaumes enthält. Unter Marker-Proteine sind Proteine zu verstehen, die in möglichst allen Spezies vorkommen und nicht horizontal gentransferiert wurden.
Als Referenzbaum werde ich den Bork-Baum (siehe Kapitel 2.1.3) verwenden, und die gleichen Marker-Proteine, die diesem Baum zugrunde liegen, benutzen (universell verteilte, nicht horizontal gentransferierte COG-Proteine). Als Proteindatenbank dient SIMAP (siehe Kapitel 2.2), da SIMAP ein Repository aller öffentlichen Proteindatenbanken darstellt, und jedes Protein mit vollständiger, einheitlicher taxonomischer Information verknüpft ist.

2.3.3. Anammox-Community Genom

Anfang 2006 gelang es, das Genom des Bakteriums *Kuenenia stuttgartiensis* zu sequenzieren [4]. Dieses Bakterium gehört zur Gruppe der Anammox-Bakterien ('anaerobic ammonium oxidation'), und war das erste und bisher einzige dieser Art Bakterien dessen Genom komplett sequenziert wurde.
Das Besondere an diesen Bakterien ist ihr einzigartiger Stoffwechsel, der Ammoniak und Nitrit unter anaeroben Bedingungen zu molekularen Stickstoff oxidiert. Deshalb ist dieses Bakterium nicht nur für Genetiker und Biochemiker äußerst interessant, sondern auch für Ökologen (Stickstoffkreislauf), Biotechnologen (z. B. bakterielle Abwasserreinigung) und andere.
Aufgrund theoretischer Überlegungen postulierte bereits 1977 der Wiener Physikochemiker Engelbert Broda die Existenz solcher, zu dieser Zeit noch unbekannten Bakterien [26]. Im Jahr 1999 gelang es holländischen Forschern um Mike Jetten von der Radboud University in Nijmegen ein solches Bakterium tatsächlich zu identifizieren [27]. Schließlich wagte man sich an die Sequenzierung dieses Bakteriengenomes. Dies war eine besondere Herausforderung, da das Bakterium nicht isoliert kultiviert werden kann. Aus diesem Grund wurden die Methoden der Metagenomik verwendet.
Da bei diesem Projekt mehr die Sequenzierung von *K. stuttgartiensis* als die Analyse der gesamten Bakteriengemeinschaft im Vordergrund stand, versuchte man dieses Bakterium in dem Bioreaktor möglichst hoch anzureichern. Dies gelang bis zu einem Anteil von $73 \pm 5\%$ *K. stuttgartiensis* im Reaktor. Die Probe wurde ebenfalls auf 16S rRNA untersucht, wobei festgestellt wurde, dass sich in dem Community Genom insgesamt mindestens 29 verschiedene taxonomische Einheiten befinden.
Obwohl das Ziel des Projektes die Sequenzierung von *K. stuttgartiensis* war, soll mit den vorhandenen Daten versucht werden, diese bakterielle Gemeinschaft um *K. stuttgartiensis* genauer zu untersuchen. Als Datengrundlage steht hierfür das Anammox-Community Genom zur Verfügung, welches bereits um die Contigs bereinigt wurde, die *K. stuttgartiensis* zugeordnet werden konnten.

2.4. Software, APIs, Spezifikationen

2.4.1. Java Enterprise Edition/Enterprise Computing

Die Java Standard Edition (J2SE) [28] stellt grundsätzliche Software-Elemente (Speicherstrukturen, Methoden, usw.) zur Verfügung. Darüberhinaus sind aber auch weitergehende Frameworks enthalten, wie z. B. Swing/AWT zur Programmierung von grafischen Benutzerschnittstellen (GUIs). Mit Hilfe der J2SE kann man einfachste Konsolenanwendungen bishin zu komplexen grafischen Standalone-Applikationen entwickeln. An die Grenzen der J2SE stößt man aber, wenn man verteilte Anwendungen entwickeln möchte. Das heißt, Anwendungen die nicht mehr nur auf einem Rechner laufen.
Methoden zur Kommunikation über Netzwerke sind zwar bereits in der J2SE enthalten, aber auf einem niedrigen Abstraktionslevel. Java-Applikationen können über sog. Sockets miteinandern kommunizieren, allerdings muß sich der Entwickler dabei um viele Dinge selbst kümmern, z. B. Verbindungsaufbau und -abbau, Synchronisation bei der Datenübertragung, usw. Eine etwas abstraktere Möglichkeit für die Kommunikation zwischen verteilten Java-Komponenten bietet RMI (Remote Method Invocation). Damit ist es möglich, Methoden bei entfernten Komponenten auszuführen, aber auch hier erwarten den Entwickler viel Handarbeit und Fallstricke. Darüberhinaus handelt es sich bei beiden Methoden um reine Java-Methoden, d. h. Komponenten, die mit einer anderen Programmiersprache entwickelt wurden, bleibt der Zutritt verwehrt (sofern man nicht zusätzlich weitere Frameworks wie z. B. Corba einsetzt).
Will man eine reine Java-Applikation entwickeln, bei der der Grad der Verteilung niedrig ist, und somit die Netzwerkfunktionalität eine untergeordnete Rolle spielt, empfiehlt es sich dennoch, die Methoden des J2SE zu verwenden, da man den typischen Overhead der Java Enterprise Edition (J2EE) [29] vermeidet. Bei Applikationen, deren Hauptaugenmerk bei Verteilung und Netzwerkfähigkeit liegt, ist jedoch J2EE das Mittel der Wahl.
Diese Art von Applikation werden unter dem Begriff 'Enterprise Applications', bzw. 'Enterprise Computing' zusammengefaßt. J2EE stellt Frameworks bereit, um verteilte Softwarekomponenten entwickeln zu können, ohne dass sich der Entwickler um Details der Kommunikation u. ä. kümmern muß. Er kann sich ganz auf die Funktionalität der Applikation konzentrieren. Neben den J2EE-Frameworks gibt es aber auch noch weitere Frameworks im Bereich Enterprise Computing, wie z. B. Webservices, SOAP, u. ä., die größtenteils auch gut mit J2EE harmonieren.

J2EE-Applikationen sind i. d. R.:

- Komponentenbasiert: Es handelt sich um eigenständige, gekapselte Softwaremodule
- Verteilt: Können sowohl auf verschiedene Softwaresysteme als auch auf verschiedene physikalische Rechner verteilt werden.
- Mehrschichtig: Können in eine mehrschichtige Struktur geordnet werden, z. B. klassiche Drei-Schichten-Architektur bestehend aus Backend, Middlelayer, und Frontend.

Eine klassische J2EE-Anwendung besteht meistens aus vielen einzelnen Komponenten und ist keine Standalone-Applikation. Eine Komponente stellt Funktionalität zur Verfügung - in ihr wird die sog. Geschäftslogik implementiert. Sie kümmert sich aber nicht darum, wie sie mit anderen Komponenten interagieren kann, d. h. sie muß nichts über das zu verwendende Netzwerkprotokoll wissen, wie die Kommunikation initiiert werden kann, usw. Für diese Funktionalität ist der sog. Applikationserver verantwortlich.

Nachdem man ein J2EE-Modul implementiert hat, übergibt man es dem Applikationsserver ('Deployment'). Der Applikationsserver stellt dann für die Komponente einen sog. Container zur Verfügung, der dafür sorgt, dass die Komponente über die jeweiligen, dafür vorgesehenen Schnittstellen erreichbar ist. Er kümmert sich um Sicherheitsaspekte, darum dass bei Clientanfragen genügend Instanzen der Komponente zur Verfügung stehen, dass Instanzen wieder 'aufgeräumt' werden, wenn sie nicht mehr benötigt werden, usw.

Einsatzbereich

J2EE, bzw. Enterprise Computing kommt dann zum Einsatz, wenn

- Softwarekomponenten verteilt sind, und über Netzwerk miteinander kommunizieren müssen;
- Dienste für externe Applikationen bereitgestellt werden müssen;
- Dienste plattformunabhängig bereitgestellt werden müssen;
- die Komponenten unabhängig von der Programmiersprache, mit der sie erstellt wurden, miteinander kommunizieren müssen.

Die SIMAP-Datenbank benötigte eine Zugriffsschicht, die sowohl die Datenzugriff abstrahiert als auch eine Geschäftslogik implementiert (z. B. zum Filtern von Ergebnissen). SIMAP sollte über ein Webinterface, sowie externen und internen Applikationen zugänglich sein. Die Architektur sollte modularsiert, erweiterbar und verteilt einsetzbar sein (siehe Abbildung 2.4, Seite 13). All diese Punkte sprechen für die Verwendung von J2EE, insbesondere Enterprise Java Beans (EJB).

Unter dem Begriff Enterprise Computing sind mehrere Frameworks zusammengefaßt. Für die Diplomarbeit relevante Frameworks werde ich im folgenden kurz vorgestellen.

JSP/Servlets

JSP (Java Server Pages) und Servlets sind Enterprise-Komponenten, die sich besonders gut für die Webentwicklung eignen [30]. Durch die Möglichkeit der Erzeugung von dynamischen Webseiten entwickelten sich angefangen von kleinen Spielereien (z. B. Besucherzähler auf Homepages) im Laufe der Zeit ganze Applikationen, die statt einer herkömmlichen grafischen Benutzeroberfläche den Webbrowser als 'Thin Client'[3] einsetzen.

Diesen Webapplikationen ist folgendes Prinzip gemein: Der Benutzer schickt eine Anfrage an den Server (Request), dieser prozessiert die Anfrage und schickt eine Antwort an den Benutzer zurück (Response). Servlets sind für diesen Zweck geschaffen. Der wesentliche Teil eines Servlets ist die Methode *service(ServletRequest request, ServletResponse response)*. Sie nimmt einen Request (z. B. von einem Webbrowser) entgegen, bearbeitet ihn und antwortet mit einem Response. Ein Servlet ist eigentlich unabhängig vom Protokoll über welches dieser Prozess stattfindet. Dennoch wird hauptsächlich das HTTPServlet eingesetzt, das HTTP als Kommunikationsprotokoll verwendet, einerseits weil Servlets meistens dazu benutzt werden, dynamischen HTTP-Content zur Verfügung zu stellen, und andererseits weil für andere Protokolle eigene Frameworks entwickelt und favorisiert wurden.

[3] 'Thin Client': Ein Client, der sich hauptsächlich auf Ein-/Ausgabe beschränkt, möglichst alle Daten von einem Server bezieht und selbst keine umfangreiche Funktionalität implementiert.

Ein Servlet nimmt dem Entwickler einige Aufgaben ab. Wie bereits erläutert, kümmert es sich um die Handhabung des Protokolls und darüberhinaus auch um das Session Handling. Aus o. g. Prinzip (Request→Verarbeitung→Response), das für die Bereitstellung von statischem Content völlig ausreichend war, ergibt sich hinsichtlich dynamischer Erzeugung von Content nämlich ein Problem: Request ist gleich Request, es läßt sich nicht zwischen verschiedenen Clients unterscheiden. Man kann keine Verbindung zum Client aufrechterhalten (HTTP ist ein sog. verbindungsloses Protokoll). Dieses Problem versucht man mit sog. 'Sessions' zu lösen. Beim ersten Request eines Clients wird eine Session-ID erzeugt, mit welcher sich der Client dann stets bei der Applikation identifiziert. Eine Session dauert üblicherweise so lange an, bis der Benutzer den Webbrowser schließt, kann aber auch dauerhaft in Form eines 'Cookies' auf dem Rechner des Benutzers abgelegt werden. Sowohl für die Handhabung von Sessions als auch Cookies gibt es Java-Klassen, die dem Entwickler bei der Benutzung von Servlets die Arbeit erleichtern.

Java Server Pages (JSP) sind HTML-Seiten mit eingebettetem Java-Code (vergleichbar mit eingebettetem PHP-Code). Diese Webseiten werden nicht direkt an den Webbrowser ausgeliefert, sondern werden vorher vom Applikationsserver prozessiert, wobei deren Java-Code kompiliert und ausgeführt wird. JSP und Servlets sehen auf den ersten Blick sehr verschieden aus. Während Servlets Java-Klassen sind, die meist HTML-Code enthalten, den sie via Response an den Client weitergeben, sind JSPs HTML-Seiten, die Java-Code enthalten. Tatsächlich sind JSP und Servlet nur zwei Seiten der gleichen Medaille. Wird eine JSP-Seite auf einem Applikationsserver bereitgestellt, wird diese prozessiert und daraus ein Servlet abgeleitet. Die Funktionalität des Servlets bildet der Java-Code, der in der JSP-Seite enthalten ist, wobei dem Servlet-Response jeweils die entsprechenden HTML-Teile der JSP-Seite hinzugefügt werden. JSP setzt man dann ein, wenn man es hauptsächlich mit statischem Webcontent zu tun hat, der einige kleine Teile dynamischen Inhalts enthält. Servlets dagegen werden dann verwendet, wenn man es hauptsächlich mit dynamischem Inhalt zu tun hat.
Beide Techniken werden gerne verwendet, um Funktionalität, die durch EJBs implentiert wird, einem Client via Webbrowser zur Verfügung zu stellen. So nutzen z. B. Java-basierte Content Management Systeme wie OpenCMS JSPs und Servlets zur Integration von EJBs in die Webumgebung.

Enterprise Java Beans (EJB)

Enterprise Java Beans verbinden das 'Java Beans'-Konzept [31] mit dem Enterprise Computing. Ein Java Bean ist eine Komponente, die einen öffentlichen Standardkonstruktor (ohne Argumente) und öffentliche Zugriffsmethoden (Getter und Setter) hat, Introspektion erlaubt und serialisierbar ist. Wird eine solche Komponente nun von einem Applikationsserver, der dafür sorgt, dass sie instanziiert wird, ihre Methoden lokal als auch entfernt angeboten werden, usw. bereitgestellt, spricht man von einem Enterprise Java Bean (EJB) [3].
Es gibt drei Arten von EJBs:

- Session Beans
- Entity Beans
- Message driven Beans

Session Beans dienen dazu, eine Clientanfrage zu beantworten. Eine Instanz des Beans wird erzeugt, sobald ein Client anfragt. Die Anfrage wird bearbeitet und die Bean-Instanz i. d. R. anschließend wieder entfernt. Eine Instanz kann aber auch an einen Client gebunden und über

eine ganze Session gehalten werden. Diese Art von Session Beans nennt man 'stateful'. Der Client kann über eine ganze Session hinweg mit dem gleichen Bean arbeiten.

- Stateless: Für jede Clientanfrage eine eigene Bean-Instanz
- Stateful: Für jeden Client eine eigene Bean-Instanz

Persistent ist aber keine Art dieser Beans. Nach Beantwortung der Anfrage (Stateless), bzw. nach einer gewissen Zeit, in der keine erneute Anfrage desselben Clients erfolgt (Stateful), wird die Bean-Instanz und damit ihr Zustand wieder entfernt.
Entity Beans dagegen sind persistent. Ein Entity Bean stellt einen bestimmten Datensatz dar, der vom Client gelesen und verändert werden kann. Das Bean selbst kann die Beschaffung und das Abspeichern der Daten übernehmen (Bean managed Persistence, BMP), dazu muß der Entwickler dann die entsprechenden Methoden implementieren, oder es bleibt dem Container überlassen (Container managed Persistence, CMP), dem man beim Deployen des Beans mitteilen muß, wie er dies tun soll. Bei Entity Beans gibt es die Unterscheidung nach stateful oder stateless nicht, sie sind per se stateful.
Message Driven Beans sind vergleichbar mit Session Beans. Ihr Einsatzzweck ist ebenfalls die Beantwortung von einzelnen Clientanfragen. Im Gegensatz zu den Session Beans verwenden sie jedoch eine asynchrone Kommunikation mittels Java Message System (JMS). Das heißt, ein Client kann eine Anfrage an das Bean schicken, weiterarbeiten und erhält irgendwann die Antwort. Er muß nicht auf das Bean warten.

Ein Bean besteht üblicherweise aus drei Teilen: Client-Interfaces, Home-Interfaces und Implementierung (bei Message driven Beans reicht die Implementierung, da die Interfaces bereits vom JMS bereitgestellt werden). In den Client-Interfaces sind alle Methoden definiert, die dem Client durch das Bean zur Verfügung gestellt werden. Es kann sowohl ein Local-Interface definiert werden, das alle Methoden für lokale Clients (und auch ausschließlich für diese) enthält, als auch ein Remote-Interface, das alle Methoden für externe Clients enthält. Lokale Clients sind z. B. Beans, die im selben Container laufen. In den Home-Interfaces (auch hier gibt es ein Local- und ein Remote-Interface) werden die Methoden definiert, die bei der Initialisierung des Beans ausgeführt werden (Abbildung 2.6).

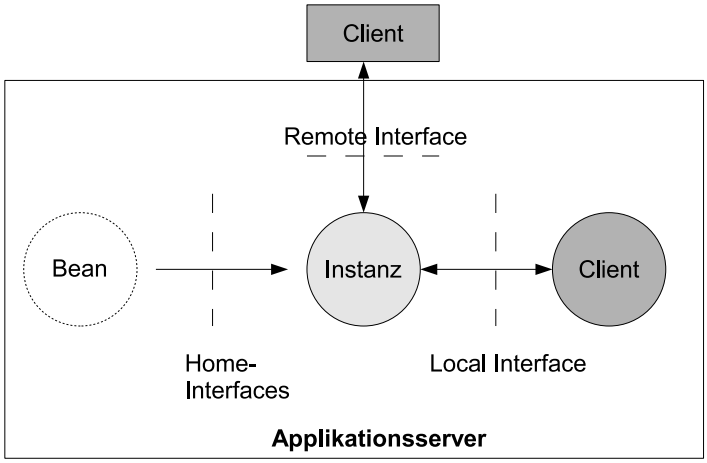

Abbildung 2.6.: EJB: Home- und Client-Interfaces
Verwendung der verschiedenen Interfaces eines Enterprise Java Beans

Durch die neue EJB Spezifikation 3.0 und Java 1.5 wurde die Entwicklung von Enterprise Java Beans erheblich erleichtert [32]. So ist es nun sogar möglich, ein Bean mit nur einer Klasse zu implementieren. Mit Hilfe von Annotations (Deskriptoren vor Methoden und Klassen) kann der Applikationsserver besagte Interfaces selbst extrahieren, benötigte Resourcen automatisch bereitstellen (z. B. Datenbankverbindungen), usw.

Für die SIMAP Datenbank schien der Einsatz von EJB ideal. Man kann Funktionalität in Form von Enterprise Java Beans hinzufügen oder entfernen, ohne andere Applikationen zu gefährden. Das System ist übersichtlich, gut wartbar, verteilbar und gut skalierbar. In SIMAP kommen hauptsächlich stateless Session Beans zum Einsatz. Session Beans deshalb, weil es keine Anwendung gibt, bei der ein Client Daten verändern und abspeichern müßte, und stateless, weil die meisten Anwendungen keine Aufrechterhaltung der Clientverbindung benötigen ('Request→Process→Response' Prinzip völlig ausreichend).

2.4.2. Grafikformate/-frameworks

Formate

Ein Bild kann als eine geordnete Sammlung von einzelnen Bildpunkten (Pixeln) betrachtet werden. Ein Pixel besteht aus einzelnen Farbwerten. Am gebräuchlichsten ist der RGB-Farbraum mit jeweils 8 bit für den Rot-, Grün- und Blauwert. Dies entspricht einem Farbraum von ca. $(2^8)^3 = 16.7$ Millionen Farben. Zusätzlich zu diesen 3 Bytes für die Grundfarben Rot, Grün und Blau gibt es bei manchen Formaten noch einen vierten Wert, den sog. Alpha-Wert, der die Transparenz eines Pixels angibt. Eine Möglichkeit ein Bild abzuspeichern ist, einfach die Farbwerte der einzelnen Pixel abzuspeichern. Diese Formate nennt man Rastergrafiken. Ein mittelgroßes Bild von 800×600 Pixeln und RGB-Farbraum mit Alphakanal nimmt so auf dem Datenträger ca. $800 \times 600 \times 4\text{byte} = 1.92$ MB in Anspruch. Bei der heutigen Kapazität und dem Preis von Datenspeichern stellt das kein großes Problem dar, im Gegensatz zu den Anfangszeiten des Computerzeitalters. Nach wie vor ist jedoch die Datenübertragung ein Problem, weshalb man versucht, die Daten bei möglichst vollständiger Erhaltung der Information soweit zu reduzieren wie möglich. Zur Komprimierung von Grafikdaten gibt es verschiedene Verfahren. Die v. a. im Internetbereich gebräuchlichsten will ich kurz vorstellen.

Graphics Interchange Format (GIF)
GIF [33] wurde 1987 von Compuserve entwickelt, und stellte zu Beginn des Internetzeitalters den de-facto Standard zur Bildübertragung im Internet dar. Die Daten werden mittels Wörterbuchalgorithmus verlustfrei komprimiert, d. h. immer wiederkehrende gleiche Bitfolgen werden durch kürzere, eindeutige Bitfolgen ersetzt, wodurch (je nach Grafik) eine hohe Kompression erreicht wird. Verlustfrei bedeutet, dass das Originalbild aus den komprimierten Daten exakt wiederhergestellt werden kann. Hinsichtlich heute üblicher Daten (aufwendige Zeichnungen, Photos, etc.) hat das Format einen gravierenden Nachteil: Es unterstützt nur max. 8 bpp (bits per pixel) und somit nur 256 Farben.

Joint Photographic Experts Group (JPEG)
Mit der Einführung des Mosaic-Browsers wurde das GIF-Format allmählich vom 1992 entwickeltem JFIF-Format (JPEG File Interchange Format, meist nur kurz als JPEG bezeichnet) [34] verdrängt. JPEG ist bis heute der de-facto Standard für Photos. Die Komprimierung erfolgt durch mehrere Schritte, wovon manche verlustbehaftet sind. Speichert man ein Bild im JPEG-Format ab, hat man meist die Möglichkeit eine bestimmte Qualitätsstufe auswählen zu

können, um einen Kompromis zwischen Dateigröße und Bildqualität zu finden. Bei Erhaltung einer guten, visuell fast verlustfreien Qualität erreicht man eine Veringerung des Datenvolumens um den Faktor 12-15 im Vergleich zu unkomprimierten Rastergrafiken.

Portable Network Graphics (PNG)
PNG [35] wurde 1995 als Ersatz für das ältere GIF-Format erschaffen und 1997 veröffentlicht. Die PNG-Komprimierung arbeitet verlustfrei und unterstützt Farbtiefen von bis zu 16 bit pro Kanal und einen Alphakanal. Die Komprimierung erfolgt in zwei Stufen. Nach einer Komprimierung im Wörterbuchverfahren (wie bei GIF) folgt eine Entropiekodierung. Es handelt sich dabei um das gleiche Prinzip, das auch bei anderen Datenkomprimierungsmethoden wie Zip, GZip, etc. eingesetzt wird. Dadurch wird ein geringes Datenvolumen ohne Qualitätsverlust erreicht.

Eine von den Rastergrafiken gänzlich verschiedene Möglichkeit Bilder abzuspeichern, ist die Methode, nicht das Bild selbst abzuspeichern, sondern die Informationen, die man benötigt, das Bild neu zeichnen zu können. Auf diese Weise werden Vektorgrafiken gespeichert. Vektorgrafiken enthalten z. B. die Start- und Endkoordinaten von Linien, Mittelpunkte und Radien von Kreisen, usw. Ein Programm zur Darstellung solcher Grafiken liest diese Informationen und stellt das Bild wieder her, in dem es es neu zeichnet.
Vektorgrafiken haben einige Vorteile: Sie lassen sich ohne Qualitätsverlust stufenlos skalieren (bei Rastergrafiken kann der Qualitätsverlust beim Skalieren nur durch Techniken wie z. B. Interpolation im begrenzten Maße ausgeglichen werden). Einzelne Objekte können selektiert und bearbeitet werden. Die Dateigröße ist meist geringer als vergleichbare Rastergrafiken und läßt sich durch jede herkömmliche Komprimierungssoftware (Zip, GZip, usw.) weiter verringern. Es handelt sich oft um Textdateien, d. h. die Datei ist auch per Texteditor les- und änderbar. Der Nachteil dieser Grafikformate ist allerdings, dass sie sich nur für Zeichnungen, bzw. geometrische Grafiken eignen. Nichtsdestotrotz wird auch an Verfahren gearbeitet, beliebige Grafiken (z. B. Photos) in einzelne geometrische Formen zerlegen und sie so ebenfalls vektorbasiert verarbeiten zu können.
Ein weit verbreitetes Vektorformat ist das SVG-Format (Scalable Vector Graphics). Es bietet zusätzlich zu anderen Vektorgrafik-Formaten noch weitere Vorteile:

Scalable Vector Graphics (SVG)
Das SVG-Format [36, 37] basiert auf XML (Abbildung 2.7). Da es in fast allen Programmiersprachen Frameworks zur Handhabung von XML gibt, hat der Entwickler bereits Werkzeuge zur SVG-Grafikerzeugung und -bearbeitung zur Hand. Im Gegensatz zu anderen Vektorgrafikformaten unterstützt SVG auch Skriptingmöglichkeiten. Mittels Javascript und DOM (Document Object Model) kann man Objekte auswählen und manipulieren. Solche Art von SVG-Grafiken werden jedoch zur Zeit noch sehr spärlich von Webbrowsern unterstützt.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="1000" height="600">
<rect id="black_stripe" width="1000" height="200" y="0" x="0" fill="#000000" />
<rect id="red_stripe" width="1000" height="200" y="200" x="0" fill="#DD0000" />
<rect id="gold_stripe" width="1000" height="200" y="400" x="0" fill="#FFCE00" />
</svg>
```

Abbildung 2.7.: Beispiel für eine SVG-Grafik (oben) mit entsprechendem XML-Code (unten) (entnommen aus Wikimedia Commons, `http://de.wikipedia.org/wiki/Bild:Flag_of_Germany.svg`)

Grafikprogrammierung in Java

Zur Erzeugung von Grafiken gibt es in Java die Graphics-API. Sowohl jede GUI-Komponente, als auch off-screen Klassen wie Image besitzen die Methode paint(Graphics g). Diese Methode wird immer dann aufgerufen, wenn die Komponente gezeichnet werden soll. Um eigene Zeichnungen zu implementieren, leitet man gewöhnlich von einer GUI-Klasse ab und überschreibt deren paint(Graphics g) Methode (Abbildung 2.8).
Das Graphics-Objekt stellt dann verschiedene Methoden zum Zeichnen (z.B. drawLine(int x1, int y1, int x2, int y2) zum Zeichnen von Linien) bereit (siehe Dokumentation zur Java-API).

```
public class Zeichnung extends Frame
{
  public Zeichnung() {}

  @Override
  public void paint( Graphics g )
  {
     //String zeichnen:
     g.drawString( "Hello World!", 120, 60 );
     //Linie zeichnen:
     g.drawLine( 10, 10, 100, 50 );
     //Rechteck zeichnen:
     g.drawRect( 10, 10, 100, 100);
     //usw.
  }
}
```

Abbildung 2.8.: Grafikprogrammierung in Java
Realisierung von eigenen Grafiken mittels Überschreiben der paint(Graphics g) Methode.

Diese Zeichnungen lassen sich auf den Bildschirm darstellen, indem die setVisible(boolean b) Methode aufgerufen wird. Sie können aber auch als Bild abgespeichert werden, z. B. mit Hilfe der Klasse ImageIO, die verschiedene Methoden zur Formatierung und zum Einlesen/Abspeichern von Bilddateien anbietet.
Zusätzlich zur Graphics-Klasse gibt es noch die davon abgeleitete Klasse Graphics2D. Diese erweitert Graphics um einige Fähigkeiten. Neben den einfachen Zeichenmethoden wie drawLine(int x1, int y1, int x2, int y2) u. ä. bietet die Graphics2D-API 2D-Objekte und eine entsprechende Zeichenmethode für diese an. Im Package java.awt.geom gibt es verschiedene 2D-Klassen (z. B. Line2D), welche mit der draw(Shape s) Methode der Graphics2D-Klasse gezeichnet werden können. Dadurch ist es möglich, grafische Objekte zu kreieren und diese erst nach Bedarf zu zeichnen. Darüberhinaus bietet Graphics2D noch andere Erweiterungen, wie Transformationen, Füllmuster, usw.

Die Erzeugung von SVG ist aber mit der J2SDK-eigenen Graphics-API nicht möglich. Für diesen Zweck gibt es den Batik-Toolkit von Apache. Dieser erweitert die API dahingehend, dass die Erzeugung und die Verarbeitung von SVG möglich wird. Batik bietet eine SVGGraphics2D-Klasse an, die von Graphics2D abgeleitet, und somit vollständig kompatibel ist. Mit dem gleichen Code, der in einem Graphics2D-Kontext zeichnet, kann man also auch in einem SVGGraphics2D-Kontext zeichnen, und so SVG-Grafiken erstellen, oder ganze GUI-Komponenten als SVG-Datei abspeichern. Der Batik-Toolkit bietet mit der Transcoder-API auch die Möglichkeit, SVG-Grafiken in andere Dateiformate umzuwandeln.

Zur Darstellung von Objekten, kann man aber auch entsprechende Grafik-Frameworks verwenden. Im J2SDK sind bereits die Frameworks AWT, und darauf aufbauend Swing enthalten. Ein Grafikframework stellt grafische Komponenten bereit, die sich z. B. auf dem Bildschirm darstellen lassen. Dazu implementieren die Komponenten die o. g. paint(Graphics g) Methode. Statt diese Methode zu überschreiben, kann man die GUI-Komponenten auch direkt benutzen, um Objekte grafisch darzustellen. Die GUI-Komponenten bieten Methoden an, um die Art ihrer Darstellung beeinflußen zu können, ohne dass man die paint(Graphics g) Methode modifierzen muß (z. B. setBackground(Color bg) um die Hintergrundfarbe zu ändern).

Im Abschnitt 'Taxonomie zur Visualisierung und Benutzerführung' (3, Seite 25) habe ich mich ausführlich mit diesen verschiedenen Herangehensweisen beschäftigt, um die für die Aufgabenstellung geeigneste zu finden.

3. Taxonomie zur Visualisierung und Benutzerführung

Für das SIMAP-Webinterface sollte eine Softwarekomponenten implementiert werden, die einen grafischen, navigierbaren taxonomischen Baum bereitstellt. Dieser Baum soll einerseits dazu dienen, die taxonomische Verteilung von SIMAP-Hits darzustellen und andererseits eine Navigationsmöglichkeit für den Benutzer bieten, dem es damit möglich sein soll, aus der Homologen-Liste, die u. U. mehrere hundert Einträge enthalten kann, nur die taxonomisch gewünschten auszuwählen.
Vor der eigentlichen Implementierung mußten zunächst die Zielsetzungen herausgearbeitet werden und die technische Umgebung untersucht werden.

3.1. Analyse

3.1.1. Zielsetzungen

Eigenschaften des taxonomischen Baumes:

- Grafisch
- Navigierbar/Dynamisch
- Darstellung der Hitverteilung (Zahlen und/oder graphisch)
- Filterung der Hitliste, hinsichtlich des ausgewählten Knotens

Weitere wichtige Punkte, die es zu beachten gilt, sind:

- Performance
- Wiederverwendbarkeit
- Aufwand

Da SIMAP hauptsächlich über das Webinterface benutzt wird, muß auf die **Performance**, insbesondere hinsichtlich Antwortzeiten ein besonderes Augenmerk gerichtet werden. Alle SIMAP-Applikationen sollten eine Antwortzeit im Milisekundenbereich, maximal jedoch im Bereich weniger Sekunden haben, da lange Wartezeiten im Webbrowser besonders störend wirken, und den Benutzer verunsichern.
Die **Wiederverwendbarkeit** von Softwarekomponenten ist generell ein Punkt auf den geachtet werden sollte. In diesem Falle heißt Wiederverwendbarkeit, dass die Komponente für die Darstellung des taxonomischen Baumes auch für andere Komponenten und in anderen Umgebungen verwendbar sein soll. Man kann sogar noch weitergehen, und das Problem 'Taxonomie' vom Problem 'Baum' trennen, um eine wiederverwendbare Komponente für das 'Tree-Drawing'-Problem zu entwickeln, die neben der Taxonomie auch in anderen Kontexten, in denen es darum geht, einen Baum grafisch darzustellen, verwendet werden kann.

Der Begriff '**Aufwand**' ist eng verknüpft mit der Wiederverwendbarkeit. Wiederverwendbare Softwarekomponenten sorgen dafür, dass der Aufwand für zukünftige Softwareprojekte verkleinert wird. Aus diesem Grund muß bei diesem Softwareprojekt auch der Frage Rechnung getragen werden, welche Komponenten es bereits gibt, die für diese Aufgabenstellung verwendet werden können. Wiederverwendbare Software ist aber keine hinreichende Bedingung für ein Minimum an Aufwand. Häufig gibt es mehrere Komponenten (Bibliotheken, Frameworks, usw.), die in Betracht kommen. Neben ihrer Leistungsfähigkeit unterscheiden sie sich gewöhnlich auch im Aufwand, der nötig ist, sie zu integrieren.
Zusammengefaßt heißt das, die Softwarekomponente(n) müssen bei möglichst geringem Implementierungsaufwand wiederverwendbar sein und eine gute Performance (= gute Antwortzeiten) aufweisen.

3.1.2. Grafikformate

Da es um die grafischen Darstellung eines taxonomischen Baumes geht, ist die Wahl des Grafikformates ein wichtiger Aspekt. Für den Einsatzzweck kommen hauptsächlich drei Formate in Frage: Die beiden gängigen Formate im Internetbereich JPEG und PNG, und das vektorbasierte Format SVG, das einige interessante Vorteile gegenüber Rastergrafikformaten bietet. Alle drei Formate habe ich hinsichtlich o. g. Kriterien genauer untersucht.

SVG

SVG ist v. a. deswegen interessant, weil es XML-basiert ist. Der Datenaustausch zwischen den SIMAP-Komponenten erfolgt ausschließlich im XML-Format. Somit würde sich SVG gut in die SIMAP-Umgebung einfügen. Außerdem unterstützt es eingebettetes Skripting (z. B. Javascript), weshalb man theoretisch die Dynamik des Baumes direkt in die SVG-Grafik implementieren könnte. Das gravierende Problem bei Verwendung von SVG ist allerdings, dass die aktuellen, gängigen Webbrowser SVG entweder nicht oder nur teilweise unterstützen. Für manche Browser gibt es Plugins zur SVG-Unterstützung, aber auch dadurch wird SVG kaum vollständig und v. a. auch bei allen Browsern einheitlich unterstützt. Das Problem ist ähnlich dem Problem der Javascript-Unterstützung. Nicht alle Browser unterstützen es vollständig, die Implementierungen weichen von Browser zu Browser ab und die Situation kann sich von Version zu Version ändern. Als Format für den Client scheidet SVG damit leider aus. Intern könnte es dennoch verwendet, und erst direkt vor der Übertragung zum Client mit Hilfe der Batik Transcoder-API in ein anderes Format wie JPEG oder PNG konvertiert werden.

PNG contra JPEG

Beide Formate sind weit verbreitet und werden von allen gängigen Webbrowsern unterstützt. Sowohl mit dem Java-SDK, als auch mit der Batik Bibliothek lassen sich JPEG- und PNG-Grafiken erzeugen. PNG hat gegenüber JPEG den Vorteil, dass es verlustfrei komprimiert. Der Implementierungsaufwand ist bei beiden Formaten gleich. Daher ist entscheidende Faktor zur Wahl zwischen den beiden Formaten die Performance. Deshalb habe ich untersucht, wie lange es dauert, eine Grafik in eines der Formate zu kodieren, abhängig von der Bildgröße und von der Anzahl der Objekte (Linien, Rechtecke, Kreise, usw.), die auf dem Bild dargestellt werden. Erzeugt wurde mittels Java Graphics-API (siehe Kapitel 2.4.2, Seite 23) jeweils ein zufälliges Bild mit gegebener Größe und Anzahl von Objekten. Gemessen wurde nur die tatsächliche Kodierungszeit.

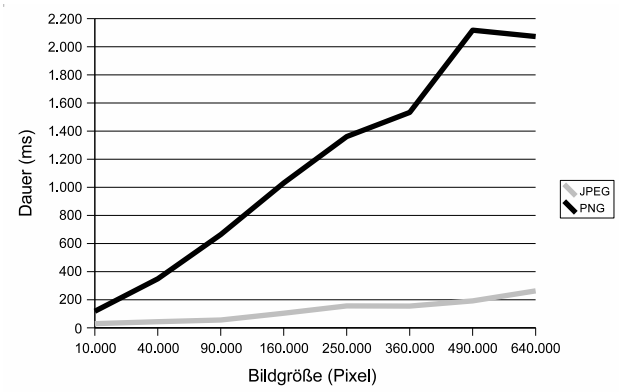

Abbildung 3.1.: Vergleich Kodierungsdauer JPEG/PNG abhängig von der Bildgröße (Objektanzahl: 100)

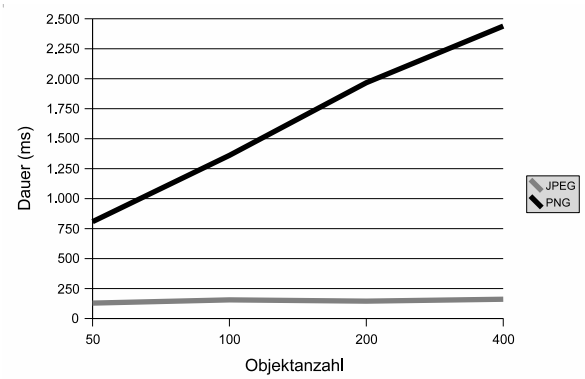

Abbildung 3.2.: Vergleich Kodierungsdauer JPEG/PNG abhängig von Anzahl dargestellter Objekte (Bildgröße: 250.000 Pixel)

Dabei zeigte sich, wie erwartet, dass die Dauer sowohl mit der Größe des Bildes, als auch mit der Objektanzahl zunimmt. Diese Abhängigkeit ist aber beim PNG-Format deutlich ausgeprägter als beim JPEG-Format. Insgesamt zeigte sich ein deutlicher Unterschied bei der Kodierungsdauer zwischen PNG und JPEG. Die JPEG-Kodierung ist in allen Situation schneller. Die Differenz zur PNG-Kodierung nimmt dabei mit steigender Bildgröße und Objektanzahl sogar noch zu (Abbildung 3.1 und 3.2; Daten siehe Appendix A.2, Seite 63).

Im Gegensatz zu dem Geschwindigkeitsunterschied ist der Qualitätsunterschied zwischen PNG und JPEG vernachlässigbar klein. Bei Verwendung der ImageIO-Klasse mit den Standardeinstellungen zur JPEG-Kodierung ist visuell kaum ein Unterschied zwischen JPEG- und PNG-Bild erkennbar. Die typischen JPEG-Artefakte, die durch die verlustbehaftete Kompression entstehen, werden erst beim Vergrößern von Bildausschnitten deutlich sichtbar. Da dies aber nicht Teil der geforderten Funktionialität ist, schlage ich das JPEG-Format zur Verwendung vor. Die Implementierung kann man dennoch flexibel gestalten, so dass man alternativ auch eine PNG-Kodierung verwenden kann.

3.1.3. Grafikframeworks

Bei der Wahl des Grafikframeworks gibt es verschiedene Möglichkeiten:

- SVG
  - Graphics/Graphics2D-API mit Batik (SVGGraphics2D) + Batik Transcoder API
  - Eigene SVG-Implementierung + Batik Transcoder-API
- PNG/JPEG
  - Graphics/Graphics2D-API des J2SDK
  - Verwendung des Swing/AWT-Frameworks

Wie bereits erläutert würde sich SVG gut in die SIMAP-Architektur einfügen. Eine Verwendung von SVG wäre möglich, wenn auch nicht auf Clientseite. Das heißt, die SVG-Grafik müßte vor Auslieferung an den Client in PNG oder JPEG kodiert werden. Die Beantwortung einer Clientanfrage umfaßt auf diese Weise vier Schritte: Layout des Baumes, Ausführen der paint(Graphics g) Methode, Übertragung ins SVG-Format und die Konvertierung der SVG-Grafik ins PNG- oder JPEG-Format. Da die SVG-Grafik nur einen Zweck erfüllt, nämlich in eine Rastergrafik umgewandelt zu werden, kann man sich den Schritt zur Erzeugung der SVG-Grafik sparen, wodurch zusätzlicher Overhead vermieden wird und eine schnellere Antwortzeit möglich ist.

Um möglichst direkt eine Rastergrafik zu erzeugen, kann man Java's Graphics-API benutzen (siehe Kapitel 2.4.2, Seite 23) und nach Ausführung der paint(Graphics g) Methode mit Hilfe der ImageIO-Klasse eine PNG- oder JPEG-Grafik erstellen. Der Implementierungsaufwand ist gleich groß wie bei Verwendung von SVGGraphics2D, aber man spart den Zwischenschritt zur SVG-Erzeugung. Für die Darstellung muß man durch Überschreiben der paint(Graphics g) Methode selbst sorgen. Dies führt allerdings zu einer Einschränkung der Wiederverwendbarkeit. Will man die Softwarekomponente neben der Erzeugung von taxonomischen Bäumen z. B. auch für die Darstellung von Dateisystem-Hierarchien verwenden o. ä. benutzen, ist es sehr wahrscheinlich, dass eine andere Darstellung gewünscht ist. Da diese aber in der überschriebenen paint(Graphics g) Methode implementiert ist, muß der Entwickler in den Code der Softwarekomponente eingreifen, um sie den neuen Anforderungen anzupassen.

Vermeiden läßt sich dies, indem man die Darstellung von der Layoutberechnung trennt. Für die Darstellung der einzelnen Komponenten des Baumes kann man wiederum auf Klassen zurückgreifen, die das J2SDK bereits anbietet. Zur Entwicklung von grafischen Benutzerschnittstellen (Graphical User Interface (GUI)) stellt Java das Swing/AWT-Framework bereit. Dieses Framework beinhaltet mehrere Komponenten, die sich auf dem Bildschirm darstellen lassen, z. B. JFrame, JPanel, JLabel, usw. Sollen die Komponenten dargestellt werden, wird deren paint(Graphics g) Methode aufgerufen.
Um die Darstellung individuell anpassen zu können, bieten die Komponenten Schnittstellen an, mit denen der Entwickler Farben, Schriftformen, Umrandungen, usw. festlegen kann, ohne dass er in den Code eingreifen müßte. Diese Komponenten kann man nun auch nutzen, um einen Knoten im Baum darzustellen.
Für das Layout ist im Swing/AWT-Framework ein sog. LayoutManager verantwortlich. Werden z. B. einem JPanel verschiedene JLabels hinzugefügt, werden diese beim entsprechenden LayoutManager des JPanels registriert. Erhält das JPanel dann die Aufforderung zur Darstellung,

wird zunächst der LayoutManager beauftragt, die Komponenten (in diesem Fall die JLabels) anzuordnen. Erst dann wird die paint(Graphics g) Methode ausgeführt, welche die Komponenten auf den Bildschirm bringt. Bei Verwendung des Swing/AWT-Frameworks zur Baumdarstellung, muß also noch ein LayoutManager implementiert werden, der für die baumartige Anordnung der Komponenten sorgt. Der Vorteil dieses Verfahrens ist die hohe Wiederverwendbarkeit. Will man diese Softwarekomponente in einem anderen Kontext verwenden, stehen einem dafür alle grafischen Komponenten und Methoden des Swing/AWT-Frameworks zur Verfügung. Die Wiederverwendbarkeit stößt aber auch bei diesem Ansatz an ihre Grenzen, wenn man die Darstellung der Verbindungslinien zwischen den Knoten ändern will. Dafür sind weder die Knotenkomponenten, noch der Layoutmanager verantwortlich. Das heißt, hier muß trotzdem in die paint(Graphics g) Methode der Komponente, die den Baum darstellt, eingriffen werden. Dieser Fall sollte aber im Vergleich zu den Fällen, bei denen man nur die Knotendarstellung individuell anpassen möchte, seltener sein.

Wegen der Wiederverwendbarkeit habe ich mich für letztere Variante entschieden. Einerseits ist die Komponente auf diese Weise auch in anderem Kontext verwendbar und andererseits, kann ich Komponenten verwenden, die bereits von J2SDK bereitgestellt werden.

3.1.4. Software-Architektur

Die SIMAP-Architektur basiert auf Enterprise Java Beans (siehe Kapitel 2.2, Seite 12). Um eine gute Integration erreichen zu können, wird die Softwarekomponente zur Darstellung/Navigation von taxonomischen Bäumen ebenfalls in Form eines EJBs implementiert.
Der bisherige 'Ansprechpartner' des Clients [1] ist das RetrievalBean. Dieses Bean erhält die Anfrage (z. B. alle Hits unterhalb des Taxonomieknotes Bacteria) über eine stateless Verbindung, ermittelt mit Hilfe des TaxonomyBeans die gewünschten Taxa, holt sich die entsprechenden Hits aus der Datenbank und gibt sie an den Client zurück (Abbildung 3.3).

Die zu implementiernde Taxonomiebaum-Komponente muß dieser Kommunikation zwischengeschaltet werden. Eine solche Architektur nennt man auch 'Business Delegate' [38]. Ein Business Delegate nimmt Clientanfragen entgegen, koordiniert Bean Interaktionen und leitet Ergebnisse an dem Client weiter. Es verbirgt damit komplexere, interne EJB-Architekturen vor dem Client. In diesem Fall nimmt das TaxonomyTreeBean die Clientanfrage entgegen, holt sich vom RetrievalBean alle Hits, ohne taxonomische Einschränkungen und gibt dem Client, je nachdem welchen Knoten der Benutzer ausgewählt hat, eine gefilterte Liste dieser Hits zurück.
Darüberhinaus ist diese Komponente auch dafür verantwortlich, stets den visuellen Baum zu aktualisieren (Auf- und Zuklappen von Knoten) und an den Client weiterzugeben. Gemäß dem Design Pattern 'Business Delegate' müsste man diese Funktionalität auf ein zusätzliches Bean auslagern. Ein Business Delegate sollte keine eigene zusätzliche Geschäftslogik implementieren. Da diese Logik hier aber nur in einem Zusammenhang (Selektion von Hits) benutzt wird, ist diese Abweichung vom Design Pattern 'Business Delegate' meiner Meinung nach zulässig.

[1] Client ist im folgenden nicht im Sinne von Benutzer gemeint, sondern als Client der EJBs. Das kann eine Java Server Page (z. B. in Form eines CMS (Content Management System)), ein Servlet, usw. sein

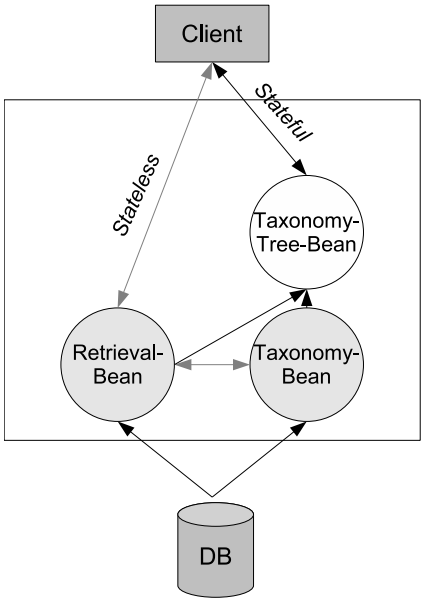

Abbildung 3.3.: Integration der Taxonomiebaum-Komponente in die Client-EJB-Kommunikation
Die bisherige Kommunikation (graue Pfeile) wird ersetzt (schwarze Pfeile), insbesondere die stateless Kommunikation zwischen Client und Retrieval-Bean durch eine stateful Kommunikation mit dem Taxonomy-Tree-Bean.

Im Gegensatz zur bisherigen Architektur muß die Client-EJB-Verbindung nun aber stateful sein. Eine stateless Implementierung wäre möglich, hat aber Nachteile:

- Der Client muß stets den kompletten Zustand des Baumes übertragen (Statt nur jeweils den Knoten, der von einer Aktion betroffen ist).
- Das RetrievalBean - und damit die Datenbank - muß bei jeder Clientanfrage angesprochen werden (Statt einmalig pro Client und anschließender Filterung der Hitliste im TaxonomyTreeBean).

3.2. Zeichenalgorithmus

Zur Lösung des 'Tree drawing'-Problems gibt es verschiedene Ansätze. Ich habe dabei einen Ansatz verfolgt, den Eric Rosé in einer Ausgabe des MacTech Magazins [2] vorgestellt hat ('ER-Algorithmus'). Der ER-Algorithmus ist relativ einfach und schnell, bietet aber nicht die optimalste Platzausnutzung, d. h. der Platz, den der Baum einnimmt, ließe sich in vielen Fällen deutlich reduzieren. Deshalb stellt Eric Rose noch einen zweiten Algorithmus ('SG-Algorithmus') vor, der auf dem gleichen Prinzip beruht wie der ER-Algorithmus, aber komplexer und langsamer ist, da er mehrere Baumdurchläufe benötigt als ER.
Wegen des Geschwindigkeitsunterschiedes und der einfacheren Handhabbarkeit habe ich mich für die Implementierung des ER-Algorithmus entschieden, zumal dieser hinsichtlich des Anwendungsbereichs 'taxonomische Darstellung von Simap-Ergebnissen' völlig ausreichend sein dürfte.

Funktionsweise

Der Kern des Algorithmus ist die Einsicht, dass jeder (Sub-)Baum durch vier, sich überlappende Rechtecke beschrieben werden kann. Diese Rechtecke beschreiben den Platzbedarf der einzelnen Elemente des Baumes (siehe Abbildung 3.4).

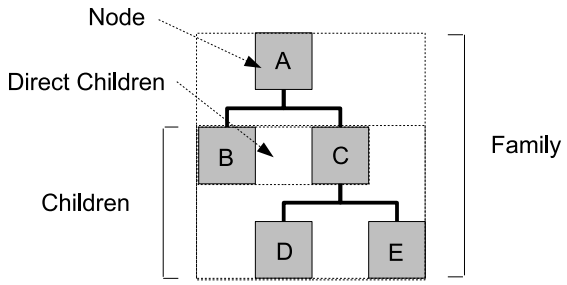

Abbildung 3.4.: ER-Algorithmus: Beschreibung eines (Sub-)Baumes mittels Rechtecken. Der Platzbedarf eines (Sub-)Baumes (Family) setzt sich zusammen aus dem Wurzelknoten (Node), den Kindern (Children) und den entsprechenden Abständen zwischen den Knoten.

'Node' ist dabei das Rechteck, welches den Knoten direkt umgibt. 'Direct Children' umfasst die direkten Kindknoten des Knotens. Innerhalb des 'Children'-Rechteckes befinden sich alle direkten als auch indirekten Kindknoten und ergibt sich aus der Summe der Family-Rechtecke der Kinder. Das 'Family'-Rechteck umfasst dann schließlich den gesamten (Sub-)Baum.
Diese Rechtecke können alle ohne eine Positionsangabe berechnet werden. Dies macht sich auch der Algorithmus zu nutze. Er umfaßt zwei Baumdurchläufe. Zuerst werden beginnend von den Blättern bis zum Wurzelknoten die Größen aller einzelnen 'Family'-Rechtecke (= Subbäume) berechnet, dann werden diese Rechtecke beginnend beim Wurzelknoten bis zu den Blättern der Reihe nach positioniert. Bei der Positionierung der Subbäume kann es ja nach gewünschter Darstellung noch notwendig sein, die Position des jeweiligen Wurzelknotens des Subbaumes zu korrigieren (z. B. über Kindknoten zentrieren, o. ä.).

Der Algorithmus umfaßt zwei Funktionen (siehe Abbildung 3.5):

- position(Point topLeft): Positioniert die Rechtecke
- calculateFamilyRectangle(): Berechnet die Größen der Rechtecke

Die Methode position(Point topLeft) auf den Wurzelknoten mit dessen Anfangskoordination (z.B. (0,0) für die äußerste Ecke oben, links) angewendet, berechnet das gesamte Layout des Baumes. Will man verschiedene Layoutvarianten implementieren, muß man in den Methoden jeweils die Berechnung der Rechtecksgrößen, bzw. die Positionsberechnung anpassen.

```
function position(Point topLeft) {
  this node's topLeft = topLeft
  // falls gewuenscht, zentriere Knoten ueber Kindknoten, o. ae.
  center(this node's topLeft)

  // Positioniere Subbaeume:
  foreach{Children child of node}
    Children's Rectangle childRect = child.calculateFamilyRectangle()
    topLeft += margin
    child.position(topLeft)
    // verschiebe Positionierungspunkt fuer naechsten Subbaum:
    topLeft += childRect
  }
}

function calculateFamilyRectangle() {
  // Family-Rechteck entspricht Node-Rechteck, falls keine Kinderknoten vorhanden
  familyRect = nodeRect
  // ...sonst beruecksichtige auch diese:
  foreach{Children child of this node} {
    familyRect += child.calculateFamilyRectangle()
  }
  return familyRect
}
```

Abbildung 3.5.: ER-Algorithmus (Pseudocode)

Die zwei wesentlichen Methoden des ER-Algorithmus: position(Point topLeft) zur Positionierung der Knoten, calculateFamilyRectangle() zur Berechnung des Platzbedarfs der Knoten.

3.3. Implementierung

3.3.1. 'Tree-Drawing' Komponente

Wegen unter 3.1.3 genannten Gründen entschied ich mich, für das Tree-Drawing Problem eine eigene gekapselte Komponente unter Verwendung des AWT/Swing-Frameworks zu implementieren (Abbildung 3.6).

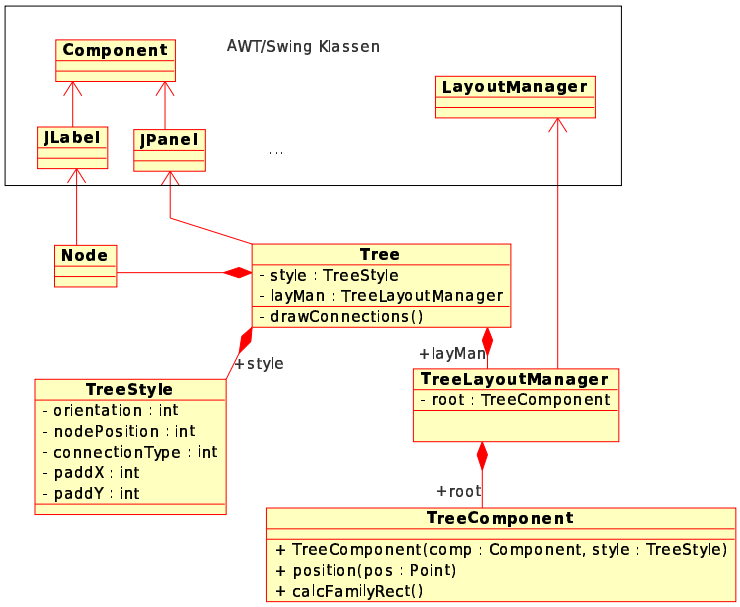

Abbildung 3.6.: 'Tree-Drawing' mit AWT/Swing unter Verwendung eines Layoutmanagers
Tree dient als Panel zur Darstellung des Baumes; der TreeLayoutManager übernimmt die Positionierung der einzelnen Knoten.

Die Klasse Tree stellt ein Panel bereit, auf welchem der Baum gezeichnet werden kann. TreeLayout definiert das Layout des Baumes (horizontal/vertikal, gebündelte/direkte Knotenverbindung, Abstände, usw.). Node kann von jeder beliebigen Component-Klasse abgeleitet werden, und übernimmt die grafische Darstellung eines Knotens. Dem Baum (Tree) können über die, in der Elternklasse Container implementierte Methode add(Component comp, Object constraints) Knoten (Node) hinzugefügt werden. Der TreeLayoutManager verwaltet diese Knoten. Bekommt der TreeLayoutManager die Anweisung, die Komponenten auszulegen, werden die Knoten in TreeComponents gepackt, die den eigentlichen rekursiven ER-Algorithmus implementieren.
Die Klassen können als jar-Datei gepackt und als Bibliothek in anderen Projekten verwendet werden. Damit lassen sich sowohl horizontale als auch vertikale Bäume erstellen (Abbildung 3.7).

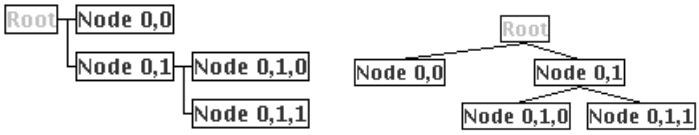

Abbildung 3.7.: Verschiedene Baumtypen mit Hilfe des TreeLayoutManagers gezeichnet;
Es werden sowohl verschiedene Baumtypen (horizontal (links), vertikal (rechts)), als auch verschiedene Verbindungstypen (gebündelt (links), direkt (rechts)) unterstützt.
(Der Quellcode zum rechten Beispiel ist im Appendix (B.1, Seite 66) zu finden)

3.3.2. Taxonomiebaum

Um die Tree-Drawing-Komponente an den speziellen Fall 'Taxonomiebaum für SIMAP' anzupassen, bedurfte es zusätzlicher Klassen (Abbildung 3.8).

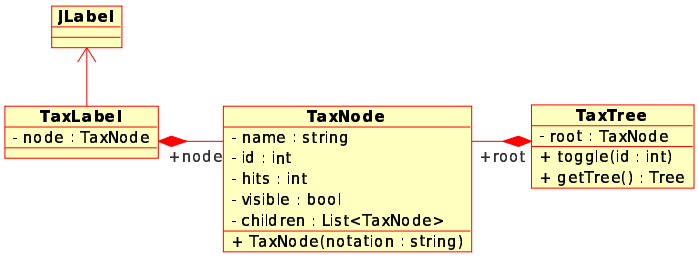

Abbildung 3.8.: Taxonomiebaum unter Verwendung des TreeLayoutManagers
Der Taxonomiebaum benötigt eine Komponente zur Darstellung der Knoten (TaxLabel), zur Verwaltung der Knoteninformation (TaxNode) und zur Verwaltung des Baumes (TaxTree).

TaxLabel ist verantwortlich für die Darstellung eines Knotens und wird von der Swing-Klasse JLabel abgeleitet. TaxNode enthält die Informationen eines Knotens (Name, Taxonomie-ID, Anzahl Hits, usw.), und implementiert Methoden, um eine Baumstruktur anhand einer Definition im Newick-Format[2] aufzubauen. TaxTree schließlich ist für die Handhabung des Baumes zuständig. Über die Methode toggle(int id) können einzelne Knoten des Baumes auf- und zugeklappt werden und der entsprechende Baum dann mittels getTree() Methode erzeugt werden.

3.3.3. Enterprise Java Bean

Bei dem Bean handelt es sich um ein stateful Bean mit Remote-Interface (Abbildung 3.9). Ein Remote-Interface wurde erstellt, weil man nicht davon ausgehen kann, dass das Bean im selben Container wie der Client (JSP, Servlet, usw.) läuft.

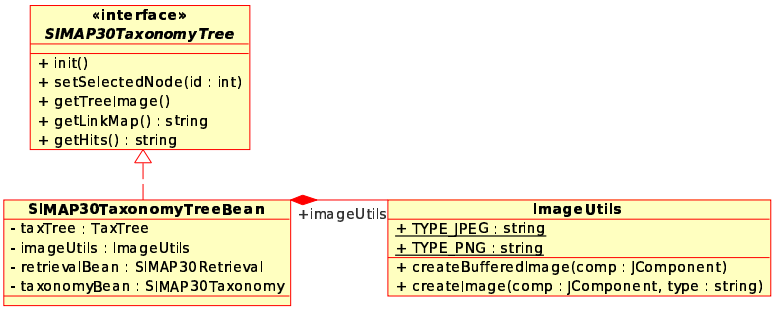

Abbildung 3.9.: SIMAPTaxonomyTreeBean
Das SIMAP30TaxonomyTreeBean stellt verschiedene Methoden zur Verwaltung des Taxonomiebaumes bereit. Ein Client kann ber das entsprechende Interface (SIMAP30TaxonomyTree) mit dem Bean kommunizieren.

[2]Knoten werden durch Komma getrennt, Knoten mit demselben Elternknoten durch Komma zusammengefaßt. Durch Dopplelpunkt getrennt, kann hinter den Knoten die Zweiglänge angegeben werden. Beispiel: ((A:1,B:1):2,C:2):1,(D:1,E:1):4)

Das Remote-Interface definiert fünf Methoden:

- init(): Initialisiert das Bean mit Suchparametern, die vom Client übergeben werden.
- setSelectedNode(int id): Wählt einen bestimmten Knoten im Baum aus.
- getTreeImage(): Erzeugt den der Knotenauswahl entsprechenden Baum und gibt ihm in Form eines JPEG- oder PNG-kodierten Byte-Arrays an den Client zurück.
- getLinkMap(): Liefert die dem Baum entsprechende HTML Link-Map[3]
- getHits(): Liefert die entsprechend der Knotenauswahl gefilterte Hit-Liste.

Dem Client, der die taxonomische Visualisierung und Navigation in das SIMAP Webinterface integriert (CMS, eigene JSP-Seite, Servlet, usw.) stehen diese fünf Methoden zur Verfügung. Er teilt dem Bean jeweils mit, welchen Knoten der Benutzer selektiert hat. Das Bean expandiert oder kollabiert den Knoten abhängig vom aktuellen Zustand des Knotens, und filtert die Hit-Liste entsprechend der Auswahl. Anschließend kann sich der Client den modifizerten Baum, die dazugehörende HTML Link-Map und Hit-Liste holen, um damit die Webseite zu aktualisieren.

3.4. Ergebnis

Die Komponenten wurden wie oben beschrieben implementiert und in einem für Testzwecke eingerichtetem lokalen JBoss-Server getestet. Die Implementierung funktionierte wie erwünscht, einzig die Filterung der Hits bereitete mit der Test-Hitliste Schwierigkeiten. Die entsprechende Methode muß vor der Integration ins Produktivsystem nochmal überarbeitet werden.

3.5. Ausblick

Nach der Überarbeitung der Methode zur Filterung der Hitliste kann das TaxonomyTree-Bean ins SIMAP-Produktivsystem integriert werden. Zur Integration ins SIMAP-Webinterface wird es nötig sein, einen Client für das Bean zu implementieren, der die Kommunikation zwischen Benutzer und Bean, und die webfähige Darstellung (HTML) ermöglicht. Dies kann z. B. mit Hilfe eines Servlets oder JSP geschehen.
Die Komponente, die für die Erstellung des Baumes zuständig ist (TreeLayoutManager), kann darüberhinaus auch in anderen Projekten zur Darstellung von hierarchischen Strukturen verwendet werden.

[3] Link-Map (oder Image Map): Mit Hilfe einer Link-Map und einem Bild erzeugt der Webbrowser eine verweissensitive ('anklickbare') Grafik

4. Taxonomische Analyse des Anammox-Community Genomes

4.1. Vorbereitung

Das Anammox-Community Genom liegt in Form eines Flatfiles, bestehend aus einzelnen Contigs (Contig-Nr. und Sequenz) vor. Da für die nachfolgenden Analyseschritte eine Datenbank benötigt wird, bestand der erste Schritt darin, ein Datenbankschema zu entwerfen, und dieses in einer MySQL-Datenbank zu realisieren. Das Datenbankschema wurde im Laufe der Analyse stetig erweitert, um weitere externe Daten (z. B. Taxonomie) oder Analyseergebnisse (Homologe in der SIMAP Datenbank) hinzufügen zu können, was letztendlich zu dem in Abbildung 4.1 gezeigtem Schema führte:

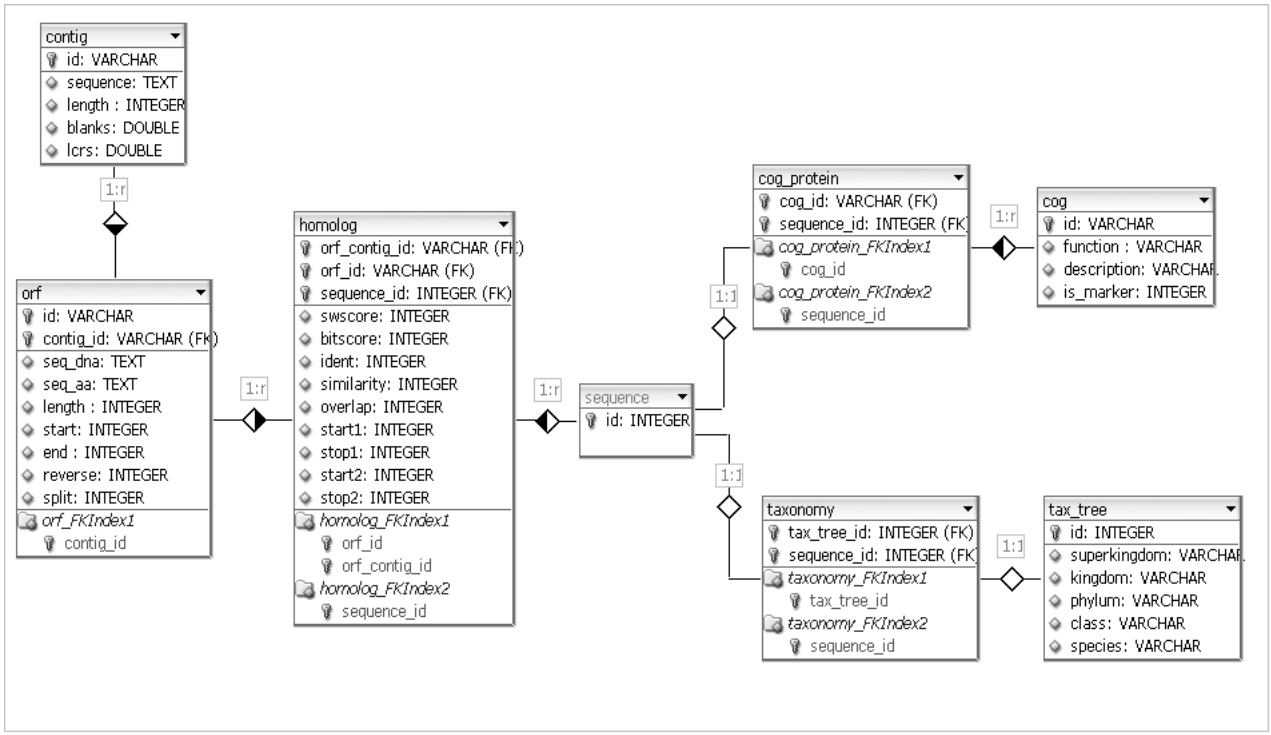

Abbildung 4.1.: Schema der Anammox-Community Datenbank
Die Datenbank enthält die primären Daten des Community Genomes (contig), sekundäre Information, die durch Analysen gewonnen wurde (orf, homolog) und Informationen, die von externen Datenquellen bezogen wurde (sequence, cog_protein, cog, taxonomy, tax_tree).

Nach der Bereitstellung der benötigen Infrastruktur, wird zuerst die in 9277 Contigs unterteilte DNA-Sequenz untersucht:

Contiglänge

Das Genom wurde mit der Shotgun-Methode sequenziert. Wie deshalb zu erwarten war, liegt die Länge der Contigs hauptsächlich im Bereich 700 bis 1.600 Nukleotide. Es gibt auch einige konkatenierte Contigs mit Längen bis zu 150.000 Nukleotide (Abbildung 4.2).

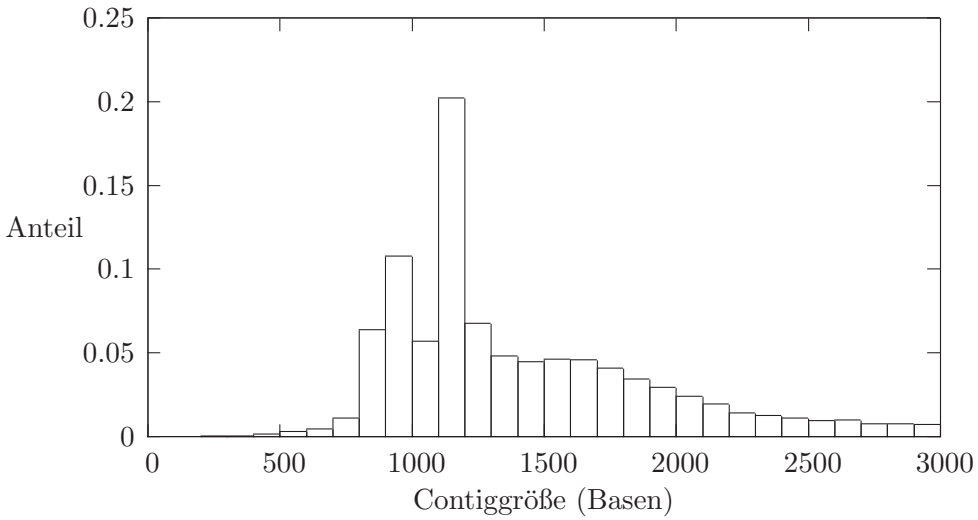

Abbildung 4.2.: Contiglängenverteilung
Die meisten Contigs liegen im Bereich 700 bis 1.600 Nukleotide.

Gap-Anteil

'Gap-Anteil' bezeichnet den Anteil nicht identifizierter (mit 'n' gekennzeichneten) Nukleotide in den Contigs. Der Großteil der Contigs (93,4%) weist einen Gap-Anteil von kleiner 1% auf. Im Bereich 0 bis 5% Gap-Anteil liegen bereits 99% der Contigs. Gap-Anteile größer 10% finden sich nur bei wenigen Ausreißern (Abbildung 4.3).

Anteil 'unsicherer' Nukleotide

Die meisten Contigs bestehen zu 10 bis 40% aus unsicheren Basen (Abbildung 4.4). Dieser Anteil gibt an, wieviele Nukleotide auf einem Contig unterhalb einer gewissen Phred-Schwelle liegen, und damit mit Kleinbuchstaben gekennzeichnet sind. Phred [39, 40] ist eine Software, die verwendet wird, um die Sequenzbruchstücke aus der Shotgun-Sequenzierung durch Overlap-Alignments wieder zusammenzufügen. Bereiche der Sequenz, in denen sich mehrere solcher Bruchstücke überlappen (die also mehrmals sequenziert wurden) gelten als sicher, Bereiche die hingegen nur wenige male oder nur einmal sequenziert wurden (keine oder nur wenig Überlappung) gelten als weniger sicher. Wie oft ein Nukleotid sequenziert werden muß bis es als sicher gilt und mit einem Großbuchstaben gekennzeichnet wird, nennt man Phred-Schwellenwert. Für das Anammox-Community Genom ist dieser Wert aber leider nicht bekannt.

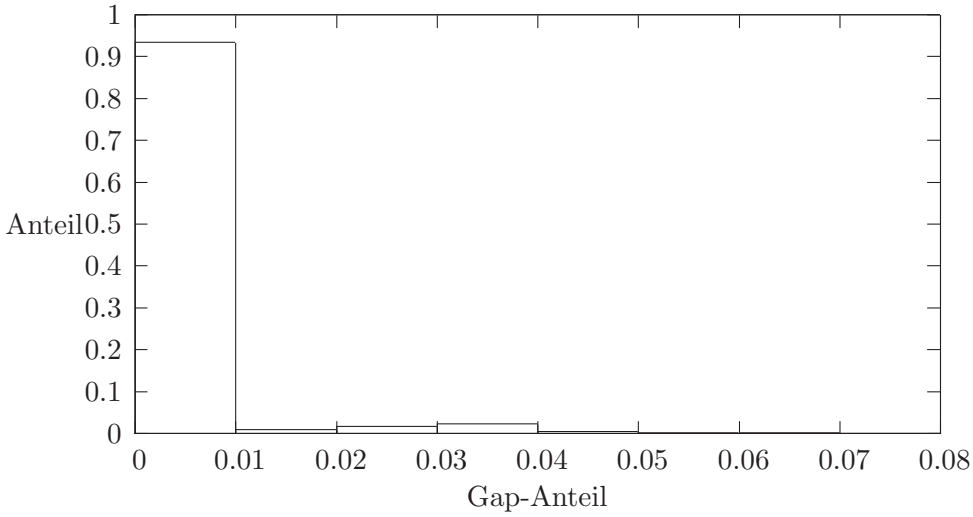

Abbildung 4.3.: Anteil nicht identifizierter Nukleotide
Die meisten Contigs (93,4%) bestehen zu weniger als 1% aus Lücken.

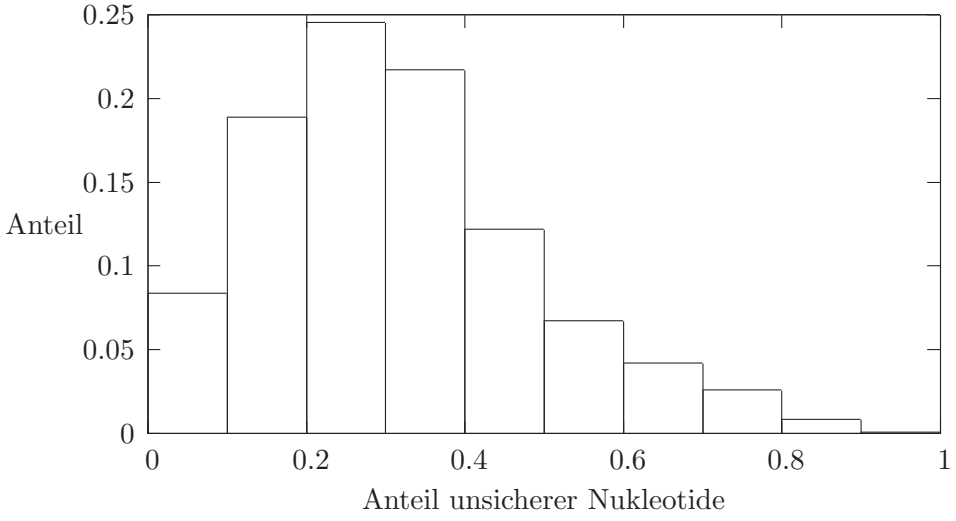

Abbildung 4.4.: Anteil 'unsicherer' Nukleotide
Die meisten Contigs bestehen zu 10 bis 40% aus 'unsicheren' Nukleotiden.

Fazit

Die Daten entsprechen dem, was man bei einem Sequenzierprojekt erwarten würde, das noch nicht vollständig abgeschlossen ist. Das Ziel des Anammox-Community Projektes war nicht die möglichst vollständige Sequenzierung aller Genome, die im Community-Genom enthalten sind, sondern die Sequenzierung des *K. stuttgartiensis* Genomes. Deshalb hat man nachfolgende Schritte, wie das Schließen der Gaps durch gezielte, wiederholte Sequenzierung dieser Abschnitte unterlassen, da das Genom von *K. stuttgartiensis* bereits in ausreichender Güte vorlag.
Für eine taxonomische Analyse des (von *K. stuttgartiensis* bereinigten) Community-Genoms sollte die vorliegende Qualität aber dennoch ausreichen.

4.2. Genvorhersage

Eine notwendige Bedingung dafür, dass eine DNA-Sequenz für ein Protein kodiert ist, dass die DNA-Sequenz einen ORF (Open Reading Frame) bildet. Deshalb ist der erste Schritt der Genvorhersage meist die Suche nach ORFs. Ein ORF ist ein Stück DNA-Sequenz, das sich zwischen einem Start- und einem Stop-Codon befindet. Bakterien unterscheiden sich aber häufig darin, welche Start-Codons sie verwenden. Deshalb verwendet man für die Suche nach ORFs in Bakteriengenomen meist die Definition eines ORFs, als DNA-Sequenz zwischen zwei Stop-Codons liegend. Da davon auszugehen ist, dass sich in dem vorliegenden Genom hauptsächlich Bakterien befinden, habe ich bei der ORF-Suche diese Definition verwendet.
Ein weiterer wichtiger Parameter ist die minimale ORF-Länge. Die Stochastik besagt, je kürzer ein ORF ist, desto wahrscheinlicher ist sein zufälliges Auftreten. Das heißt, wählt man eine zu kurze minimale ORF-Länge, erhält man viele ORFs, die aber zu einem großen Teil rein zufallsbedingt sind und keine biologische Funktion haben. Wählt man aber eine zu große minimale ORF-Länge, könnte man kleine, aber dennoch biologisch relevante ORFs übersehen. Für die ORF-Suche im Anammox-Community Genom habe ich eine minimale ORF-Länge von 100 Nukleotiden gewählt. Bei Genomen die annähernd geschlossen vorliegen, verwendet man normalerweise eine größere minimale ORF-Länge. Da aber in diesem Fall das Genom in Form einzelner Contigs vorliegt, ist damit zu rechnen, das einige ORFs an den Contiggrenzen abgeschnitten sind, was die kleinere minimale ORF-Länge rechtfertigt.

Zur ORF-Suche wird das Programm getorf aus dem EMBOSS-Paket [41] mit folgenden Parametern verwendet:

- minsize 100: Minimale ORF-Länge von 100 Nukleotiden
- reverse: Betrachte Richtung 5'→3' und 3'→5'
- find 2: Suche Stop-Stop
- find 0: Suche Stop-Stop und anschließende Translation in die entsprechende Proteinsequenz

Insgesamt wurden 298.253 ORFs gefunden, wovon fast 60% eine Länge von 100 bis 200 Nukleotiden besitzen. Wie bereits erwähnt, hat diese Häufung im Bereich kleiner ORF-Längen keine biologische Entsprechung. Ein Großteil dieser ORFs ist zufallsbedingt.

Um die biologische Relevanz dieser ORFs zu ermitteln, werden im Anschluß gewöhnlich weitere Methoden angewendet, wie die Untersuchung der ORFs nach charakteristischen Merkmalen (z. B. GC-Gehalt), Mustererkennung, usw. Eine wichtige Methode ist die Suche nach homologen Proteinen. Bei einem ORF, der eine hohe Homologie mit einem bekannten Protein aufweist, handelt es sich sehr wahrscheinlich um einen biologisch relevanten ORF.

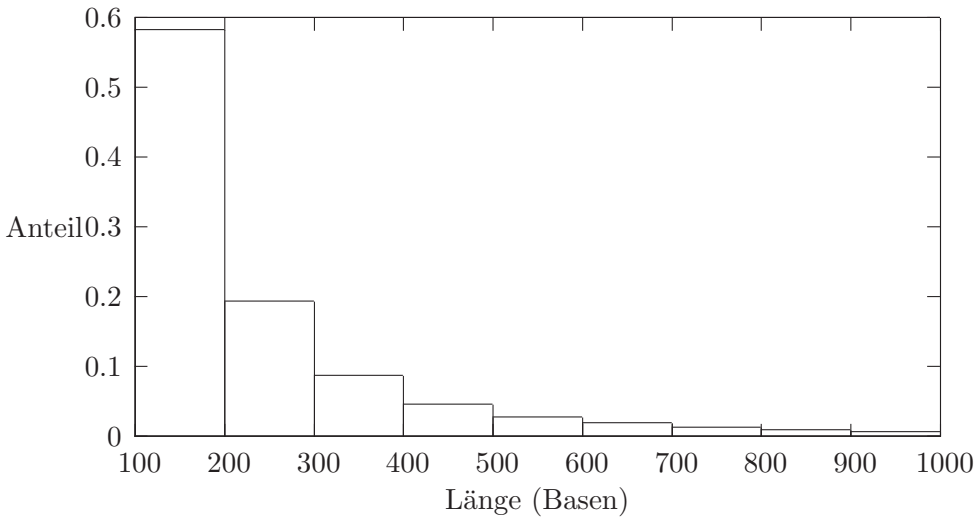

Abbildung 4.5.: Längenverteilung der im Anammox-Community Genom gefundenen ORFs
Deutlicher Schwerpunkt bei kurzen ORFs; die meisten Contigs sind kürzer als 300 Nukleotide.

4.3. Homologe in der SIMAP-Datenbank

Als Quelle für Proteinsequenzen stand die SIMAP-Datenbank zur Verfügung. Mit Hilfe des Rechenclusters wurden alle Homologen der ORFs in der SIMAP-Datenbank ermittelt. Die Berechnung erfolgte mit dem Programm FASTA [5] mit folgenden Parametern:

- -H, -z3, -m10: Dient der Ausgabeformatierung
- -E 10: Homologe dürfen einen max. EValue von 10 haben
- ktup 2: Es wird eine Wortlänge von 2 verwendet

Es wurde ein EValue von 10 gewählt, um auch noch schwache Homologien zu finden, und eine Wortlänge von 2. Die Wortlänge bestimmt die Schärfe der Heuristik, mit der FASTA nach Homologen sucht. Je kleiner der Wert, desto sicherer findet FASTA Homologe, aber desto länger dauert die Berechnung.
Zu 173.356 ORFs (58%) konnten auf diese Weise Homologe in der SIMAP-Datenbank gefunden werden. Dass diese ORFs nicht alle biologische relevant sind, ergibt sich aus dem hohen EValue, der verwendet wurde.

4.4. Taxonomische Analyse

Wie bereits in der Einleitung erwähnt, ist nicht jedes Protein als Marker für die taxonomische Analyse geeignet (siehe Kapitel 2.1.3). Für die weitere Analyse habe ich mich deshalb entschieden, die gleichen Marker-Proteine zu verwenden, die Peer Bork für seinen Baum verwendet hat (siehe Tabelle A.2, Seite 65), bzw. die ORFs zu betrachten, die Homologien zu diesen Proteinen aufweisen. Dabei handelte es sich um 431 ORFs (0,14%), die auf insgesamt 261 Contigs (2,81%) lagen.

4.4.1. 'Beste Homologen'-Methode

Die naheliegendste Methode zur Untersuchung dieser ORFs war die Betrachtung der besten SIMAP-Homologen dieser ORFs. Ich habe dazu jeweils die drei besten Homologen eines ORFs, und den ORF als taxonomisch zuordenbar betrachtet, wenn drei Bedingungen erfüllt wurden:

- die Homologen einen EValue von max. 1×10^{-10} haben;
- die Homologen in der zu untersuchenden taxonomischen Hierarchie übereinstimmen;
- oder bei Nichtübereinstimmung eine EValue-Differenz um mindestens den Faktor 10 vorliegt.

Das heißt, wird ein ORF hinsichtlich der Spezies analysiert, gilt der ORF als zuordenbar, wenn die Spezies der drei besten Homologen übereinstimmt, oder falls das zweit- oder drittbeste homologe Protein davon abweicht, mindestens ein EValue-Differenz um den Faktor 10 zum besten, bzw. zweitbesten homologen Protein aufweist.

Ergebnisse

Mehr als die Hälfte der ORFs (281) hatten homologe Marker-Proteine in der SIMAP-Datenbank mit einem EValue kleiner als 1×10^{-10}. Die meisten ORFs hatten dennoch keine starke Homologie mit den entsprechenden Marker-Proteinen (die Evalues liegen in den wenigsten Fällen unter 1×10^{-100}). Das heißt eine eindeutige Zuordnung der ORFs zu bestimmten Spezies ist nicht möglich, obwohl zu manchen Spezies mehrere homologe ORFs gefunden wurden (Tabelle 4.1), jedoch nicht mit aussagekräftigen Evalues. Aber man kann davon ausgehen, dass die Homologien auf Verwandtschaftsverhältnisse hindeuten. So kann man vermuten, dass die meisten Genome, die im Anammox-Community Genom enthalten sind, von Proteobakterien, Bacili, Clostridien und Thermotogae stammen, bzw. verwandt mit diesen Klassen sind (Tabelle 4.2).

Spezies	**Anzahl ORFs**
Bacillus halodurans	9
Clostridium acetobutylicum	15
Corynebacterium glutamicum	6
Deinococcus radiodurans	7
Mesorhizobium loti	7
Neisseria meningitidis	9
Ralstonia solanacearum	23
Thermotoga maritima	15
Xylella fastidiosa	6

Tabelle 4.1.: Spezies, zu denen mehrere homologe ORFs im Anammox-Community Genom gefunden werden konnten ('Beste Homologen'-Methode)
(Details siehe Appendix C.2, Seite 74)

Klasse	Anzahl ORFs
Alphaproteobacteria	29
Bacilli	43
Betaproteobacteria	37
Clostridia	15
Gammaproteobacteria	25
Thermotogae (class)	15

Tabelle 4.2.: Klassen, zu denen mehrere homologe ORFs im Anammox-Community Genom gefunden werden konnen ('Beste Homologen'-Methode)
(Details siehe Appendix C.2, Seite 74)

Die Probleme der Methode liegen darin, dass die Parameter willkürlich gewählt sind. Man betrachtet nur eine bestimmte Anzahl von Homologen. Nur das beste homologe Protein, nur die drei besten, usw. Bei einer bestimmten Anzahl muß man ein Grenze ziehen. Diese kann man fest vorgeben (z. B. die besten drei), oder variabel gestalten (z. B. ist der EValue des viertbesten Homologen ähnlich dem drittbesten, zieht man es auch noch mit hinzu, usw.). Diesen Cut-Off dynamisch zu ermitteln, wäre tatsächlich eine Verbesserung dieser Methode. Aber eine andere Methode ist naheliegender, eine phylogenetische Methode.

4.4.2. Referenzbaum-Methode

In einem Community Genom wird immer eine Vielzahl an bisher noch nicht sequenzierten Genomen verborgen liegen, d. h. man kann in den meisten Fällen nur Verwandschaftsaussagen machen. Für diese Art von Analyse ist die Phylogenie prädestiniert.
Anstatt phylogenetische Bäume mit dem Community Genom zu rechnen, was sehr aufwendig wäre (siehe Kapitel 2.1.3), wird versucht, die einzelnen ORFs in einem Referenzbaum einzuordnen. Als Referenzbaum wird der Bork-Baum und als Datengrundlage zur Berechnung der ORF-Distanzen werden die Homologiedaten zu den Markerproteinen verwendet.

Berechnung der Distanzmatrix des Referenzbaumes

Der Bork-Baum (bewurzelte Version) liegt mit Angabe der Zweiglängen im Newick-Format vor. Im Newick-Format sind die einzelnen Knoten durch Komma getrennt, zusammengehörende Knoten (Knoten mit demselben Elternknoten) werden durch Klammern zusammengefaßt. Die Zweiglänge (Distanz Knoten zu Elternknoten) kann durch Doppelpunkt getrennt, hinter dem Knotennamen angegeben werden.
Beispiel: ((A:1,B:1):2,C:2):1,(D:1,E:1):4); (Abbildung 4.6 und 4.8)

Die Distanzmatrix besteht aus Distanzvektoren. Jeder Knoten im Bork-Baum besitzt einen Distanzvektor, der die Distanzen zu den Blattknoten des Baumes angibt, und aus der Angabe der Zweiglängen berechnet werden kann (Abbildung 4.6). Es handelt sich um eine nicht-quadratische $m \times n$ Matrix mit m =Anzahl Knoten und n =Anzahl Blattknoten.

Verhältnis von Blattknoten zu internen Knoten im binären, gewurzelten Baum:

$$Ni = Nb - 1 \tag{4.1}$$

Ni : Anzahl interne Knoten; Nb : Anzahl Blattknoten

Der Bork-Baum umfaßt 191 Spezies ($n = 191$) und ist ein binärer Baum. Damit folgt aus Gleichung 4.1, daß die Distanzmatrix aus $m = 191 + (191 - 1) = 381$ Distanzvektoren besteht.

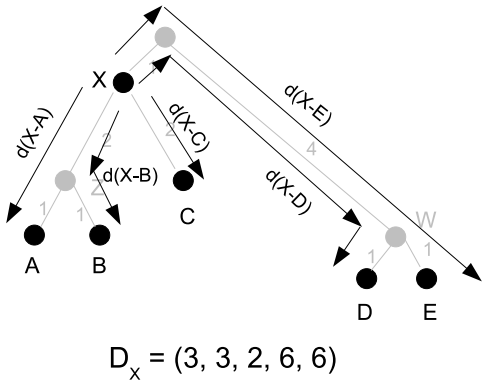

Abbildung 4.6.: Beispiel: Berechnung eines Distanzvektors
Die Distanzen ergeben sich aus der Summe der einzelnen Zweiglängen.

Verknüpfen des Referenzbaumes mit der NCBI Taxonomie

Zusätzlich zur Berechnung der Distanzvektoren, muss jedem Knoten die entsprechende NCBI Taxonomie zugewiesen werden. Für diese Aufgabe wurde ein Python-Skript erstellt. Sowohl der Bork-Baum als auch der NCBI-Baum werden eingelesen und im Speicher gehalten. Der Bork-Baum wird Knoten für Knoten durchlaufen und die Distanz zu allen Blattknoten ermittelt. Daraus wird der Distanzvektor gebildet. Diesem Distanzvektor werden dann die entsprechenden NCBI Taxononmie-IDs zugewiesen.
Die Blattknoten des Bork-Baums repräsentieren Spezies, weshalb ihnen ziemlich einfach NCBI Taxonomie-IDs zugewiesen werden können. Ein solcher Knoten kann mehrere IDs haben, wenn der NCBI-Baum z. B. zwischen mehreren Stämmen dieser Spezies unterscheidet. Problematisch sind die inneren Knoten des Bork-Baumes. Da der Bork-Baum nicht dem NCBI-Baum entspricht, gibt es zu solchen Knoten in vielen Fällen kein Analogon im NCBI-Baum. In diesen Fällen wird ermittelt, welche Blattknoten der jeweilige Bork-Knoten verbindet. Anschließend wird der NCBI-Baum beginnend vom ersten ermittelten Knoten nach oben durchlaufen, bis ein Knoten gefunden wurde, der auch im NCBI-Baum alle entsprechenden Blattknoten miteinander verbindet. Die Taxonomie-ID dieses Knoten wird nun dem internen Knoten des Bork-Baumes zugewiesen.
Bei Ausführung des Skriptes wird gleichzeitig die durchschnittliche Distanz jedes internen Knotens zu seinen Blattknoten ermittelt. Dieser Wert wird später bei der Adaption der ORF-Distanzvektoren an die Baum-Distanzvektoren benötigt. Nach Ausführung des Skriptes liegen dann drei Dateien vor: Die Distanzmatrix des Baumes, die Bezeichnung der Knoten in Form der NCBI-Taxonomie, und die durchschnittliche Blattdistanz jedes internen Knotens.

Berechnung ORF-Distanzen

Durch die vorherige Arbeit liegen zu den 431 interessanten ORFs (zu Marker-Proteinen homologe ORFs) bereits die Homologen zu allen in SIMAP enthaltenen Proteinen, und somit auch zu den Proteinen der 191 Spezies des Bork-Baumes vor. Daraus kann nun für jeden ORF ein Distanzvektor berechnet werden, wobei Homologien mit einem E-Wert größer als 1×10^{-10} ausgeschlossen werden.

$$\text{Distanzvektor } Dm = (d_0, d_1, \ldots, d_n)$$

$$d_n = -\begin{cases} \ln\left(\frac{S}{S_\mathrm{s}}\right) & \text{für: ORF liegt am Rand eines Contigs} \\ \ln\left(\frac{\frac{S}{S_\mathrm{s}}+\frac{S}{S_\mathrm{sh}}}{2}\right) & \text{für: sonst} \end{cases} \tag{4.2}$$

S : Smith-Waterman-Score; S_s : Selfscore des ORFs; S_sh : Selfscore des SIMAP-Hits

Für jeden ORF werden die Homologen zu den Proteinen der jeweiligen Spezies (Blattknoten) zur Berechnung der Distanz herangezogen. Gibt es mehrere homologe Proteine pro Spezies und ORF, wird das beste homologe Protein verwendet. Die Distanz ist der negative natürliche Logarithmus aus dem Mittelwert der Score zu Selfscore Verhältnisse (Score zu Selfscore des ORFs und Score zu Selfscore des Hits). Liegt der ORF am Rand eines Contigs, wird nur das Score zu Selfscore Verhältnis des ORFs verwendet. Der Grund hierfür ist, dass ein solcher ORF durch die Contig-Grenze abgeschnitten sein könnte. Somit ist der ORF wahrscheinlich nur zu einem Bruchteil des entsprechenden homologen Proteins homolog. Das Score zu Selfscore(Hit) Verhältnis würde in einem solchen Fall sehr klein, und somit auch die Distanz unverhältnismäßig klein.

Analyse

Eine taxomische Aussage über einen ORF wird möglich, wenn man den ORF in den Baum einordnen kann. Dazu wird der Distanzvektor des ORFs mit den Distanzvektoren der Knoten verglichen, und die besten (ähnlichsten) Knoten ausgewählt. Da die Werte eines ORF-Distanzvektors nicht direkt mit denen eines Knoten-Distanzvektors verglichen werden können, muß der ORF-Distanzvektor erst entsprechend skaliert werden. Für diese Skalierung müssen zwei Parameter geschätzt werden, ein Multiplikator f und ein Summand s.

$$Do' = (d_0 * f + s, d_1 * f + s, \ldots, d_n * f + s)$$

$$f = \frac{\sum_{i=0}^{n} dn_n}{\sum_{i=0}^{n}(do_n + s)} \tag{4.3}$$

$$s = f * \bar{dl} \tag{4.4}$$

Do' : skalierter ORF-Distanzvektor; dn : Knoten-Distanz; do : ORF-Distanz;

dl : durchschnittliche Distanz zu Blattknoten

Da f und s voneinander abhängen, werden sie iterativ mit den Startbedingungen $s = 0$ und $f = 1$ geschätzt (Stopbedingung: $|f_i - f_{i+1}| \leq 1 \times 10^{-6}$ oder $i \geq 1000$ (1000 Zyklen durchlaufen)).

Nach Anwendung der Parameter ist das Niveau der ORF-Distanzen in etwa mit dem der Knotendistanzen vergleichbar (siehe Abbildung 4.7).

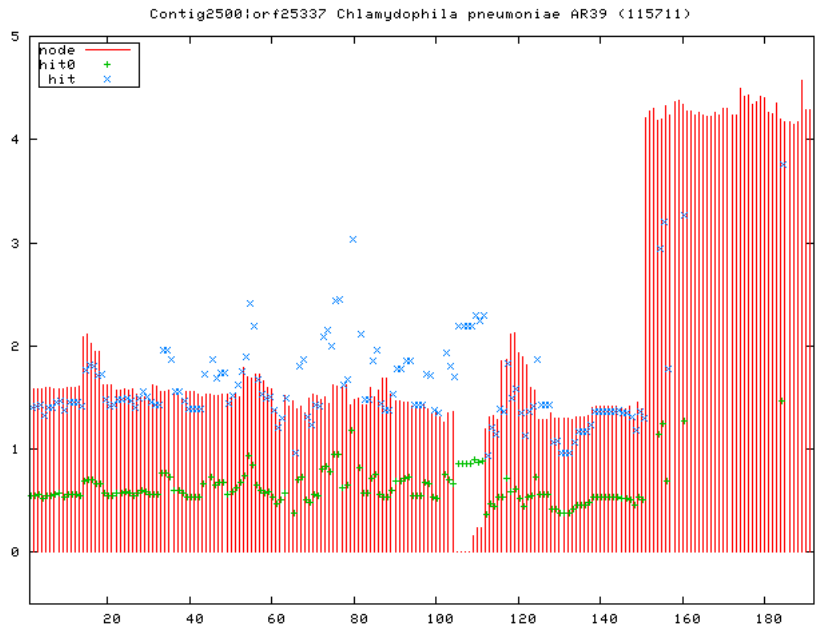

Abbildung 4.7.: Anpassung des ORFs ORF25337 an den Distanzvektor von *C. pneumoniae*
Linien: Distanzen des Referenzknotens (*C. pneumoniae*)
+: Rohdistanzen des ORFs
×: Angepaßte Distanzen des ORFs
X-Achse: Numerische ID der Blattknoten des Referenzbaumes (Spezies)
Y-Achse: Distanz
Die Rohdistanzen des ORFs werden durch Anwendung der geschätzten Parameter f und s auf das Niveau des zu vergleichenden Distanzvektors des Referenzbaum-Knotens gebracht.

Der skalierte ORF-Distanzvektor wird dann mit allen Knoten-Distanzvektoren des Bork-Baumes verglichen. Die ähnlichsten Knoten werden über die Summe der Fehlerquadrate bestimmt:

$$e = \sum_{i=0}^{n} (dn_n - do'_n)^2 \tag{4.5}$$

Die Knoten mit denen der ORF-Distanzvektor die kleinsten Fehlerquadrate aufweist, sind die zu dem ORF ähnlichsten Knoten.

Beispiel:
Der Knoten Y mit $D_y = (4.1, 3.9, 3.5, 0.4, 0.6)$ soll in den Baum (Abbildung 4.8) eingeordnet werden.

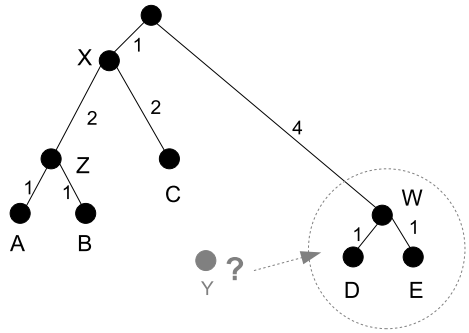

Abbildung 4.8.: Beispiel: Zuordnung eines Knoten nach der Methode der kleinsten Fehlerquadrate

$$D_a = (0, 2, 5, 9, 9)$$
$$D_b = (2, 0, 5, 9, 9)$$
$$D_c = (5, 5, 0, 8, 8)$$
$$D_d = (9, 9, 8, 0, 2)$$
$$D_e = (9, 9, 8, 2, 0)$$
$$D_z = (1, 1, 4, 8, 8)$$
$$D_x = (3, 3, 2, 6, 6)$$
$$D_w = (8, 8, 7, 1, 1)$$
$$D_y = (4.1, 3.9, 3.5, 0.4, 0.6)$$

Jeder Knoten des Baumes wird durchlaufen, wobei der Distanzvektor an jeden Knoten angepaßt und dann der Fehler berechnet wird:

- Knoten A: $s = 0.0; f = 2.976 \Rightarrow D'_y = (12.20, 11.61, 10.42, 1.19, 1.78) \Rightarrow e = 383.57$
- ...
- Knoten D: $s = 0.0; f = 2.314 \Rightarrow D'_y = (9.49, 9.03, 8.10, 0.93, 1.39) \Rightarrow e = 1.48$
- Knoten E: $s = 0.0; f = 2.353 \Rightarrow D'_y = (9.65, 9.18, 8.24, 0.94, 1.41) \Rightarrow e = 3.62$
- Knoten W: $s = 1.312; f = 1.312 \Rightarrow D'_y = (6.69, 6.43, 5.90, 1.84, 2.10) \Rightarrow e = 7.30$

Über die Betrachtung des Fehlers kann man den Knoten Y dem Zweig (W,D,E) zuordnen.

Ergebnis der Referenzbaum-Analyse

Oben genannte Prozedur wurde auf alle 431 ORFs angewendet. Betrachtet wurden jeweils die drei ähnlichsten Knoten, die im Bork-Baum gefunden wurden.
147 ORFs konnten aufgrund mangelnder Homologien nicht analysiert werden. Das heißt, bei diesen ORFs konnten kein Distanzvektoren berechnet werden. Bei weiteren 26 ORFs konnten nur weniger als 50 Werte (von 192) für die Distanzvektoren berechnet werden. Diese ORFs konnten ebenfalls nicht berücksichtigt werden.

Die Analyse zeigte, dass in allen drei Domänen (Archäen, Eukaryoten und Bakterien) Treffer gefunden wurden (siehe Abbildung 4.9). Bei den Archäen kann im Gegensatz zu den Bakterien keine genauere Zuordnung getroffen werden. Auch die Domäne der Eukaryonten wurde mehrmals getroffen. Als konkreter Vertreter dieser Domäne wurde nur ein Verwandter von *Giardia lamblia* gefunden, was aber nicht bedeutet, dass dies der einzige Eukaryot ist, der in dem Community-Genom vorkommt. Mehrere Treffer wurden für einen Verwandten von *Fibrobacter succinogenes* und *Gemmata obscuriglobus* gefunden. *Gemmata obscuriglobus* ist wie *K. stuttgartiensis* ein Planctomycet[1]. Da *K. stuttgartiensis* noch nicht in der SIMAP-Datenbank ist und somit *G. obscuriglobus* wahrscheinlich der nächste Verwandte ist, könnte es sich bei diesen ORFs durchaus um Teile des *K. stuttgartiensis* Genomes handeln, die nicht aus dem Community-Genom entfernt wurden. (Details siehe Appendix C.3, Seite 76)

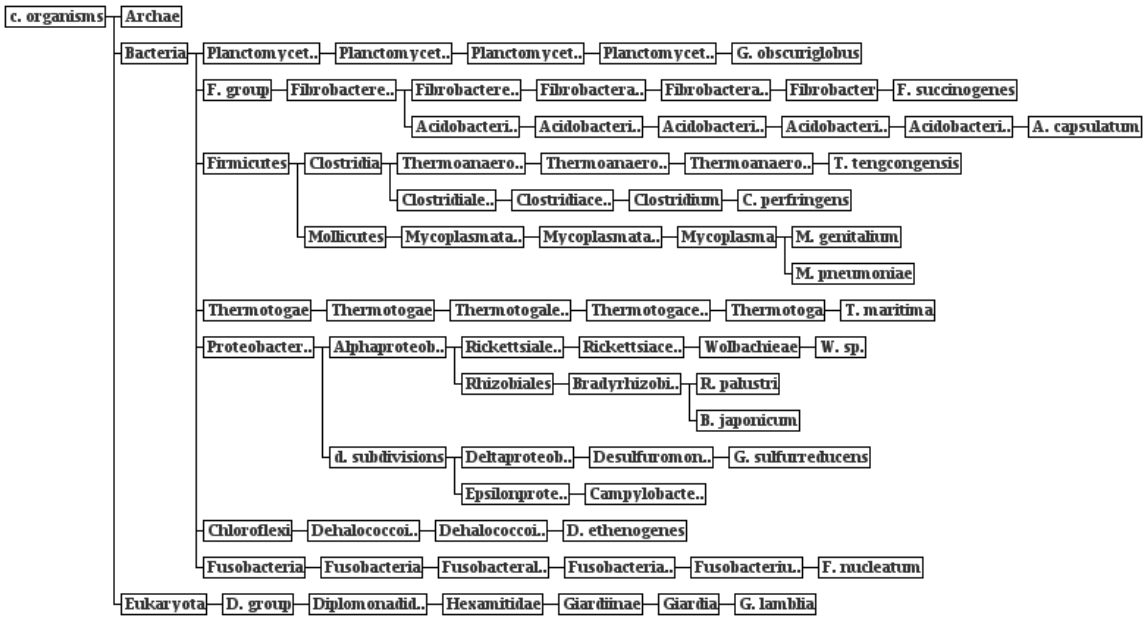

Abbildung 4.9.: Taxa im Anammox-Community Genom (Referenzbaum-Methode)
In allen drei Domänen (Archaeen, Bakterien und Eukaryoten) wurden Treffer gefunden. Im Gegensatz zu den Archaeen war bei den Bakterien eine genauere Unterscheidung möglich.

[1]Planctomyceten: Fakultativ anaerobe Bakterien, wovon die meisten Arten aquatisch leben

4.5. Ergebnis

Die Ergebnisse der beiden Methoden sehen auf den ersten Blick sehr unterschiedlich aus. Betrachtet man z. B. den Contig 9317, auf den ein Marker-Protein-ORF (ORF 232122) gefunden wurde: Die Beste-Homologen-Methode würde den ORF der Spezies *Lactococcus lactis*, die zur Gattung Firmicutes[2] gehört, zuordnen. Die Referenzbaum-Methode schlägt dagegen *Gemmata obscuriglobus*, gehörend zu den Planctomyceten, vor. Der Grund dafür ist, dass sich beide trotz Zugehörigkeit zu verschiedenen Gattungen phylogenetisch sehr ähnlich sind.
Die Beste-Homologen-Methode hat nur jeweils die besten Homologen berücksichtigt. Und hier wies *L. lactis* eine etwas höhere Homologie als *G. obscuriglobus* auf. Die Referenzbaum-Methode dagegen berücksichtigt möglichst alle Homologien und kommt so zu einem anderen Ergebnis. Aus den Vergleichen ORF232122 - *L. lactis* (Abbildung 4.10) und ORF232122 - *G. obscuriglobus* (Abbildung 4.11) wird ersichtlich, daß unter Berücksichtigung möglichst aller Homologien (Referenzbaum-Methode) die Zuordnung zu *G. obscuriglobus* besser paßt als zu *L. lactis*, wie es die Beste-Homologen-Methode vorgeschlagen hätte. Dadurch, dass die Referenzbaum-Methode mehr Information berücksichtigt, liefert sie zuverlässigere Ergebnisse.

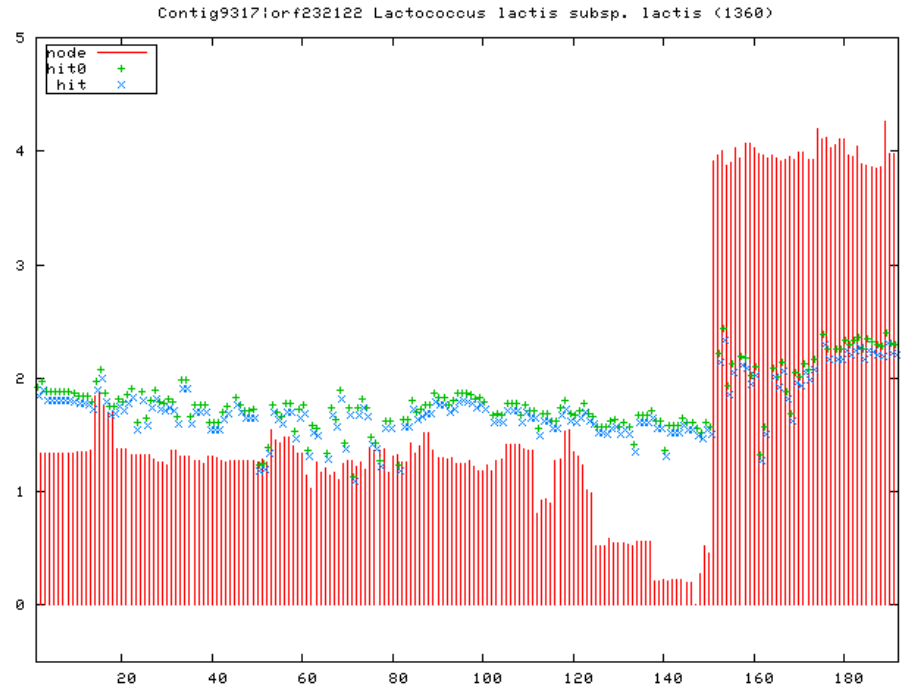

Abbildung 4.10.: Vergleich der Distanzvektoren ORF232122 - *Lactococcus lactis*
Die geringen Distanzen des Referenzbaum-Knotens im Bereich 125-150 spiegeln sich nicht im Distanzvektor des ORFs wider.

[2]Firmicutes: Artenreicher Bakterienstamm, der hauptsächlich aus Gram-positiven Baktieren besteht (mit Ausnahme der Mycoplasmen). Typische Vertreter: Streptokokken, Milchsäurebakterien, Mycoplasmen, usw.

Abbildung 4.11.: Vergleich der Distanzvektoren ORF232122 - *Gemata obscuriglobus*
Der Distanzvektor dieses Referenzbaum-Knotens zeigt eine bessere Übereinstimmung mit dem ORF-Distanzvektor als der des Referenzbaum-Knotes *Lactococcus lactis* (Abbildung 4.10)

Bei der Referenzbaum-Methode tritt häufig die Gruppe *G. obscuriglobus* und *F. succinogenes*, obwohl auch diese aus taxonomischer Sicht weit voneinander entfernt sind (Planctomyceten und Firmicutes). Auch hier ist der Grund die enge phylogenetische Verwandtschaft. Ein Vergleich von Abbildung 4.11 mit Abbildung 4.12 verdeutlicht die Ähnlichkeit. Die beiden Spezies unterscheiden sich in den Distanzen Nr. 80 (*G. obscuriglobus*) und 101 (*F. succinogenes*), also jeweils in den Distanzen zueinander. Zu allen anderen Taxa haben sie sehr ähnliche Distanzen. Aus phylogenetischer Sicht ist das häufige Auftreten dieser Gruppe kein Problem. Die Schwierigkeiten entstehen dann, wenn man davon eine taxonomische Einordnung ableiten möchte.

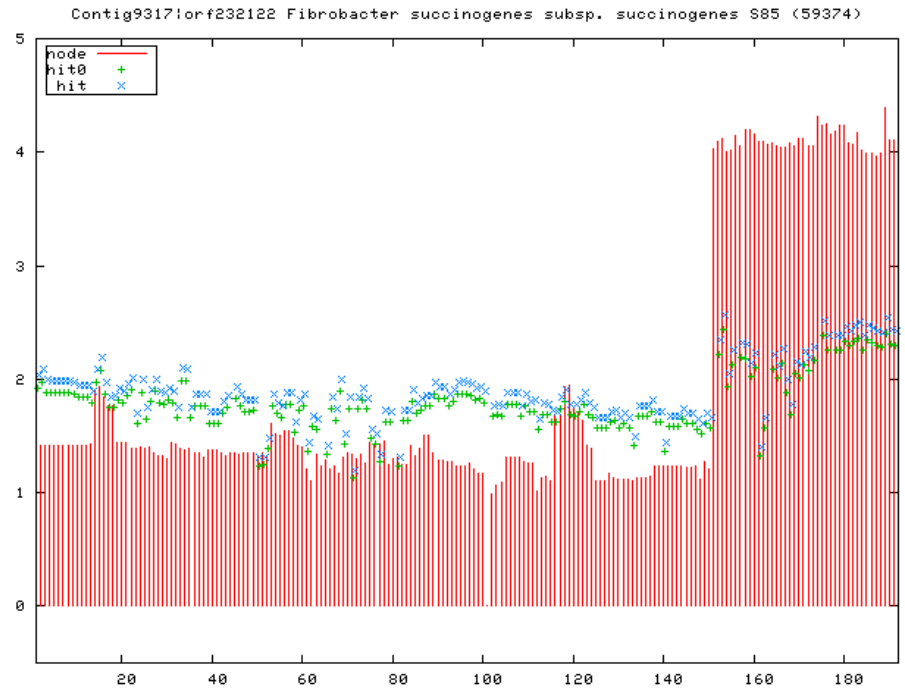

Abbildung 4.12.: Vergleich der Distanzvektoren ORF232122 - *Fibrobacter succinogenes*
Der Distanzvektor von *Fibrobacter succinogenes* ist dem von *Gemata obscuriglobus* (Abbildung 4.11) sehr ähnlich. Lediglich die Distanzen Nr. 80 und 101 (jeweils die Distanzen zueinander) unterscheiden sich deutlich.

Ursächlich begründet sind diese Probleme in der Diskrepanz zwischen verwendetem Referenzbaum und dem Taxonomiebaum. Diese Diskrepanz hat einen weiteren Effekt: Dadurch erklärt sich auch das häufige Auftreten der drei Domänen (v. a. die Domäne Archäen), ohne daß eine besonders ähnliche Gattung oder Spezies ermittelt werden konnte. Beim Mapping des Referenzbaumes auf die NCBI-Taxonomie (zu einem phylogenetischen Cluster wird eine Entsprechung im NCBI-Baum gesucht), mußte im NCBI-Baum häufig bis in die Hierarchie der Domänen aufgestiegen werden, um einen entsprechenden NCBI-Knoten finden zu können.

Zusammenfassend läßt sich sagen:

- Die Referenzbaum-Methode ist besser geeignet als die Beste-Homologen-Methode
- Probleme entstehen bei noch nicht in der Datenbank vorhandenen Spezies: Referenzbaum-Methode ermöglicht phylogenetische Zuordnung, eine Übertragung in die Taxonomie ist aber schwierig.
- Probleme nehmen mit der Diskrepanz zwischen Referenz- und Taxonomiebaum zu.

4.6. Ausblick

Die Referenzbaum-Methode ist geeignet, einen Überblick über die Zusammensetzung eines Community-Genomes zu bekommen. Um sowohl die Performance als auch die Aussagekraft der Ergebnisse zu verbessern, gibt es verschiedene Ansatzpunkte:

- Referenzdaten:
  - Schaffung einer kompakten Datenbank, die nur die Marker-Proteine aller sequenzierten Spezies enthält.
  - Suche nach weiteren Proteinen, die als Marker verwendet werden können.
  - Wahl eines geeigneten Referenzbaumes (möglichst ähnlich dem Taxonomiebaum oder Beschränkung auf phylogenetische Analyse)
- Algorithmen:
  - Verwenden einer anderen Gleichung zur Distanzberechnung (Gleichung 4.2)
  - Verbesserte Parameterschätzung, zur Anpassung der ORF-Distanzen an die Referenzbaum-Distanzen (Gleichungen 4.3 und 4.4)
  - Alternativen zur Kleinste-Fehlerquadrate-Methode zur Auswahl des ähnlichsten Knotens (Gleichung 4.5)
- Architektur:
  - Schaffung eines Softwarepakets zur automatischen Analyse von Community-Genomen
- Validierung:
  - Systematische Validierung anhand bekannter Genome, deren Proteine zum Test aus dem Referenzbaum entfernt und dann mittels der Referenzbaum-Methode klassifiziert werden (sog. Jack-Knife Verfahren).

5. Zusammenfassung

Um Proteinhomologe taxonomisch analysieren zu können, ist es notwendig, taxonomische Information interaktiv darstellen zu können. Für diesen Zweck wurde eine Softwarekomponente in Form eines Enterprise Java Beans [3] zur Visualisierung von und Navigation in taxonomischen Bäumen implementiert. Als Grafikframework wurde AWT/Swing verwendet, da es äußert anpassbar ist, und die Funktionalität 'Baumdarstellung' neben der Darstellung von taxonomischen Bäumen im SIMAP-Webinterface [1] auch in anderen Kontexten eingesetzt werden kann. Zur Positionierung der Knoten wurde der ER-Algorithmus [2] verwendet, da er einen guten Kompromiss zwischen Komplexität, Performance und Qualität darstellt. Das Enterprise Java Bean kann nun in das SIMAP-Webinterface z. B. mittels JSP oder Servlet eingebunden werden.
Ein weiterer Schwerpunkt der Arbeit war die Entwicklung einer Methode zur taxonomischen Analyse von Community-Genomen am Bespiel des Anammox Community-Genomes [4]. Es wurden die potentiellen Proteine, die in dem Community-Genom vorkommen, ermittelt. Durch Betrachtung der dazu homologen Proteine, die in SIMAP gefunden wurden, wurde versucht, taxonomische Informationen über das Anammox Community-Genom gewinnen zu können. Die Herangehensweise, nur die jeweils besten Proteinhomologen zu betrachten, erwies sich als wenig erfolgsversprechend. Die Referenzbaum-Methode, bei welcher Distanzvektoren aus der Homologieinformation berechnet und mit einem phylogenetischen Referenzbaum (Peer Bork's Baum des Lebens [6])verglichen wurden, stellte sich als besseres Verfahren heraus, da es mehr Information berücksichtigt. Probleme entstehen aber durch die Diskrepanz von phylogenetischen und taxonomischen Bäumen, so dass in vielen Fällen trotz phylogenetischer Zuordenbarkeit keine ausreichende taxonomische Klassifikation möglich ist. Dennoch hat die Methode das Potential zu einem schnellen, automatischen Analyseverfahren für Community-Genome weiterentwickelt zu werden.

Literaturverzeichnis

[1] Rattei, Thomas ; Arnold, Roland ; Tischler, Patrick ; Lindner, Dominik ; Stümpflen, Volker ; Mewes, H. W.: SIMAP: the similarity matrix of proteins. In: *Nucleic Acids Research* 34 (2006), Jan, S. 252–6

[2] Rosé, Eric: C++ objects to manage and draw trees dynamically. In: *MacTech* 11 (1995), Apr

[3] DeMichiel, Linda ; Keith, Michael: *EJB Core Contracts and Requirements*, Dec 2005

[4] Strous, Marc ; Pelletier, Eric ; Mangenot, Sophie ; Rattei, Thomas ; Lehner, Angelika ; Taylor, Michael W. ; Horn, Matthias ; Daims, Holger ; Bartol-Mavel, Delphine ; Wincker, Patrick ; Barbe, Valerie ; Fonknechten, Nuria ; Vallenet, David ; Segurens, Beatrice ; Schenowitz-Truong, Chantal ; Medigue, Claudine ; Collingro, Astrid ; Snel, Berend ; Dutilh, Bas E. ; Camp, Huub J. M. O. ; Drift, Chris van d. ; Cirpus, Irina ; Pas-Schoonen, Katinka T. d. ; Harhangi, Harry R. ; Niftrik, Laura van ; Schmid, Markus ; Keltjens, Jan ; Vossenberg, Jack van d. ; Kartal, Boran ; Meier, Harald ; Frishman, Dmitrij ; Huynen, Martijn A. ; Mewes, Hans-Werner ; Weissenbach, Jean ; Jetten, Mike S. M. ; Wagner, Michael ; Paslier, Denis L.: Deciphering the evolution and metabolism of an anammox bacterium from a community genome. In: *Nature* 440 (2006), Apr, S. 790–4

[5] Pearson, WR ; Lipman, DJ: Improved tools for biological sequence comparison. In: *Proc. Natl. Acad. Sci. USA* 85 (1988), Apr, S. 2444–2448

[6] Ciccarelli, Francesca D. ; Doerks, Tobias ; Mering, Christian von ; Creevy, Christopher J. ; Snel, Berend ; Bork, Peer: Toward Automatic Reconstruction of a Highly Resolved Tree of Life. In: *Science* 311 (2006), Mar, S. 1283–7

[7] Wheeler, David L. ; Chappey, Colombe ; Lash, Alex E. ; Leipe, Detlef D. ; Madden, Thomas L. ; Schuler, Gregory D. ; Tatusova, Tatiana A. ; Rapp, Barbara A.: Database resources of the National Center for Biotechnology Information. In: *Nucleic Acids Research* 34 (2006), Jan, S. 173–180

[8] Gibas, Cynthia ; Jambeck, Per: *Einführung in die praktische Bioinformatik*. O'Reilly, 2001. – ISBN 3–89721–289–7

[9] Haubold, Bernhard ; Wiehe, Thomas: *Introduction to Computational Biology. An Evolutionary Approach.* Birkhäuser, 2006. – ISBN 3–76436–700–8

[10] Feng, Da-Fei ; Doolittle, Russell F.: Converting Amino Acid Alignment Scores into Measures of EvolutionaryTime: A Simulation Study of Various Relationships. In: *Journal of Molecular Evolution* 44 (1997), Apr, S. 361370

[11] Sokal, R.R. ; Michener, C.D.: A statistical method for evaluating systematic relationships. In: *University of Kansas Science Bulletin* 38 (1958), S. 14091438

[12] Felsenstein, Joseph: Confidence Limits on Phylogenies: An Approach Using the Bootstrap. In: *Evolution* 39 (1985), Jul, S. 783–791

[13] Elfron, B. ; Tibshirani, R. J.: *An Introduction to the Bootstrap*. Chapman & Hall, 1994. – ISBN 0–41204–231–2

[14] Linné, Carl: *Systema naturae per regna tria naturae, secundum classses, ordines, genera, species, cum characteribus, differentiis, synonymis, locis*. Salvius, 1758

[15] Page, Roderic D. M. ; Charleston, Michael A.: From Gene to Organismal Phylogeny: Reconciled Trees and the Gene Tree/Species Tree Problem. In: *Molecular Phylogenetics and Evolution* 7 (1997), Apr, S. 231–240

[16] Bromhan, L. ; Penny, D.: The modern molecular clock. In: *Nature Reviews Genetics* 4 (2003), Mar, S. 216–224

[17] Snel, Berend ; Bork, Peer ; Huynen, Martijn A.: Genome phylogeny based on gene content. In: *Nature Genetics* 21 (1999), Jan, S. 108–110

[18] Konstantinidis, Konstantinos T. ; Tiedje, James M.: Towards a Genome-Based Taxonomy for Prokaryotes. In: *Journal of Bacteriology* 187 (2005), Sept, S. 62586264

[19] Wilson, Cyrus A. ; Kreychman, Julia ; Gerstein, Mark: Assessing annotation transfer for genomics: quantifying the relations between protein sequence, structure and function through traditional and probabilistic scores. In: *Journal of Molecular Biology* 297 (2000), Mar, S. 233–249

[20] Smith, Temple F. ; Waterman, Michael S.: Identification of Common Molecular Subsequences. In: *Journal of Molecular Biology* 147 (1981), Mar, S. 195–197

[21] Tringe, S. G.: Comparative metagenomics of microbial communities. In: *Science* 308 (2005), Apr, S. 554–557

[22] Anderson, S.: Shotgun DNA sequencing using cloned DNase I-generated fragments. In: *Nucleic Acids Research* 9 (1981), Jul, S. 30153027

[23] Giovannoni, Stephen J. ; Britschgi, Theresa B. ; Moyer, Craig L. ; Field, Katharine G.: Genetic diversity in Sargasso Sea bacterioplankton. In: *Nature* 345 (1990), May, S. 60–63

[24] Chen, Kevin ; Pachter, Lior: Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities. In: *PLoS Computational Biology* 1 (2005), Jul, S. 106–112

[25] Venter, J. C. ; Remington, Karin ; Heidelberg, John F. ; Halpern, Aaron L. ; Rusch, Doug ; Eisen, Jonathan A. ; Wu, Dongying ; Paulsen, Ian ; Nelson, Karen E. ; Nelson, William ; Fouts, Derrick E. ; Levy, Samuel ; Knap, Anthony H. ; Lomas, Michael W. ; Nealson, Ken ; White, Owen ; Peterson, Jeremy ; Hoffman, Jeff ; Parsons, Rachel ; Baden-Tillson, Holly ; Pfannkoch, Cynthia ; Rogers, Yu-Hui ; Smith, Hamilton O.: Environmental genome shotgun sequencing of the Sargasso Sea. In: *Science* 304 (2004), Apr, S. 66–74

[26] Broda, Engelbert: Die Evolution der bioenergetischen Prozesse. In: *Biologie in unserer Zeit* 7 (1977), S. 33–40

[27] Strous, Marc ; Fuerst, John A. ; Kramer, Evelien H. M. ; Logemann, Susanne ; Muyzer, Gerard ; Pas-Schoonen, Katinka T. d. ; Webb, Richard ; Kuenen, J. G. ; Jetten, Mike S. M.: Missing Lithotroph Identified as New Planctomycete. In: *Nature* 400 (1999), Jul, S. 446–449

[28] Gosling, James ; Joy, Bill ; Steele, Guy ; Bracha, Gilad: *The Java Language Specification*. 3. Addison-Wesley, 2005. – ISBN 0–321–24678–0

[29] *Java2 Platform Enterprise Edition Specification, v1.4*. `http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`, Nov 2003

[30] Farley, Jim ; Crawford, William: *Java Enterprise in a Nutshell*. 3. O'Reilly, 2005. – ISBN 0–59610–142–2

[31] Hamilton, Graham: *Sun Microsystems JavaBeans*, Aug 1997

[32] Stearns, John ; Chinnici, Roberto ; Sahoo: Update: An Introduction to the Java EE 5 Platform. (2006), May

[33] *Spezifikation des GIF-Formates*. `http://www.w3.org/Graphics/GIF/spec-gif89a.txt`, Jan 1995

[34] Hamilton, Eric: *JPEG File Interchange Format*, 1992

[35] Roelofs, Greg: *PNG: The Definitive Guide*. O'Reilly. – ISBN 1–56592–542–4

[36] Anderssoni, Ola: *Scalable Vector Graphics (SVG) 1.1 Specification*, Jan 2003

[37] Eisenberg, J. D.: *SVG Essentials*. O'Reilly, 2002. – ISBN 0–59600–223–8

[38] Alur, Deepak ; Crupi, John ; Malks, Dan: *Core J2EE Patterns: Best Practices and Design Strategies*. 2. Prentice Hall PTR, 2003. – ISBN 0–13142–246–4

[39] Ewing, Brent ; Hillier, LaDeana ; Wendl, Michael C. ; Green, Phil: Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment. In: *Genome Research* 8 (1998), Mar, S. 175–185

[40] Ewing, Brent ; Green, Phil: Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. In: *Genome Research* 8 (1998), Mar, S. 186–194

[41] Rice, P. ; Longden, I. ; Bleasby, A.: EMBOSS: The European Molecular Biology Open Software Suite. In: *Trends in Genetics* 16 (2000), Jun, S. 276–277

[42] *Wikipedia*. `http://www.wikipedia.org`, Okt 2006

Abkürzungsverzeichnis

Anammox	Anaerobic ammonium oxidation.
AWT	Abstract Window Toolkit.
BMP	Bean Managed Persistance.
CMP	Container Managed Persistance.
CMS	Content Management System.
COG	Clusters of Orthologous Groups.
EJB	Enterprise Java Beans.
EMBL	European Molecular Biology Laboratory.
EMBOSS	European Molecular Biology Open Software Suite.
GIF	Graphics Interchange Format.
HGT	Horizontaler Gentransfer.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
J2EE	Java Enterprise Edition.
J2SDK	Java Software Development Kit.
J2SE	Java Standard Edition.
JFIF	JPEG File Interchange Format.
JPEG	Joint Photographic Experts Group.
JSP	Java Server Pages.
NCBI	National Center for Biotechnology Information, USA.
ORF	Open Reading Frame.
PDB	Protein Databank.
PFAM	Protein Families.
PNG	Portable Network Graphics.
RMI	Remote Methode Invocation.

SIMAP Similarity Matrix of Proteins.

SVG Scalable Vector Graphics.

TrEMBL Translated EMBL.

UniProt Universal Protein Database.

UPGMA Unweighted Pair Group Method with Arithmatic Mean.

XML Extensible Markup Language.

Stichwortverzeichnis

A

Anammox 16, 36
Anammox-Datenbank 36
Apache Batik 24
Applikationsserver 19
AWT/Swing 24, 28

B

Beste-Homologen-Methode 41, 67
Bootstrapping 11
Business Delegate 29

C

Client-Interface 20
Clustering 10
Clusters of Orthologous Groups (COGs) 12, 65
Community-Genom 14, 16, 42
Content Management System (CMS) 13
Contigs 14, 37

D

Datenbanken 7
Distanzmatrix 42
Distanzvektor 42, 44

E

EJB 29, 34
EMBOSS 39
Enterprise Computing 17
Enterprise Java Beans (EJB) 19
Entity Beans 20
ER-Algorithmus 30

F

FASTA 12, 40

G

Genvorhersage 39
getorf 39
GIF 21
Grafikformate 21, 26, 63
Grafikframeworks 23, 28, 63
Graphics,Graphics2D 23, 28

H

Home-Interface 20
Homologe 40
Horizontaler Gentransfer (HGT) 12, 65
HTML 19
HTTPServlet 18

I

Implementierung 32
Interpro 7

J

J2EE 17
J2SE,J2SDK 17
Java Beans 19
JPEG 21, 26, 63
JSP 18

K

Kleinste-Fehlerquadrate 45

L

Local-Interface 20

M

Marker-Proteine . . . . . 41 f
Message Driven Beans 20
Metagenomics 14

N

NCBI 8, 43
Newick 42

O

ORF 15, 39

P

PDB 8
Peer Bork's Tree of Life 12, 64
PFAM 8
Phred 37
Phylogenie . . . . . 10 f, 15
PNG 22, 26, 63
Prosite 7

R

Rastergrafik 21
Referenzbaum-Methode 16, 42, 74
Remote-Interface 20, 35
rRNA (16S) 14

S

Score 44
Selfscore 44
Servlets 18
SessionBeans 19
Shotgun-Sequenzierung 14
SIMAP 12, 62
Smith-Waterman-Algorithmus 12
Stateful 19
Stateless 19
SVG 22, 26
SVGGraphics2D 28

T

Taxonomie 8, 11, 43
Taxonomiebaum 33
TaxonomyTreeBean 34
Thin Client 18
Tree-Drawing 30, 32
TreeLayout-Manager 32, 66

U

Uniprot/TrEMBL 7
UniRef 7

V

Vektorgrafik 22

W

Webapplikation 18
Webinterface 13, 35

X

XML 22, 26

Tabellenverzeichnis

2.1. Überblick: Taxonomische Information in Proteindatenbanken 8
2.2. Umfang der SIMAP Datenbank 14

4.1. Spezies, zu denen mehrere homologe ORFs im Anammox-Community Genom gefunden werden konnten ('Beste Homologen'-Methode) 41
4.2. Klassen, zu denen mehrere homologe ORFs im Anammox-Community Genom gefunden werden konnen ('Beste Homologen'-Methode) 42

A.1. Vergleich zwischen PNG- und JPEG-Kodierungsdauer 63
A.2. Universell verteilte, nicht horizontal gentransferierte COGs 65

C.1. Anammox-Community Genom, Zusammenfassung 67
C.2. Ergebnis: Taxonomische Zuordnung der ORFs nach der 'Beste Homologe'-Methode 74
C.3. Ergebnis: Zuordnung der Contigs nach der Referenzbaum-Methode 76

Abbildungsverzeichnis

2.1. Die zwei wichtigsten Tabellen der NCBI-Taxonomie 9
2.2. Struktur des NCBI Taxonomie-Baums 9
2.3. Beispiel: Fiktiver phylogenetischer Baum mit Angabe der Bootstrap-Unterstützung 10
2.4. SIMAP Architektur 13
2.5. Assemblierung von Shotgun-Sequenzen 15
2.6. EJB: Home- und Client-Interfaces 20
2.7. Beispiel für eine SVG-Grafik (oben) mit entsprechendem XML-Code (unten) . . 23
2.8. Grafikprogrammierung in Java 23

3.1. Vergleich Kodierungsdauer JPEG/PNG abhängig von der Bildgröße (Objektanzahl: 100) 27
3.2. Vergleich Kodierungsdauer JPEG/PNG abhängig von Anzahl dargestellter Objekte (Bildgröße: 250.000 Pixel) 27
3.3. Integration der Taxonomiebaum-Komponente in die Client-EJB-Kommunikation 30
3.4. ER-Algorithmus: Beschreibung eines (Sub-)Baumes mittels Rechtecken. 31
3.5. ER-Algorithmus (Pseudocode) 32
3.6. 'Tree-Drawing' mit AWT/Swing unter Verwendung eines Layoutmanagers 33
3.7. Verschiedene Baumtypen mit Hilfe des TreeLayoutManagers gezeichnet; 33
3.8. Taxonomiebaum unter Verwendung des TreeLayoutManagers 34
3.9. SIMAPTaxonomyTreeBean 34

4.1. Schema der Anammox-Community Datenbank 36
4.2. Contiglängenverteilung 37
4.3. Anteil nicht identifizierter Nukleotide 38
4.4. Anteil 'unsicherer' Nukleotide 38
4.5. Längenverteilung der im Anammox-Community Genom gefundenen ORFs 40
4.6. Beispiel: Berechnung eines Distanzvektors 43
4.7. Anpassung des ORFs ORF25337 an den Distanzvektor von *C. pneumoniae* . . . 45
4.8. Beispiel: Zuordnung eines Knoten nach der Methode der kleinsten Fehlerquadrate 46
4.9. Taxa im Anammox-Community Genom (Referenzbaum-Methode) 47
4.10. Vergleich der Distanzvektoren ORF232122 - *Lactococcus lactis* 48
4.11. Vergleich der Distanzvektoren ORF232122 - *Gemata obscuriglobus* 49
4.12. Vergleich der Distanzvektoren ORF232122 - *Fibrobacter succinogenes* 50

A.1. Screenshot der SIMAP-Webseite 62
A.2. Peer Bork's 'Tree of Life' [6] 64

B.1. Beispiel: Anwendung des TreeLayouts 66

A. Einleitung

A.1. SIMAP

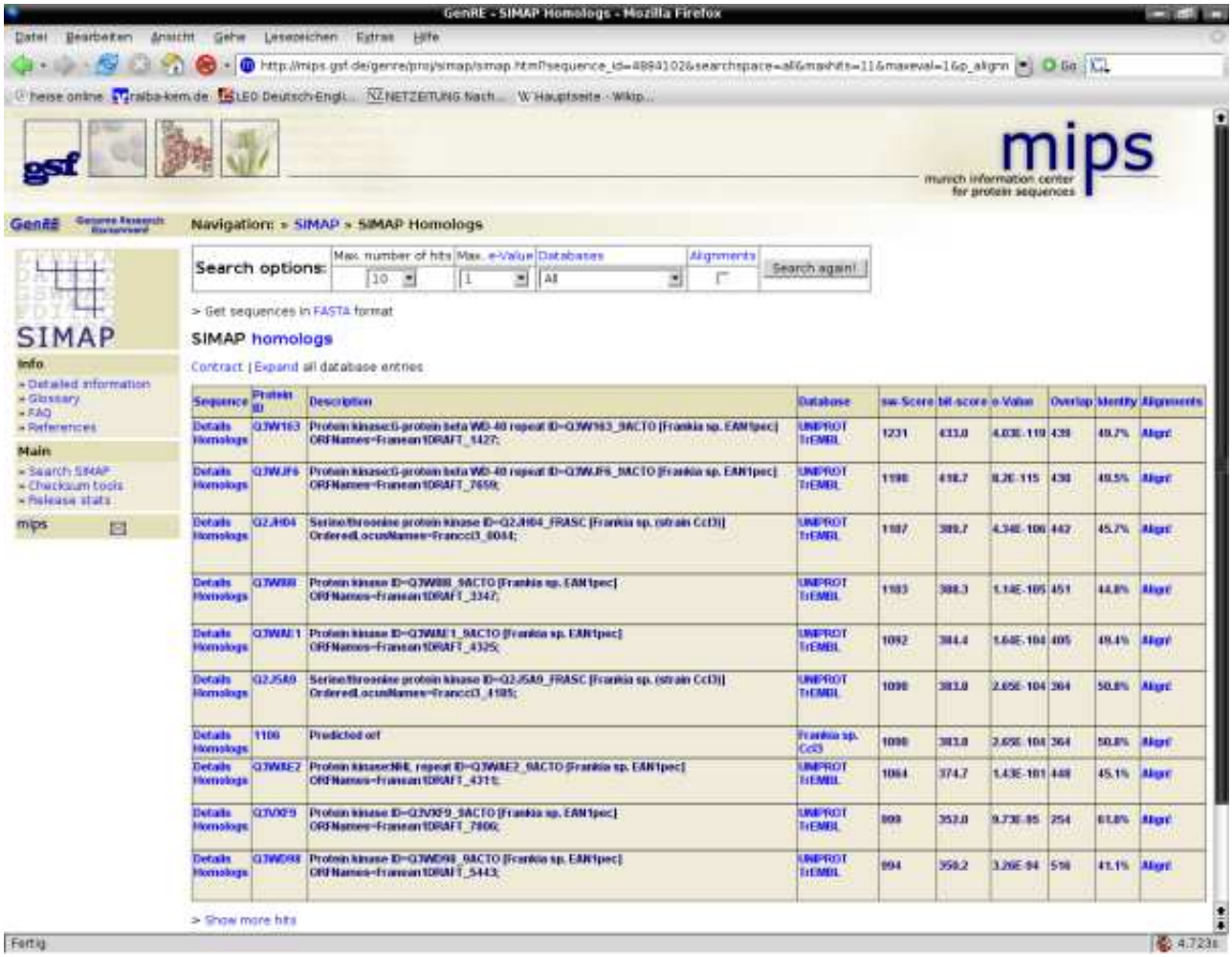

Abbildung A.1.: Screenshot der SIMAP-Webseite
(`http://mips.gsf.de/proj/simap/`)

A.2. Grafikformate und -frameworks

Vergleich zwischen PNG- und JPEG-Kodierung

Objektanzahl	**Bildgröße**	**Zeit**	
		JPEG	**PNG**
50	10.000	25	91
	40.000	30	245
	90.000	52	465
	160.000	101	652
	250.000	129	808
	360.000	144	994
	490.000	167	1288
	640.000	210	1527
100	10.000	30	118
	40.000	44	349
	90.000	56	664
	160.000	104	1031
	250.000	156	1361
	360.000	155	1533
	490.000	192	2118
	640.000	263	2073
200	10.000	68	107
	40.000	56	421
	90.000	81	855
	160.000	147	1445
	250.000	145	1967
	360.000	168	2312
	490.000	196	3128
	640.000	262	3703
400	10.000	29	79
	40.000	33	365
	90.000	68	903
	160.000	120	1625
	250.000	161	2439
	360.000	224	3181
	490.000	208	4120
	640.000	296	4991

Tabelle A.1.: Vergleich zwischen PNG- und JPEG-Kodierungsdauer

A.3. Peer Bork's 'Tree of Life'

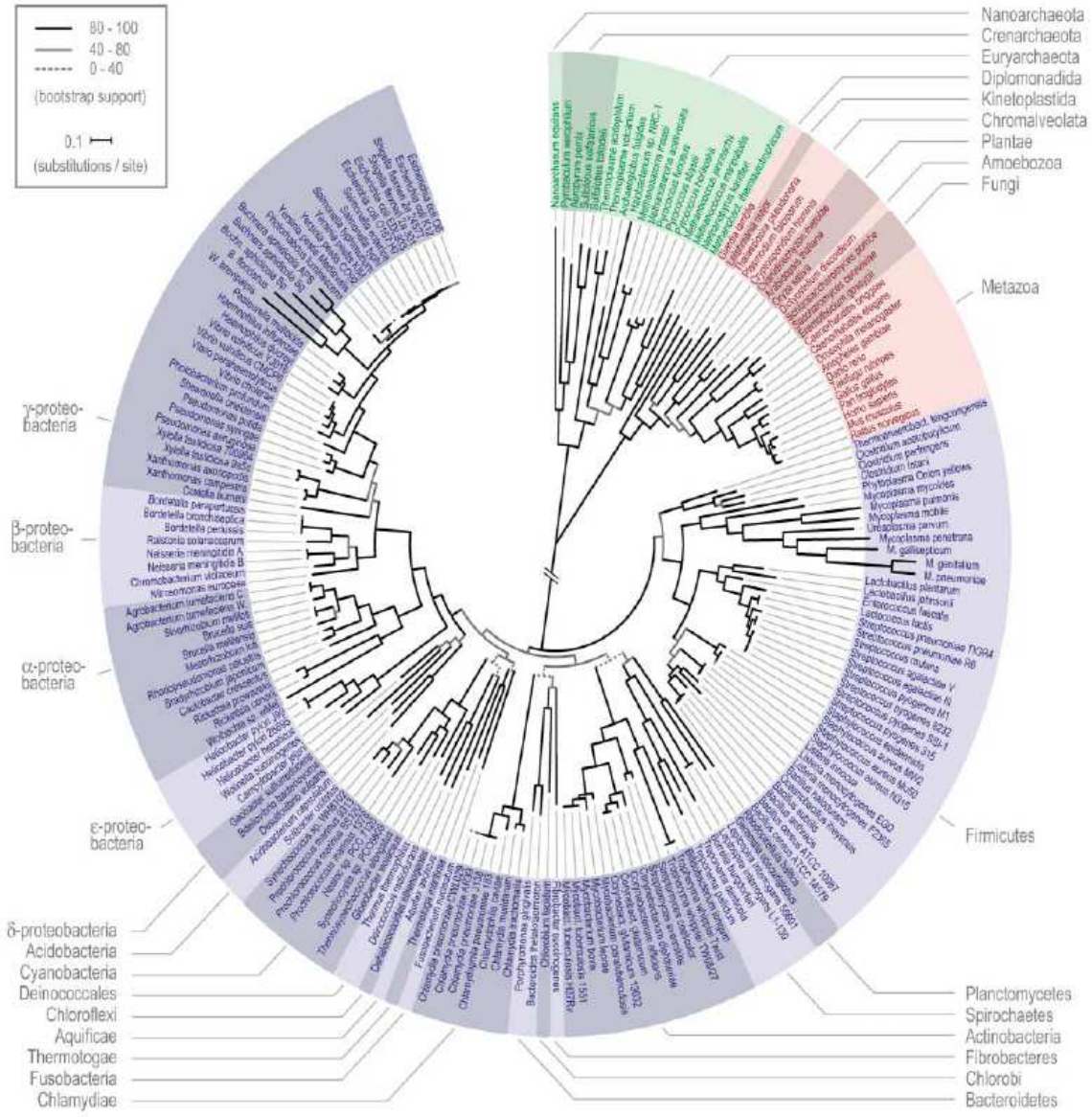

Abbildung A.2.: Peer Bork's 'Tree of Life' [6]

A.4. Universelle, nicht-HGT COGs

Im Supporting Material zu Peer Bork's Methode [6] werden 36 universelle COGs genannt, wovon jedoch 6 als HGT und/oder nicht eindeutig alignierbar gekennzeichnet sind. Somit verbleiben 30 universelle Marker-COGs. Welchen COG Peer Bork dennoch als 31. Marker-COG verwendete, geht leider nicht eindeutig hervor. Deswegen habe ich als Marker nur die folgenden 30 COGs verwendet.

COG-ID	**Beschreibung**
COG0012	Predicted GTPase, probable translation factor
COG0016	Phenylalanine-tRNA synthethase alpha subunit
COG0048	Ribosomal protein S12
COG0049	Ribosomal protein S7
COG0052	Ribosomal protein S2
COG0080	Ribosomal protein L11
COG0081	Ribosomal protein L1
COG0087	Ribosomal protein L3
COG0091	Ribosomal protein L22
COG0092	Ribosomal protein S3
COG0093	Ribosomal protein L14
COG0094	Ribosomal protein L5
COG0096	Ribosomal protein S8
COG0097	Ribosomal protein L6P/L9E
COG0098	Ribosomal protein S5
COG0099	Ribosomal protein S13
COG0100	Ribosomal protein S11
COG0102	Ribosomal protein L13
COG0103	Ribosomal protein S9
COG0172	Seryl-tRNA synthetase
COG0184	Ribosomal protein S15P/S13E
COG0186	Ribosomal protein S17
COG0197	Ribosomal protein L16/L10E
COG0200	Ribosomal protein L15
COG0201	Preprotein translocase subunit SecY
COG0202	DNA-directed RNA polymerase, alpha subunit
COG0256	Ribosomal protein L18
COG0495	Leucyl-tRNA synthetase
COG0522	Ribosomal protein S4 and related proteins
COG0533	Metal-dependent proteases with chaperone activity

Tabelle A.2.: Universell verteilte, nicht horizontal gentransferierte COGs

B. Taxonomie zur Visualisierung und Benutzerführung

B.1. Tree-Layout

```
public class TreeLayoutExample extends JFrame {

public TreeLayoutExample() {
//Determine the tree's style
TreeStyle style = new TreeStyle();
style.setConnection(TreeStyle.CONNECTION_DIRECT);
style.setNodePosition(TreeStyle.NODE_CENTER);
style.setOrientation(TreeStyle.ORIENTATION_VERTICAL);

Tree tree = new Tree(style);
this.buildTree(tree);
this.add(tree);
}

private void buildTree(Tree tree) {
//Create nodes (of type (J)Component)
JLabel node0 = new JLabel("Root");
JLabel node00 = new JLabel("Node 0,0");
JLabel node01 = new JLabel("Node 0,1");
JLabel node010 = new JLabel("Node 0,1,0");
JLabel node011 = new JLabel("Node 0,1,1");
//...

//Style nodes
node0.setBorder(BorderFactory.createLineBorder(Color.BLACK));
node0.setForeground(Color.LIGHT_GRAY);
node00.setBorder(BorderFactory.createLineBorder(Color.BLACK));
node01.setBorder(BorderFactory.createLineBorder(Color.BLACK));
node010.setBorder(BorderFactory.createLineBorder(Color.BLACK));
node011.setBorder(BorderFactory.createLineBorder(Color.BLACK));
//...

//Add nodes to the tree
//The second parameter says to which parent node this node should be added
tree.add(node0, "");
tree.add(node00, "0");
tree.add(node01, "0");
tree.add(node010, "0,1");
tree.add(node011, "0,1");
//...
}

public static void main(String[] args) {
TreeLayoutExample example = new TreeLayoutExample();
example.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
example.pack();
example.setVisible(true);
}
}
```

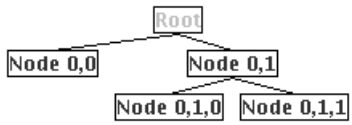

Abbildung B.1.: Beispiel: Anwendung des TreeLayouts

C. Taxonomische Analyse des Anammox-Community Genomes

Anzahl Contigs	9277
Anzahl Nukleotide	17.654.753
Anzahl Lower Case	4.185.818 (23,7%)
Anzahl Upper Case	13.435.616 (76,1%)
Anzahl 'Gaps'	33.319 (0.2%)
Contig-Länge (min/avg/max)	231/1939/150.503
Anzahl ORFs	298.253
ORF Länge min/avg/max	100/249/6.591
Anzahl kurze ORFs (kleiner 350)	247.253 (82,9%)
Anzahl lange ORFs (größer/gleich 350)	51.000 (17,1%)
Anzahl ORFs mit SIMAP-Homologen(*)	173.356 (58,12%)
Anzahl ORFs mit COG-Homologen(*)	23.655 (7,93%)
Anzahl ORFs mit Marker-COG-Homologen(*)	431 (0,14%)
Anzahl Contigs mit SIMAP-Homologen(*) Gesamtlänge	9.275 (99,97%) 17.653.462 (99,99%)
Anzahl Contigs mit COG-Homologen(*) Gesamtlänge	8.143 (87,78%) 16.169.607 (91,59%)
Anzahl Contigs mit Marker-COG-Homologen(*) Gesamtlänge	261 (2,81%) 1.462.588 (8,28%)

Tabelle C.1.: Anammox-Community Genom, Zusammenfassung
*) max. e-Wert: 10

C.1. 'Beste Homologe'-Analyse

In der Spalte 'EValue' ist der EValue des bestimmenden Hits angegeben. '-' in Spalten Species, Class, usw. bedeutet, dass hier keine Zuordnung möglich war. ORFs, die auf dem selben Contig liegen, sind farbig unterlegt.

Contig	**ORF**	**Species**	**Class**	**Phylum**	**SuperK.**	**EValue**
1022	510	Borrelia sp. NE581	Spirochaetes (class)	Spirochaetes	Bacteria	1.08E-099
1033	715	-	-	-	Bacteria	2.86E-016
1033	722	-	-	-	Bacteria	7.11E-013
1038	804	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	3.99E-046
106	1252	Lactococcus lactis	Bacilli	Firmicutes	Bacteria	2.54E-014
107	1438	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	3.63E-040
1079	1618	-	Alphaproteobacteria	Proteobacteria	Bacteria	1.23E-043
1192	4075	-	-	Proteobacteria	Bacteria	2.09E-062

1262	5492	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	7.93E-050
1343	7252	-	Alphaproteobacteria	Proteobacteria	Bacteria	2.55E-025
1343	7250	-	Alphaproteobacteria	Proteobacteria	Bacteria	1.93E-027
1429	8980	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	6.41E-018
1429	8979	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.57E-034
1429	8978	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	4.41E-014
1431	9039	-	-	-	Bacteria	8.42E-026
1439	9190	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	5.94E-012
1439	9187	-	-	Firmicutes	Bacteria	8.31E-034
145	9411	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.30E-029
1503	10638	-	Alphaproteobacteria	Proteobacteria	Bacteria	7.73E-045
1528	11170	Staphylococcus aureus	Bacilli	Firmicutes	Bacteria	3.10E-069
1528	11169	-	Bacilli	Firmicutes	Bacteria	5.20E-034
1540	11470	-	Alphaproteobacteria	Proteobacteria	Bacteria	1.04E-057
1599	12649	-	Alphaproteobacteria	Proteobacteria	Bacteria	6.91E-077
1599	12650	-	Alphaproteobacteria	Proteobacteria	Bacteria	8.13E-016
177	14159	Caulobacter vibrioides	Alphaproteobacteria	Proteobacteria	Bacteria	5.31E-046
1835	15526	-	Gammaproteobacteria	Proteobacteria	Bacteria	6.17E-031
1835	15528	-	-	-	Bacteria	4.36E-011
1866	16131	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	4.40E-140
1916	17107	-	-	-	-	5.51E-100
1935	17518	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	1.78E-080
1958	17971	Pseudomonas aeruginosa	Gammaproteobacteria	Proteobacteria	Bacteria	2.52E-087
1993	18654	-	-	-	Bacteria	7.71E-038
2210	23356	-	-	-	Bacteria	5.96E-021
2210	23358	-	-	-	Bacteria	1.69E-099
2231	23829	Escherichia coli	Gammaproteobacteria	Proteobacteria	Bacteria	8.20E-013
2231	23823	-	-	-	Bacteria	5.08E-052
2231	23822	-	-	Firmicutes	Bacteria	3.22E-012
2232	23836	-	Bacilli	Firmicutes	Bacteria	9.22E-071
2281	24911	-	-	Proteobacteria	Bacteria	7.48E-018
2299	25269	-	-	Proteobacteria	Bacteria	1.39E-078
2500	25337	-	Bacilli	Firmicutes	Bacteria	5.67E-045
2514	25611	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	4.99E-043
2516	25636	-	-	-	Bacteria	2.77E-075
2632	27842	Staphylococcus aureus	Bacilli	Firmicutes	Bacteria	1.51E-035
2682	28843	-	-	-	Bacteria	5.13E-062
2837	32722	Pseudomonas aeruginosa	Gammaproteobacteria	Proteobacteria	Bacteria	8.70E-049
2890	34049	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	1.83E-118
2945	35454	-	Actinobacteria (class)	Actinobacteria	Bacteria	7.10E-028
3005	36985	-	-	-	Bacteria	7.27E-026
3005	36983	Anabaena variabilis	null	Cyanobacteria	Bacteria	5.91E-061
304	37911	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	2.83E-052
310	39442	-	-	Proteobacteria	Bacteria	1.64E-014
310	39445	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	6.18E-059
3173	41347	-	-	-	Bacteria	1.77E-013
3173	41341	-	Actinobacteria (class)	Actinobacteria	Bacteria	1.58E-017

319	41786	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	4.46E-033
319	41781	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	1.44E-048
3197	41947	Mycoplasma pulmonis	Mollicutes	Firmicutes	Bacteria	7.64E-037
3347	45758	Escherichia coli	Gammaproteobacteria	Proteobacteria	Bacteria	6.02E-050
3347	45754	-	-	Proteobacteria	Bacteria	2.89E-046
340	47168	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	1.97E-058
3622	52698	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	3.24E-084
3622	52692	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	2.03E-060
3643	53202	Pyrobaculum aerophilum	Thermoprotei	Crenarchaeota	Archaea	4.66E-029
3668	53813	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	4.41E-096
3793	57112	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	2.17E-063
3803	57351	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	6.91E-054
381	57529	Caulobacter vibrioides	Alphaproteobacteria	Proteobacteria	Bacteria	2.01E-081
3882	59225	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	7.31E-029
3882	59220	Anabaena sp.	null	Cyanobacteria	Bacteria	3.00E-011
3882	59227	Synechocystis sp. PCC 6803	null	Cyanobacteria	Bacteria	2.01E-022
39	59664	-	-	-	Bacteria	5.47E-035
3911	59995	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	4.98E-026
3911	59998	-	-	Proteobacteria	Bacteria	1.95E-035
3911	59990	-	Bacilli	Firmicutes	Bacteria	1.24E-025
3913	60040	Treponema pallidum	Spirochaetes (class)	Spirochaetes	Bacteria	2.59E-088
4065	63832	-	-	-	Bacteria	3.85E-021
4065	63837	Borrelia sp. NE581	Spirochaetes (class)	Spirochaetes	Bacteria	1.83E-019
4075	64076	Rickettsia conorii	Alphaproteobacteria	Proteobacteria	Bacteria	1.29E-040
4170	66462	-	-	-	Bacteria	5.87E-060
4228	67915	-	-	-	Bacteria	5.57E-012
4228	67924	-	-	-	Bacteria	7.30E-051
4228	67919	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	3.88E-035
4254	68566	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	1.34E-041
4328	70401	-	Bacilli	Firmicutes	Bacteria	5.44E-043
4328	70397	-	-	-	Bacteria	9.67E-028
4456	73724	-	Gammaproteobacteria	Proteobacteria	Bacteria	5.57E-047
448	74362	-	Betaproteobacteria	Proteobacteria	Bacteria	1.35E-027
448	74364	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	5.38E-015
4637	78344	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	6.59E-051
4637	78343	Brucella melitensis	Alphaproteobacteria	Proteobacteria	Bacteria	1.11E-069
4661	78923	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	7.70E-092
476	81457	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.69E-052
4811	82775	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	1.70E-114
4861	84010	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	2.50E-028
4925	85574	-	-	Proteobacteria	Bacteria	5.16E-070
4945	85969	Listeria monocytogenes	Bacilli	Firmicutes	Bacteria	1.42E-061
4958	86267	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	2.56E-012
4958	86270	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	2.30E-045
4990	87029	-	Alphaproteobacteria	Proteobacteria	Bacteria	4.31E-031

5037	88162	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	3.16E-095
505	88487	-	Bacilli	Firmicutes	Bacteria	9.66E-017
51	89710	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.07E-054
5119	90184	Brucella melitensis	Alphaproteobacteria	Proteobacteria	Bacteria	6.02E-057
5119	90189	Sinorhizobium meliloti	Alphaproteobacteria	Proteobacteria	Bacteria	3.04E-036
5214	92590	Streptococcus pneumoniae	Bacilli	Firmicutes	Bacteria	8.40E-062
5214	92586	Streptococcus pneumoniae	Bacilli	Firmicutes	Bacteria	5.35E-037
5230	93019	-	-	-	Bacteria	3.26E-064
5328	95550	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	1.05E-074
5389	97045	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	2.31E-058
5389	97049	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	1.99E-013
5389	97050	-	Bacilli	Firmicutes	Bacteria	4.35E-039
5421	97848	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	2.37E-069
5500	99686	-	Gammaproteobacteria	Proteobacteria	Bacteria	3.54E-020
5500	99688	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	1.26E-016
5500	99685	-	Actinobacteria (class)	Actinobacteria	Bacteria	1.97E-020
5500	99682	-	-	-	Bacteria	1.02E-015
5500	99690	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	1.45E-041
5505	99791	Sinorhizobium meliloti	Alphaproteobacteria	Proteobacteria	Bacteria	5.14E-030
5530	100360	-	Betaproteobacteria	Proteobacteria	Bacteria	4.14E-021
5565	101233	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	7.52E-020
5568	101297	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	5.48E-118
5685	104205	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	1.99E-056
5685	104201	-	-	-	Bacteria	3.65E-025
5693	104402	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.33E-082
5693	104396	Escherichia coli	Gammaproteobacteria	Proteobacteria	Bacteria	1.89E-016
5770	106191	-	Actinobacteria (class)	Actinobacteria	Bacteria	9.47E-016
5770	106189	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.09E-097
5805	107014	-	-	Firmicutes	Bacteria	8.07E-026
586	108260	-	-	-	Bacteria	2.74E-033
586	108261	Chlamydophila pneumoniae	Chlamydiae (class)	Chlamydiae	Bacteria	3.36E-026
6005	111803	-	Bacilli	Firmicutes	Bacteria	4.38E-078
6005	111805	Listeria innocua	Bacilli	Firmicutes	Bacteria	8.51E-028
6327	120020	-	-	-	Bacteria	2.18E-030
6327	120019	-	-	-	Bacteria	1.10E-016
6327	120015	-	-	-	Bacteria	3.25E-021
6410	122384	-	-	Proteobacteria	Bacteria	4.19E-026
6410	122388	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	5.42E-040
6434	123077	-	Alphaproteobacteria	Proteobacteria	Bacteria	2.01E-121
6493	124884	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	3.21E-023
7054	142094	-	-	-	Bacteria	6.56E-051
7054	142095	-	Bacilli	Firmicutes	Bacteria	1.35E-020
7054	142096	-	-	-	Bacteria	5.39E-012

7116	143936	-	Gammaproteobacteria	Proteobacteria	Bacteria	2.54E-084
7116	143939	-	Gammaproteobacteria	Proteobacteria	Bacteria	2.20E-018
7116	143930	-	Gammaproteobacteria	Proteobacteria	Bacteria	2.81E-023
7116	143929	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	1.37E-023
7116	143938	-	Gammaproteobacteria	Proteobacteria	Bacteria	2.88E-045
7116	143940	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	3.89E-033
717	145447	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	3.53E-068
7172	145514	-	Actinobacteria (class)	Actinobacteria	Bacteria	6.59E-014
7172	145508	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	1.53E-069
7172	145513	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.85E-046
7204	146507	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	3.13E-082
7279	148727	-	Alphaproteobacteria	Proteobacteria	Bacteria	1.49E-022
7295	149280	Aeropyrum pernix	Thermoprotei	Crenarchaeota	Archaea	3.63E-096
7308	149705	-	Thermococci	Euryarchaeota	Archaea	5.59E-116
7370	151576	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	5.11E-068
7390	152142	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.58E-050
7390	152138	Fusobacterium nucleatum	Fusobacteria (class)	Fusobacteria	Bacteria	4.67E-140
7422	153036	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	5.97E-092
7422	153033	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	5.76E-132
7476	154580	Brucella melitensis	Alphaproteobacteria	Proteobacteria	Bacteria	2.83E-108
7497	155113	-	-	-	Bacteria	8.04E-045
7550	156698	-	-	-	Bacteria	6.07E-015
7709	161776	-	Betaproteobacteria	Proteobacteria	Bacteria	1.36E-027
7709	161775	Vibrio cholerae	Gammaproteobacteria	Proteobacteria	Bacteria	3.82E-036
7709	161769	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	3.59E-065
7744	163075	-	Bacilli	Firmicutes	Bacteria	1.21E-027
7744	163071	-	Actinobacteria (class)	Actinobacteria	Bacteria	2.95E-073
7840	166628	Methanopyrus kandleri	Methanopyri	Euryarchaeota	Archaea	7.25E-125
7948	170548	Sinorhizobium meliloti	Alphaproteobacteria	Proteobacteria	Bacteria	1.53E-020
7958	170886	-	-	Proteobacteria	Bacteria	5.42E-061
8133	176998	-	-	Firmicutes	Bacteria	2.29E-104
8166	178012	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	4.33E-136
820	179144	-	-	Proteobacteria	Bacteria	6.48E-023
820	179145	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	2.68E-075
8226	180020	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	2.75E-102
8319	183362	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	6.13E-031
8326	183643	Archaeoglobus fulgidus	Archaeoglobi	Euryarchaeota	Archaea	6.53E-056
8445	188728	-	-	Proteobacteria	Bacteria	1.14E-021
8546	192855	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.19E-128
8701	199339	Archaeoglobus fulgidus	Archaeoglobi	Euryarchaeota	Archaea	4.67E-039
8845	205752	Pseudomonas aeruginosa	Gammaproteobacteria	Proteobacteria	Bacteria	8.53E-116
8845	205745	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	5.00E-011
8885	207741	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	5.59E-100
8916	209353	Anabaena sp.	null	Cyanobacteria	Bacteria	2.36E-043

8975	212203	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	2.54E-278
9078	217244	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	3.00E-048
9078	217229	Listeria monocytogenes	Bacilli	Firmicutes	Bacteria	1.54E-046
9078	217248	-	-	Firmicutes	Bacteria	7.89E-027
9110	219037	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	3.78E-077
9110	219033	-	Betaproteobacteria	Proteobacteria	Bacteria	6.12E-066
9146	221211	Pyrococcus horikoshii	Thermococci	Euryarchaeota	Archaea	4.59E-022
9168	222453	-	-	-	Bacteria	1.15E-173
9216	225480	Neisseria meningitidis	Betaproteobacteria	Proteobacteria	Bacteria	2.73E-034
9252	227854	-	-	Firmicutes	Bacteria	2.18E-029
9287	230018	-	-	-	Bacteria	7.58E-018
9287	230024	-	-	Firmicutes	Bacteria	6.20E-041
9317	232122	Lactococcus lactis	Bacilli	Firmicutes	Bacteria	3.63E-081
9350	234461	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	3.59E-053
9403	239269	-	-	Proteobacteria	Bacteria	6.82E-035
9403	239283	-	Betaproteobacteria	Proteobacteria	Bacteria	9.04E-078
9403	239279	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	4.15E-030
9403	239285	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	7.49E-115
9403	239294	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.12E-061
9403	239265	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	3.62E-069
9403	239274	-	-	Proteobacteria	Bacteria	5.67E-056
9403	239291	-	-	Proteobacteria	Bacteria	5.39E-056
9429	242173	Listeria innocua	Bacilli	Firmicutes	Bacteria	2.25E-094
9444	243686	Ralstonia solanacearum	Betaproteobacteria	Proteobacteria	Bacteria	1.53E-238
9470	246285	Haemophilus influenzae	Gammaproteobacteria	Proteobacteria	Bacteria	2.23E-106
9483	247606	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	1.15E-062
9512	251629	Escherichia coli	Gammaproteobacteria	Proteobacteria	Bacteria	1.42E-096
9566	258356	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	2.52E-046
9566	258353	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.47E-079
9581	260685	Streptococcus pneumoniae	Bacilli	Firmicutes	Bacteria	2.01E-139
9587	261721	Synechocystis sp. PCC 6803	null	Cyanobacteria	Bacteria	1.44E-088
9603	263503	-	Gammaproteobacteria	Proteobacteria	Bacteria	8.13E-094
962	264674	Xylella fastidiosa	Gammaproteobacteria	Proteobacteria	Bacteria	1.94E-094
9620	264756	Mesorhizobium loti	Alphaproteobacteria	Proteobacteria	Bacteria	2.61E-235
9647	266695	Bacillus subtilis	Bacilli	Firmicutes	Bacteria	2.65E-276
9647	266595	-	-	Proteobacteria	Bacteria	1.07E-130
9650	267176	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	2.75E-302
9683	269911	Staphylococcus aureus	Bacilli	Firmicutes	Bacteria	3.16E-080
9683	269909	-	Bacilli	Firmicutes	Bacteria	1.74E-053
9725	275607	Borrelia sp. NE581	Spirochaetes (class)	Spirochaetes	Bacteria	6.36E-142
9729	276826	Helicobacter pylori	Epsilonproteobacteria	Proteobacteria	Bacteria	4.06E-075
9733	278049	Bacillus subtilis	Bacilli	Firmicutes	Bacteria	2.35E-139

9735	278778	Corynebacterium glutamicum	Actinobacteria (class)	Actinobacteria	Bacteria	5.01E-054
9735	278775	-	-	-	Bacteria	8.12E-061
9735	278772	-	Bacilli	Firmicutes	Bacteria	5.15E-026
9735	278809	Aquifex aeolicus	Aquificae (class)	Aquificae	Bacteria	1.55E-037
9735	278827	Listeria monocytogenes	Bacilli	Firmicutes	Bacteria	1.34E-086
9735	278815	Brucella melitensis	Alphaproteobacteria	Proteobacteria	Bacteria	3.49E-120
9735	278823	-	-	-	Bacteria	1.22E-046
9735	278791	-	-	-	Bacteria	4.52E-054
9735	278825	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	2.99E-069
9735	278820	-	-	-	Bacteria	6.44E-040
9735	278794	Listeria innocua	Bacilli	Firmicutes	Bacteria	1.06E-029
9735	278783	-	Bacilli	Firmicutes	Bacteria	7.28E-066
9735	278788	-	Bacilli	Firmicutes	Bacteria	8.60E-042
9735	278773	-	Alphaproteobacteria	Proteobacteria	Bacteria	1.75E-074
9735	278777	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	1.54E-027
9739	279671	Bacillus subtilis	Bacilli	Firmicutes	Bacteria	2.09E-026
9740	279894	-	-	-	Bacteria	4.32E-087
9745	280660	-	-	-	Bacteria	3.87E-102
9745	280650	Deinococcus radiodurans	Deinococci	Deinococcus-Thermus	Bacteria	1.69E-220
9756	284962	Borrelia sp. NE581	Spirochaetes (class)	Spirochaetes	Bacteria	8.20E-168
9763	286079	Pyrococcus horikoshii	Thermococci	Euryarchaeota	Archaea	1.73E-158
9767	286476	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	7.01E-093
9774	287240	-	-	-	Bacteria	3.69E-059
9774	287218	-	-	-	Bacteria	4.08E-059
9774	287221	-	-	-	Bacteria	4.41E-057
9780	292521	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	2.09E-307
9780	291725	Aquifex aeolicus	Aquificae (class)	Aquificae	Bacteria	4.29E-084
9780	291718	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	1.71E-039
9780	291716	Synechocystis sp. PCC 6803	null	Cyanobacteria	Bacteria	7.04E-049
9786	296768	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	5.35E-059
9786	296738	-	-	Firmicutes	Bacteria	3.91E-066
9786	296744	-	Bacilli	Firmicutes	Bacteria	3.18E-031
9786	296731	Vibrio cholerae	Gammaproteobacteria	Proteobacteria	Bacteria	8.34E-021
9786	296733	-	-	-	Bacteria	7.76E-046
9786	296729	-	-	Firmicutes	Bacteria	1.28E-045
9786	296742	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	7.36E-041
9786	296770	Listeria monocytogenes	Bacilli	Firmicutes	Bacteria	3.18E-083
9786	297507	-	-	Firmicutes	Bacteria	2.09E-173
9786	296752	Chlamydia trachomatis	Chlamydiae (class)	Chlamydiae	Bacteria	1.79E-109
9786	296713	-	-	-	Bacteria	4.83E-065
9786	296747	Staphylococcus aureus	Bacilli	Firmicutes	Bacteria	1.58E-041
9786	296656	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	1.48E-070
9786	296654	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	1.36E-054
9786	296743	-	Actinobacteria (class)	Actinobacteria	Bacteria	9.54E-050
9786	296723	-	-	-	Bacteria	6.83E-026
9786	296766	-	Alphaproteobacteria	Proteobacteria	Bacteria	4.35E-045
9786	296745	Bacillus halodurans	Bacilli	Firmicutes	Bacteria	6.01E-049
9786	296760	-	-	-	Bacteria	2.45E-036
9786	296699	Thermotoga maritima	Thermotogae (class)	Thermotogae	Bacteria	4.96E-048
9786	296697	-	-	-	Bacteria	2.55E-055
9786	296727	-	-	-	Bacteria	1.22E-073

982	297908	Pseudomonas aeruginosa	Gammaproteobacteria	Proteobacteria	Bacteria	6.59E-040
994	298143	Clostridium acetobutylicum	Clostridia	Firmicutes	Bacteria	5.89E-116

Tabelle C.2.: Ergebnis: Taxonomische Zuordnung der ORFs nach der 'Beste Homologe'-Methode

C.2. Referenzbaum-Methode

Contig-Nr.	Verwandte
39	Thermoanaerobacter tengcongensis, Deinococci
51	Fibrobacter succinogenes, Gemmata obscuriglobus
107	Gemmata obscuriglobus, Fibrobacter succinogenes
145	Acidobacteria, Deltaproteobacteria
177	Archaea
304	Archaea
310	Acidobacterium capsulatum, Acidobacteria, Geobacter sulfurreducens
448	Acidobacterium capsulatum
476	Acidobacterium capsulatum
505	Thermotoga maritima, Aquifex aeolicus, Desulfovibrio vulgaris
717	Clostridium perfringens
982	Acidobacterium capsulatum, Desulfovibrio vulgaris
994	Archaea
1022	Wolinella succinogenes, Campylobacter jejun, Helicobacter hepaticus
1038	Betaproteobacteria, Burkholderiales
1079	Caulobacter vibrioide, Acidobacteria
1192	Mycoplasma genitalium, Mycoplasma pneumoniae
1262	Gemmata obscuriglobus, Fibrobacter succinogenes
1429	Fibrobacter succinogenes
1431	Fibrobacter succinogenes, Thermoanaerobacter tengcongensis, Bacteria
1439	Gemmata obscuriglobus, Fibrobacter succinogenes
1503	Fibrobacter succinogenes
1528	Bacteroides thetaiotaomicron, Porphyromonas gingivalis
1540	Gemmata obscuriglobus, Fibrobacter succinogenes
1835	Fibrobacter succinogenes, Gemmata obscuriglobus
1866	Thermoanaerobacter tengcongensis
1916	Firmicutes
1935	Wolbachia sp. wMel, Gemmata obscuriglobus
1958	Fibrobacter succinogenes, Dehalococcoides ethenogenes
1993	Archaea
2231	Fibrobacter succinogenes, Gemmata obscuriglobus
2232	Gemmata obscuriglobus, Fibrobacter succinogenes
2299	Acidobacterium capsulatum, Solibacter usitatus, Gemmata obscuriglobus
2500	Thermoanaerobacter tengcongensis, Clostridium acetobutylicum
2514	Fibrobacter succinogenes, Gemmata obscuriglobus
2516	Archaea
2632	Archaea
2682	Gemmata obscuriglobus, Fibrobacter succinogenes
2837	Mycoplasma genitalium
2890	Fibrobacter succinogenes
2945	Gemmata obscuriglobus
3173	Archaea
3197	Fibrobacter succinogenes

3622	Archaea
3643	Giardia lamblia
3668	Archaea
3793	Archaea
3803	Fibrobacter succinogenes
4170	Gemmata obscuriglobus, Treponema pallidum
4254	Acidobacteria, Helicobacteraceae
4456	Thermoanaerobacter tengcongensis
4661	Coxiella burnetii, Nitrosomonas europaea, Acidobacteria
4811	Bacteria
4861	Archaea
4925	Acidobacterium capsulatum, Bdellovibrio bacteriovorus, Solibacter usitatus
4945	Fibrobacter succinogenes, G. obscuriglobus, Thermoanaerobacter tengcongensis
4958	Clostridium perfringens, Dehalococcoides ethenogenes
4990	Acidobacteria, Deltaproteobacteria, Acidobacterium capsulatum
5037	Archaea
5119	Acidobacterium capsulatum, Fibrobacter succinogenes
5230	Fibrobacter succinogenes
5328	Bacteria, Geobacter sulfurreducens
5421	Archaea
5565	Rhizobiales
5568	Bradyrhizobiaceae, Rhodopseudomonas palustri, Rhizobiaceae
5693	Treponema pallidum, Gemmata obscuriglobus
5805	Bacteroidale, Chlorobaculum tepidum, Thermoanaerobacter tengcongensis
6410	Acidobacteria, Acidobacterium capsulatum
6434	Chlorobaculum tepidum, Fibrobacter succinogenes
6493	Acidobacterium capsulatum, Fibrobacter succinogenes
7204	Acidobacterium capsulatum, Bdellovibrio bacteriovorus
7279	Acidobacterium capsulatum, Geobacter sulfurreducens, Solibacter usitatus
7295	Archaea
7308	Fibrobacter succinogenes
7370	Acidobacterium capsulatum, Solibacter usitatus
7422	Archaea
7476	Bradyrhizobium japonicum, Mesorhizobium loti
7497	Fibrobacter succinogenes, Acidobacterium capsulatum
7948	Fusobacterium nucleatum, Thermotoga maritima, Acidobacterium capsulatum
7958	Fibrobacter succinogenes
8133	Fibrobacter succinogenes, Gemmata obscuriglobus
8226	Fibrobacter succinogenes
8319	Fibrobacter succinogenes, Thermoanaerobacter tengcongensis
8546	Bacteria
8701	Eukaryota
8885	Wolbachia sp. wMel, Bdellovibrio bacteriovorus, Dehalococcoides ethenogenes
8916	Eukaryota
8975	Archaea
9146	Eukaryota
9168	Fibrobacter succinogenes, Gemmata obscuriglobus
9216	Geobacter sulfurreducens, Desulfovibrio vulgaris
9252	Thermoanaerobacter tengcongensis, Clostridium
9287	Archaea
9317	Gemmata obscuriglobus
9350	Bacteroidales
9429	Gemmata obscuriglobus, Borrelia burgdorferi
9444	Clostridia, Thermoanaerobacter tengcongensis
9470	Clostridium perfringens
9483	Archaea

9512	Mycoplasma genitalium, Mycoplasma pneumoniae
9566	Archaea
9581	Fibrobacter succinogenes, Gemmata obscuriglobus
9587	Fusobacterium nucleatum, Dehalococcoides ethenogenes
9603	Mycoplasma genitalium , Mycoplasma pneumoniae
9620	Geobacter sulfurreducens, Thermoanaerobacter tengcongensis, Bradyrhizobia
9650	Archaea
9725	Archaea
9733	Thermoanaerobacter tengcongensis
9740	Dehalococcoides ethenogenes, Gemmata obscuriglobus
9745	Fibrobacter succinogenes
9756	Wolbachia sp. wMel, Buchnera aphidicola
9763	Archaea
9767	Mycoplasma pulmonis, Borrelia burgdorferi
9777	Archaea

Tabelle C.3.: Ergebnis: Zuordnung der Contigs nach der Referenzbaum-Methode

D. Quellcode

Auf der beiliegen CD befindet sich der Quellcode, der im Rahmen der Diplomarbeit erstellt wurde (sensible Informationen wie Passwörter wurden durch '***' ersetzt).

Struktur der CD:

- SimapTaxonomyTreeBean: Quellcode zu Kapitel 3
- Anammox-Community: Quellcode zu Kapitel 4
  - Scripts: Perl- und Pythonskripte
  - src: Java-Quellcode
  - procedure.txt: Die einzelnen Arbeitsschritte im Detail
- TreeLayout: TreelayoutManager zur Verwendung als externe Bibliothek
  - src: Quellcode
    - example: Anwendungsbeispiel
  - treelayout.jar: Binary Jar