Lehrstuhl für Mikrobiologie der
Technischen Universität München

ARB:
Entwicklung eines Programmsystems zur Erfassung, Verwaltung und Auswertung von Nuklein- und Aminosäuresequenzen

Oliver Strunk

Vollständiger Abdruck der von der Fakultät für Chemie der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Natruwissensccchaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Dr. Adelbert Bacher
Prüfer der Dissertation	1. Univ.-Prof. Dr. Karl-Heinz Schleifer
	2. Univ.-Prof. (komm.) Dr. Thomas Ludwig
	3. Univ.-Prof. Dr. Arndt von Haeseler, Universität Leibzig (schriftliche Beurteilung)

Die Dissertation wurda am 18.01.2001 bei der Technischen Universität München eingereicht und durch die Fakultät für Chemie am 01.10.2001 angenommen.

Eine kleine Geschichte ...

Eine der entscheidenden Erfahrungen, die auch ich bei dieser Arbeit gemacht hatte, hat einmal Ross Overbeek in einer persönlichen Anekdote zusammengefaßt. Ross Overbeek ist wohl einer der besten Computerspezialisten im Bereich künstliche Intelligenz, der mir begegnet ist. Er ist seit einigen Jahren im Bereich Bioinformatik erfolgreich tätig.

Er erzählte mir folgendes:

> Es kamen immer wieder Biologen und Mikrobiologen zu mir und schwärmten davon, wie genial es doch wäre, ein Computerprogramm zu haben, mit dem man alle Fragen der Genetik lösen könnte. Ich programmierte daraufhin ein komplexes Program und als es fertig war, sagte ich ihnen, daß ich nun alle Fragen in bezug auf genetische Sequenzen lösen könne und die ersten Fragen erwarte. Aber plötzlich fielen den Biologen keine Fragen mehr ein.

Dies ist eine seltsame, aber dennoch wahre Geschichte. Das Problem, das Ross damals wohl hatte, war, daß sich die Biologen ein ideales Programm mit idealen Daten vorstellten. Dabei war weniger das Programm das Problem als die Daten. Auch heute noch schwebt den Biologen eine ideale Datenbank vor. Wir Informatiker können den Biologen nicht das perfekte Programm liefern, wir sollten uns darauf konzentrieren, ein solides System zu schaffen, das Biologen einen Einblick in ihre Daten gibt und sie dabei gezielt auf die Unzulänglichkeiten und Löcher in ihren Datensätzen hinweist.

Danksagung

Ich möchte mich bei allen Personen herzlich bedanken, die mir nicht nur bei dieser Arbeit geholfen haben, sondern ohne die diese Arbeit überhaupt nicht möglich gewesen wäre.

Zum ersten bedanke ich mich bei meinen beiden Doktorvätern Prof. Schleifer und Prof. Bode, die mir ein motivierendes und kreatives Arbeitsumfeld geschaffen und mich in vielfältiger Weise unterstützt haben.

Unser Team wurde professionell von Dr. Thomas Ludwig und Dr. Michael Lenke unterstützt.

Der entscheidende Dank gebührt natürlich Dr. Wolfgang Ludwig, der mit seinen Ideen und Kommentaren unser Team immer wieder zu Höchstleistungen anspornte und der als biologischer Anwender, bzw. Beta-Tester, uns das nötige Feedback gab. Mit seiner wissenschaftlichen Neugier war er uns immer ein Vorbild. Seine großen Erfahrungen im Bereich der Biologie und sein Forscherdrang beflügelten uns, sein hohes Niveau auch in der Umsetzung seiner Ideen zu halten.

Die Ideen kamen aber nicht nur aus dem eigenen Labor, sondern Kontake zu amerikanischen Universitäten ermöglichten den Gedankenaustausch unter anderem mit Dr. Niels Larsen und Dr. Ross Overbeek. Diese beiden Informatiker konnten unserem Projekt viele kleine, aber entscheidende Ideen beisteuern. Sie waren gewissermaßen neutrale Beobachter, die unserem Team erlaubten, unsere eigene Arbeit mit globalen Augen zu sehen und so auf eine global einsetzbares Ergebnis hinzuarbeiten.

Zusätzlich möchte ich natürlich meiner Freundin Stephanie für all ihre Hilfe danken, indem sie mich selbst bei in tiefster Nacht bei der Arbeit seelisch und mit einer Apfelsaftschorle unterstützt hat, und ihrer Schwester Cordula Mraz zu der gewissenhaftesten Überarbeitung eines Textes, die man sich vorstellen kann.

Nur durch eine solche Kombination von Personen konnte eine solche Arbeit angepackt und zu einem erfolgreichen Ende geführt werden.

Inhaltsverzeichnis

I Übersicht

I.1 Kurzübersicht (Deutsch)

In der heutigen Zeit hat sich die vergleichende Sequenzanalyse der ribosomalen RNS zur Rekonstruktion des phylogenetischen Stammbaumes von Mikroorganismen weitgehend durchgesetzt. Weiterhin hat sich aufgrund moderner automatischer Sequenziertechniken die für diesen Zweck verwendbare Sequenzdatenmenge exponentiell vervielfältigt. Die Aufgabe dieser Arbeit bestand nun darin, ein Gesamtsystem zur phylogenetischen Analyse und Verwaltung dieser großen Sequenzdatenmengen zu schaffen. Dabei erlaubt ein in dieser Arbeit entwickeltes Integrationskonzept, Hunderte von existierenden Algorithmen und Programmen in ein einheitliches Programmpaket (ARB) mit einer einheitlichen grafischen Benutzeroberfläche zu integrieren. Eine speziell optimierte Datenbank ermöglicht die Datenströme zwischen diesen Algorithmen auch über Rechnergrenzen hinweg effektiv und effizient zu steuern. Diese Datenbank wurde so konzipiert, daß sowohl Sequenzdaten, Benutzerdaten, wie auch phylogenetische Informationen zusammen gespeichert werden können. Diese Kombination verschiedener Datentypen war auch die Basis für die Entwicklung und den Einsatz einer ganzen Reihe von Analysealgorithmen, von denen einige exemplarisch in der schriftlichen Arbeit vorgestellt werden: Die Berechnung eines für einen Organismus charakteristischen Sequenzabschnittes, die schnelle visuelle Kontrolle von Sequenzen auf Sequenzierfehler und die Bewertung der statistische Signifikanz von großen phylogenetischen Bäumen. Den Biologen konnte somit ein wertvolles Werkzeug in die Hand gegeben werden, das es ihnen erlaubt, ihre Daten schnell, einfach und effektiv zu analysieren.

I.2 Abstract (English)

Today comparative sequence analysis has been widely established as a means to reconstruct the phylogenetic tree of micro-organisms. Also, modern automatic sequencing methods allow for an exponential growth of sequence data. The task of this project was to design and implement an integrated solution to handle and analyze a huge amount of gene sequence data. By using a new integration method, hundreds of existing algorithms and programms have been integrated into a single package (ARB) with one graphical user interface. With the help of a highly optimized database system, sequence data, user data as well as phylogenetic tree data can be transferred efficiently and effectively between the different modules. This combination of different data types in a single environment is the basis for a variety of new applications. A selection of those applications is described in this thesis: The (probe) search to calculate a characteristic and identifying sequence part for a given microorganim, a fast visual verification tool to find sequence errors and the analysis of the statistical significance of any given big tree topology. As a result, a valuable package was developed, which helps microbiologists all over the world to analyse their databases in an easy and effective manner.

I.3 Übersicht

Schon seit den Anfängen der systematischen und wissenschaftlich betriebenen Artenkunde ist es ein Bestreben ihrer Verteter, die verschiedenen Lebewesen in einem Klassifikationssystem anzuordnen. Während bei Tieren ihre Klassifikation aufgrund äusserer Merkmale (morphologische Ontologie) wie Knochenbau oder innerer Organe relativ leicht fällt, war es bei den Bakterien bis zur Mitte dieses Jahrhunderts sowohl aufgund ihrer geringen Größe als auch bedingt durch fehlende Klassifizierungsmerkmale nicht nur schwierig, sondern schlichtweg unmöglich, einen Stammbaum zu entwickeln, der der tatsächlichen Verwandschaft (Phylogenie) der Bakterien nahe kam. Erschwerend kam bei den Bakterien hinzu, daß es keine fossilen Funde gibt, die einen Blick in die Vergangenheit erlaubt hätten. Erst mit der Entdeckung der DNS[1] durch Watson und Crick sowie der Idee von Kandl, Zucker und Pauling [Zukerkandl 65], daß sich die Verwandtschaft aller Lebewesen in der Ähnlichkeit ihrer genetischen Sequenzen in eben dieser DNS widerspiegelt, war ein Weg entdeckt, der es zum ersten mal erlaubte, einen möglichen Stammbaum der Bakterien zu ermitteln.

Um nun ausgehend von einer kleinen Menge an Sequenzen und einem statistischen Modell der Evolution einen Stammbaum automatisch 'berechnen' zu können, wurden in der Vergangenheit weltweit eine Vielzahl unterschiedlicher Algorithmen und noch mehr meist singuläre Programme entwickelt. Dabei war der Stammbaum nur ein Ergebnis, das zwar publiziert wurde, jedoch selten für die weitere Analyse der Sequenzen eingesetzt wurde.

So erkannte W. Ludwig aufgrund langjährige Erfahrung die Bedeutung des Stammbaumes für die Analyse der ihm zugrundeliegenden Sequenzen. Diese Idee war es auch, auf der diese Arbeit basiert und damit zur Entwicklung des Softwarepaketes ARB geführt hat:

So stellte sich z.B. bereits in den ersten Wochen dieses Projektes heraus, daß die traditionelle Sicht, Sequenzen und phylogenetische Bäume als unabhängige Objekte zu betrachten, sowohl zu unflexibel als auch zu ineffizient war, um den wachsenden Anforderung der Biologie auch in Zukunft zu genügen. So wurde bereits in dieser Frühphase festgelegt, in diesem Projekt Sequenzen und zugehörigen Stammbaum als untrennbare Einheit zu betrachten. Durch diese Verbindung konnten neue Verfahren entwickelt werden, die neue Einblicke in die Evolution auf der Ebene der Sequenzen der Erbanlagen in der DNS eröffnen. Außerdem hat diese neuartige Betrachtungsweise zu Programmen geführt, die mit einer noch nie gekannten Eleganz die interaktive Sequenzeingabe und deren Überprüfung erlaubt. Aus praktischen Gründen ist dies wohl der einzige Weg, der es ermöglicht, mit mehreren Tausend Sequenzen zu arbeiten und diese zu verwalten.

Da nun die Möglichkeit gegeben war, viele DNS-Sequenzen auf einmal zu bearbeiten, begann W. Ludwig, in Reaktion auf die oben genannten Programme, eine große Anzahl dieser Sequenzen zu sammeln. Ergebnis dieser unermüdlichen Tätigkeit war eine Menge an Daten, die nicht mehr von traditioneller Software zu bewältigen war. Einerseits sprengte diese Datenflut deren handhabbare Datensätze bei weitem, andererseits waren die dort eingesetzten Algorithmen von zu großer Komplexität, um in akzeptabler Zeit Ergebnisse berechnen zu können. Infolgedessen mußte eine ganze Reihe existierender Algorithmen entscheidend geändert werden. Als Ergebnis dieser Änderungstätigkeiten entstanden z.B. Methoden, die die Anwendung genetischer Algorithmen bei der Stammbaumberechnung erlauben.

[1] Desoxyribonukleinsäure

Durch die enorme Anzahl der von W.Ludwig gesammelten Daten wurde jedoch nicht alleine das Problem der mangelnden Leistungsfähigkeit der existierenden Algorithmen aufgeworfen, auch die traditionelle Art der Datenverwaltung mußte vor dieser Datenflut kapitulieren. Deshalb mußte ein Integrationskonzept entwickelt werden, das es erlaubt, Hunderte von Algorithmen in ein Programmpaket (**ARB**) zu integrieren, und dabei die Datenströme zwischen diesen auch über Rechnergrenzen hinweg effektiv und effizient zu steuern.

Damit konnte ein Gesamtsystem zur phylogenetischen Analyse und Verwaltung großer Sequenzdatenmengen geschaffen werden, das aufgrund einer flexiblen inneren Struktur weitreichende Möglichkeiten bietet, die noch bei weitem nicht ausgeschöpft sind und auch in Zukunft den Biologen wertvolle Werkzeuge in die Hand geben werden, ihre Daten schnell und einfach zu analysieren.

Diese Gliederung dieser Arbeit orientiert sich im wesentlichen an der strukturierten Vorgehensweise der Mikrobiologen bei der Analyse und Auswertung der Sequenzdaten. Es wurde dabei versucht, die verschiedenen Einzelthemen möglichst als unabhängige Teilgebiete darzustellen, dennoch sind diese Teilgebiete eng miteinander verknüpft und erst durch die gesamtheitliche Lösung konnte ein elaganter Einblick in die Zusammenhänge zwischen Evolution, Stammbaum und Sequenzen geschaffen werden.

II Bemerkungen des Autors

Ziel dieser Arbeit war es nicht so sehr, eine **einzige** neue Technik zu entwickeln, als vielmehr existierende Programme in einem benutzerfreundlichen Gesamtpaket zu integrieren, fehlende Teile zu ergänzen und das gesamte Paket auf seine Praxistauglichkeit zu testen und zu optimieren. Erst durch Kombination einer **Vielfalt** bearbeiteter Themen konnte die **notwendige Einsicht** in die Schwachstellen konventioneller Algorithmen und Lösungsansätze erreicht werden. Bei einer vollständigen wissenschaftlichen Analyse eines Einzelthemas hätte sich das Ergebnis nur in die bestehenden Teillösungen eingereiht. Es ist der ganzheitliche Ansatz, der ARB zu einem Novum auf dem Gebiet der Sequenzanalyse und Stammbaumberechnung macht und ihm den Erfolg sichert. Um das Lesen dieser Arbeit zu erleichtern , wurde versucht die entsprechenden Themen möglichst unabhängig darzustellen

Mit den herkömmlichen Methoden stand bisher niemandem auch nur annähernd eine derartige Menge von phylogenetisch organisierten Sequenzen und Analysewerkzeugen auf einmal zur Verfügung. Die vorliegende Arbeit öffnet das Tor zu einer neuen Dimension auf diesem Gebiet der Biologie. Aus diesem Vorstoß resultieren natürlich auch neue Probleme, zu deren Lösung das Program ARB unterstützend herangezogen werden kann.

Es wurden mehrere hunderte von kleinsten, kleinen und großen Algorithmen entwickelt, umgesetzt, getestet, verbessert, statistisch ausgewertet und dokumentiert. Es würde den Rahmen sprengen, zu versuchen, alle in dieser Richung unternommenen Arbeiten hier schriftlich niederzulegen. Hier sollen nur die wichtigsten zentralen Themen konzentriert wiedergegeben werden. Themen, die absolut notwendig für den Erfolg dieses und eines nachfolgenden integrierten Softwarepaket für Biologen sind. Themen, die sicherlich zusammengenommen nur etwa 30% der geleisteten Arbeit ausmachen.

Durch die intensive Nachfrage nach ARB seitens der Benutzer bereits in der Anfangsphase entstand für das Gesamtprojekt sehr früh der Druck, alle Teilprojekte in einem integrativen Ansatz zu einem praxisnahen und doch wissenschaftlichen Erfolg zu führen. Nur durch Entwicklung leistungsfähiger Konzepte zur Automatisierung wiederkehrender Tätigkeiten, zur sicheren Integration neuer Algorithmen und konsequentes Vorausplanen zukünftiger Anforderungen konnte es gelingen, ein so umfangreiches Programm mit so wenigen zum Teil nicht vollständig ausgebildeten Mitarbeitern in so kurzer Zeit der Öffentlichkeit vorzustellen.[1]

[1] siehe auch **?**

III Begriffe

- **Base**
  Grundbaustein der Erbinformation. Ketten dieser Nucleotiden bezeichnet man als:
- **Sequenzen**
  Zeichenkette, die die Abfolge der Basen in einem Ausschnitt des Genoms beschreibt.
- **sequenzieren**
  Bestimmung der Basenabfolge eines Genes im Labor
- **Alignment**
  Horizontale und vertikale Ausrichtung von Sequenzen mittels Füllzeichen, so daß homologe Basen untereinander zu liegen kommen.
- **alignen**
  Fügt man in mehrere verwandte Sequenzen so Füllzeichen ein, daß man ein Alignment erhält, nennt man diesen Vorgang alignen.
- **Komplementäre Basen**
  Zwei Basen a und b heißen komplementär, wenn die eine Adenin und die andere Thymin oder die eine eine Guanin und die andere ein Cytosin ist. Solche Basenpaare haben die Eigenschaft, Wasserstoffbrücken zwischen den Partnern aufzubauen.
- **Helix**
  Struktur einer Sequenz, die sich durch das Verbinden zweier antiparalleler komplementärer Sequenzabschnitte bildet.
- **Heuristik**
  ist ein Algorithmus, der darauf verzichtet, immer die korrekte Lösung zu berechnen, dafür aber in praktikabler Zeit eine brauchbare Lösung liefert.
- **Homologe Basen**
  Basen, die aus derselben 'Ur'-Base hervorgegangen sind.
- **Mismatch**
  Nicht paarende oder fehlende Basen in Helices.
- **NP**—vollständig
  Bezeichnung der exponentiellen Komplexität eines Algorithmus. Das bedeuted, das mit steigender Problemgröße der Aufwand zur Lösung dieses Problems exponentiel steigt. In der Praxis sind vieler dieser Probleme aus rechenzeitgründen nicht lösbar.
- **Primärstruktur**
  Die Abfolge der Basen in einer Sequenz.
- **Phylogenie**
  Der Stammbaum der Lebewesen
- **Pfad**
  Der Weg in einem Stammbaum von einem Knoten zu einem anderen Knoten.
- **Sekundärstruktur**
  Zweidimensionale Struktur einer Sequenz, die sich durch das Zusammenfalten

der Sequenz aufgrund der Verbindung antiparalleler, komplementärer Sequenzabschnitte ergibt.

- **Distanz zweier homologer Sequenzen**
  Die Anzahl der Mutationenen längs des Pfades
- **RDP**
  Ribosomal **D**atabase **P**roject: Eine wissenschaftliche Arbeitsgruppe in den USA, die zum Ziel hat, alle ribosomalen RNS Sequenzen zu sammeln, zu alignen und mit Hilfe des WWW allen Forschern zur Verfügung zu stellen. Dieses hehre Ziel konnte jedoch nicht optimal erreicht werden, da sie nicht mehr mit der Geschwindigkeit des Datenwachstums mithalten konnten, weil ihnen leistungsfähige Softwarewerkzeuge fehlten.
- **Hammingabstand**
  Die Summe aller Unterschiede bei einem positionsweisen Vergleich zweier Zeichenketten wird Hammingabstand genannt.

IV Molekularbiologishcer Hintergrund und phylogenetische Ziele

IV.1 Einführung

Die Erbinformation jeder Zelle ist in Form von Desoxyribonukleinsäure-Ketten (DNS) gespeichert. Mit der Möglichkeit, DNS zu sequenzieren [Hillis 90], ergaben sich für die Biologen neue Methoden der phylogenetischen Analyse der Bakterien. Mit Hilfe hoch konservierter Gene, wie zum Beispiel der ribosomalen RNS, ist es unter anderem möglich, einen Blick in die Vergangenheit zu werfen und für die Bakterien einen Stammbaum zu finden.

Dies soll an einem sehr einfachen Beispiel einmal durchgespielt werden: Dazu wird eine mögliche Evolution eines kurzen Genabschnittes an einem Beispiel näher betrachtet. Dieses Genstück besteht aus einer Folge der Basen **A**denin, **C**ytosin, **G**uanin und **T**hymin:

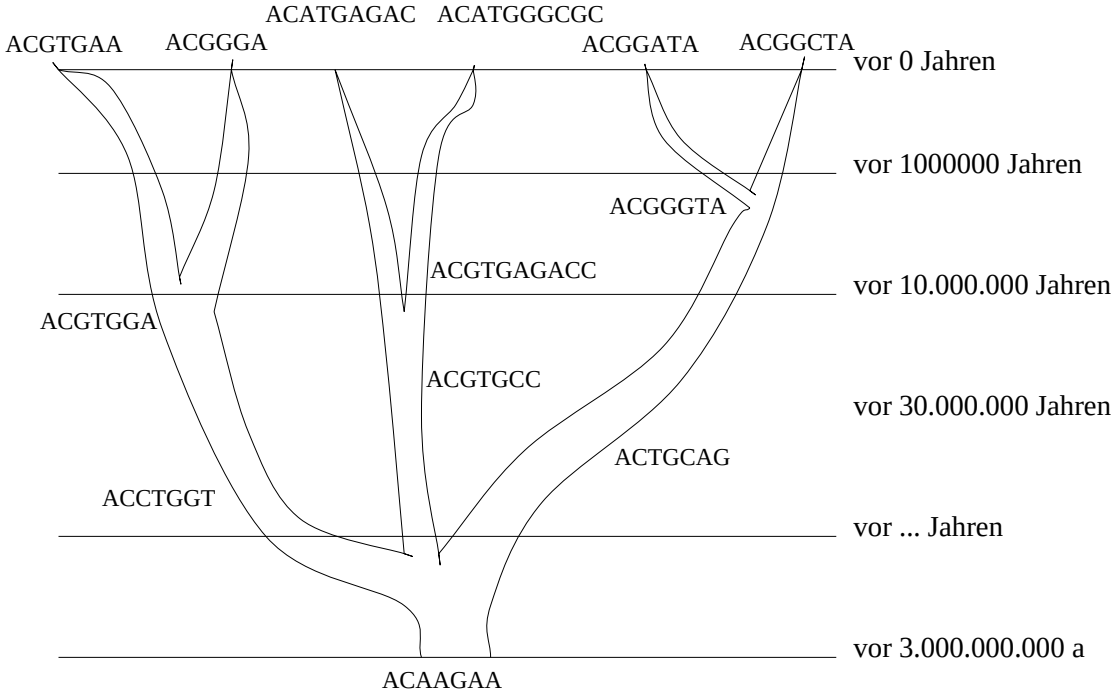


Abb. 1 Mögliche Evolution einer kurzen DNS-Sequenz

IV.1

Am unteren Ende des Baumes und damit am Beginn unserer Zeitrechung stand einmal die kurze Sequenz 'ACAAGAA', die sich im Laufe der Zeit zu den heutigen sichtbaren Genausschnitt am oberen Ende des Baumes mutiert hat. Dabei kann folgendes auftreten:

- Da sich alle Lebewesen vermehren, kann der Genausschnitt beliebig häufig kopiert werden. Jeder so entstandene Ast mutiert dann unabhängig von den anderen.
- Durch die Evolution kann es vorkommen, daß eine erfolgte Mutation wieder rückgängig gemacht wurde. So scheint die letzte Base des Beispiels bei den linken und den rechten Ästen jeweils Adenin zu sein, und man könnte dem Irrtum erliegen, daß es an dieser Position im Laufe der Entwicklung zu keiner Mutation gekommen wäre.
- Es gibt konservative und variable Abschnitte innerhalb der DNS, die Aussagen über verschiedene Zeitabschnitte der Evolution geben können. So sind die ersten beiden Basen unseres Beispiel-Gens sehr konservativ, die letzte nicht ganz so konservativ und die drittletzte ziemlich variabel. Ein Charakter heißt konservativ, wenn sein Wert sich im Laufe der Evolution relativ selten ändert.
- Da im Laufe der Evolution nicht nur einfach Basen ausgetauscht, sondern auch neue Bereiche eingebaut (**Insertion**) oder weggelassen (**Deletion**) werden, ist es notwendig, die Sequenzen auszurichten (= **alignen**), so daß die **homologen** Abschnitte wieder an der gleichen Position stehen, d.h. Sequenzteile von verschiedenen Organismen, die sich aus demselben Sequenzstück eines gemeinsamen Urgens entwickelt haben, werden an dieselbe Stelle im Alignment plaziert.

Tatsächliche Genome sind allerdings weitaus länger als in unserem Beispiel. Um ein Gefühl für die Größe existierender Genome zu bekommen, soll eine kurze Übersicht gegeben werden:

Organismus	**Nukleotidpaare**
Simian Virus	5.243
Bakteriophage X 174	5.375
Bakteriophage T7	39.936
Bakteriophage T4	180.000
Escherichia coli	4.000.000
Hefe (Saccharomyces cerevisiae)	$13.5 * 10^6$
Fruchtfliege (Drosophila melanogaster)	$160*10^6$
Seeigel (Strongylocentrotus purpuratus)	$3000*10^6$
Säugetiere	$3000*10^6$

Abb. 2 Größe von Genomen (entnommen der Diplomarbeit von Anton Ginhart)

Der Begriff Genom bezeichnet die Gesamtheit der Gene eines Organismus. Die Genome lassen sich in verschiedene Gene unterteilen, die jeweils eine unterschiedliche Funktion im

Lebewesen haben. Je nach Wichtigkeit dieser Funktion haben die entsprechenden Bereiche unterschiedliche Eigenschaften.

IV.2 Die Ribosomale RNS (rRNS)

Man könnte annehmen, daß sich eine phylogenetische Analyse mit beliebigen Teilen der Erbinformation durchführen ließe. Dies ist im Prinzip auch korrekt, jedoch müssen für eine umfassende phylogenetische Analyse gewisse Grundanforderungen an den betrachteten DNS-Ausschnitt gestellt werden, um diesen als **phylogenetischen Marker** akzeptieren zu können:

- Sie enthalten hoch konservative Abschnitte, die eine Sequenzierung und das Alignen unterstützen (**homologe Bereiche**).
- Dieser phylogenetischer Marker kommt ohne Ausnahme bei allen zu vergleichenden Lebewesen vor (**ubiquitär**).
- Sie übernehmen in allen zu vergleichenden Lebewesen dieselbe Aufgabe (**funktionelle Konstanz**).
- Die Gesamtlänge dieser Bereiche ist nicht zu kurz, um eine statistische Analyse betreiben zu können, und nicht zu lang, um nicht unnötigen Aufwand in die Analyse stecken zu müssen. Als praktikabel haben sich je nach Anwendung Längen zwischen 250 und 5000 Basen herausgestellt.

Historisch hat sich ein ganz bestimmter Bereich als gewissermaßen ideal für viele Anwendungen herausgestellt. Es handelt sich dabei um den Bereich, der für die Codierung der ribosomalen Ribonukleinsäure (RNS) zuständig ist.

Das **Ribosom** ist eine Zellkomponente des Proteinbiosyntheseapparats [czihak 84] [Alberts 83]. Dabei wird die Erbinformation mit Hilfe von Enzymen kopiert (Transkription) und das einsträngige Ergebnis (**Messenger RNS)** aus dem Zellkern zum Ribosom transportiert. Dieses liest die darin gespeicherten Informationen und baut entsprechende Aminosäureketten zusammen, die sich dann zu Eiweißen selbständig zusammenfalten.

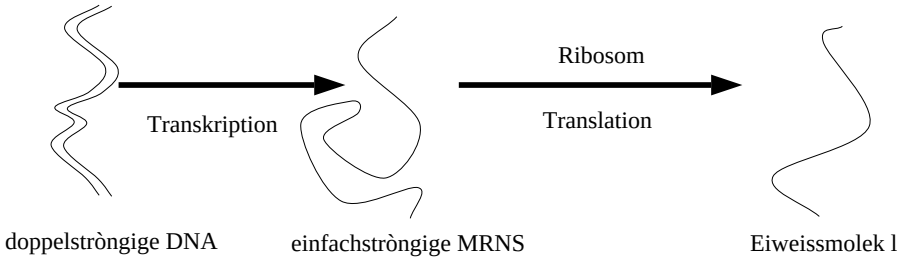


Abb. 3 Proteinbiosynthese

Das Ribosom enthält neben einer Vielzahl verschiedener Proteine auch Ribonukleinsäure—Ketten (RNS). Diese sind aus den Basen Adenin, Guanin, Cytosin und Uracil aufgebaut. Drei dieser Ketten bilden das ’Rückgrat’ des Ribosoms: die kurze **5S** (etwa 120 Basen), die **16S** (ca. 1500 B.) und die lange **23S** (ungefähr 2300 Basen) rRNS Sequenz.

Bis heute (Mitte 1999) sind etwa 16000 16S-rRNS Sequenzen bekannt. Allerdings kommen jedes Jahr etwa 8000 neue Datensätze hinzu, so daß alle im Rahmen dieses Projektes

durchgeführten Komplexitätsbetrachtungen beliebiger Algorithmen auf entsprechende Datenmengen hochgerechnet werden müssen.

Im Laufe der Evolution der Organismen werden immer wieder einzelne Basen an scheinbar zufälligen Sequenzpositionen einer 5S, 16S oder 23S-rRNS Sequenz ausgetauscht. Eine solche Mutation ist ein ganz entscheidender Eingriff in das Erbgut der einzelnen Organismen, die dazu führen kann, daß ein mutierter Organismus abstirbt. Entsprechend dem geringen Wissen über die jeweilige genaue Funktion einzelner Basen an bestimmten Positionen kann man heutzutage Analysen nur mit Hilfe statistischer Methoden durchführen. Dabei kann man allerdings auf verschiedene Beobachtungen zurückgreifen:

- Es gibt Bereiche der rRNS, die für eine Organismenspezies oder für alle Organismen vollkommen identisch sind.
- Die verschiedenen Positionen einer rRNS haben eine unterschiedliche Wichtigkeit für das Überleben des Organismus. Aus diesem Grund ergeben sich für die Basen an diesen Positionen eine scheinbar unterschiedliche Mutationsrate, weil eine nicht erfolgreiche Mutation für uns heute nicht mehr sichtbar ist.
- Eine **Transition** (Austausch der Base A mit G oder C mit T) ist häufiger als eine **Transversion**(A mit CT, C mit AG, G mit CT oder T mit AG)[1].
- Es können mitten im Gen Basen eingebaut und eingesetzt werden. Dabei kann es sich um einzelne Basen handeln, oder, wesentlich seltener, um ganze Basenketten.
- Aufgrund der räumlichen Struktur eines Moleküls können Abhängigkeiten verschiedener Charaktere entstehen. Bei der ribosomalen RNS wurde eine räumliche Struktur anhand der Abhängigkeiten bestimmter Charaktere vermutet, die sich dann auch im Experiment bestätigt hat[2]. Diese Abhängigkeiten beruhen darauf, daß jede Base Wasserstoffbrücken-Bindungen mit einer bestimmten anderen eingehen kann (z.B. A mit T oder C mit G). Wenn nun bei fast allen Organismen zwei bestimmte Bereiche in der rRNS Sequenz weitestgehend komplementär sind, schließt man daraus, daß aufgrund der räumlichen Struktur des Moleküls diese beiden Bereiche eine Bindung eingegangen sind. Die entsprechende Faltung der rRNS nennt man **Sekundärstruktur**.

Viele dieser Abhängigkeiten können nicht exakt berechnet werden. Aus diesem Grund sollten die mathematischen Algorithmen, die zu ihrer Modellierung verwendet werden, transparent und durch freie Parameter nachträglich modifizierbar sein.

[1] [S&O] Seite 462 Verweis auf W.M. Brown, E. M. Prager and A. C. Wilson 1982. **Mitochondrial DNS Sequences of primates: Tempo and mode of evolution** J. Mol. Evol. 18:225–239

[2] [grum] 6ff

IV.3 Phylogenie der Bakterien

Nachdem die ersten rRNS Sequenzen vorlagen, begann C. Woese am Ende der 70er Jahre die bekannten Bakterien aufgrund der rRNS-Analyse in phylogenetische Gruppen einzuteilen und für diese jeweils einen Stammbaum zu zeichnen. Somit war eine mögliche neue Klassifizierung und Ordnung der Bakterien geschaffen.

Zur Ermittlung des Verwandschaftsgrades zweier Bakterien nimmt man also, sehr vereinfacht ausgedrückt, zwei rRNS-Stränge, vergleicht beide und zählt die Differenzen. Je größer die relativen Unterschiede zwischen den Sequenzen ausfallen, desto entfernter die Verwandschaft:

Ein Beispiel mit 5 Organismen:

Bakterium:[1]	Sequenz[2]
6dssvari:	CGGAUAACGCCCACAGG
6decmult:	CGGAUGACAGUCAUUUG
6debwidd:	CGGAUAACCAUCUUUGG
6dtmorie:	CGGAUAUGCUUAGAUUA
6dtmrumi:	CGGAUAUGGCUAGUAGA

Die Organismen 6dssvari, 6decmult und 6debwidd bilden eine Gruppe, ebenso 6dtmorie und 6dtmrumi:

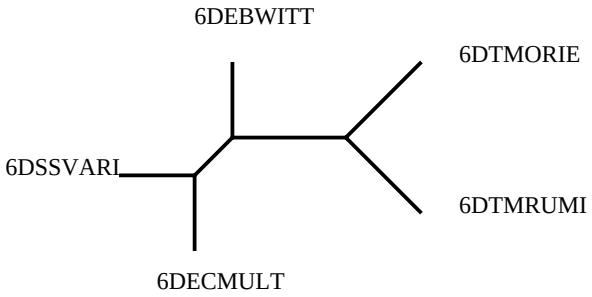


Abb. 4 Rekonstruierter Stamm-Baum

Leider lassen sich nur relative Angaben über das Alter des gemeinsamen Vorfahren machen, da entsprechende “fossile” Organismen fehlen[3].

[1] Es handelt sich hier um eindeutige Abkürzungen, da die biologisch korrekten Namen der Bakterien zum Teil sehr lang sind

[2] 16S-rRNS:

[3] [Woese 87]

IV.4 Gen-Sonden

Phylogenie wird in der Biologie und Mikrobiologie nicht als Sebstzweck betrieben, sondern vor dem Hintergrund praktischer Anwendungen durchgeführt. Mögliche Anwendungen sind z.B. Umweltanalysen, die Extrapolation morphologischer Merkmale bestimmter Bakterien und das Aufspüren und Kennzeichnen unbekannter Bakterien mit Hilfe von **Gen-Sonden**.

Die herkömmliche Methode, unbekannte Bakterien in ihrem Lebensraum zu identifizieren, besteht darin, den Organismus zu isolieren, auf einem geeigneten Nährboden zu züchten und die enstandene Kolonie identischer Bakterien zu untersuchen. Diese Methode setzt allerdings voraus, daß sich die Bakterien auch auf einem Nährboden vermehren lassen. Da in der Vergangenheit das Identifizieren eines Organismus nur mit dieser Methode geschehen konnte, also seine Kultivierbarkeit voraussetzte, sind die meisten der heute bekannten Bakterien kultivierbar. Erst in letzter Zeit wurde festgestellt, daß die meisten noch unbekannten Arten nicht zur Reagenzglaszüchtung geeignet sind.

Diese Arten können seit kurzem einzeln in ihrem Lebensraum aufgespürt und identifiziert werden [Amann 95] [Amann 90a]. Dazu verwendet man die Methode der Sondenhybridisierung.

Hierbei wird ein für das Bakterium charakteristischer rRNS-Ausschnitt ausgesucht und ein dazu komplementäres Oligonukleotid synthetisiert. Diese komplementäre Sequenz wird mit einer radioaktiven oder fluorezierenden Markierung versehen. Die so geschaffene Gen-Sonde kann daraufhin zu einer zu bestimmenden Probe mit fixierten Bakterien zugegeben werden. Falls das gesuchte (fixierte) Bakterium vorhanden ist, lagert sich die Sonde an die entsprechende rRNS-Sequenz an. Mit Hilfe eines Epifluoreszenzmikroskop läßt sich dann das Bakterium leicht ausfindig machen.

IV.4

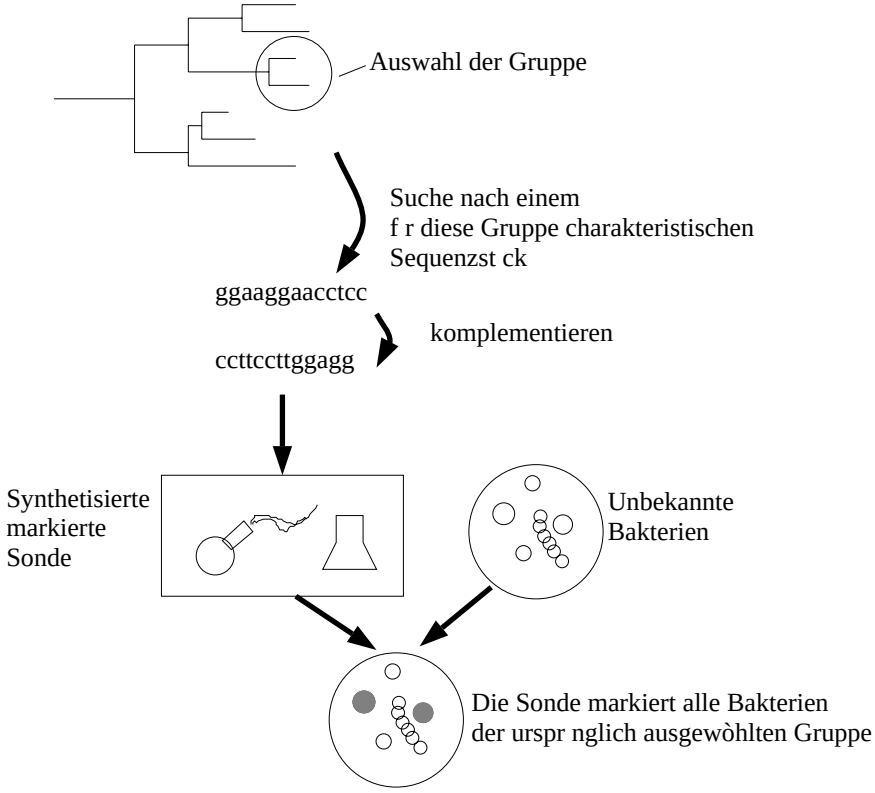


Abb. 5 Prinzip der Sondenanwendung

Um eine Gensonde zu konstruieren, ist ein schneller Zugriff auf alle vorhandenen Daten wünschenswert und notwendig. Nur so kann praxisgerecht und schnell der charakteristische rRNS Ausschnitt gefunden werden.

V Stand der Forschung

Schon in den 80er Jahren entstanden vielfältige Forschungsarbeiten, die sich mit der Frage der Analyse der sequenzierten Gene beschäftigten. Um die Ergebnisse dieser Arbeiten zu belegen, wurden viele kleine, singuläre Computerprogramme entwickelt. Einzelne Forscher wie zum Beispiel Felsenstein versuchten auch erste einfache integrierte Programmpakete zu entwickeln, die jedoch nur sehr spezielle Aufgaben lösten. Eine integrierte Entwicklungsumgebung sprengte dann jedoch in jedem Fall die damaligen Möglichkeiten und die informatische Leistungsfähigkeit der meisten Lehrstühle und Forschungseinrichtungen. So gibt es heute nur wenige, meist kommerzielle, integrierte Lösungen; selbst diese sind in ihrer Leistungsfähigkeit oft eingeschränkt.

Um befriedigende Gesamtlösung zu erreichen, müßten die Teilmodule eines integrierten Gesamtprogrammes drei Mindestanforderungen genügen:

- Die errechneten Ergebnisse müssen den wissenschaftlichen Anfordungen genügen.
- Auch bei steigenden Datenmengen müssen Berechnungen innerhalb annehmbarer Zeiten ablaufen.
- Die Schnittstellen müssen so ansprechbar sein, daß sich das Modul in ein integriertes Programmpaket sauber einfügen läßt.

Im Folgenden sollen die bereits existierenden Programme anhand ihrer Leistungsfähigkeit in den Bereichen Alignment, Phylogenie, Design von Sonden und Gesamtstruktur auf diese Mindestanforderungen hin überprüft werden.

V.1 Alignment

V.1.1 Das allgemeine Problem

Das Alignment von Sequenzen ist der erste wichtige Schritt bei der Sequenzanalyse. Leider ist die Forschung heute noch nicht in der Lage, einen Algorithmus vorzustellen, der eine perfekte Lösung in akzeptabler Rechenzeit anbietet.

Das Alignen von Sequenzen gehört zu einer Klasse von Problemen, die aufgrund ihrer Komplexität (für Mathematiker: NP-vollständig) nur mit extremen Rechenaufwand und auch nur für kleine Probleme in praktikabler Zeit lösbar sind. Das heißt, das Problem hat zwar prinzipiell eine Lösung, doch diese zu finden ist aus praktischen Gründen nureingeschränkt möglich. Um dennoch sinnvolle Ergebnisse zu erhalten, wurde eine Vielzahl von empirischen Näherungslösungen (Heuristiken) entwickelt, die jedoch die Wirklichkeit nur begrenzt modellieren können. Wohl aus diesem Grunde ist die wissenschaftliche Forschung immer noch damit beschäftigt, die Grundlagen neuer Methoden und Programme zum automatischen Alignment zu entwickeln.

V.1.2 Die existierenden Lösungen

Aufgrund der Komplexität des Grundproblems gibt es heute leider kein Programm, das auch nur eine, geschweige denn alle, oben genannten Mindestanforderungen befriedigend erfüllt. Dennoch sollen im folgenden einige bisher verwendete Lösungen diskutiert werden:

- **Needleman und Wunsch** [Needleman 1970] entwickelten die wohl wichtigste Heuristik zum Alignen von Sequenzen basierend auf der dynamischen Optimierung. Das Grundprinzip dieser Idee ist, daß sie nur bestimmte Abhängigkeiten innerhalb einer Sequenz zuläßt und daß aufgrund dieser Vereinfachung der Vorgang mit Hilfe einer sehr großen Matrix effizient in quadratischer Zeit gelöst werden kann. Da es zur Zeit keine wirkliche AlteRNStive für diese Heuristik gibt, ist sie in allen eingesetzten Programmen, zum Teil auch in abgewandelten Formen, verwendet.
- **clustalv bzw. clustalw**: Dieses Programmpaket ist wohl die am häufigsten eingesetzte Software zum Alignen von Sequenzen. Es handelt sich hierbei um ein integriertes Paket mit flexibel steuerbaren Algorithmen, schnellen Heuristiken und oft brauchbaren Ergebnissen. Es kommt auch mit 100 oder mehr Sequenzen gut zurecht. Aus diesem Grunde wurden der wohl größte Teil heutiger Alignments mit diesem Programm erstellt. Ein weiterer Vorteil dieses Programmes ist, daß es sich vollständig von der Kommandozeile bedienen läßt, wodurch dieses Programm leicht in andere Programmpakete integriert werden kann und konnte. Leider verwendet dieses Programm eine Heuristik, die dazu führt, daß weitere

phylogenetische Analysen schlechte Ergebnisse liefern.[1] Hinzu kommt, daß dieses Program diskret arbeitet, es sich also in jeder Situation für ein konkretes Alignment entscheidet und dann diese Entscheidung niemals wieder revidiert.

- **J. Hein**, [hein 89a/b] [hein 94a/b] ist einer der zahlreichen Wissenschaftler, die den Versuch unternehmen, durch Verwendung aller verfügbaren Information die Qualität des Alignments zu verbessern, ohne den Grundalgorithmus von Needleman und Wunsch zu verändern.
- Zur Zeit entwickelt **M. Vingron** in einer Forschungsgruppe einen neuen Aligner, der in der Lage ist, mit unscharfen phylogenetischen Bäumen umzugehen. Allerdings können in der jetzigen Version nur bis zu 25 Sequenzen auf einmal in praktikabler Zeit “alignt” werden[2].

V.1.3 Problematik der existierenden Lösungen

Zum automatischen Alignment von ribosomalen RNS Sequenzen muß davon ausgegangen werden, daß effiziente Algorithmen nur unzureichende Ergebnisse liefern und daß alle existierenden Programme bei weitem nicht in der Lage sind, die heutigen Mengen an ribosomaler RNS auch nur annähernd brauchbar zu alignen. Auch berücksichtigt keiner der existierenden Algorithmen die Sekundärstruktur der Sequenz in irgendeiner Weise. Hinzu kam, daß alle Programme als ’Black Box’ ausgelegt sind und somit kein interaktives Arbeiten erlauben.

Die heutige Datenmenge der ribosomalen RNS macht das vollständige Berechnen eines akzeptablen Gesamtalignments mit den heutigen Grundalgorithmen unmöglich. Daher wurde für ARB ein teilinteraktiver Ansatz gewählt, der den Benutzer als Entscheidungsträger in den Prozeß des Alignens mit einbindet.

Diese Vorgehensweise warf folglich neben dem Grundproblem noch eine weitere Fragestellung auf:

- Der existierende Algorithmus des Programmpaketes ClustalW mußte so modifiziert werden, daß er mit einer deutlich höheren Anzahl an Sequenzen zurecht kommt.
- Es mußten interaktive Alignmentbewertungsalgorithmen entwickelt werden, die Problemzonen mittels eines graphischen Editors aufzeigen.

[1] Diese Heuristik beruht im Prinzip darauf, daß mit einem sehr einfachen schnellen Algorithmus eine Phylogenie der noch nicht alignten Sequenzen erstellt wird, und diese dann die Grundlage für den eigentlichen Alignvorgang darstellt. Auf diese Weise wird ein später eingesetztes aufwendiges Phylogenieberechnungsprogramm stark von der zuerst erstellten Phylogenie beeinflußt.

[2] nach persönlichen Aussagen M. Vingrons

V.2 Phylogenie

V.2.1 Das allgemeine Problem

Phylogenie(griechisch: Lehre der Abstammung) beruht unter anderem auf vergleichenden Sequenzanalysen und bildet die Grundlage einer Vielzahl von weitreichenden Anwendungen.

Die Probleme bei der Berechnung des Stammbaumes sind denen des Alignments ähnlich. Das Grundproblem ist NP-vollständig, entsprechende Heuristiken oft zu ungenau oder zu rechenintensiv. Im Gegensatz zum Alignment ist es sehr schwer, wenn nicht gar unmöglich, das Ergebnis manuell zu bewerten. So kam es beispielsweise öfters vor, daß mehrere Publikationen verschiedene Bäume für dieselbe Bakteriengruppe vorstellten [Achenbach 87] [Woese 91].

V.2.2 Die existierenden Lösungen

Da die Bestimmung des Stammbaumes sowohl ein spannendes wie auch ein schwieriges, aber zugleich auch ein äußerst wichtiges Thema ist, gibt es eine große Vielfalt von Algorithmen, Analysen und einfachen Programmen. Die wichtigsten Methoden lassen sich im wesentlichen in zwei Klassen einteilen:

1. Bei den **Distanzverfahren** wird für eine gegebene Menge von Sequenzen der paarweise phylogenetische Abstand bestimmt. Mit Hilfe dieser Abstandsmatrix läßt sich dann in polynomialer Zeit der Baum berechnen. Während für diese abschließende Baumberechnung gute Algorithmen bekannt sind [Saitou 87] [Soete 83], ist die Berechnung des phylogenetischen Abstandes nicht trivial. Dabei müssen aus Rechenzeit- und praktischen[1] Gründen Vereinfachungen gemacht werden, die der Wirklichkeit nur sehr bedingt entsprechen. So gehen die meisten Programme davon aus, daß sich jede Base in einer Sequenz unabhängig von allen anderen mutiert. Erst in letzter Zeit haben zum Beispiel M. Schöniger und A. von Häseler versucht, diese Vereinfachung für helikale Bereiche der Sekundär-Struktur einer Sequenz aufzuweichen [Schöniger 94].
   Sehr vielversprechend ist der Ansatz von H-J. Bandelt [Bandelt 92a&b], der die Kanten des phylogenetischen Baumes durch sogenannte Splits ersetzt, eine Vorgehensweise, bei dem ein Anwender aufgetretene Probleme und Inkonsistenzen erkennen kann: D.h. kann sich der Algorithmus an einer Kante nicht eindeutig für eine Topologie entscheiden, wird diese Kante durch ein Parallelogram ersetzt. Eine Seite des Parallelograms kann dazu verwendet werden, dieses Parallelogram wieder in eine Kante umzuwandeln. Dazu werden einfach alle Längen der anderen

[1] Jede Erweiterung des primitivsten Grundalgorithmus führt in den meisten Fällen zu komplexen mathematischen Modellen, die nicht trivial gelöst und programmiert werden können.

Parallelogramseiten auf 0 gesetzt. Die Länge der ursprünglichen Parallelogramseite gibt an, mit welcher Wahrscheinlichkeit die tatsächliche Topoligie mit der Topologie übereinstimmt, die sich diese 'Seite in Kante' Operation ergibt.

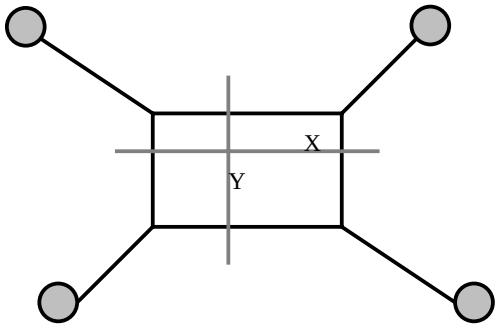


Abb. 6 In diesem Fall kann sich der Algorithmus im Zentrum des Baumes nicht eindeutig für eine Kante entscheiden. Dieser Sachverhalt wird dadurch dargestellt, indem die fragliche Kante durch eine Box ersetzt ist, d.h. man erhält die beiden möglichen Baumtopologien, indem man die Box entweder in X oder in Y Richtung zusammenpreßt.

2. Bei den **Parsimony/Maximum-Likelihood** Methoden gibt es eine 'Rechenvorschrift', die einem gegebenen Baum und Alignment eine Wahrscheinlichkeit zuordnet. Durch Bewerten aller möglichen Baumtopologien kann man alle Bäume mit optimaler, d.h. höchster, Wahrscheinlichkeit herausfinden. Der optimale Baum wird dann als Lösung interpretiert. Leider ist auch dieses Problem in der praktischen Anwendung nicht korrekt lösbar (NP-vollständig), die bekannten Heuristiken [Dueck 88] [Lundy 85] [Swofford 90] sind für kleine Problem annähernd optimal, wenn auch für die Datenmenge der rRNS-Sequenzen bei weitem nicht ausreichend.[1]

Des weiteren gibt es natürlich eine ganze Menge anderer Methoden, die jedoch meist Forschungscharakter besitzen und bis heute nur vereinzelt den Weg in ein anwendbares Programm gefunden haben.

Die wichtigsten dieserAlgorithmen wurden in verschiedenste Programmpakete integriert, unter anderem in

- **phylip:** Das bis jetzt wohl bedeutendste Programm wurde schon sehr frühzeitig von Joseph Felsenstein entwickelt [Felsenstein 81a] und beinhaltet alle wichtigen Algorithmen. Obwohl dieses Paket vom softwaretechnischen Standpunkt suboptimal ist, ist die verwendete Mathematik doch weit in die Zukunft gedacht und heute noch den meisten anderen Programmen überlegen.

[1] So berichtete zum Beispiel Ross Overbeek, wie er für eine Tagung einen Baum berechnet hatte: Zuerst wurde mit einem schnellen Verfahren ein Ausgangsbaum generiert. Dieser wurde in sich überlappende Teilbäume gespalten. Jeder dieser Teilbäume wurde auf einem Supercomputer mehrere hundert mal mit leicht veränderten Heuristiken neu berechnet, wobei ein Lauf viele Stunden Rechenzeit benötigte. Dann wurde versucht, diese Teilbäume, teils maschinell, teils mit der Hand wieder zu einem Baum zusammenzufügen.

- **molphy:** Ein von einem japanischen Team entwickeltes Paket zur phylogenetischen Analyse von Proteinen. Obwohl mathematisch sehr weit fortgeschritten, steckt es seitens der Implementierung noch weitestgehend in den Kinderschuhen, so daß es noch keine weite Verbreitung gefunden hat.
- **paup:** Hierbei handelt es sich um ein leistungsfähiges Parsimonypaket [Swofford 90]. Mit vielen Optionen und für Macintosh-Benutzer mit graphischer Benutzeroberfläche ausgestattet, war es bis zum Entstehen von ARB mit Abstand das professionellste Phylogenieprogramm.
- **Mc Clade:** Dies ist ein interaktives Parsimonyprogramm von Maddison [Maddison 92]. Hier berechnet nicht der Computer die möglichen Topologien, sondern der Anwender verändert den Baum interaktiv. Der Rechner bewertet nun diesen Baum und hilft so dem Benutzer, die optimalen Entscheidungen zu treffen.
- **fastDNSml:** Dieses optimierte Maximum-Likelihood-Programm entstand auf der Grundlage von phylip [Olsen 94]. Da viele Wissenschaftler, unter anderen G. Olsen, zu dem Schluß gekommen waren, daß Felsensteins Maximum-Likelihood-Konzept mathematisch allen anderen Programmen überlegen war, jedoch aufgrund seiner schlechten algorithmischen Umsetzung ineffizient arbeitete, investierte G. Olsen fast ein Jahr intensiver Arbeit in seine Optimierung und Verbesserung. Ergebnis ist ein zwar ’gehacktes’, aber äußerst brauchbares Programm, das bis heute das ’non plus ultra’ bei der Berechnung großer Bäume darstellt.

Bereits theoretisch ist es ausgeschlossen, daß je ein Programm in der Lage sein wird, die Phylogenie immer korrekt zu berechnen. Je nach Anwendungsfall haben jedoch die verschiedenen Verfahren eine geringere oder höhere Wahrscheinlichkeit, die richtige Lösung zu ’raten’.

V.2.3 Problematik der existierenden Lösungen

Problematisch war im hier zu lösenden Anwendungsfall wieder einmal die Datenmenge, für die ein Baum berechnet werden sollte. Schnell stellte sich heraus, daß die meisten Algorithmen bei mehr als 100 oder gar 1000 Sequenzen vollkommen versagten, hier jedoch für zehntausende die Stammbäume berechnet werden sollten. Oft war es aus prinzipiellen Gründen schwer möglich, alle Algorithmen so zu verändern, daß sie für die großen vorhandenen Datenmengen verwendbar wurden. Selbst die Anwendung der Algorithmen auf kleine Datenmengen erwies sich als sehr mühsam, zu unterschiedlich waren die Schnittstellen existierender Programme, um sie von Nichtinformatikern bedienbar zu machen.

Die Aufgabe bestand nicht darin, ein neues Verfahren zu entwickeln, sondern vielmehr darin, eine angemessene Näherungslösung für die Phylogenie zu berechnen. Da kein Verfahren auch nur annähernd perfekt arbeitet, stellte sich die Aufgabe als Integration aller existierenden Programme, Optimierung existierender Algorithmen und Entwicklung leistungsfähigerer Heuristiken dar.

V.2

Es sollte den Biologen ein Instrument in die Hand gegeben werden, mit dem sie in kürzester Zeit und mit wenig Einarbeitungsaufwand für eine begrenzte Menge von Sequenzen möglichst viele verschiedene Verfahren ausprobieren, anwenden und vergleichen konnten.

Um nun aber auch große Bäume berechnen zu können, sollte das Parsimonyverfahren möglichst praktisch anwendbar umgesetzt werden. So mußten unter anderem leistungsfähige Strategien zur Ausnutzung von Datenabhängigkeiten, schnelle Undo-Funktionen auf Bäumen und leistungsfähige Heuristiken entwickelt, umgesetzt und optimiert werden. Da alle bekannten Heuristiken bei großen Datenmengen versagen, sollte mit der Entwicklung eines Konzeptes für den Einsatz genetischer Algorithmen eine vielversprechende Heuristik geschaffen werden.

V.3 Sonden

V.3.1 Das allgemeine Problem

Bei der Sondenberechnung geht es im wesentlichen darum, einen kurzen, meist 15 bis 20 Zeichen langen Ausschnitt zu finden, der eine gegebene Menge an Sequenzen möglichst gut charakterisiert, d.h. in allen diesen Sequenzen vorkommt und zu allen Teilsequenzen innerhalb aller anderen bekannten Sequenzen möglichst unterschiedlich ist. Im Gegensatz zu den beiden schon beschriebenen Problemen ist dieses relativ neu und es existieren nur wenige Forschungsarbeiten und höchstwahrscheinlich kein einsatzfähiges Programm zur Berechnung.

Um ein eindeutiges Sondenstück für eine gegebene Sequenzgruppe zu finden, müssen viele Stringsuchoperationen durchgeführt werden. Da die zu durchsuchende Sequenzmenge nicht unerheblich ist (>100 * 10^6 Basen) lassen sich auch traditionelle Algorithmen nur schlecht unmodifiziert anwenden. Ausgehend von Einzelsonden mußte zusätzlich ein Konzept entwickelt werden, das es erlaubt, optimale Kombinationen solcher Sonden zu finden.

V.3.2 Existierende Lösungen

Zur Zeit gibt es (meines Wissens nach) nur eingeschränkte Lösungen, die eine Konstruktion einer Sonde in praktikabler Weise auf einem lokalen Rechner ermöglicht. Allerdings gibt es (erst zum Ende dieser Arbeit hin) vereinzelte einfache Lösungsversuche, die meistens über das WWW angeboten werden (z.B. http://www.cme.msy.edu).

Die Aufgabe für diese Arbeit lag darin, einen schnellen, einfachen und gut skalierbaren Algorithmus zu entwickeln, mit dem effizient eine qualitativ hochwertige Sonde konstruiert werden kann.

V.3.3 Problematik der zu erstellenden Lösung

Die zu bewältigende Aufgabe gliedert sich in verschiedene Teilgebiete:

- Es muß eine Funktion gefunden werden, die die Qualität einer Sonde oder Sondenkombination bewertet.
- Ein schneller Suchalgorithmus muß implementiert werden, der mit Hilfe dieser Bewertungsfunktion die optimale Sonde findet.
- Das Programm muß schließlich dergestalt in ein größeres Programmpaket integriert werden, daß auch Biologen, die wenig bis keine Erfahrung im Umgang mit Computern besitzen, effizient arbeiten können.

- Schließlich soll ein theoretisches Konzept entwickelt werden, das die Grenzen des gerade entwickelten Algorithmus aufzeigt. Diese Abschätzung ist notwendig, um bei exponentiell steigender Sequenzmenge auch in Zukunft sinnvoll Sonden und Signaturen berechnen zu können.

Da das entwickelte Programm, im nachhinein gesehen, das einzig praktikabel einsetzbare seiner Art war, kam noch eine relativ große Integrationsaufgabe hinzu. So wurden zum Beispiel höchst leistungsfähige Kompressionsalgorithmen für java-Applets entwickelt, die es erlauben, dieses Programm auch von Internet-Browsern optimal zu bedienen.

V.4 Integrierte Softwarelösungen

V.4.1 Das allgemeine Problem

Traditionell entstand in der Bioinformatik eine Vielzahl kleiner Module. Erst durch die Kombination verschiedener solcher Komponenten konnten sinnvolle Resultate erzielt werden. Dabei werden im einfachsten Fall einfach die Ausgaben eines Programmes als Eingabe eines anderen verwendet:

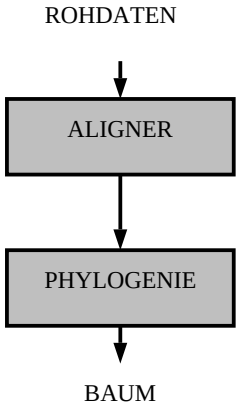


Abb. 7 Einfache Bearbeitungspipeline

Dieses Vorgehen entspricht im wesentlichen den schon bei Unix bekannten ’Pipes’. Da diese Lösung zu einer nicht sehr beliebten kommandozeilen-orientierten Arbeitsweise führt, wurde nach neuen Lösungen gesucht. Als ersten Schritt versuchten einige Arbeitsgruppen diesen Ablauf zu automatisieren. Allerdings wurden wohl oft die technischen Schwierigkeiten so sehr unterschätzt, daß viele dieser Projekte bereits im Prototypenstadium versandeten.

Im wesentlichen sind also zwei Aufgaben zu lösen:

- Algorithmen müssen zu einer sinnvollen Zusammenarbeit gebracht werden. Da als Ausgangspunkt eine Vielzahl von mathematisch anspruchsvollen Algorithmen mit einer sehr einfachen Implementierung vorlagen, stand die Wahl offen, diese Algorithmen neu zu programmieren oder entsprechende Schnittstellen für die existierenden Implementierungen zu schaffen.
- Alle Teilmodule müssen in einer konsistenten, einfachen Weise von einem Benutzer angesteuert werden können. Idealerweise geschieht dies mit Hilfe einer graphischen Benutzeroberfläche.

V.4.2 Die existierende Lösungen

Betrachtet man die existierenden Software-Pakete, so stellt man fest, daß unterschiedliche Konzepte zur Integration verfolgt und verwirklicht wurden.

- **phylip:** Das Paket von Felsenstein basiert auf der Idee, ein genormtes Dateiformat für alle Sequenz- und Matrixdaten zu verwenden. Dadurch ließen sich alle Module einfach auf Kommandozeilenebene verbinden. Die Grenzen dieser Lösung sind schnell erreicht. Zum einen erlauben die verwendeten Dateiformate keinerlei Erweiterungen, die Anzahl der zu verarbeitenden Daten ist aufgrund fester Grenzen deutlich eingeschränkt, und es gibt keinerlei Module zur Bearbeitung und Auswahl der Sequenzen. So blieb vielen Anwendern keine andere Wahl, als mit dem Unix-Editor 'vi' die Sequenzen zu editieren.
- **Mc Clade/Paup:** Diese beiden Programme basieren auf dem Nexus-Format, das auf Macintosh-Computern üblich ist. Dieses Format erlaubt es, beliebige Inhalte in einer Datei abzulegen. So konnten eine Vielzahl verschiedenster Datentypen, Sequenzen, Bäume und Matrizen in ein und derselben Datei gespeichert werden. Aufbauend auf diesem Dateiformat sind beide Programme mit einer hervorragenden Benutzeroberfläche ausgestattet. Trotz des guten Ansatzes konnte sich dieses Paket nicht durchsetzen. Zum einen war es speziell für den Macintosh entwickelt worden, ein Computer, der außerhalb der USA nur bedingt anzutreffen ist, zum anderen wird dieses Format nur von diesen beiden Programmen verwendet.
- **GCG:** Zentrales Element dieses Programmpaketes ist eine einfache, aber leistungsfähige Sequenzdatenbank, an die eine Vielzahl von Algorithmen angepaßt wurde. Es handelt sich hier um eines der ersten Programme, die gut skalierend auch mit vielen tausend Sequenzen spielend umgehen konnten. Die Ergebnisse der Algorithmen waren meist illustrative Graphiken, die auch als solche behandelt wurden. Da dieses Programm aufgrund der Komplexität nicht von einem Forschungsinstutut entwickelt werden konnte, wurde die Entwicklung kommerzialisiert. Infolgedessen kostet eine Lizenz mehrere tausend Mark. In den letzten Jahren wurde die ursprünglich kommandozeilenbasierte Eingabeschnittstelle durch eine komfortable graphische Benutzeroberfläche erweitert. Der Schwerpunkt dieses Paketes liegt ganz eindeutig im Bereich der Proteinanalyse, Funktionen zur Rekonstruktion der Phylogenie und zur Editierung der Sequenzen sind nur äußerst rudimentär vorhanden:

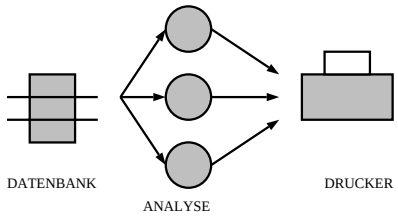


Abb. 8 Vereinfachtes Übersichtsschema des GCG Paketes

- **GDE:** Weltweit existieren drei Forschungsgruppen, die systematisch und im großen Stil ribosomale RNS Sequenzen sammeln: die DeWachter Gruppe in Ant-

werpen, der Lehrstuhl für Mikrobiologie der TU München und die RDP (Ribosomal Database Project) in East Lansing (USA). Die RDP startete zu diesem Zweck das GDE (Genetic Data Environment) Projekt. Die Grundidee dabei war, daß ein leistungsfähiger Sequenzeditor mit Hilfe einfacher Konfigurationsdateien so geändert werden konnte, daß existierende Programme einfach zu integrieren waren. Dazu mußte man nur das Dateiformat des externen Moduls kennen, seine Kommandozeilenoptionen und schon konnte man mit Hilfe eines kleinen Shell-Scripts und der eben erwähnten Konfigurationsdatei innerhalb von wenigen Stunden viele existierende Programme verwenden.

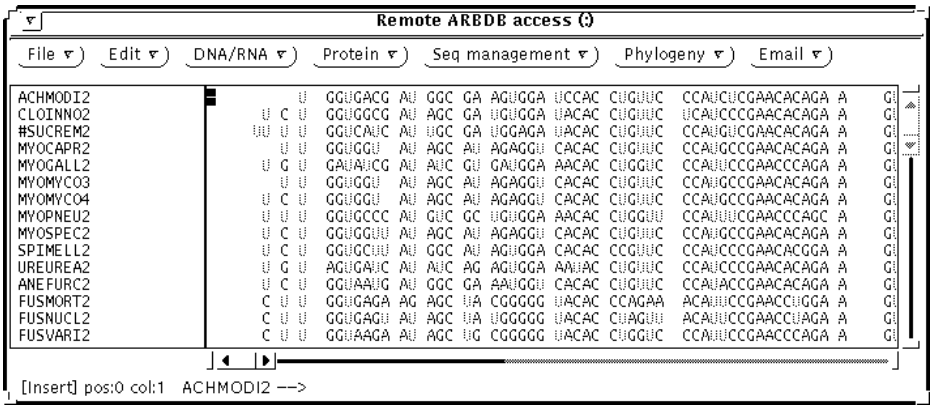


Abb. 9 Screenshot des GDE Editors

Primäres Ziel dieses Editors war seine Einfachheit der Bedienung, ein Ziel, das er voll und ganz erfüllt hat. Dieser Editor ist vollkommen sequenzorientiert, eine Eigenschaft, die die Bedienung stark vereinfacht, die jedoch dazu führt, daß sich dieser Editor nur schlecht skalieren läßt. Mit wenigen hundert Sequenzen überschreitet man leicht die Grenze zur Unübersichtlichkeit. Dabei ist das Problem weniger der Ressourcen-Verbrauch als vielmehr die fehlende hierarchische Strukturierungsmöglichkeit der Sequenzen.

- **ALE:** Da sich der GDE-Editor aufgrund seiner sehr einfachen inneren Struktur nicht ohne weiteres erweitern ließ, wurde als Nachfolgeprojekt das Projekt ALE ins Leben gerufen. Die Idee dabei war, den existierenden und sehr leistungsfähigen 'emacs'-Editor so zu konfigurieren, daß sich mit seiner Hilfe sinnvoll Sequenzen bearbeiten lassen. Der nach 2 Mannjahren Entwicklungszeit fertiggestellte Editor zeigt eine Eleganz bei der Bedienung, wie sie bis heute unerreicht ist. So kann man mit der Maus auf einfache Weise ganze Teilsequenzen innerhalb einer Gruppe von Sequenzen verschieben. Es stellte sich jedoch heraus, daß der 'emacs'-Editor als Basis bei weitem zu langsam ist, um interaktiv mit vielen Sequenzen arbeiten zu können. Selbst bei einer Verzehnfachung der Leistungsfähigkeit der Hardware wäre flüssiges Arbeiten nicht möglich. Hinzu kommt, daß dieser Editor wie auch sein Vorgänger GDE rein sequenzbasiert ist und sich somit Strukturinformationen nicht abspeichern lassen.
- **tkdcse:** Die Gruppe um DeWachter entwickelte einen Sequenzeditor auf 'tcl/tk' Basis. Dies ist eine einfache interpretierte Programmiersprache, die die Erstellung graphisch orientierter Programme stark vereinfacht. Das Neue an diesem Editor

war, daß von ihm auch die Sekundärstruktur der Sequenzen konsequent mitgespeichert und angezeigt wurde. Auch dieser Editor verwendet keine leistungsfähige Datenbank und skaliert mit steigender Sequenzmenge schlecht. Hinzu kommt, daß die Sekundärstruktur ineffizient gespeichert wird, so daß die Sequenzdateien auf mehr als hundert MegaByte anwachsen können. Schließlich war das verwendete 'tcl/tk' System bei der Darstellung unnötig langsam, so daß ein flüssiges Arbeiten den Besitzern einer schnellen Silicon Graphics Workstation vorbehalten blieb.

V.4.3 Problematik der existierenden Lösungen

Bei genauerer Untersuchung lassen sich die Pakete im wesentlichen in zwei Klassen einteilen: Die einen verwenden eine Datenbank, die anderen nicht. Die datenbankorientierten Programme waren alle mit einer nicht interaktiven graphischen Oberfläche ausgestattet, die gut bedienbaren Editoren hingegen skalierten mit der Anzahl der Sequenzen schlecht. Zudem gab es fast kein Programm, das mit anderen Inhalten als Sequenzen gut umgehen kann. So kann man zum Beispiel mit den meisten Programmen Bäume berechnen, jedoch lassen sich diese Bäume höchstens an den Drucker schicken und nicht als Eingabe zur Kontrolle der Sequenzen oder Alignments verwenden. Schließlich waren bei allen Programmen die internen Module nur lose zusammengeflickt, eine interaktive Zusammenarbeit verschiedenster Module war damit von vornherein ausgeschlossen.

Die hier zu bewältigende Aufgabe war es also, die aufgezeigten Probleme existierender Lösungen zu umgehen. Es genügte, die Ideen anderer Wissenschaftler zu adaptieren, um ein leistungsfähiges Grundkonzept zu erweitern und zu implementieren. Die entscheidende Idee, die allen Programmen bis dato fehlte, war die konsequente Entwicklung einer Datenbank, die verschiedenste Inhalte — und nicht nur Sequenzen — effizient speichern, und die als Synchronisationsmedium zwischen verschiedenen Anwendungmodulen dienen konnte. Erst durch die Entwicklung dieser Datenbank konnten alle Module sich gleichzeitig und synchron über dieselben Datenobjekte 'unterhalten'. Ausgehend von diesem Grundsystem sollten alle vorhanden Programme, die als Source-Code vorlagen, zu dem System ARB integiert werden. So wurde beispielsweise dem GDE Editor die Ansteuerfunktionen für externe Module entnommen, dem phylip-Paket die wichtigsten Phylogenieprogramme, es wurde clustalw integriert und außerdem die wichtigsten Algorithmen in einer optimierten und interaktiven Form implementiert.

VI Zu lösende Aufgaben

Die Anforderungen, die an das Projekt ARB gestellt wurden, entstanden nicht vollständig zu Projektbeginn, sondern erst nach und nach unter Berücksichtigung der bis dahin gesammelten Erfahrung. Während am Anfang die funktionalen Anforderungen im Vordergrund standen, zeigte es sich im weiteren Verlauf, daß die aus der praktischen Anwendung enstehenden Nebenbedingungen eine immer wichtigere Rolle spielten. Während es Anfangs bereits einen Erfolg darstellte, die wenigen weltweit bekannten Sequenzen auf primitive Weise analysieren zu können, mußte mit der Zeit immer mehr Zeit und Energie darin investiert werden, aus der weltweit einsetzenden Datenflut die für die Analyse relevanten Sequenzen herauszufiltern.

An einem kurzen Beispiel soll diese Anforderungsverschiebung gezeigt werden:

Die eigentliche, ursprüngliche Aufgabe, die ARB erfüllen sollte, war es, bestimmte Eigenheiten einzelner Basen oder Sequenzabschnitte zu erkennen und anzuzeigen. Diese Aufgabe konnte nur dadurch gelöst werden, daß möglichst viele Sequenzen miteinander verglichen wurden. Dies wiederum setzte einen leistungsfähigen Sequenz-Editor und ein effizientes Programm zur Stammbaumberechnung voraus. Allerdings sollten Stammbäume nicht nur berechnet, sondern auch graphisch angezeigt werden. Aus diesem Grunde wurde ein weiteres Unterprojekt gestartet, im Rahmen dessen ein Stammbaum-Editierprogram entwickelt wurde. Um Mehrfacharbeiten zu vermeiden, wurden sowohl der Stammbaum-Editor wie auch der Stammbaumberechnungs-Algorithmus mit einer flexiblen Datenschnittstelle ausgestattet, die es erlaubte, die Daten schnell und einfach zwischen beiden Programmen auszutauschen. Erst jetzt ließ sich die ursprünglich geforderte Funktionalität, das Erkennen von Baseneigenheiten, implementieren. Diese Datenschnittstelle (**ARBDB**) wurde im Lauf der Zeit immer weiter verbessert und stellt heute den entscheidenden Kern des gesamten ARB-Projektes dar.

Dadurch, daß erst eine geeignete Umgebung für funktionale Analysen geschaffen werden mußte, verzögerte sich zwar die Lösung der eigentlichen Aufgabe, es konnten jedoch auf diesem Fundament die Sequenzanalyse-Algorithmen einfach, elegant und vor allem interaktiv aufgesetzt werden.

Die Anforderungen an ARB können also in die funktionalen Grund– und die notwendigen praktischen Erweiterungsanforderungen gegliedert werden:

VI.2 Funktionale Grund-Anforderungen

Wie schon erwähnt, war es die ursprüngliche Anforderung, Sequenzen nach verschiedenen Kriterien zu vergleichen (vergleichende Sequenzanalyse). Um jedoch Sequenzen miteinander vergleichen zu können, müssen eine ganze Reihe von Vorarbeiten geleistet worden sein, die logisch aufeinander aufbauen:

- **Sequenz-Rohdaten sammeln, verwalten und editieren:** Es existiert weltweit eine ganze Reihe großer Sequenzdatenbanken, die alle bekannten Sequenzen in meist unsortierter Form öffentlich zur Verfügung stellen. Mit Hilfe des Internet lassen sich diese Datenbanken durchsuchen und Sequenzen anhand bestimmter Stichwörter finden sowie auf den eigenen Rechner kopieren. Aus diesen Rohsequenzen müssen nun die relevanten Sequenzinformationen herausgeschnitten werden. Alle diese Sequenzen mit ihren Beschreibungen sollen nun in einer lokalen Datenbank verwaltet, editiert und veröffentlicht werden.
- **Rohdaten vergleichbar machen:** Normalerweise werden nur solche Sequenzen miteinander verglichen, die sich aus einer gemeinsamen Ursequenz entwickelt haben. Um nun solche Sequenzen miteinander optimal vergleichen zu können, ist es sinnvoll, die Sequenzen auszurichten (**alignen**). Dieser Vorgang muß durch leistungsfähige Hilfsprogramme unterstützt werden:

  a. **Eine gegebene Menge von Sequenzen soll alignt werden:** Da die Forschung in diesem Bereich schon viel Vorarbeit geleistet hat, reicht es aus, existierende Algorithmen in das zu schaffende Programmpaket zu integrieren.
  b. **Zu einer gegebenen alignten Menge von Sequenzen soll eine neue Sequenz dazu-alignt werden:** Auch hier kann auf existierende Algorithmen zurückgegriffen werden, allerdings setzt deren interaktive Integration in ARB ein leistungsfähiges Datenkommunikationskonzept voraus.
  c. **Ein gegebenes Alignment soll überprüft werden:** Mit Hilfe einer Online-Primär- bzw. Sekundär-Strukturanalyse soll ein gegebenes Alignment auf Unstimmigkeiten untersucht werden. Dazu sollen diese in einem Sequenzeditor farbig markiert dargestellt werden.
  d. **Eine Sequenz eines Alignments soll überprüft werden:** Sequenzen, die noch nicht vollständig sequenziert und kontrollgelesen wurden, zeichnen sich oft durch viele Fehler aus. Um nun diese Fehler möglichst schnell zu finden, sollten mögliche Fehlerstellen farbig markiert am Bildschirm angezeigt werden.

- **Phylogenie-Rekonstruktion:** Basierend auf einem Alignment läßt sich eine mögliche Verwandtschaftsbeziehung der Sequenzen schätzen. Die Aufgabe war es nicht, neue Algorithmen zu finden, sondern:

  a. **für große und sehr große Anzahl von Sequenzen Stammbäume zu generieren:** Viele der bekannten Verfahren (sog. Distanzverfahren) werden bei steigender Sequenzanzahl zu ungenau. Untersuchungen haben gezeigt, daß

die Ungenauigkeit der Stammbaumschätzung exponentiell mit dem Radius des Baumes zunimmt. Andere Verfahren (Maximum Likelihood/ Parsimony), die diesem Problem nicht unterliegen, benötigen oft eine nicht mehr tolerable Rechenzeit. Durch Erweiterung existierender **Parsimony-** und **Maximum-Likelihood** Verfahren um **genetische Algorithmen** soll eine neue Heuristik entwickelt werden, die es erlaubt, für das NP-vollständige Grundproblem in kürzerer Zeit eine brauchbare Lösung zu finden.

b. **die Topologie großer Bäume zu bewerten**: Die klassischen Verfahren zur Bewertung der Stabilität einer Stammbaumtopologie benötigen viele hundert Berechnungen ein und desselben Baumes mit leicht veränderten Eingabedaten. Da dies aufgrund fehlender Rechenressourcen bei großen Bäumen nicht möglich ist, sollen Verfahren entwickelt werden, die aufbauend auf dem Parsimony Algorithmus die Stabilität einzelner Kanten im Baum schätzen.

- **Berechnung von Oligonukleotid-Sonden:** Basierend auf einer Menge von (meist rRNS) Sequenzen und ihrem dazugehörigen Stammbaum, sollen kurze, d.h. 15–25 Basen lange Sequenzstücke (**Oligonukleotid-Sonden**) gesucht werden, die eindeutig eine Sequenz oder eine Sequenzgruppe identifizieren:

  a. **Suche nach eindeutigen Sequenz oder Sequenzgruppen-Identifikatoren:** Gesucht werden soll ein kurzes Sequenzstück (= Oligonukleotid), das in einer benutzergegebenen Sequenzmenge möglichst oft vorkommt, und das zu allen anderen Sequenzabschnitten aller anderen Sequenzen einen möglichst hohen Hammingabstand aufweist.

  b. **Suche nach einem Mehrfacholigonukleetid-Tupel:** Basierend auf einer gegebenen Menge von Oligonukleotid-Sonden soll diejenige Dreierkombination gefunden werden, die eine gegebene Sequenzmenge optimal identifiziert.

VI.3 Praktische Anforderungen

Die Praxis des Softwareerstellungsvorganges muß die Wünsche zweier unterschiedlicher Parteien berücksichtigen. Zum einen stellen die Endbenutzer Forderungen an die Bedienbarkeit und Schnelligkeit des Programms sowie an den optischen Gesamteindruck der Benutzeroberfläche, zum anderen ist es im Sinne des Programmierteams wünschenswert, eine sinnvolle und effiziente Arbeits- und Ablauforganisation zu finden.

VI.3.1 Anforderungen aus Benutzersicht

Mit dem Aufkommen graphischer Benutzeroberflächen erwartet der typische Benutzer heute eine moderne, leicht zu erlernende und leistungsfähige Oberfläche, die alle Funktionalitäten eines Programmpaketes durch eine einfache Bedienung mit der Maus zugänglich macht. Darüber hinaus soll ARB folgende Eigenschaften bekommen:

- **Hohe funktionale Vielfalt:** ARB soll mindestens alle Funktionen integrieren, die schon im letzten Kapitel ausführlich besprochen wurden.
- **Orthogonale Funktionalität:** Jede Teilfunktion soll mit jeder beliebig anderen Teilfunktion frei kombiniert werden können, und zwar so, wie man es als Anwender intuitiv erwarten würde. Zum Beispiel soll ein gerade berechneter Stammbaum dazu dienen, das Alignment der Sequenzen zu überprüfen.
- **Graphische Benutzerschnittstelle:** Nur durch den konsequenten Einsatz einer graphischen Oberfläche für **alle** Funktionen von ARB kann die Akzeptanz von ARB durch seine Anwender langfristig sichergestellt werden.
- **Integration existierender Software:** Es gibt im Internet eine Vielzahl höchst leistungsfähiger Algorithmen zur Sequenzanalyse. Da diese meist noch keine eigene graphische Benutzeroberfläche besitzen, sollen sie mit einer solchen versehen und in ARB integriert werden.
- **Integration bestehender Algorithmen:** Da es manchmal nicht gelingen wird, existierende Software in ARB so zu integrieren, daß diese interaktiv bedienbar bleibt, sollen die zugrundeliegenden Algorithmen umprogrammiert und um ein interaktives Kontrollprogramm ergänzt werden.
- **Integration von ARB in bestehende Analyseumgebungen:** Da Sequenzdaten heute meist mit Hilfe von Computern verwaltet werden, sind die Anwender meist mit einer bestimmten Anwendung bereits sehr gut vertraut. ARB soll nun so implementiert werden, daß es bestehende Systeme nicht ersetzt, sondern ergänzt. Dies setzt voraus, daß Daten leicht zwischen den verschiedenen Paketen ausgetauscht werden können.
- **Fehlertoleranz:** Da zu Versuchszwecken immer wieder neue Algorithmen implementiert werden sollen, darf ein Fehlverhalten der Anwender nicht automatisch zu Datenverlust führen. Aus diesem Grund soll ein konsequentes **'UNDO'**– Konzept die Rückgängigmachung aller Funktionen garantieren.
- **Interaktiv:** Es soll eine Analyseumgebung für die wissenschaftliche Forschung entwickelt werden, d.h. daß die Algorithmen so implementiert werden, daß sie für

den Anwender sichtbar bleiben. Das sogenannte 'Black-Box'-Prinzip ist unbedingt zu vermeiden, da dabei unter Umständen Fehlerquellen zu spät erkannt und nicht mehr eliminiert werden können. Soweit möglich, sollen also alle Algorithmen interaktiv bedienbar sein, d.h. keine oder nur eine geringe Reaktionszeit auf die Eingaben des Benutzers haben. So soll z.B. ein Sequenzeditor entwickelt werden, der schon während der Eingabe mögliche Sekundärstruktur-Fehler erkennt und anzeigt.

VI.3.2 Praktische Nebenbedingungen

Zusätzlich zu den Anforderungen aus der Sicht der Benutzer muß das Program ARB eine ganze Reihe praktischer Bedingungen erfüllen, die sich aus den verschiedenen zu betrachtenden Kontextsituationen ableiten. Hierzu zählt zum einen der Kontext, in dem die Entwicklung durchgeführt wird, zum anderen der Kontext, der für eine spätere Anwendung maßgeblich ist.

- **Implementierung durch eine lose Gruppe von Informatikern unterschiedlichster Qualität:** Selbst unerfahrene Studenten sollten nach möglichst kurzer Einarbeitung Erfolge bei der Entwicklung neuer Funktionalitäten vorweisen können.
- **Unabhängigkeit der Module bei der Programmierung:** Da gleichzeitig meist von mehreren Teams an verschiedenen Stellen des Programmpaketes gearbeitet wurde, soll eine Entwicklungsumgebung geschaffen werden, die es den Teams erlaubt, unabhänging voneinander zu entwickeln und zu testen.
- **Fehlertoleranz:** Da Fehler in einer Software nie auszuschließen sind und die beteiligten Studenten oft noch wenig Programmiererfahrung vorweisen konnten, muß das Gesamtsystem so fehlertolerant und sicher sein, daß selbst ein Absturz eines Moduls auf keinen Fall die Konsistenz der Daten gefärdet.
- **Offenheit des Systems:** Für zukünftige Softwareentwickler soll es einfach sein, weitere Funktionalitäten in ARB zu integrieren. Deswegen soll eine ganze Reihe von Schnittstellen implementiert werden, die eine Erweiterung von ARB auf verschiedenstem Niveau garantieren: **C**, **C++**, **PERL** und **JAVA** Schnittstellen für erfahrene Softwareentwickler, **ASCII** Schnittstellen für Programierneulinge.
- **Platformunabhängigkeit:** Da die einschlägigen Labors bereits Rechner mit den verschiedensten Architekturen besitzen, sollte ARB nicht auf eine bestimmte Rechnerarchitektur festgelegt werden. Da mit UNIX ein platformunabhängiges Betriebssystem zur Verfügung stand, sollte ARB vor allem auf allen bekannten UNIX-Derivaten zum Einsatz kommen.
- **Public Domain:** Da ARB aus Forschungsmitteln finanziert wurde, sollten keine kommerziellen Softwaremodule integriert werden, damit das Gesamtprogramm vollständig frei von Rechten Dritter bleiben würde. Wie sich später herausstellte, führte diese Bedingung zu einer Datenbank, die in diesem speziellen mikrobiologischen Kontext allen bekannten kommerziellen Datenbanken weit überlegen war.
- **Schnell und gut skalierend:** Das Programmpaket soll auch in Zukunft mit stark steigenden Datenmengen zurechtkommen, ohne sofort auf den Einsatz von neuen, leistungsstarken und dann auch entsprechend teuren Rechnern angewiesen zu sein.

VI.3

Alle diese hier angesprochenen Probleme und Forderungen tauchten im Rahmen des ARB Projektes auf und wurden zur Zufriedenheit aller Beteiligten gelöst. Diese Lösungsansätze werden im Folgenden dokumentiert.

VII Optimierung des Alignment

VII.1 Einführung und wissenschaftliche Ziele

Bei den vergleichende Sequenzanalysen müssen die Sequenzen in einer Tabelle so angeordnet und ausgerichtet werden, daß pro Zeile eine Sequenz und pro Spalte jeweils die homologen Basen angeordnet werden. Dieser Vorgang wird als **alignen**, das Ergebnis als **Alignment** bezeichnet. Da im Laufe der Evolution diese homologen Bereiche verschoben wurden, dabei aber leider keine irgendwie geartete Markierung tragen, muß eine Reihe verschiedener Indizien dafür eingesetzt werden, diese zu finden:

- Man versucht mit Hilfe der **Primärstruktur** die Sequenzen so auszurichten, daß die Anzahl der spaltenweisen Unterschiede zweier Sequenzen minimal wird.
- Da die **Sekundärstruktur** oft wesentlich konservativer ist als die Primärstruktur, versucht man die Sequenzen so zu alignen, daß gleiche Sekundärstrukturbereiche in derselben Spalte zu stehen kommen.
- Oft kann man die unterschiedliche **Konservativität** einzelner Sequenzpositionen ausnützen, um konservative homologe Bereiche zu identifizieren.

Nun gibt es eine ganze Reihe effizienter Algorithmen, die Sequenzen anhand ihrer ungewichteten Primärstruktur alignen. Diese Algorithmen besitzen eine Zeitkomplexität O(Sequenzlänge * Sequenzlänge * Anzahl_der_Sequenzen). Sie basieren im wesentlichen auf einem Grundalgorithmus von Needleman und Wunsch [Needleman 70], der den kostengünstigsten Weg durch eine Matrix sucht. Leider lassen diese Algorithmen keine praktische Berücksichtigung der Sekundärstruktur zu, sie würden dadurch sofort zu einem Algorithmus mit NP[1]-vollständiger Komplexität ausarten. Außerdem basieren die meisten heutigen Programme ausschließlich auf dem paarweisen Alignment von Sequenzen, so daß beim Alignen von Sequenzmengen auf Heuristiken zurückgegriffen werden muß.

Aufgrund großer praktischer Erfahrung verschiedener Labors kann folgendes behauptet werden:

Die existierenden Programme zum Alignen einer gegebenen Menge von Sequenzen liefern oft mittelmäßige Ergebnisse[2]

Wesentlich besser sieht die Situation aus, wenn eine neue Sequenz zu einem gut alignten Datensatz hinzugefügt werden soll. Dies ist der Standardfall für 16S– und 23S–rRNS Sequenzen. Hat man also einmal einen gut alignten Grunddatensatz zur Verfügung, dann

[1] Non Polynomial. D.h. Es gibt keinen Algorithmus, der dieses Problem innerhalb polynomialer, d.h. praktikabler Rechenzeit löst.

[2] Dies liegt vor allem daran, daß zum Alignen von mehr als 4 Sequenzen ein interner Baum aufgebaut werden muß. Die Blätter des Baumes entsprechen den Sequenzen, die inneren Knoten den Konsensussequenzen ihrer Blättern. Die Sequenzen werden nun so alignt, daß man rekursiv an allen inneren Knoten die beiden Söhne paarweise alignt. Stimmt der aufgrund statistischer Heuristiken aufgebaute Baum nicht, erhält man ein nicht optimales Alignment.

wird auch das Hinzualignen neuer Sequenzen problemlos vor sich gehen. Somit ergibt sich als erstes Teilziel für ein gutes Alignment des Gesamtdatensatzes, einen ausreichenen Grunddatensatz interaktiv so zu alignen, daß er als Basis für das automatisierte Hinzufügen weiterer Sequenzen dienen kann.

Als Gesamtziel für das Alignment war es nicht so wichtig, neue Algorithmen zum automatischen Alignment zu entwerfen, sondern es sollten vielmehr leistungsfähige Alignment-Kontroll-Programme entwickelt werden, die es einem Benutzer erlauben, schnell die fehlerhaften Stellen in seinem Alignment zu finden.

Um ein primärstrukturbasiertes Alignment maschinell überprüfen zu können, bietet es sich an,

- die **Sekundärstruktur** zu überprüfen und
- **statistische** Informationen über die einzelnen Spalten der Alignments auszuwerten.

Die Umsetzung eines an diesen Kriterien orientierten Programms zur Alignment– Sequenzüberprüfung stellte einen wesentlichen Kernpunkt bei der Entwicklung von ARB dar. Da ein gutes Alignment interaktiv überprüft werden sollte, sollten auch alle zu entwickelten-den Hilfsprogramme interaktiv eingesetzt werden können. Interaktiv heißt in diesem Fall, daß eine Änderung einer Sequenz im Alignment sofort die Anzeige dieser Alignmentkontrollprogramme beeinflußt. Erst dadurch kann ein effizientes Arbeiten garantiert werden.

VII.2 Sekundärstruktur

VII.2.1 Einführung

Haben sich zwei Sequenzen unterschiedlicher Organismen, die von derselben Ursequenz abstammen, entsprechend weit auseinander mutiert, wird es aus praktischen Gründen oft schwierig, längere Subsequenzen zu finden, die in beiden Sequenzen vorkommen und die als Indikator für homologe Bereiche dienen können. Also sucht man bei der Erstellung eines Alignments möglichst nach noch konservativeren Merkmalen als der Primärstruktur. Unsbesondere bei den ribosomalen RNS-Sequenzen hat man solche Merkmale gefunden:

Bei allen bekannten ribosomalen RNS-Sequenzen hat man festgestellt, daß etwa die Hälfte aller Basen einer Sequenz mit einer bestimmten anderen Base derselben Sequenz eine physikalische-chemische Bindung[1] (**Wasserstoffbrücken**) eingeht und somit korreliert sind. In den Bereichen, in denen sich längere Teile einer Sequenz mit anderen Teilbereichen derselben Sequenz Wasserstoffbrücken eingehen, bildet sich um die Achse dieser Bindungen jeweils eine doppelhelikale Struktur (**Helix**) aus. Die Platzierung der so verbundenen Teilbereiche innerhalb einer Sequenz ist höchst konservativ und wird als **Sekundärstruktur** bezeichnet. Um diese Sekundärstruktur im Alignment zu berücksichtigen, wurde zunächst als einfachster möglicher Ansatz die These aufgestellt, daß aus der bekannten Sekundärstruktur einer Sequenz auf diejenige aller Sequenzen eines Alignments geschlossen werden könne[2]. Erstaunlicherweise führte dieser einfache, fast schon naive Ansatz zu äußerst brauchbaren Ergebnissen, und ließ auch eine reichlich elegante, d.h. rechenzeit– und speicherplatzsparende Implementierung zu.[3]

VII.2.2 Aufgabe und existierende Lösungen

Das Ziel bei der Erstellung eines Alignments ist es, ein möglichst realitätsnages Alignment zu erstellen. Dies gelicht umso besser, je mehr Information zur Verügung seht. Die Aufgabe für diese Arbeit ist es, die Informationen, die sich aus der Sekundärstruktur ergeben, dem Benutzer in eleganter Form zugänglich zu machen. Dazu mußte

- ein Modell gefunden werden, die jeweils leicht unterschieliche Sequndärstruktur für alle Bakterion effizient zu speichern.

---

[1] Die wichtigsten möglichen Basenpaarungen sind die Watson Crick Paarungen GC und AT. In der Praxis gibt es allerdings auch weitere Paarungsmöglichkeiten.

[2] Anderen Bioinformatikern fiel es schwer, diesen Ansatz zu akzeptieren. N. Larsen von der RDP zum Beispiel konnte das Funtionieren dieses naiven Ansatzes erst dann akzeptieren, als er seine Ergebnisse sah. Die Programmiergruppe von de Wachter hatte schon eine allgemeinere Helixdarstellung implementiert, die jedoch Haupt- und Plattenspeicherintensiv war und sich zudem nur schwierig in einen interaktiven Editor integrieren ließ

[3] Somit war es möglich, einen Algorithmus zu wählen, der erst bei der Darstellung der Sequenz die entsprechenden Sekundärstrukturinformation der dargestellten Sequenzteile berechnet und somit vollkommen auf eine Speicherung dieser Information verzichten kann.

- diese gespeicherte Sequndärstruktur so dem Benutzer präsentiert werden, daß dieser ein Alignment optimieren kann.

Existierende Lösungen in diesem Bereich beinhalten zum Beispiel den Editor dkdcse (von der deWachter Gruppe) oder Ale (RDP, unveröffentlicht). Leider skalieren diese beiden Editoren mit der Anzahl der Sequenzen nur sehr schlecht, des weiteren ist die Sekundärstrukturanalyse relativ primitiv.

VII.2.3 Theoretisches Konzept

Nomalerweise läßt sich die Sekundärstruktur mit Hilfe der Grammatik wohlgeformter Klammerausdrücken (jede geöffnete Klammer wird auch geschlossen) darstellen. Die mathematisch korrekte Definition eines eine Sekundärstruktur beschreibenden wohlgeformten Klammerausdruckes *K* gibt die folgende Abbildung:

K::= **K K**
| **(B K B)**
| eps
B::= **'[ACGTU]'**

Abb. 10 Sekundärstruktur als wohlgeformter Klammerausdruck

Dabei paart die Base, die auf eine Klammer folgt, mit der Base, die vor der zugehörigen schließenden Klammer steht.

In dem folgenden Beispiel paart demnach das erste G mit dem letzten C, sowie das zweite G mit dem vorletzten C:

```
(G (G A G U G C) C)
    +---------+
  +---------------+
```

Abb. 11 Vom Klammerausdruck zur paarenden Position

Praktisch bedeutet dies, daß sich die Sekundärstruktur in einer sogenannten Faltblattform zeichnen läßt.

Die Natur hält sich jedoch nicht an mathematische Formeln und weicht von dieser einfachen, schönen rekursiven Struktur vielfach ab. Um dennoch mit wohlgeformten Klammerausdrücken die Sekundärstruktur darstellen zu können, wurde eine 'Multi-Layer' Klammerschreibweise entwickelt und eingesetzt:

- Die Darstellung der Sekundärstruktur basiert auf zwei Zeichenketten: Einem Klammerausdruck und einer Zeichenkette zu Identifizierung verschiedener Klammerschichten.

- Um herauszufinden, welche Klammer mit einer gegebenen Klammer paart, werden alle Klammern ausgefiltert, die einer anderen Ebene angehören als der gegebenen. Anhand des übriggebliebenen wohlgeformten Klammerausdruck läßt sich leicht die entsprechende paarende Klammer finden.

Beispiel:

Klammerausdruck	`((( - ((( - ))) - )))`
Ebenenzuweisungbeschreibung	`000 - 111 - 000 - 111`
Klammerausdruck der Ebene 1	`xxx  x ((( x xxx x )))`
Klammerausdruck der Ebene 0	`((( x xxx x ))) x xxx`

Abb. 12 Beispiel eines Mehrfachebenen-Klammerausdruck

Aus praktischen (Lesbarkeits)-Gründen werden die Klammersymbole '()', '[]' sowie '<>' vom Programm gleichwertig behandelt, der Benutzer kann die verschiedenen Symbole dazu verwenden, die Lesbarkeit der Zeile zu erhöhen. Außerdem können Ebenenzuweisungen dann unterlassen werden, wenn zugehörige Klammern direkt nebeneinander stehen, da Zuweisungsinformationen dann redundant sind.

Aus einer gegebenen zweizeiligen Sekundärstrukturbeschreibung einer Sequenz läßt sich eine partielle Funktion

$$SF : P \rightarrow P$$

definieren, wobei **P** die Menge aller Sequenzpositionen ist. **SF** gibt somit an, welche Sequenzposition mit einer anderen paart. Da nicht alle Basen einer Sequenz an der Sekundärstruktur beteiligt sind, ist **SF** partiell.

Da die Sekundärstruktur der ribosomalen RNS konservativer ist als die Primärstruktur, liegt es nahe, nicht für jede einzelne Sequenz eine Funktion **SF** zu finden, sondern eine für das gesamte Alignment. Man definiert sich analog eine partielle Funktion

$$S_{ges} : P_A \rightarrow P_A$$

mit $\mathbf{P_A}$ gleich der Menge aller Positionen in einem Alignment.

Um nun für eine Base einer Sequenz die entsprechende paarende zu finden, wendet man die Funktion S_{ges} auf die Alignmentposition *p* dieser Base an:

- Ist $q = S_{ges}(p)$ definiert, ist q die gesuchte Alignment-Position der paarenden Base,
- sonst gibt es für diese Position keine Sekundärstrukturinformation.

VII.2.4 Praktische Umsetzung

Ziel ist es, mit Hilfe einer Paarungsfunktion S_{ges} eine Form der Sekundärstruktur-Darstellung zu finden, die es erlaubt, ein gegebenes Alignment zu verbessern. Im wesentlich gibt es dazu zwei Möglichkeiten:

VII.2

I. Man zeigt die Paarungen von Basen innerhalb einer Sequenz an,

   a. indem man die Sequenz so faltet, daß sich die paarenden Basen gegenüberliegen

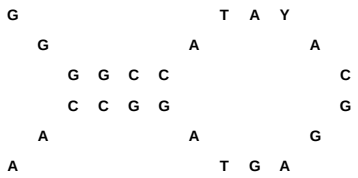


Abb. 13 Sekundärfaltblattstruktur

   b. indem man die paarenden Basen mit Linien verbindet:

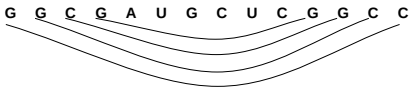


Abb. 14 Sekundärdarstellung durch Bögen

II. Man stellt nur dar, ob eine Base an der Alignment-Position *p* mit der Base an der Position $q = S_{ek}(p)$ eine Wasserstoffbrückenbindung eingehen kann. Können beide Basen keine Verbindung eingehen, liegt eine **Helixverletzung** vor, die man dadurch anzeigt, daß man entweder

   a. die entsprechende Positionen in einem Alignmenteditor farbig hinterlegt,

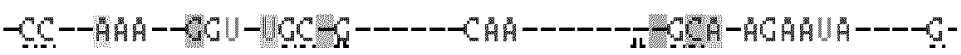


Abb. 15 Anzeige der Sekundärstrukturfehler durch Farben

   b. oder alle paarenden Basen innerhalb einer Helix durch ein leicht erkennbares Symbol '~' und '-' anzeigt.

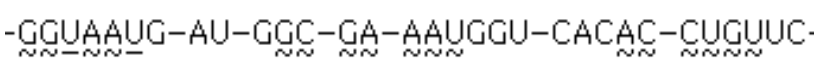


Abb. 16 Anzeige der Sekundärstruktur mit Hilfe von Symbolen

Dabei geht man davon aus, daß sich die Sekundärstruktur innerhalb desselben Alignments nicht ändert und man von festen Positions-Paaren ausgehen kann.[1]

[1] Bei ribosomalen RNS Sequenzen ist dies der Fall.

VII.2

Da die Aufgabe ist, ein Alignment zu überprüfen, scheiden die Fälle Ia und Ib aus, da sie jeweils nur eine Sequenz anzeigen können. Außerdem muß für diese beiden Fälle zuerst die tatsächliche Sekundärstruktur gefunden werden, während man in den anderen Fällen nur überprüft, ob die Basen zweier Alignment-Positionen eine Paarung eingehen können.

Offentsichtlich gilt:

- Der Mensch kann schneller eine große Zahl verschiedener Symbole erkennen als dieselbe Anzahl verschiedener Farben (z.B. chinesische Schriftzeichen). Mit Hilfe verschiedener Symbole läßt sich daher mehr Information übertragen. Somit läßt sich nicht nur darstellen, ob zwei Basen eine Paarung eingehen oder nicht, sondern auch, welche Basen an dieser Paarung beteiligt sind. Somit läßt sich bei einem kurzen Blick auf die eine Hälfte einer Helix die andere Hälfte exakt rekonstruieren.

- Neuere Editoren nutzen zudem die Farbdarstellung zur unterschiedlichen Einfärbung der verschiedenen Basen einer Sequenz, z.B. **A**denin rot, **C**ytosin gelb usw. Färbt man nun auch noch den Hintergrund der Sequenzen, leidet der Kontrast und die Buchstaben werden schlechter lesbar.

- Symbole lassen sich auch auf Schwarz-Weiß–Druckern leicht ausdrucken.

- Das Einfärben des Hintergrundes bleibt als Option für andere Zwecke erhalten.

Dadurch, daß für das gesamte Alignment eine global gültige Funktion S_{ges} gefunden werden kann, können für jede Sequenz in kürzester Zeit alle Helixverletzungen ermittelt werden. Es ist daher nicht notwendig und auch nicht sinnvoll, diese Informationen in der Datenbank zu speichern. Es hat sich herausgestellt, daß es nicht einmal notwendig ist, alle Helixverletzungen einer Sequenz zu generieren, **es reicht aus, nur für die Teile der Sequenz, die gerade am Bildschirm dargestellt werden, die Sekundärstrukturverletzungen zu berechnen**. Dies führt zu einem minimalen Hauptspeicherbedarf des Sequenzeditors, eine Größe, die bei der Bearbeitung von bis zu 10000 Sequenzen der Länge 10000 Zeichen nicht zu veRNSchlässigen ist. Außerdem führt diese Vorgehensweise dazu, daß immer die aktuellen Helixverletzungen korrekt angezeigt werden. Aus diesem Grund ist es auch sinnvoll, bei einer interaktiven Änderung einer einzelnen Base durch einen Benutzer die gesamte Sequenz auf dem Bildschirm neu zu zeichnen, damit alle Helixverletzungen im betrachteten Ausschnitt neu dargestellt werden können.

Für ARB wurde also die Darstellungsart mit Symbolen gewählt. Diese können durch eine interaktive Maske vom Benutzer ausgewählt werden.:

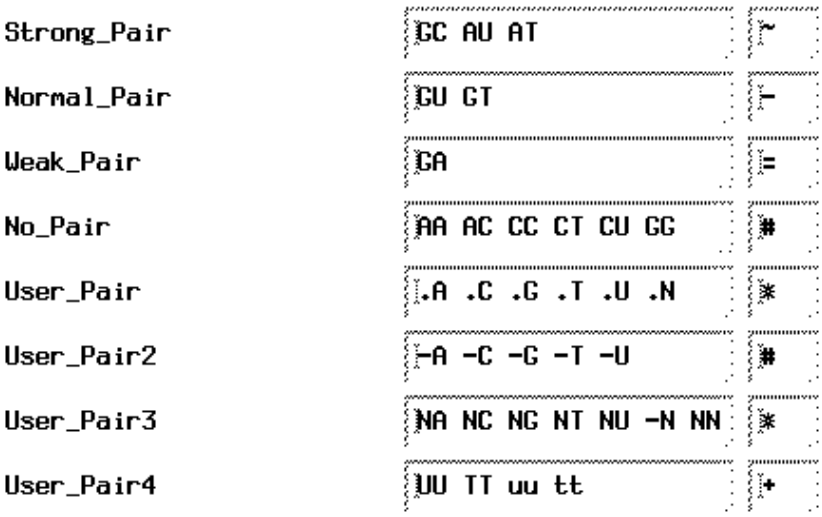


Abb. 17 Beispiel einer ausgefüllter Eingabemaske für den Sequenzeditor. Damit lassen sich für beliebige Kombinationen von Basen jeweils ein Symbol auswählen, das diese Paarung grafisch unterlegt. Z.B. erhält die Watson Crick Paarung das Symbol '~' und die nicht Paarung 'AA' das auffällige Symbol '#'. Auf diese Weise läßt sich sehr einfach und übersichtlich erkennen, welche Bereiche korrekt paaren.

Zusammengefaßt hat sich also folgendes Vorgehen als sinnvoll erwiesen:

- Die Helix-Information für ein Alignment wird in zwei ASCII-Zeichenketten kodiert, wobei die eine Zeichenkette aus einer Vielfachebenen-Klammerstruktur besteht, während die andere die Ebene der einzelnen Klammern der ersten Zeichenkette bestimmt.
- Beim Start eines Editors oder eines anderen Programms, das die Helixinformation benötigt, werden diese beiden Zeichenketten zu einer S_{ges}–Funktion dekodiert. Diese S_{ges}–Funktion läßt sich als eine einfache Tabelle speichern.
- Anhand einer benutzerdefinierten Tabelle wird bestimmt, welche Basenpaare zu bestimmten Helix-Symbolen führen.
- Soll eine Sequenz mit den jeweiligen Helixverletzungen dargestellt werden, werden diese Verletzungen erst bei der eigentlichen Bildschirmdarstellung generiert. Auf diese Weise vermeidet man unnötige Redundanz.
- Bei einer Sequenzänderung muß der gesamte sichtbare Abschnitt neu dargestellt werden, um die Helixverletzungs-Information auf dem Bildschirm immer auf dem neuesten Stand zu halten.

Abb. 18 Vorgehen zum Darstellen der Helixinformation in einem Sequenzeditor

VII.2.5 Ergebnisse und Diskussion

Der einfache Ansatz, für ein gesamtes Alignment eine feste Positions-Positions-Zuordnung festzulegen, die dann die Sekundärstruktur der einzelnen Sequenzen beschreibt, hat sich

als überaus erfolgreich erwiesen. Es konnte nicht nur auf eine aufwendige Berechnung und Speicherung der Sekundärstruktur verzichtet, sondern eine immer aktuelle Darstellung der Sekundärstrukturverletzungen in einem Editor elegant implementiert werden. Diese Darstellung erlaubt die direkte und einfache Kontrolle eines Alignments und ist äußerst hilfreich bei der Bearbeitung einzelner Sequenzen.

Wurde einmal von einem Datenbankverwalter die Alignment-Sekundärstruktur festgelegt, so können beliebige Anwender diese Funktionalität effektiv nutzen, ohne sich entsprechend einarbeiten zu müssen oder sich durch aufwendige Menüstrukturen durchzuarbeiten.

Ein typischer Auschnitt aus dem Sequenzeditor ARB_EDIT zeigt deutlich die Sekundärstrukturverletzungen, die mit dem Symbol '#' dargestellt sind.

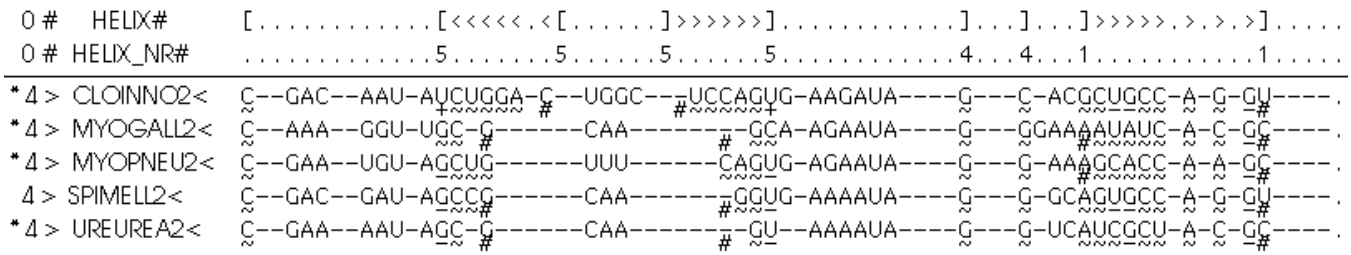


Abb. 19 Auschnitt aus dem ARB_EDIT Sequenz-Editor: Sequenzen mit Helixinformation

Welches Vorgehen ist nun aber angebracht, wenn sich die Sekundärstruktur einzelner Gruppen entgegen der globalen Sekundärstruktur teilweise ändert? Eine für ein ganzes Alignment gültige Sekundärstrukturbeschreibung kann natürlich nicht die Sekundärstrukturänderungen in einzelnen Sequenzen berücksichtigen. Sekundärstrukturen ändern sich jedoch nur aufgrund von entsprechenden Mutationen der Primärstruktur. Dies bedeutet, daß eine Sequenzanalyse auf Primärstrukturbasis zwischen Sequenzen unterschiedlicher Sekundärstrukturen unsinnig ist, da

- bereits die Primärstruktur keine signifikanten Ähnlichkeiten mehr aufweisen kann, und zusätzlich
- ein veränderter Selektionsdruck auf diesen unterschiedlichen Sekundärstrukturen herrscht, der zu einer unterschiedlichen Mutationsgeschwindigkeit dieser Bereiche führt. Hierdurch wird wiederum die Vergleichbarkeit in Frage stellt.

Ein sinnvoller Ausweg aus dieser Situation ist es, die Teilbereiche des Alignments, die unterschiedliche Sekundärstrukturen besitzen, zu trennen:

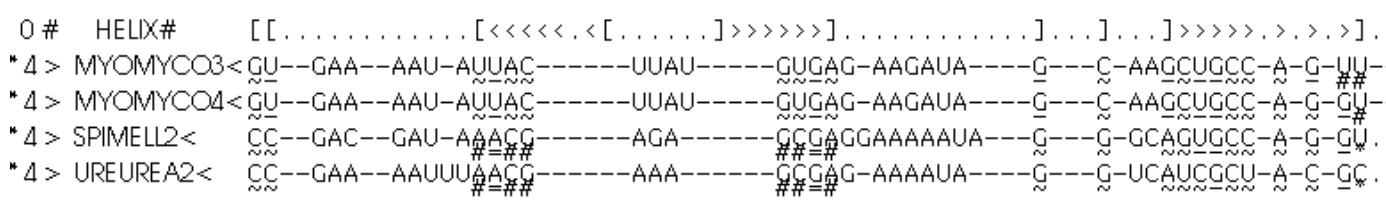


Abb. 20 Beispiel einer sich geänderten Sekundärstruktur

Im obigen Beispiel hat sich die Sekundärstruktur der beiden letzten Sequenzen geändert. Dies erkennt man mit einem Blick daran, daß viele Positionen dieser Sequenzen keine

Paarung mit dem durch S_{ges} angegebenen Nachbarn eingehen, hier dargestellt durch das '#' Zeichen. Aus diesem Grunde sollte obiges Alignment folgendermaßen geändert werden:

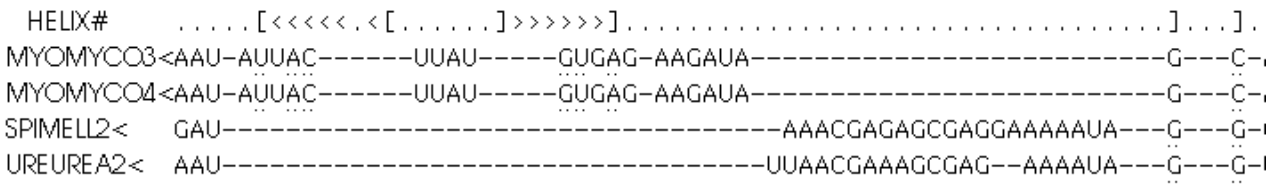


Abb. 21 Korrektur des Alignments mit geänderter Sekundärstruktur

Die Integration einer einfachen Sekundärstrukturanalyse in einem Sequenzeditor stellt ein mächtiges und effektives Werkzeug dar. Unklarheiten in einem Alignment können schnell und klar herausgearbeitet werden. Auf lange Sicht wäre es jedoch wünschenswert, bereits in den automatischen Alignvorgang Sekundärstrukturinformationen mit einzuarbeiten. Die Lösung dieses Problems ist allerdings nicht trivial und soll nachfolgenden Programmierergenerationen überlassen bleiben.[1]

Mit steigendem Sequenzaufkommen könnte es jedoch sein, daß ein solches Programm immer notwendiger wird, damit mit vertretbaren Aufwand ein Gesamtalignment der ribosomalen RNS verwalten werden kann. Eine spätere Integration eines solchen Programmes in ARB sollte kein entscheidendes Problem darstellen.

[1] Dennoch wurden Vorarbeiten durchgeführt, in denen untersucht wurde, wie sich die Sekundärstrukur-Information in einem Align–Algorithmus mit verarbeiten ließe. Die Ergebnisse lassen sich folgendermaßen zusammenfassen:

- Vor dem eigentlichen Alignvorgang muß die Sekundärstruktur der Sequenz ermittelt werden. Da dies für eine komplette ribosomale RNS Sequenz aus Rechenzeit- und algorithmischen Gründen noch nicht möglich ist, muß abwechselnd grob alignt und Sekundärstrukturinformation aus den noch nicht alignten Teilbereichen gewonnen werden.
- Mit Hilfe der Sekundärstrukturinformation kann nun folgende Heuristik zum alignen verwendet werden: Die Bewertungen der einzelnen Matrixelemente des Needleman und Wunsch-Algorithmus müssen die jeweils paarenden Basen mitberücksichtigen, sowohl auf der Sequenz– wie auch auf der Alignmentseite.
- Da es sich nur um eine Heuristik handelt, muß das Ergebnis folgendermaßen geprüft werden:
  Alignt man die Sequenz gegen das Alignment sowie die invertierte, d.h. rückwärtsgelesene, Sequenz mit dem invertierten Alignment und vergleicht beide Ergebnisse, so deuten gleiche Teilergebnisse auf ein brauchbares primär- und sekundärstrukturbasiertes Alignment hin. Unterscheiden sich Teile der Ergebnisse, hat die Heuristik in diesem Teilbereich versagt.

VII.3 Basenwahrscheinlichkeit

VII.3.1 Übersicht

Heutige Situation

Bei der Entwicklung von Sequenzanalysealgorithmen legt man typischerweise eine definierte Menge von Annahmen zugrunde. Die Qualität eines solchen Modells testet man im allgemeinen dadurch, daß eine Wahrscheinlichkeit (**Likelihood**) errechnet wird, die Aussagen darüber macht, wie gut das Modell eine Situation beschreibt. Dies wird für das Modell wie für ein Referenzmodell durchgeführt. Erhöht sich die Likelihood durch dieses neue Modell signifikant, wird es akzeptiert.

Zielsetzung

Im vorliegenden Anwendungsfall soll jedoch genau andersherum vorgegangen werden: Zu einem gegebenen Modell und Datensatz sollen solche Stellen im Datensatz gefunden werden, die die Likelihood am stärksten reduzieren, also nicht modellkonform sind. Dazu werden die Daten des Datensatzes systematisch einzeln verändert und die Likelihood neu berechnet. Kann durch eine Änderung eines Datenelementes *D* eine höhere Likelihood erhalten werden, wird diese Erhöhung als Attribut *H* dieses Elementes *D* gespeichert, andernfalls wird *H* von *D* auf 0 gesetzt. Damit ergeben sich zwei wichtige Anwendungsgebiete:

- Diese Likelihoodänderung *H* wird für jede Base eines Alignments ermittelt und dient zur visuellen Markierung auffälliger Basen **(Online Statistik)**. Somit können schnell in einem Datensatz:

  a. Sequenzierfehler
  b. Alignmentfehler
  c. außergewöhnliche Mutationen
  d. normale Mutationen

erkannt, analysiert und ausgewertet werden.

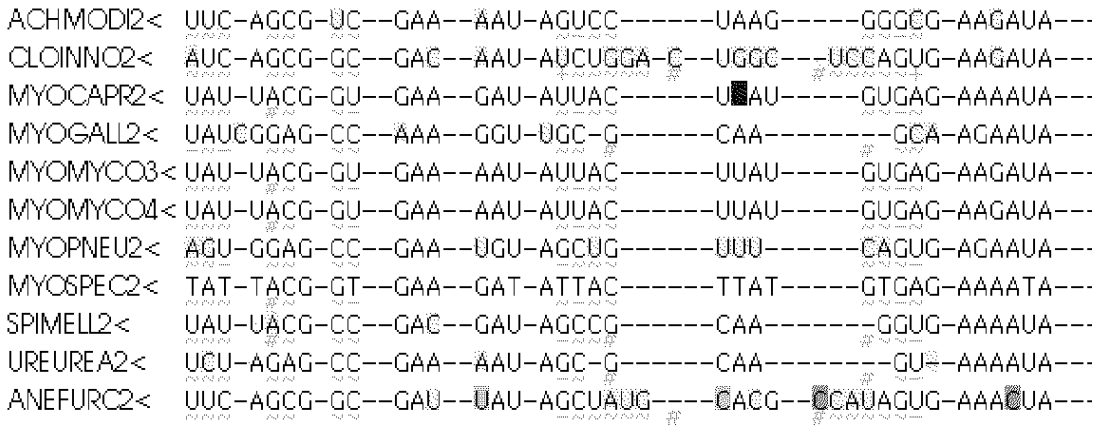


Abb. 22 Markieren derjenigen Basen, die die Likelihood erniedrigen

- In einem zweiten Schritt wird das Alignment in *n* gleichlange Teile zerlegt und in jedem Teil für jeweils jede Sequenz die mittlere Erhöhung H_m aus allen *H* der entsprechenden Basen berechnet. Signifikante Unterschiede zwischen den unterschiedlichen H_m– Werten einer Sequenz deuten auf:

  a. ein nicht adäquates Modell.
  b. eine schlecht alignte Sequenz.
  c. eine Sequenz mit vielen Sequenzierfehlern.
  d. außergewöhnliche evolutionäre Vorgänge

Mit diesen beiden Hilfsmitteln kann man in einem Alignment schnell problematische Stellen aufspüren und korrigieren. Auch hierbei ist es von Vorteil, wenn die Algorithmen so implementiert werden, daß eine interaktive Darstellung der *H*–Werte möglich ist. Im Gegensatz zur Berechnung der Sekundärstruktur ist die Berechnung der *H*–Werte nicht lokal für einzelne Basen möglich, es mußte erst eine leistungsfähige Cache-Strategie entwickelt werden, die für ein interaktives Arbeiten eine ausreichend schnelle Berechnung ermöglicht.

VII.3.2 Basistheorie

Die meisten der zur Zeit eingesetzten Evolutionsmodelle gehen von folgenden Grundannahmen aus:

- Die einzelnen Spalten des Alignments sind voneinander unabhängig. Diese Annahme ist sicherlich vollkommen falsch, da die Basen einer Sequenz meistens eine sinnvolle Funktion im Wechselspiel mit allen anderen Basen des Genoms haben. Diese Vereinfachung ist jedoch oft sinnvoll und notwendig, um überhaupt Sequenzanalyse betreiben zu können. Da im vorliegenden Anwendungsfall hauptsächlich die Qualität des Alignment selbst überprüft werden soll, d.h. unhabhängig vom Rest der Sequenz einzelne Spalten ausgewertet werden sollen, ist diese Annahme

vorerst sinnvoll. Weitergehende Untersuchungen in dieser Richtung erscheinen jedoch in jedem Fall angebracht.

- Für jede Spalte des Alignments kann eine von der evolutorischen Zeit abhängige Mutationsmatrix angegeben werden, die die Wahrscheinlichkeit beschreibt, mit der eine gegebene Base dieser Spalte in eine andere mutiert:

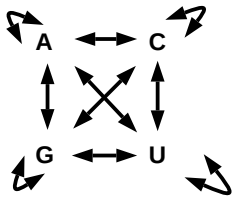


Abb. 23 Basenmutation

Das Mutationsverhalten einzelner Basen läßt sich auch als konventionelle Matrix *M(t)* darstellen. Diese Matrix läßt sich nicht direkt angeben, daher geht man zunächst von ihrer Ableitung, der Wahrscheinlichkeitsmatrix M' aus:

$$M' = \begin{pmatrix} \alpha_{aa} & \alpha_{ca} & \alpha_{ga} & \alpha_{ua} \\ \alpha_{ac} & \alpha_{cc} & \alpha_{gc} & \alpha_{uc} \\ \alpha_{ag} & \alpha_{cg} & \alpha_{gg} & \alpha_{ug} \\ \alpha_{au} & \alpha_{cu} & \alpha_{gu} & \alpha_{uu} \end{pmatrix}$$

wobei α_{ij} die relative Wahrscheinlichkeit ist, daß ein *i* innerhalb einer sehr kurzen Zeit *t* in ein *j* mutiert. Dabei gilt

$$\forall i \in \{a,c,g,u\} \ni \alpha_{ii} = - \sum_{j \in \{a,c,g,u\}: j \neq i} \alpha_{ij}$$

Dabei soll gelten: *M(t1 + t2) = M(t1) * M(t2)*, d.h. die Wahrscheinlichkeiten ändern sich im Laufe der Evolution nicht, sie sind für eine Spalte des Alignments festgelegt.
Jeder diskreten Base kann also ein Vektor zugeordnet werden:

$$v_a = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}; v_c = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}; v_g = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}; v_u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Die Wahrscheinlichkeit, daß eine Base *i* während des Zeitraumes *t* zu der Base *j* mutiert, beträgt:

$$L_{ij} = v_j * \left(\left(\lim_{\partial t \to 0^+} \left(\prod_{n=0}^{t/\partial t} \left(M' * \partial t + I \right) \right) \right) v_i \right)^T$$

oder mit

$$M(t) = \lim_{\partial t \to 0^+} \left(\prod_{n=0}^{t/\partial t} \left(M' * \partial t + I \right) \right)$$

wobei gilt:

- L_{ij} ist der Eintrag an der Spalte *i* und Zeile *j* der Matrix *M(t)*
- *I* ist die Einheitsmatrix
- M^T ist die Transponierte der Matrix M

Jukes-Cantor-Modell

Unter Annahme, daß alle Übergangswahrscheinlichkeiten α gleich groß sind, erhält man das Jukes-Cantor[1]–Modell:

$$M' = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}$$

$$a = \frac{1 + 3e^{-4\alpha t}}{4}$$

$$b = \frac{1 - e^{-4\alpha t}}{4}$$

$$M(t) = \begin{pmatrix} a & b & b & b \\ b & a & b & b \\ b & b & a & b \\ b & b & b & a \end{pmatrix}$$

Kimura Zwei-Parameter-Modell

Berücksichtigt man zusätzlich zum Jukes-Cantor-Modell noch eine unterschiedliche Mutationsrate zwischen Transitionen (Mutationen A-G bzw. C-U) und Transversionen (Mutationen A-C, A-U, C-G bzw. G-U), so erhält man das Zwei-Parameter-Modell von Kimura:

$$M' = \begin{pmatrix} -\alpha - 2\beta & \beta & \alpha & \beta \\ \beta & -\alpha - 2\beta & \beta & \alpha \\ \alpha & \beta & -\alpha - 2\beta & \beta \\ \beta & \alpha & \beta & -\alpha - 2\beta \end{pmatrix}$$

$$a = \frac{1 + 2e^{-2(\alpha+\beta)t} + e^{-4\beta t}}{4}$$

$$b = \frac{1 - e^{-4\beta t}}{4}$$

$$c = \frac{1 - 2e^{-2(\alpha+\beta)t} + e^{-4\beta t}}{4}$$

$$M(t) = \begin{pmatrix} a & b & c & b \\ b & a & b & c \\ c & b & a & b \\ b & c & b & a \end{pmatrix}$$

[1] [Swofford/Olson 1990]

Allgemeines Modell

Das allgemeine Modell

$$M' = \begin{pmatrix} \alpha_{aa} & \alpha_{ca} & \alpha_{ga} & \alpha_{ua} \\ \alpha_{ac} & \alpha_{cc} & \alpha_{gc} & \alpha_{uc} \\ \alpha_{ag} & \alpha_{cg} & \alpha_{gg} & \alpha_{ug} \\ \alpha_{au} & \alpha_{cu} & \alpha_{gu} & \alpha_{uu} \end{pmatrix}$$

läßt sich nicht explizit lösen. Es soll später noch gezeigt werden, daß der phylogenetische Abstand zwischen zwei Sequenzen als **gegeben** voraussetzt werden kann, und weiterhin davon ausgegangen werden kann, daß dieser gegebene Abstand mit großer Varianz behaftet ist. Folglich reicht es aus, **die Zeit** *t* **genügend genau zu diskretisieren** und auf eine explizite Integration der Matrix *M'* zu verzichten. Stattdessen berechnet man eine Matrix $M_{t\text{-}min}$ für eine sehr kleine Zeitdifferenz *t-min*. Alle anderen Matrizen der Zeiten *n*t-min* erhält man durch *n*–fache Multiplikation dieser Basismatrix.

Durch einen geschickten Algorithmus, der auf der Binärdarstellung der Zahl *n* beruht, läßt sich der Rechenaufwand bei der *n*–fachen Multiplikation sehr klein halten, was am folgenden Beispiel aufgezeigt werden soll:

Die Zeit zwischen [0..2] soll in 2000 gleiche Schritte zerteilt werden. *M(t=0.160)* ist dann

$$M\left(\frac{2}{2000}\right)^{160} = M(0.001)^{160} = M^{32*(1+4)} = M^{32} * \left(M^{32}\right)^4$$

wobei $M^{(2^x)}$ durch einfaches Potenzieren der Matrix berechnet werden kann.

In der Praxis verwendet man für jede Spalte des Alignments eine eigene Mutationsrate *r*. Die Matrix *M(t,r)* berechnet sich dann aus *M(t*r)*, d.h. eine höhere Mutationsrate wird rein rechnerisch gleichgesetzt mit einer schneller ablaufenden Zeit. Da nun je nach Mutationsrate eine unterschiedliche Matrix *M(t*r)* benötigt wird, berechnet man einfach im Vorfeld für jeden möglichen Wert *r*t* die Matrix *M*.

Mit dieser Methode nimmt man bewußt kleine Fehler bei der Auswertung der Distanz zweier Sequenzen in Kauf, dies ist jedoch unumgänglich, da durch die Einfachheit und Schnelligkeit der Algorithmen erst die notwendige Geschwindigkeit erreicht werden kann, die für interaktives Arbeiten notwendig ist.

VII.3.3 Likelihood eines gegebenen Baumes

Ist ein Alignment von Sequenzen gegeben sowie der dazugehörige phylogenetische Baum mit Astlängen, läßt sich die Likelihood dieses Baumes berechnen. Dazu geht man nach folgendem Schema vor:

1. In dem ungewurzelten phylogenetischen Baum (drei Söhne je innerer Knoten) wird willkürlich ein an ein Blatt angrenzender innerer Knoten zur Wurzel erklärt, um einen leichter verrechenbaren Binärbaum zu generieren. Die Position der Wurzel hat keinen Einfluß auf das Ergebnis des Algorithmus, sehr wohl aber auf die Rechengeschwindigkeit. Aus diesem Grund soll auch die Wurzel dynamisch im

Baum verschoben werden, um möglichst schnelle Antwortzeiten des Algorithmus zu erreichen:

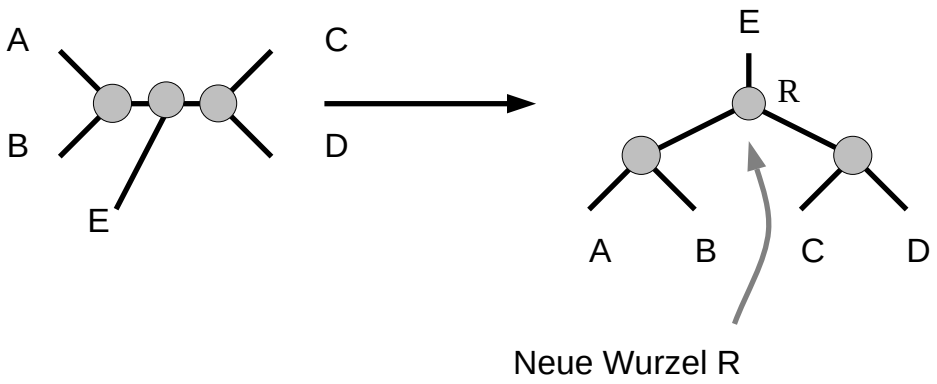
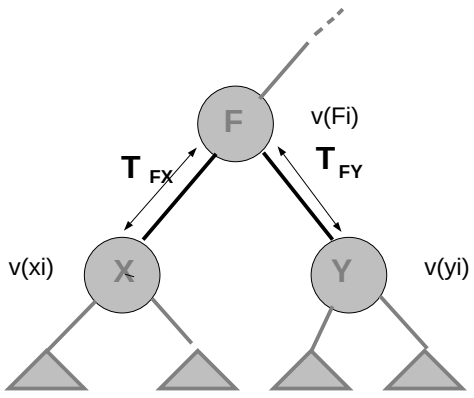


Abb. 24 Setzen der Wurzel in einem ungewurzelten Baum:

2. Jede Base *B* einer Sequenz *S* wird durch einen Vektor *v* repräsentiert, der die Wahrscheinlichkeiten des Auftretens einzelner Basen bzw. das Fehlen einer Base (GAP) angibt, also **A**denin wird durch den Vektor $v_a = \begin{pmatrix} 1.00 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, **C**ytosin durch $v_c = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, etc, repräsentiert.
3. Ausgehend von den Blättern des Baumes werden rekursiv die inneren Likelihood-Vektoren des Baumes rekonstruiert:



Sei:

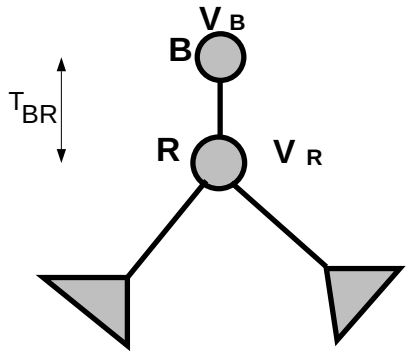
- *X,Y* zwei benachbarte Knoten des Baumes *B* mit dem Vater *F*.
- v_{xi} der Vektor, der die Auftretenswahrscheinlichkeit der Basen des Knotens *X* an der Sequenzposition *i* angibt, sowie v_{yi} der Vektor des Knotens *y* derselben Sequenzposition.
- t_{fx} die Länge der Kante zwischen *x* und seinem Vater *f*, sowie t_{fx} die Länge der entsprechenden anderen Kante. Diese Länge entspricht der Zeit, die für eine

Mutation der Sequenzen zwischen den entsprechenden Knoten zur Verfügung stand.

Dann berechnet sich v_{fi} als Zeilenprodukt der um den Abstand T_{xf} bzw. T_{yf} korrigierten Vektoren v_{xi} und v_{yi}:

$$v_{fi}[j] = \big(M\big(t_{xf}\big)v_{ai}\big)[j] * \big(M\big(t_{yf}\big)v_{bi}\big)[j]$$

4. Ist diese rekursive Berechnung abgeschlossen und hat als letzten Vater den Wurzelknoten *R* erreicht, so bleibt als letzte zu berechnende Einheit noch der Abstand zwischen *R* und dem letzten zu *R* gehörigen Blatt *B*.



Die Likelihood L_i der Spalte *i* aus dem Vektor v_{ri} des Wurzelknotens *R* und der letzten Sequenz v_{bi} wie folgt berechnet werden:

$$L_i = (M(t_{br})v_{ri}) * v_{bi}^T$$

wobei v^T die Transponierte von *v* ist.

5. Die Gesamtlikelihood berechnet sich als das Produkt der Likelihoodwerte der einzelnen Spalten:

$$L = \prod_{i=0}^{i<Alignmentlaenge} L_i$$

VII.3.4 Einfluß einzelner Basen auf die Likelihood des Baumes

Angenommen, in einem Datensatz mit 10000 Sequenzen tritt an einer bestimmten Alignmentposition immer die Base **A**denin auf, bis auf eine einzige Sequenz $S_?$. Diese hochkonservierte Position führt konsequenterweise zu einem sehr niedrigen Mutationsratenkoeffizienten *r*. Stünde anstelle der abweichenden Base ebenfalls ein Adenin, so würde sich die Gesamtlikelihood des Baumes drastisch vergrößern. Generell kann also gesagt werden:

Je mehr sich die Gesamtlikelihood durch die Veränderung einer Base erhöhen läßt, desto unwahrscheinlicher[1] ist das Auftreten dieser Base und desto höher ist die Wahrscheinlichkeit eines Sequenzier- oder Alignmentfehlers.

Um diese mögliche Erhöhung des Likelihoodwertes *dH* durch Veränderung einer Base einer Sequenz *B* an der Position *i* zu berechnen, werden folgende Aktionen durchgeführt:

- Der innere Knoten des Baumes, der direkt mit einer Kante mit dem Blatt *B* verbunden ist, wird zur Wurzel gemacht:

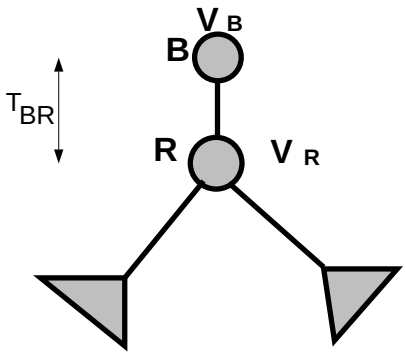


Abb. 25 Verschieben der Wurzel in die direkte Nachbarschaft der zu untersuchenden Sequenz *B*

- Es wird ein Hilfsvektor v_{ci} eingeführt:

$$v_{ci} = M(t_{br})v_{ri}$$

wobei t_{br} die Länge der Kante zwischen dem Blatt *B* und der angrenzenden Wurzel *R* ist.

- Dann berechnet sich *dH* aus

$$dH = \frac{\max_j (v_{ci}[j])}{v_{ci} * v_{bi}}$$

wobei der Zähler der größte auftretende und der Nenner der konkret angenommene Wert des Vektors v_{ci} ist, d.h. durch eine entsprechende Mutation der Base v_{bi} kann eine Likelihooderhöhung um den Faktor *dH* erreicht werden.

[1] Und damit umso größerer mathematischer Informationsgehalt

VII.3.5 Einfluß von Sequenzabschnitten auf die Likelihood des Baumes

Die Suche nach Stellen im Alignment, die die Likelihood des zugehörigen Baumes erhöhen können, kann auch auf ganze Bereiche ausgedehnt werden. Dazu werden die *dH*–Werte eines Sequenzabschnittes für jede Alignmentposition über den betrachteten Bereich geometrisch gemittelt und diese Mittelwerte der einzelnen Sequenzen miteinander verglichen

- Man berechnet für jede Sequenz *S* und jede Position *i* im Alignment den *dH(S,i)*–Wert.
- Das Alignment der Länge *n* wird in *m* gleiche Teile der Länge *n/m* geteilt.
- Über alle Positionen *i* in einem Sequenzabschnitt *j* mit *i>=j*n/m); i<(j+1)*n/m* wird das geometrische Mittel der *dH*–Werte berechnet, sinnvollerweise werden dabei nur die Positionen berücksichtigt, an denen die Sequenz keine unsequenzierten Bereiche aufweist:

$$dH_M(S,j) = \sqrt[n/m]{\prod_{i=(j*n/m)}^{i<(j+1)*n/m} dH(S,i)}$$

  Es wird die mögliche Verbesserung der Likelihoodwerte über ganze Sequenzbereiche gemittelt.
- Schließlich wird das geometrische Mittel aller $dH_M(S,j)$–Werte desselben Sequenzabschnittes *j* über alle Sequenzen des Alignments gebildet:

$$dH_M(j) = \sqrt[|MS|]{\prod_{s \in MS} dH_M(S,j)}$$

  MS sei dabei die Menge aller im Baum vertretenen Sequenzen. Damit kann $dH_M(S,j)$ in Relation zum allgemeinen Mittelwert $dH_M(j)$ gesetzt werden und man erhält **einen Vergleich zwischen der möglichen Likelihoodverbesserung einer konkreten Sequenz und der mittleren Likelihoodverbesserung aller Sequenzen**. Um die Normaldarstellung der Likelihood in Logarithmusform zu erreichen, wird dieser Quotient nun noch logarithmiert:

$$K(S,j) = log \frac{dH_M(S,j)}{dH_M(j)}$$

  Ist das Alignment einer Sequenz in einem bestimmten Bereich fehlerhaft, so drückt sich dies durch in einem signifikant hohen Quotienten aus.
- Eine weitere Möglichkeit der Auswertung besteht darin, nicht über jeweils einen Sequenzabschnitt des gesamten Alignments zu mitteln, sondern das geometrische Mittel verschiedener Abschnitte derselben Sequenz *S* zu betrachten. Die Formeln und auch die Aussage des Wertes K(j,S) entsprechen der des vorherigen Absatzes. In der jetzigen Implementierung von ARB wurde dieser zweite Ansatz gewählt.

Jede Sequenz *S* der Länge *n* wird im Zuge dieses Verfahrens in eine Sequenz aus $K(j,S)_{j=0..m-1}$ der Länge *m* konvertiert. Diese *K(j,S)*–Werte lassen sich normieren, indem sie durch ihre stochastische Varianz dividiert werden.[1] Die normierten *K(j,S)*–Werte werden in der aktuellen Version von ARB schließlich nach folgender Regel in symbolhafte Ziffern umgewandelt:

Wertebereich	Ergebnis
falls abs(K(j,S)/Varianz) < 0.5	'-'
sonst	'5' + round(K(j,S)/Varianz)

D.h. Zahlen kleiner als 5 weisen auf eine niedrige Rate außergewöhnlicher Mutationen hin, Zahlen größer als 5 geben eine hohe Rate an. Das Ergebnis ist ein kurzer Ziffern-String, der die Situation der Sequenz in einer leicht interpretierbaren Form wiederspiegelt.

VII.3.6 Realisierung

Übersicht

Die nun bis zu diesem Punkt entwickelte Theorie ist sinnvoll und nützlich, doch grau ist alle Theorie, wenn sie nicht sinnvoll in die Praxis umgesetzt werden kann. Sinnvoll soll in diesem Falle heißen, daß interaktives Arbeiten auch mit vielen Sequenzen nicht durch viele große Wartezeiten unterbrochen werden sollte. Ziel einer brauchbaren Realisierung mußte also sein, ein gleichmäßiges Arbeiten ohne nennenswerte Zeitverzögerung zu ermöglichen (**Online Statistik)**. Dies war alles andere als eine triviale Aufgabe, da bereits für die Berechnung eines einzigen *dH*–Wertes einer einzelnen Base die Likelihood des gesamten Baumes berechnet werden muß.

Aus diesem Grund lohnt es sich, noch einen kurzen Blick auf die konkrete Umsetzung der Algorithmen zu werfen.

Die entscheidende Idee bei der Implementierung war, benötigte Informationen immer nur bei tatsächlichem Bedarf zu berechnen. Um unnötige Mehrfachberechnungen zu vermeiden, wurden Cache-Strategien entwickelt, die die Basis der Implementierung bilden. Im weiteren wurden die einzelnen Algorithmen in einen objektorientierten Sequenzeditor integriert, dessen Arbeitsweise im nächsten Kapitel erklärt werden soll.

Numerik

Bei der Berechnung der inneren *v*–Vektoren zur Bestimmung der Likelihood eines Baumes ergeben sich extrem kleine Werte.

[1] Die biologische Varianz der *K(j,S)*–Werte ist von einer Vielzahl von unbekannten Werten abhängig. In der jetzigen 'Test'-Implementierung wurde als Varianz die aus der Stochastik bekannte Formel $Varianz = \frac{\sum K^2(j) - \left(\sum K(j)\right)^2}{\sqrt{m}}$ verwendet.

Beispiel: Gegeben sind zwei Sequenzen *X* und *Y* mit den Basen **A** und **C** an einer bestimmten Alignmentposition *i*. Sie haben einen gemeinsamen Vater *F*, dessen v_F berechnet werden soll. Die Astlängen zwischen *X* und *F* sowie *Y* und *F* sind nahezu Null, d.h. *M(t)* hat die folgende Form:

$$\begin{pmatrix} 1-3a & a & a & a \\ a & 1-3a & a & a \\ a & a & 1-3a & a \\ a & a & a & 1-3a \end{pmatrix}$$

mit *a->0*$^+$. v_F hat folgendes Aussehen:

$$v_f = \begin{pmatrix} (1-3a)a \\ (1-3a)a \\ aa \\ aa \end{pmatrix}$$

Für sehr kleine *a* geht also auch die Länge des Vektors v_f gegen Null.

Je weiter entfernt von den Blättern *v*–Vektoren berechnet werden, desto kleiner wird deren Betrag. Schon bei relativ kleinen Bäumen sinkt dieser Wert schnell unter die von normalen Maschinenzahlen darstellbaren Wertebereich. Es bieten sich im wesentlichen zwei Lösungen an:

- Man geht zum Logarithmus der Zahlen über: Multiplikationen werden zu Additionen. Dies hat den Nachteil, daß in den entsprechenden Algorithmen sehr häufig die langsame Logarithmusfunktion aufgerufen werden muß, was einen entscheidend negativen Einfluß auf die Rechenzeiten hat.
- Man ersetzt jeden Vektor *v* durch einen korrigierten Vektor *v'* und einen Skalar *s*, so daß für das Produkt von *s* und *v'* gilt:$v = s*v'$. Dieser Skalar soll ausschließlich Potenzen von 1/2 als Werte annehmen und sich dadurch als Exponent zur Basis 2 speichern lassen. Verwendet man ein 32Bit–System zu seiner Speicherung, wird der Zahlraum des Vektors etwa um den Faktor $2^{2.000.000.000}$ erweitert. Durch diesen Trick kann der Betrag von *v'* im maschinendarstellbaren Bereich gehalten werden.
  Bei dieser Lösung muß von Zeit zu Zeit der Betrag des Vektors *v'* überprüft werden. Unterschreitet dieser den Wert *0.5*, so wird der Exponent des Skalars um 1 erniedrigt und der Vektor *v'* mit dem Skalar 2.0 multipliziert.

Praktische Anwendungsüberlegungen

Als Vorüberlegungen für die Entwicklung konkreter Algorithmen zur Online-Darstellung der berechneten Likelihood-Daten auf dem Bildschirm, ist es sinnvoll, ein paar theoretische Abschätzungen durchzuführen. Diese Abschätzungen basieren auf dem Stand der technischen Gegenheiten im Sommer 1997:

- Es sollen 2000 Sequenzen gleichzeitig editiert werden.
- Die Alignmentlänge soll 10000 betragen.
- Eine typische Workstation berechnet effektiv 100 Millionen Fließkomma-Operationen pro Sekunde,
- und ist mit 128 MegaByte Hauptspeicher ausgestattet, wovon maximal die Hälfte für den Sequenzeditor zur Verfügung stehen.

Unabhängig von der technischen Entwicklung lassen sich folgende Konstanten festlegen:

- Zur Speicherung eines *v*–Vektors werden bei niedriger Maschinenauflösung 4+4*5 = 24 Bytes benötigt. (4 Byte Skalar, 4 Bytes pro *float*–Wert für jeweils **ACGT**-).
- Für einen Baum mit *g* Blättern müssen *g-1* innere Knoten berechnet werden.

Mehrere Untersuchungen des typischen Benutzerverhaltens brachten als Ergebnis, daß folgende Aktivitäten hauptsächlich durchgeführt werden:

- Vertikales Scrolling: Wird der auf dem Bildschirm sichtbare Alignmentausschnitt nach rechts oder links verschoben, muß am Rand ein kleiner Streifen neu berechnet und gezeichnet werden. Dieser Streifen ist meist nur ein oder zwei Zeichen breit.
- Horizontales Scrolling: Wird der sichtbare Ausschnitt nach oben oder unten bewegt, müssen wenige Sequenzabschnitte der sichtbaren Länge neu dargestellt werden.
- Wird eine Sequenz editiert, muß diese oft innerhalb kurzer Zeit neu dargestellt werden.

Dabei ist zu beobachten, daß eine Aktivität meist eine bestimmte Zeit lang durchgeführt wird. Ist der Benutzer gerade dabei, eine Sequenz zu verändern, wird mit hoher Wahrscheinlichkeit auch seine nächste Aktion eine Editoroperation sein.

Prinzipiell läßt sich der *dH*–Wert für jede Alignmentspalte einzeln berechnen, dies hat aber zur Folge, daß der Rechner mehr Zeit damit verbringt, die gesamte Baumhierarchie zu durchlaufen und die entsprechenden Daten zu sammeln, als er sich mit der eigentlichen Aufgabe, nämlich der Bestimmung der Likelihoodwerte, beschäftigt.[1] Deshalb geht man dazu über, jeweils für kurze Spalten-Bereiche die Likelihood auszurechnen.

Implementierung des Algorithmus zur Visualisierung einzelner auffälliger Basen

Die praktische Umsetzung basiert im wesentlichen auf folgenden Ideen:

- **Datensammlung**: Alle notwendigen Sequenzen und phylogenetischen Informationen müssen erfaßt werden. Die Sequenzen werden dabei **nicht** gelesen, sondern nur eine Referenz auf die Datenbank vermerkt, d.h. man überläßt es der Datenbank, die Sequenzen für die interaktive Darstellung schnell genug bereitzustellen.
- **Integration der Algorithmen** in einen Editor: Der Editor soll **dynamisch** bei Bedarf die notwendigen Informationen eines darzustellenden Teilabschnittes anfordern. Erst dann sollen die *dH*–Werte berechnet werden.
- Zusätzlich zu den geforderten Daten werden vorrausschauend (**optimistisch**) die nächsten angrenzenden Spalten mitberechnet, da davon ausgegangen wird, daß diese in der nächsten Operation benötigt werden. Dabei werden, soweit möglich, alle schon einmal durchgeführten Berechnungen zwischengespeichert.
- **Dynamische** Beobachtung der Datenbank mit Hilfe von **Triggerfunktionen**, um möglicherweise interne Zwischenspeicher löschen zu können.

[1] Eine falsch vorhergesagte ’wenn’ ’dann’ Anweisung kostet bei modernen Prozessoren (Pentium Pro) oft mehr als 10 arithmetische Fließkommaoperationen.

Damit ergibt sich folgender Informationsfluß:

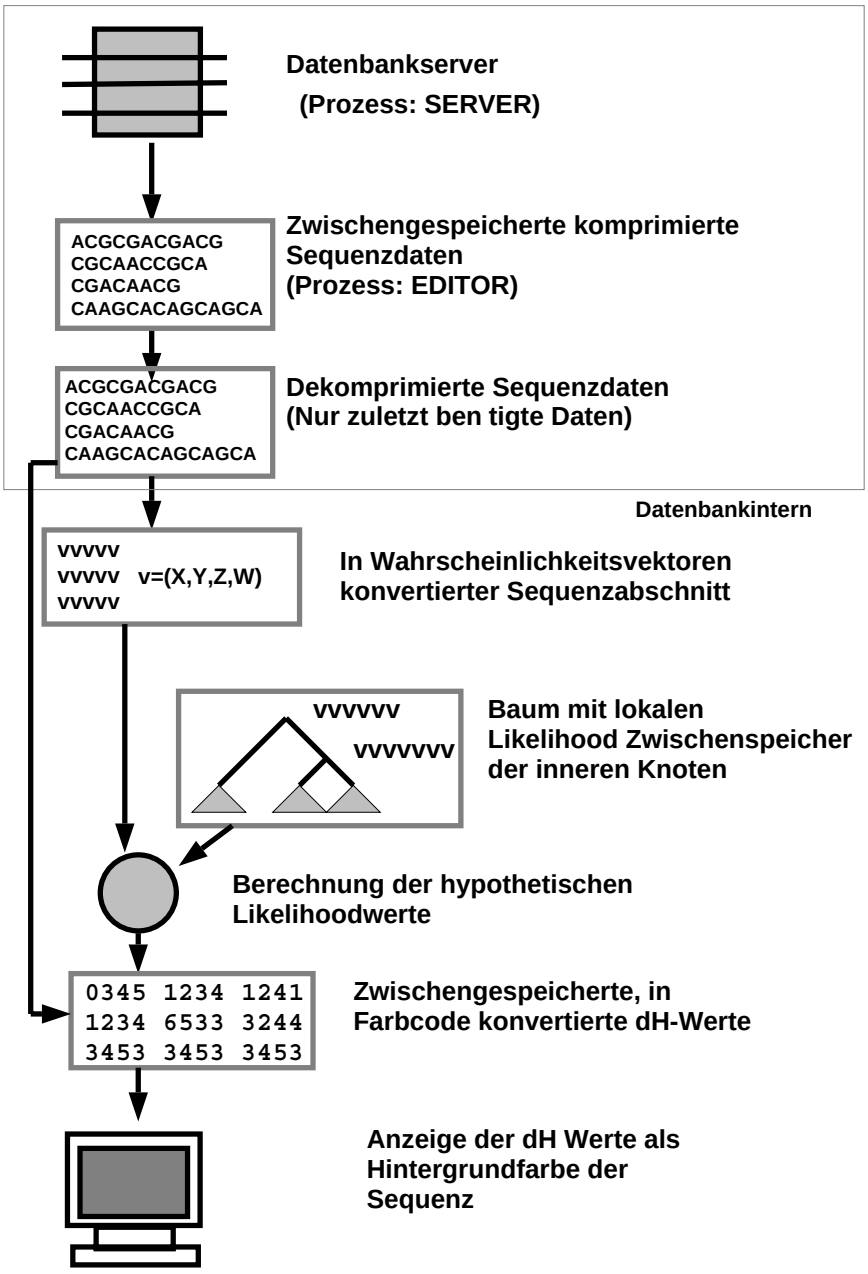


Abb. 26 Informationsfluß bei der Darstellung der *dH*–Werte in einem Sequenzeditor

Da von allen Teilalgorithmen die Information erst auf Anforderung bereitgestellt wird, soll der Algorithmus für die Berechnung der Likelihoodwerte ausgehend von der obigen Darstellung beschrieben werden:

Bereitstellung der Visualisierungsinformation der *dH*–Werte

Ziel ist es, hohe hypothetische Likelihoodverbesserungen *dH* durch Austausch einzelner Basen dazu zu verwenden, die entsprechenden Positionen im Alignment **auffallend zu markieren**. Es hat sich dabei als praktikabel herausgestellt, den Hintergrund der entsprechenden Base farbig einzufärben:

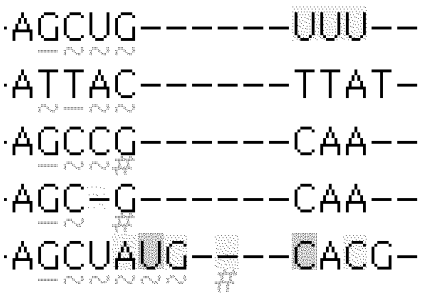


Abb. 27 Einfärben des Hintergrundes auffälliger Basen

Sinnvollerweise verwendet man mehrere Farben: je größer der *dH*–Wert, desto auffälliger der Hintergrund. In der jetzigen Implementierung werden **zehn verschiedene Farbabstufungen** eingesetzt, doch hat es sich in der Praxis gezeigt, daß diese von einem Benutzer optisch nicht mehr sauber voneinander getrennt werden können, so daß auch eine geringere Anzahl ausreichend gewesen wäre.

Da die meisten Benutzer länger mit einem Alignment arbeiten, ist es sinnvoll, alle schon einmal berechneten *dH*–Werte zwischenzuspeichern (cachen), um auf eine erneute Berechnung verzichten zu können. Es wird allerdings nicht der *dH*–Wert, sondern nur die sich daraus ergebende Hintergrundfarbe gespeichert. Somit reicht **ein Byte zur Speicherung des Farbwertes** aus. Bei den geforderten 2000 Sequenzen a 10000 Position hätte dies einen Speicherbedarf von 20 MegaByte zur Folge, was durchaus akzeptabel ist.

Es läge nahe, einen bestimmten Byte-Wert für 'noch nicht berechnet' zu definieren. Damit könnte dann sehr einfach ermittelt werden, welche Positionen noch eine *dH*–Wert Berechnung benötigen. Dies hieße aber, daß bei einer Sequenzänderung alle schon berechneten Werte durch diesen Wert ersetzt werden müßten. Bei 20 MegaByte Farbspeicher hieße dies, daß bei jeder interaktiven Änderung der Sequenz 20 MegaByte gelöscht werden müßten, was eine nicht akzeptable Verzögerung ergäbe. Deshalb wird zusätzlich zur Hintergrundfarbe eine **virtuelle Uhrzeit gespeichert**, die angibt, wann zum letzten Mal der Farbwert berechnet wurde. Vergleicht man nun diese Zeit mit der der letzten Sequenzänderung, erhält man die gewünschte Aussage. Als virtuelle Zeit bietet es sich an, nicht auf eine echte Uhrzeit zurückzugreifen, sondern den Transaktionszähler der Datenbank zu verwenden. Dieser läßt sich bequem in einer 32 Bit breiten Speicherzelle speichern.

Speichert man für jeden Farbwert seine letzte Berechnungsuhrzeit, benötigt man für die 2000 Sequenzen jetzt aber 100 MegaByte Hauptspeicher, ein Wert, der über der 64 MegaByte-Grenze liegt, die von der technischen Ausstattung vorgegeben ist. Daher liegt es nahe, diese Uhrzeit für jeweils eine **kleine Gruppe von Alignmentpositionen** zu speichern. Dazu wird das Alignment in kleine Bereiche der Länge *ax* (in der momentanen Implementierung ist diese Länge auf 16 eingestellt) aufgeteilt. Alle Berechnungen, die auf Alignmentabschnitten arbeiten, werden nur noch auf diesen Bereichen durchgeführt: werden zum Beispiel die Hintergrundfarbinformationen für die Positionen 2 bis 12 benötigt, werden dennoch die Positionen 0 bis 15 berechnet (falls *ax*=16). Dieses Vorgehen hat drei wesentliche Vorteile:

- Der **Ressourcenverbrauch** zur Speicherung der Zeit der letzten Berechnung **verringert** sich um den Faktor *ax*.
- Durch Begrenzen der Anzahl von Positionen, die zu berechnen sind, wird der **algorithmischen Overhead**, der sich durch Baummanipulationen ergibt, begrenzt.

- Wird das Alignment horizontal verschoben, sind mit hoher Wahrscheinlichkeit die **nächsten Positionen** innerhalb der schon sichtbaren Alignmentgruppen **berechnet**.

Diese Art der Implementierung hat allerdings ein etwas ungleichmäßiges vertikales Scrolling zur Folge, was aber bei entsprechend schnellen Rechnern keinen allzu störenden Einfluß hat.

Berechnung der Likelihoodänderungen *dH*

Zur Berechnung der *dH*–Werte eines Abschnittes einer Sequenz *S* müssen im wesentlichen folgende Schritte durchgeführt werden:

- Die **Wurzel des Baumes** muß auf die Kante, die mit der Sequenz *S* verbunden ist, **gesetzt** werden.
- Ausgehend von den Blättern (=**bottom up**) werden dann die Likelihoodwerte der **inneren Knoten berechnet**.
- Dann können die *dH*–Werte als einfache **Quotienten** des jeweils **maximalen** Zeilenwertes des Likelihoodvektors der Wurzel durch den jeweils **tatsächlichen** Zeilenwert berechnet werden.

Es hat sich in der Praxis herausgestellt, daß sich die **Reihenfolge** der Sequenzen in der Datenbank und damit im Sequenzeditor an der **Phylogenie orientiert**. Konkret bedeutet dies, daß nah verwandte Organismen auch direkt nacheinander abgefragt werden. Beläßt man also die Wurzel des Baumes nach einer Berechnung dort, wo sie gerade plaziert wurde, muß diese bei der nächsten Berechnung mit hoher Wahrscheinlichkeit nur um wenige innere Knoten verschoben werden.

Weiterhin gilt natürlich die triviale Feststellung, daß, falls sich ein Teilbaum nicht ändert, sich auch die Likelihoodvektoren an dessen Wurzel nicht ändern. Deshalb werden berechnete Likelihoodvektoren nicht verworfen, sondern bei den entsprechenden Knoten zwischengespeichert (**Likelihood-Vektoren-Cache**). **Verschiebt** man die **Wurzel**, müssen nur alle inneren Likelihoodvektoren neu berechnet werden, die auf dem **Pfad zwischen neuer und alter Wurzelposition** lagen:

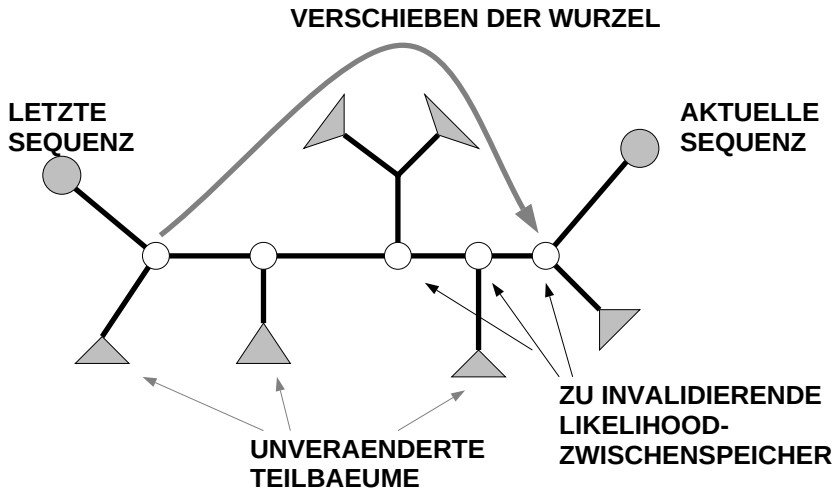


Abb. 28 Invalidierung innerer Likelihoodvektoren-Zwischenspeicher beim Verschieben der Wurzel:

Um nicht jedesmal vom Betriebssystem Speicher für die inneren Likelihood-Vektoren anfordern zu müssen, wird ein **kleiner Speicherbereich** bei jedem Knoten für diese Vektoren fest eingerichtet. Dieser Bereich muß nur der einzigen Anforderung genügen, größer als die Länge des maximal angeforderten Alignmentauschnittes zu sein. Am einfachsten wäre es natürlich, seine Größe der Länge des Alignments anzugleichen, allerdings hätte dies unsinnig viel Hauptspeicherverbrauch zur Folge. Weiterhin speichert ein **zusätzliches Flag**, ob die Likelihood-Werte in einem Knoten gültig sind. Damit müssen ausgehend von den Blättern nur noch die inneren Likelihood-Vektoren berechnet werden, die als ungültig markiert sind. Mit Hilfe dieser Techniken kann der Berechnungsaufwand weitestgehend minimiert werden, und zwar soweit, daß selbst mit einfachen Rechnern (Pentium PCs) ein interaktives Arbeiten ohne nennenswerte Verzögerung möglich ist. Durch diese Art der Implementierung ist die Berechnungsgeschwindigkeit unabhängig von der Länge des Alignments und somit auch für extrem lange Gene einsetzbar.

Bereitstellung der Sequenzinformation

Zur Darstellung von Sequenzen muß als Grundvoraussetzung selbstverständlich zunächst einmal die **Sequenzinformation verfügbar** sein. Im Gegensatz zur herkömmlichen Vorgehensweise wird bei ARB nicht die Sequenz, sondern nur ein **Datenbankverweis auf eine Sequenz gespeichert**. Jedesmal, wenn die Information benötigt wird, wird eine Datenbanktransaktion geöffnet und die Sequenz aus der Datenbank gelesen. Dies bietet eine ganze Reihe von Vorteilen:

- Änderungen der Sequenz in der Datenbank müssen nicht explizit ausgewertet werden.
- Die Datenbank stellt leistungsfähige **Kompressionsalgorithmen** zur Verfügung, die dafür sorgen, daß der Editor mit minimalem Ressourcenverbrauch arbeitet.
- Die Datenbank stellt leistungsfähige **undo**/**redo** Funktionen zur Verfügung, eine Funktionalität, die sonst nur mit hohem Aufwand implementierbar wäre.

Diese Vorgehensweise stellt höchste **Geschwindigkeitsanforderungen** an die Datenbank. Zehntausend und mehr Lesezugriffe pro Sekunde sind keine Seltenheit, eine Anzahl, die nur mit Hilfe von leistungsfähigen Cache-Strategien erreicht werden kann. Dazu besitzt die Datenbank einen **lokalen Speicher**, der die **dekomprimierte Sequenzinformation** bereitstellt. Zur Berechnung eines *dH*–Wertes einer Spalte wird jedoch die Spalteninformation aller Sequenzen benötigt. Müssen hierzu jedoch alle Sequenzen dekomprimiert werden, sinkt die Performance des Editors um Größenordnungen ab. Aus genau diesem Grund **muß der lokale Zwischenspeicher für dekomprimierte Daten der Datenbank mindestens die Größe aller Sequenzdaten haben.** Dadurch verliert man zwar den Vorteil des äußerst geringen Hauptspeicherbedarfs des Editors, für die interaktive Bedienbarkeit ist dieses Vorgehen jedoch unumgänglich.

Implementierung des Algorithmus zur Bewertung von Sequenzabschnitten

Bei der Bewertung ganzer Alignmentabschnitte werden so viele Rechenoperationen benötigt, daß **interaktives Arbeiten** nicht mehr in Betracht gezogen werden kann. Ist man von dieser

Forderung einmal abgerückt, so ist die benötige Rechenleistung nur noch sekundär. Somit kann die Implementierung im wesentlichen der schon beschriebenen Theorie folgen. Allerdings hat die Praxis gezeigt, daß die Berücksichtigung folgender Punkte den **Ressourcenverbrauch**, d.h. Hauptspeicher und Rechenleistung, **minimieren** kann:

- **Unterteilt man die Länge des Alignments** in Abschnitte konstanter Länge *l* und berechnet jeweils für alle Sequenzen die *dH*–Werte eines Abschnitts, müssen nur *l* Vektoren in jedem inneren Baumknoten zwischengespeichert werden; der Hauptspeicherbedarf kann klein gehalten werden.
- **Sortiert man die Sequenzen anhand der Topologie** des Baumes und berechnet die *dH*–Werte jeweils für die sortierte Reihenfolge der Sequenzen, sinkt die benötigte Rechenleistung, da die Wurzel im Baum immer nur über wenige innere Knoten hinweg verschoben werden muß. In der praktischen Anwendung konnte so eine Rechenleistungssteigerung um den Faktor *4* bis *9.6* gemessen werden.

Messungen ergaben folgende benötige Zeiten zur Bewertung eines 16S–rRNS Alignments.

Rechnerarchitektur	benötige Rechenzeit 430 Sequenzen der Länge 1400	Rechenzeit für 2849 Sequenzen der Länge 1400
PC Cyrix P166+ 133MHz	66.4 sec	n.d.
PC Pentium Pro 180MHz	24 sec	144 sec
Sun Ultra Sparc 143 MHz	43 sec	256 sec
Digitial Alpha 400 MHz	12 sec	102 sec

Bei der in ARB eingesetzten Implementierung bleibt die benötigte Rechenleistung so niedrig, daß genügend **Spielraum** sowohl für **komplexere Analysen** wie auch für den Einsatz bei **größeren Datenmengen** bleibt.

VII.3.7 Ergebnisse und Diskussion

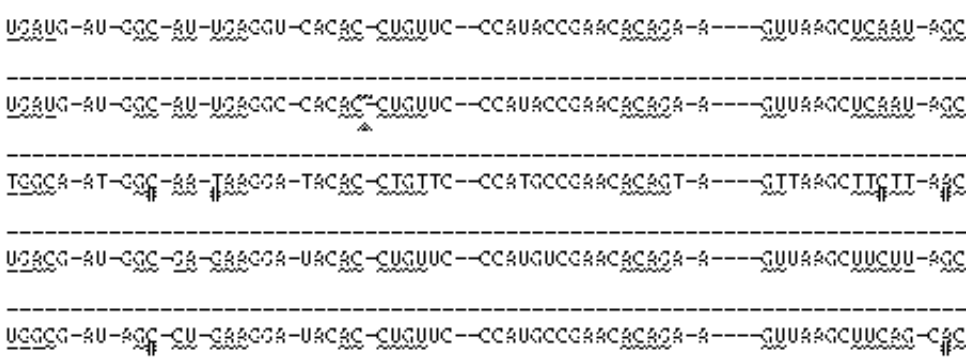
Ziele der Entwicklung und Implementierung der oben angeführten Algorithmen war es, dem Anwender ein effizientes Intrument zur Verfügung zu stellen, ein gegebenes Alignment schnell und einfach auf ’Unstimmigkeiten’ hin zu untersuchen. Dazu wurde die schon bekannte Methode des Maximum-Likelihood Algorithmus so umgewandelt, daß sich durch künstliche Manipulation der Sequenz die ’Unstimmigkeiten’ der Original-Sequenz leicht erkennen lassen. Dieser Algorithmus wurde in zwei Versionen implementiert: einmal zur interaktiven (farbig dargestellten) Bewertung jeder einzelnen Base der Sequenz und als Bewertung ganzer Sequenzabschnitte. Anwender, die nur wenige Sequenzen zu verwalten haben, werden insbesondere mit der interaktiven Form arbeiten; während sogenannte ’Power User’, die große Alignments verwalten, auch das Testen ganzer Alignmentbereiche effektiv einsetzen können, da es einen schnellen Überblick über alle Daten vermittelt.

Immer wieder hat sich gezeigt, daß die **Akzeptanz** einer neuen Funktion sehr davon abhängt, wie einfach sie zu bedienen und wie leicht die Ergebnisse zu interpretieren sind.

’Einfach’ meint in diesem Zusammenhang, daß jeder das Programm bedienen können muß, ohne den dahinterstehenden Algorithmus zu verstehen. Im Fall der interaktiven Alignmentkontrolle besteht die Ausgabe von ARB lediglich in einem farbig markierten Hintergrund im Sequenzeditors. Ein **einfacher Mausklick** auf einen Menüpunkt **genügt**, um die automatischen Likelihood-Schätzungen auf dem Bildschirm sichtbar zu machen. Auch die Interpretation der Ergebnisse liegt auf der Hand: Je dunkler der Hintergrund, desto auffälliger die Base. Erste Erfahrungen mit Anwendern haben gezeigt, daß insbesondere bei der Erstellung eines neuen Alignments dieser Algorithmus auf breite Zustimmung trifft.

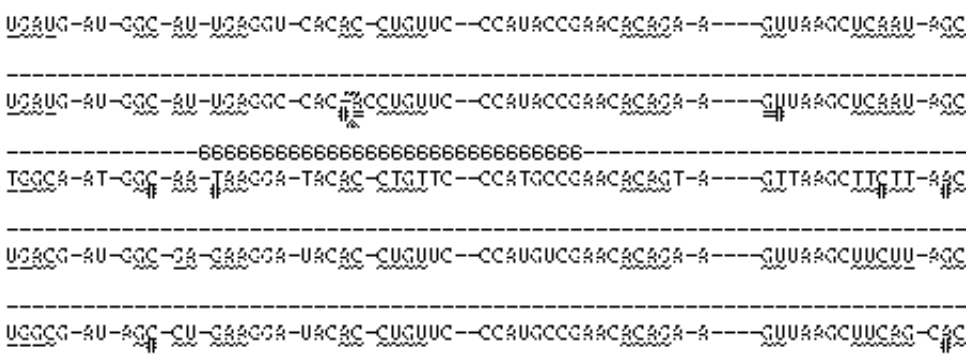
Die Umsetzung der Idee, Likelihoodalgorithmen auf ganze Aligmentbereiche anzuwenden, zeigte schnell, daß damit einfach ein guter Überblick über das Mutationsverhalten einer gesamten Sequenzmenge erreicht werden kann.

An dieser Stelle soll nur an einem kleinen Beispiel die Wirkungsweise dieser Bereichsbewertung demonstriert werden. Die folgende Abbildung zeigt ein von Hand erstelltes gutes Alignment. Die Bereichsbewerung findet keine offenkundigen Mißstimmigkeiten (- – – – Linien unterhalb der Sequenzen)



Um eine schlecht alignte Sequenz zu simulieren, wird nun das Alignment einer einzelnen Sequenz minimal modifiziert und eine neue Bewertung durchgeführt.

Deutlich sieht man, daß die kleine Änderung als signifikant schlechteres Alignment erkannt wurde (666–Zeichen unterhalb der Sequenz).



Im Bereich der Biologie ist diese Art des Einsatzes von Likelihood-Algorithmen vollkommen neu. Da nur bestehende Daten (und Modelle) getestet werden und dem Algorithmus nur eine bewertende Funktion zukommt, ist es leicht, eine Vielzahl von neuen Hypothesen im

Kontext großer Datensätze zu testen. Um eine derartige Testfunktion zu generieren, muß nur die Likelihood-Funktion entsprechend angepaßt werden:

- Geht man dazu über, statt einzelner Basen immer Doubletten sich paarender Basen in der Doppelhelix zu untersuchen, kann die Analyse um Sekundärstrukturinformationen erweitert werden.
- Während bei DNS/RNS die visuelle Kontrolle eines Alignments noch relativ leicht fällt, ist sie bei Aminosäuren aufgrund der Vielzahl unterschiedlicher Aminosäuren (20 Aminosäuren statt 4 Basen) aufgrund der Unübersichtlichkeit relativ schlecht durchführbar. Eine Färbung des Hintergrundes schlecht alignter Bereiche erleichtert hier dem Benutzer die Arbeit ganz wesentlich.

Eine vollständige Analyse aller sich daraus ergebenden Möglichkeiten ist noch nicht abzusehen, so daß hier nur der Grundalgorithmus entwickelt und seine Leistungsfähigkeit nachgewiesen wurde.

VII.4 Der Alignment-Editor ARB_EDIT4

VII.4.1 Aufgabe

Um Sequenzen bearbeiten zu können, bedarf es eines speziellen Sequeneditors. Die Aufgaben dieses Editors sind im wesentlichen:

- Ein Alignment aus Sequenzen darzustellen.
- dem Benutzer zu erlauben, dieses Alignment zu verändern,
- und bei Bedarf unterschiedlichste Informationen zu dem Alignment mit anzuzeigen.

Dabei sollte dieser Editor

- sich nahtlos in das bestehende Programmsystem einpassen,
- sich einfach und schnell bedienen lassen,
- mit vielen Sequenzen zurechtkommen,
- und flexibel erweiterbar sein.

Hinzu kommt, daß in der Vergangenheit der Vorgang des Alignens und der der Sequenzeditierung im Prinzip als unabhängige Vorgänge angesehen wurden. Im Gegensatz dazu sollte hier versucht werden, den mit Hand durchgeführten Vorgang des Alignens mit Hilfe eines Sequenzeditors entscheiden zu verbessern.

VII.4.2 Derzeitiger Stand der Forschung

In regelmäßigen Zeitabschnitten werden weltweit neue Sequenzeditoren programmiert und öffentlich zur Verfügung gestellt. So entstanden im Laufe der Zeit **AE2**, **GDE, ALE, SEQEDIT** und **TKDCSE**. Am Lehrstuhl für Mikrobiologie der TU-München wurden die beiden Programme **EDTU** und **ARB_EDIT** entwickelt. Wozu also brauchte es nochmals einen neuen Editor?

Das gemeinsame Merkmal der oben genannten Programme ist, daß sie

- **eine Liste**
- **von realen**
- **Sequenzen bearbeiten**

Genau diese Eigenschaften sind es, die alle bekannten Editoren für den Einsatz im Rahmen von ARB unzweckmäßig machen:

- **Liste** bedeutet, daß die Datensätze ohne Hierarchie angeordnet sind. Mit steigender Sequenzanzahl wird die Bearbeitung solch ungeordneter Sequenzen immer unhandlicher und fehleranfälliger. Aus diesem Grund wurde für ARB erstmals eine hierarchische Organisation der Sequenzen umgesetzt, die aus der Phylogenie der Organismen generiert wird.
- **Real** bedeutet in diesem Fall, daß alles, was diese Sequenzeditoren darstellen, auch physikalisch gespeichert ist. Will man bei dieser Technik die Darstellung um weitere Zusatzinformationen, z.B. Helixinformationen, erweitern, wird unverhältnismäßig viel Arbeitsspeicher benötigt, wie z.B. bei TKDCSE. Um dem entgegenzuwirken, wurde ARB_EDIT4 so programmiert, daß die darzustellenden Daten erst bei Bedarf erzeugt werden. Für rechenintensive Algorithmen wurde ein Cache implementiert, der es erlaubt, Teile der berechneten Daten zwischenzuspeichern, um sie bei einer erneuten Darstellung nicht neu berechnen zu müssen.
- **Sequenzen** bedeutet, daß die interne Sequenzdarstellung streng mit den graphischen Layoutfunktionen verknüpft sind. So konnte zwar die Programmierung einfach gehalten werden, doch ist dann auch die Darstellung auf Sequenzen beschränkt. In dem neuen Editor sollten die Darstellungsfunktionen von den Inhalten vollkommen getrennt werden. Der neue Sequenzeditor kann daher nicht nur Sequenzen sondern theoretisch jede Art graphischer Information effizient darstellen.

VII.4.3 Sequenz–Organisation

Hierarchie von Sequenzen

Bei der Verwaltung von Sequenzen sollte bei ARB die Phylogenie ins Zentrum der Betrachtung gestellt werden. Logischerweise sollte sich dann auch der Sequenzeditor an der Phylogenie der Organismen orientieren. Dazu mußte die traditionelle hierarchielose Struktur der Sequenzen durch eine hierarchische ersetzt werden. Ein Blick auf einen typischen Stammbaum zeigt, daß Untereinheiten eines Baumes, die bestimmte Gemeinsamkeiten aufweisen, zu einer Gruppe zusammengefaßt werden.

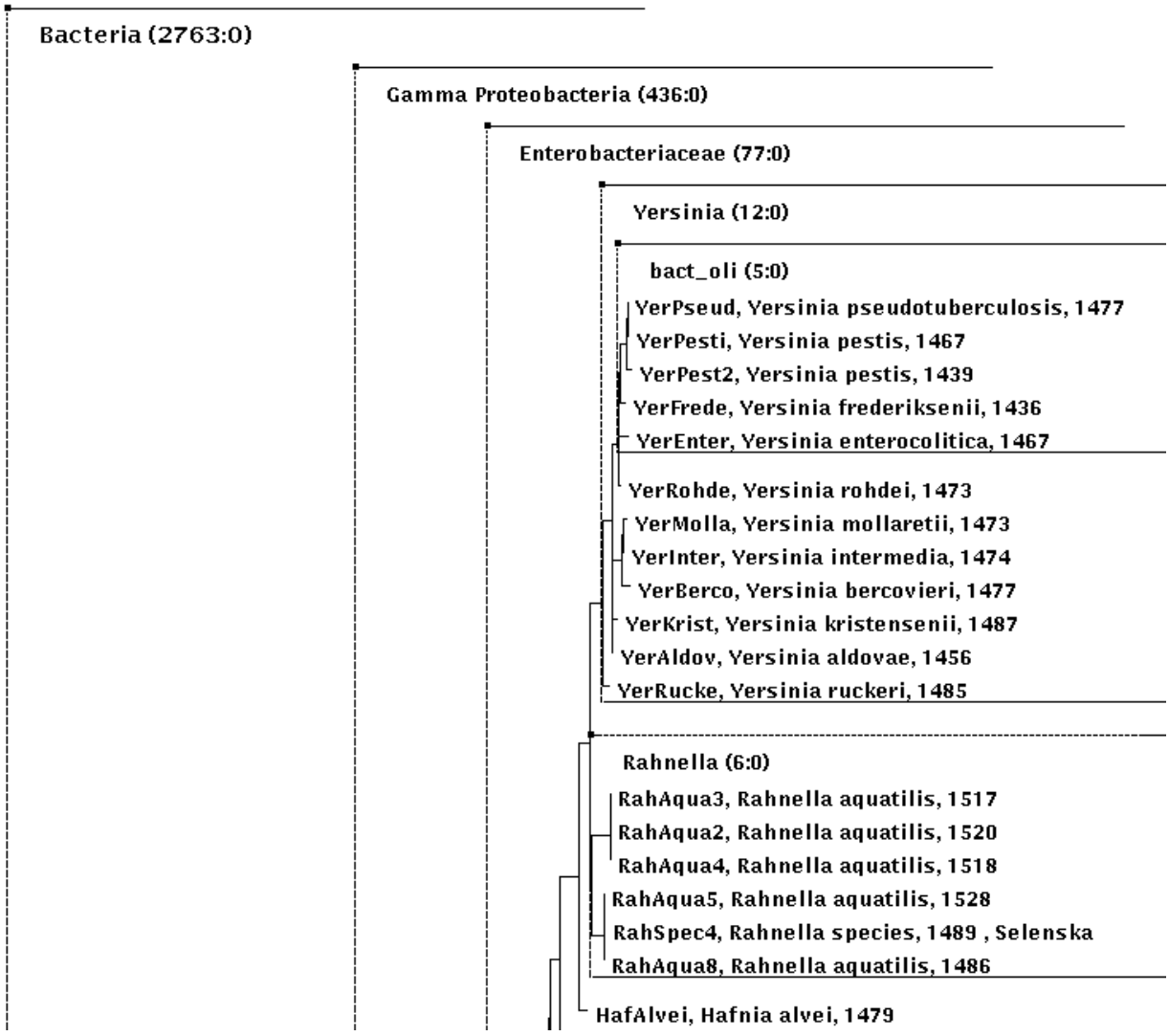


Abb. 29 Typischer Stammbaum

Es macht wenig Sinn, im Editor die Phylogenie zu detailliert anzugeben, vielmehr sollte eine Konzentration auf die Gruppen stattfinden. Außerdem soll die Reihenfolge, mit der die Sequenzen im Editor erscheinen, exakt mit der Reihenfolge der Organismen im Stammbaum übereinstimmen:

Für jede Gruppe von Sequenzen wird nun dynamisch eine Konsensus-Sequenz erstellt. Dies ist eine Sequenz, die sich aus der den jeweils häufigsten Basen ihrer Gruppenmitglieder zusammensetzt. Dieser Konsensus steht gewissermaßen symbolisch für alle Sequenzen seiner Gruppe und kann auch so behandelt werden. Ändert ein Benutzer den Konsensus mit Hilfe der Maus und Tastatur, ändert er automatisch alle Sequenzen seiner Gruppe. Dadurch erreicht man, daß die Anwender auf einem höheren Abstraktionsniveau als mit Einzelsequenzen arbeiten können. In vielen Fällen ist man sogar an der Information der Einzelsequenzen einer Gruppe nicht mehr interessiert, der Konsensus reicht für die weitere Verarbeitung vollkommen aus. Aus diesem Grund erlaubt der Editor das **Zusammenfalten** von Sequenzgruppen:

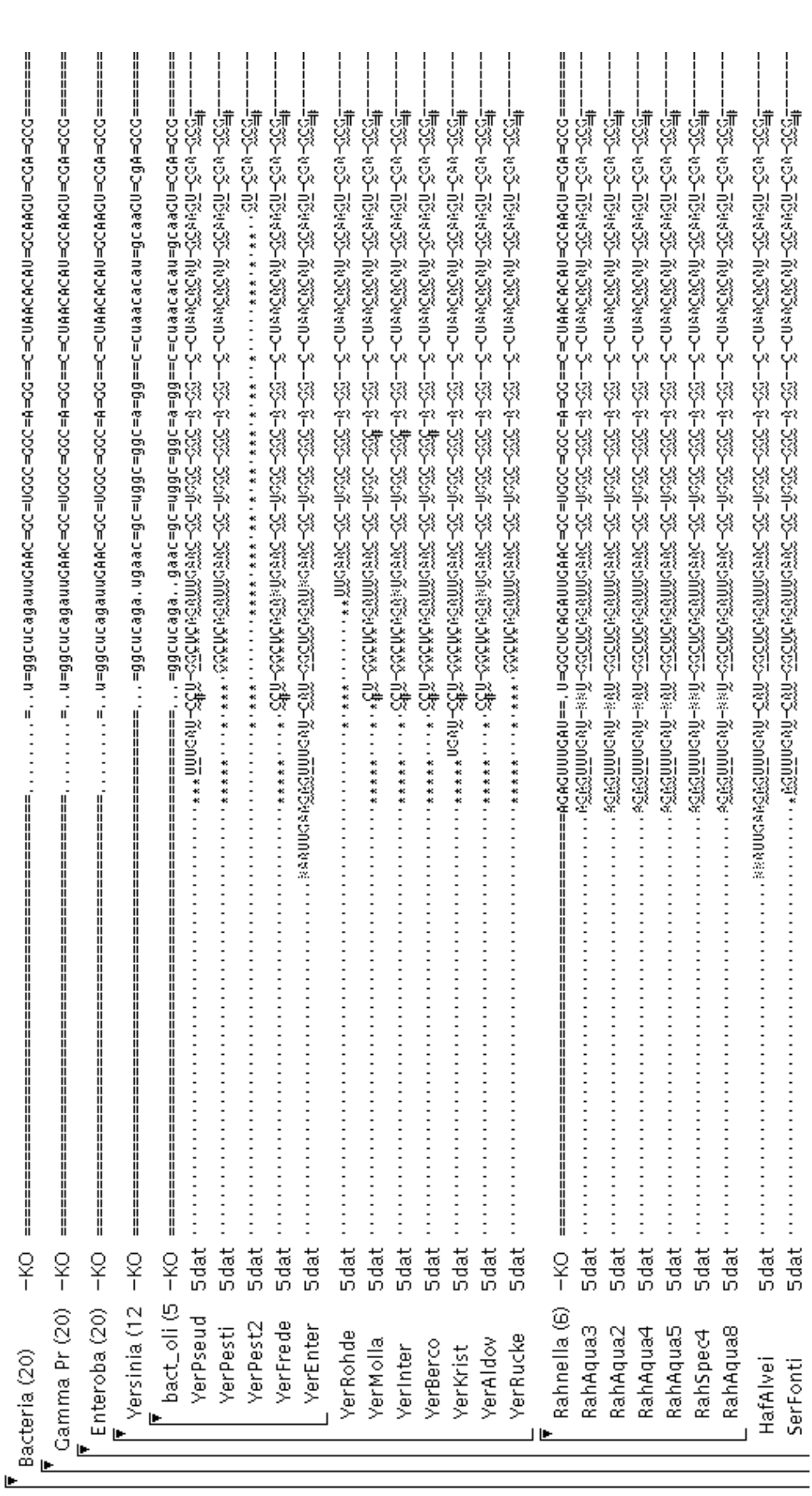


Abb. 30 Typischer ARB_EDIT4

Abb. 31 Editor mit zusammengefalteten Gruppen:

So präsentiert sich der Editor ähnlich wie der Windows Explorer von Microsoft, selbst die Benutzung der Maus wurde so umgesetzt, wie es der Benutzer intuitiv erwarten würde.

Arbeitsbereiche

Alle bekannten Sequenzeditoren arbeiten auf allen Sequenzen der Datenbank. Nur wenige Ausnahmen erlauben das selektive Auswählen einzelner Sequenzen für den Editiervorgang. Dieses für den praktischen Einsatz zu einfache Vorgehen wurde bei ARB durch das flexiblere Konzept der **Arbeitsbereiche** ersetzt. Ein Arbeitsbereich ist ein Ausschnitt aus der Menge der Sequenzen und ihrer hierarchischen Organisation. Er erlaubt, in einem Bereiche in einem Datensatz zu editieren. Alle Information der Arbeitsbereiche wird innerhalb einer Datenbank gespeichert. Dies garantiert die einfache Weitergabe der vollständigen Daten an weitere Benutzer.

Diese Arbeitsbereiche können nicht nur dazu dienen, Sequenzen zu editieren, sondern sie können auch den Zustand der Datenbank zu einem bestimmten Zeitpunkt speichern. So können sie zum Beispiel dazu eingesetzt werden, neue Sequenzen zu identifizieren.

In diesem Arbeitsbereich sind nicht die Sequenzen selbst gespeichert, sondern nur symbolische Referenzen auf den Namen des Organismus. In der jetzigen Implementierung von ARB wird bei Änderung eines Namens eines Organismus ein solcher Verweis ungültig.

VII.4.4 Darstellung einer Sequenz

Farben und Zeichen

Bis 1994 gab es zwei Sequenzeditoren mit denen ARB arbeiten konnte: Den GDE-Editor (Farbdarstellung), der jedoch nur eingeschränkt mit der ARB-Datenbank zusammenarbeitete, und ARB_EDIT (schwarz-weiß Darstellung). Es zeigte sich, daß der Einsatz von Farbe nicht nur von der Anwendergemeinde gefordert wurde, sondern auch die Qualität eines solchen Editors entscheidend verbessern konnte. So wurde für den neuen Editor ARB_EDIT4 ein möglichst flexibler Umgang mit farbiger Darstellung vorgesehen.

Ein Blick auf einen Ausschnitt des Sequenzeditors zeigt die unterschiedlichen Bereiche, die zur Darstellung von Zusatzinformationen genutzt werden:

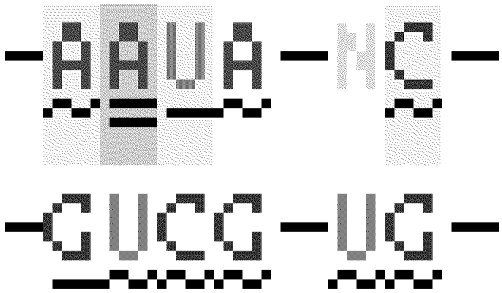


Abb. 32 Ausschnitt aus dem ARB_EDIT4 Sequenzeditor

- Die **Zeichen** der **Sequenz** zeigen normalerweise die Basen der zugrundeliegenden Sequenz an, können jedoch bei Bedarf modifiziert werden.
- Die **Hintergrundfarbe** der einzelnen **Sequenzzeichen** dient dazu, Informationen mit hohem Informationsgehalt darzustellen, wie z.B. Alignmentfehler, außergewöhnliche Basen etc.
- Die **Farbe** der **Sequenzzeichen** soll dazu verwendet werden, die einzelnen Buchstaben besser unterscheidbar zu machen. In vielen Fällen wird mit einem sehr kleinen Zeichensatz gearbeitet, die Farben können helfen, schnell einen Überblick über das Alignment zu bekommen.
- Die **Zeichen** der **Zusatzzeile** (unter den Sequenzen) wird dazu eingesetzt, Helixinformationen einzublenden. Man könnte auch in diesem Falle Farbcodierung einsetzten, dann würden jedoch zu viele verschiedene Farben mehr Verwirrung als Ordnung stiften.

Praktische Erfahrung hat gezeigt, daß mehr als vier verschiedene Darstellungsformen (Bereiche) als verwirrend empfunden werden. Gibt es jedoch mehr als vier darzustellenden Informationen, müssen sich gewisse Informationen eine Darstellungsform teilen. Es hat sich als sinnvoll herausgestellt, alle seltenen Informationen, also solche mit hohen Entropiegehalt, nach geordneten Prioritäten zusammenzufassen und das Ergebnis über die Hintergrundfarbe der Sequenzzeichen darzustellen.

Berechnung der darzustellenden Information

Es gibt eine nahezu unendliche Anzahl möglicher Informationen, die in einem Editor dargestellt werden können. Um einen Benutzer nicht visuell zu überforden, mußte eine sinnvolle Auswahl verschiedener Algorithmen getroffen werden. Da bis auf die Farbdarstellung der Sequenzen im Bereich der Biologie noch keine weiteren Informationen in einen Editor integriert wurden, fehlte zunächst jegliche praktische Erfahrung, wie die einzelnen Informationen darzustellen und die entsprechenden Algorithmen zu konfigurieren seien. So wurden alle diese Algorithmen auf eine Weise implementiert, die die nachträgliche Modifikation der zugehörigen Parameter erlaubte. Diese Parameter wurden sogar durch den Benutzer einstellbar gestaltet, so daß schließlich oft die Algorithmen wesentlich flexibler und effektiver eingesetzt werden konnten als ursprünglich gefordert.

Zwei der komplexeren Algorithmen zur Generierung von Zusatzinformationen wurden in diesem Kapitel exemplarisch vorgestellt, der zur Gewinnung von Helix-Information und der zur Berechnung der hypothetischen Basenwahrscheinlichkeit.

Wie bereits erwähnt, soll die darzustellende Information möglichst dynamisch berechnet werden, um knappen Speicherplatz zu sparen. Die dargestellten Informationen werden also immer neu aus den gespeicherten Sequenzen berechnet; Insgesamt ergibt sich folgender Informationsfluß:

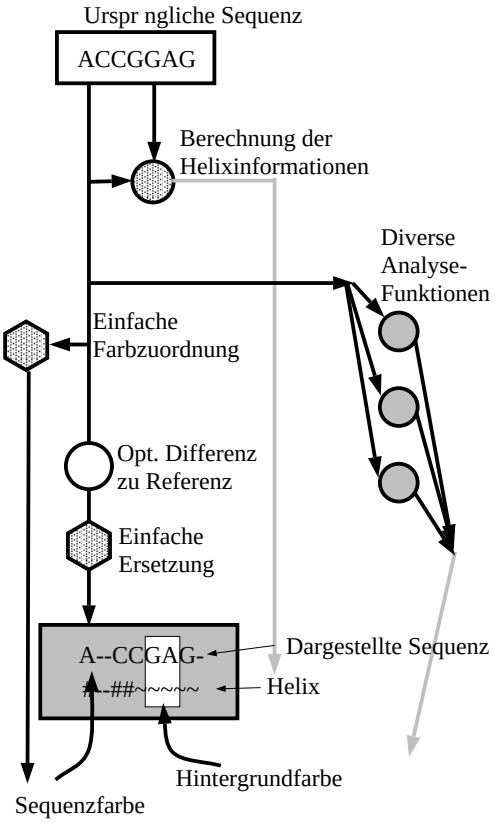


Abb. 33 Informationsfluß innerhalb des Sequenzeditors

- In vielen Fällen sind die auf dem Bildschirm angezeigten Sequenzen bis auf minimale Unterschiede gleich. Um genau diese Unterschiede herauszustellen, können in dem Sequenzeditor alle Zeichen, die identisch zu den Zeichen einer Referenzsequenz sind, durch das unauffällige Symbol '.' ersetzt werden. So bekommt man einen schnellen Überblick über die Unterschiede zwischen den Sequenzen.
- Jeder darzustellenden Base kann mit Hilfe einer einfachen Tabelle eine Zeichenfarbe zugeordnet werden.
- Insbesondere bei Analyse von Aminosäuren lassen sich die darzustellenden Zeichen in biochemische Gruppen aufteilen. Diese Gruppen haben die Eigenschaft, daß bei Mutationen die Gruppe selten gewechselt wird (konservative Mutationen). Ersetzt man mit Hilfe einer einfachen Ersetzungstabelle alle Aminosäuren einer Gruppe durch ein Gruppensymbol, oder ordnet man jeder dieser Gruppen eine eindeutige Farbe zu, kommt man zu einer Darstellung, bei der wesentlich weniger verschiedene Zustände sichtbar sind. Dadurch wird das Überprüfen des Alignments wesentlich vereinfacht.
- Mit Hilfe von Information über Sekundärstruktur kann entschieden werden, ob in einer Sequenz eine Position eine Bindung eingeht oder ob sie die Helix verletzt.

Kommt es zu Helixverletzungen, wird unterhalb der Sequenz ein ausgezeichnetes Symbol dargestellt[1].

- Diverse Algorithmen bestimmen die Hintergrundfarbe der Sequenz:

  a. Basen, die dem Programm 'unwahrscheinlich' erscheinen, werden farbig hervorgehoben.
  b. Sequenzen, die aufgrund von Datenbankanfragen markiert wurden, werden farbig mit einem leichten Hellgrau hinterlegt.
  c. Vom Benutzer selektierte Blöcke werden farbig markiert.
  d. Ergebnisse einer Sequenzmustersuchfunktion werden farbig hinterlegt:
     Alle bisher bekannten Suchfunktionen funktionieren nach dem Schema 'suchen->finden'. Die innovative Idee für eine neue Suchfunktion war, daß immer alle Suchtreffer angezeigt werden sollten und man einfach mit <shift>+<cursor right/left> zwischen den verschiedenen Treffern hin- und herspringen können sollte. Auf dieser Basis ist es gelungen, nicht nur Suchergebnisse darzustellen, sondern auch Primärbindungsstellen, Sondenbindungsbereiche, Signaturen usw. einfach einzublenden. Der große Vorteil dieses Vorgehens ist es, daß sich diese Funktionen gewissermaßen intuitiv in den Gesamteditor integrieren lassen, es ist nicht nötig, die Anwender entsprechend einzuarbeiten.

Die Möglichkeiten, verschiedenste Informationen darzustellen, sind bei weitem noch nicht ausgeschöpft. Es herrscht jedoch bei den Anwendern Einigkeit, daß es sinnvoll und notwendig ist, solche Zusatzinformationen in den Editor zu integrieren. Da die Entwicklung dieses Programmoduls noch fortgeführt wird, wird in der Zukunft wohl ein immer leistungsfähigerer und benutzerfreundlicher Editor entstehen.

VII.4.5 Layout-Verwaltung

Bei allen bisherigen Sequenzeditoren waren die Sequenzen einfach zeilenweise angeordnet. Jede Zeile hatte dieselbe graphische Höhe. So war es leicht, den darzustellenden Bildschirmausschnitt zu bestimmen. Von diesem Konzept sollte in mehrfacher Hinsicht abgewichen werden, die Darstellung von mehr als einer Sequenz pro Organismus sollte möglich die Höhe einer Sequenz beliebig einstellbar sein. Außerdem sollten die Sequenzen hierarchisch organisiert sein. Zur Lösung dieser Probleme wurde ein Konzept aus Objekten zur Darstellung und Objekten zum Layout entwickelt. Die darstellenden Objekte, im weiteren **Terminale** genannt, wie Name eines Organismus, Sequenz oder Helixinformation kümmern sich nur um die Darstellung und ihren auf dem Bildschirm benötigten Platzbedarf. Diese Objekte werden von sogenannten Layout-Managern verwaltet. Ein Manager hat die Aufgabe, seine Elemente horizontal bzw. vertikal anzuordnen. Dabei wechseln sich vertikal mit horizontal anordnenden Managern in der Hierarchie ab. Alle Layout-Manager besitzen dieselben

[1] Es gibt eine benutzerdefinierbare Konfigurationstabelle, die genau festlegt, welche Zeichen unterhalb der Sequenz erscheinen, in dieser Tabelle gibt es jedoch Voreinstellungen, von denen im weiteren ausgegangen werden soll.

Schnittstellen (**Signatur**), dadurch kann jeder Layout-Manager durch jeden beliebigen anderen ersetzt werden.

So entstand eine komplexe Hierarchie aus Terminalen und Layout-Managern. Diese Hierarchie baut auf wenigen Grundstrukturen auf: Der Layouthierarchie des Organismus, der Layouthierarchie einer Gruppe, der Layouthierarchie des gesamten Editors:

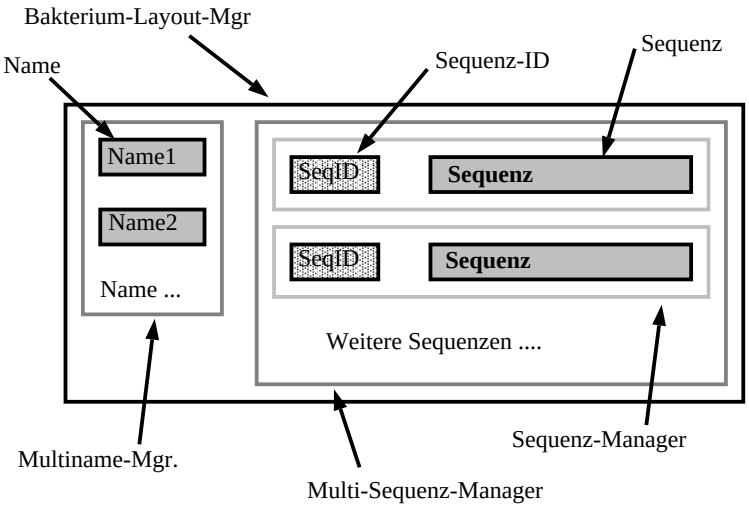


Abb. 34 Organisation der Terminale eines Organismus

Da jedem Organismus sowohl mehrere Namen wie auch mehrere Sequenzen zugeordnet sein können, werden diese mit Hilfe von MULTINAME-Managern bzw. MULTISEQUENZ-Managern verwaltet. Zur eindeutigen Identifizierung einer Sequenz besitzt diese noch einen Sequenzidentifier, der jedoch praktisch nur eine untergeordnete Rolle spielt.

Eine Liste von Organismen kann zu einer Gruppe zusammengefaßt werden. Damit später Organismen und Gruppen von Organismen einfacher weiterverarbeitet werden können, sind die Programmierschnittstellen einer Gruppe identisch mit denen eines Organismus.

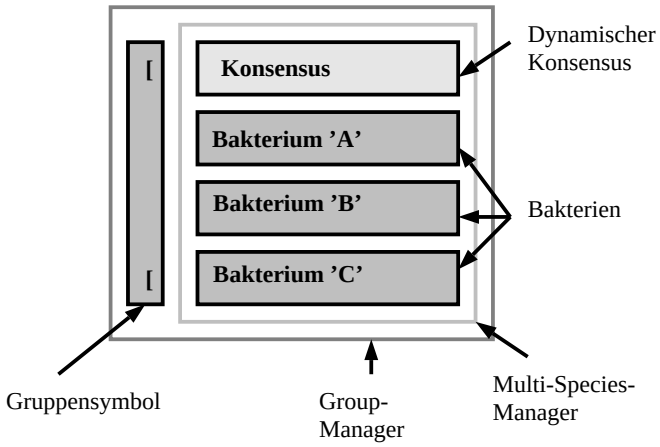


Abb. 35 Organisation der Objekte einer Gruppe

Optisch wird eine Gruppe durch ein übergreifendes Klammersymbol angedeutet. Der schon erwähnte dynamische Konsensus wird parallel zu den Bakerien in die Gruppe eingehängt.

Alle Gruppen wiederum sind in einer weiteren Hierarchie organisiert, der des gesamten Editors. Dabei sind insbesondere die beiden höchsten Ebenen besonders ausgezeichnet: Die oberste (root-) Ebene und die folgende Gruppe, die jeweils für einen Scroll-Bereich zuständig ist:

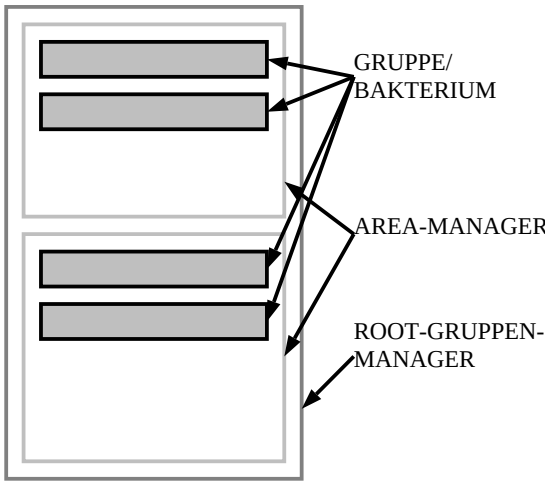


Abb. 36 Organisation der Hauptobjekte

Dem Konsensus des Root-Gruppen-Managers kommt eine besondere Bedeutung zu:

Betrachtet man typische 16S-rRNS-Alignments genauer, findet man große Bereiche im Alignment, in denen nur wenige Sequenzen Basen aufweisen. In diesen Bereichen erscheinen bei allen anderen Sequenzen auf dem Bildschirm entsprechend große Lücken auf dem Bildschirm. Tatsächlich bestehen sogar meist etwa 80% des Alignments aus ungenutzten Lücken. Um dennoch sinnvoll editieren zu können, sollten diese Leerzeichen dynamisch entfernt werden. So wurden in der jetzigen ARB_EDIT4–Version Algorithmen implementiert, die mit Hilfe des Konsensus des Root-Gruppen-Managers ungenutzte Alignment-Bereiche ausblenden sollten. Diese Algorithmen sind jedoch nicht trivial und in der jetzigen Version des Editors noch nicht ausgereift.

VII.5 Zusammenfassung und Ausblick

In der Vergangenheit wurde der Vorgang des Alignens und der der Sequenzeditierung als im Prinzip unabhängige Vorgänge angesehen. Im Gegensatz dazu wurde bei ARB versucht, den mit Hand durchgeführten Vorgang des Alignens mit Hilfe eines Sequenzeditors entscheidend zu verbessern. Dabei entstand ein Sequenzeditor, mit dem es nicht nur wesentlich leichter fällt, Sequenzen zu alignen, sondern mit dem auch ein gegebenes Alignment schnell und sicher auf Probleme und Fehler untersucht werden kann.

Dazu wurden zwei Algorithmen entwickelt, die eine Bewertung eines Alignments erlauben:

- Die Sekundärstruktur dient dazu, jede Sequenz einzeln für sich auf Sekundärstrukturfehler zu untersuchen.
- Die Berechnung hypothetischer Likelihoodwerte erlaubt es, jede Spalte des Alignments unabhängig auf ’unwahrscheinliche’ Basen hin zu testen.

Da der eine Algorithmus auf Zeilen, der andere auf Spalten des Alignments arbeitet, liegen hier zwei aufeinander orthogonale Analysesysteme vor. Darunter versteht man zwei Verfahren, die sich gegenseitig nicht beeinflußen und deren Ergebnisse einfach kombiniert werden können.

Diese Algorithmen wurden in einem neuartigen Sequenzeditor integriert. Neuartig ist dieser Editor deswegen, weil er sich von der traditionellen Art, Listen von Sequenzen zu editieren, abhebt und einen wesentlich flexibleren, effizienteren, hierarchischen und objektorientierten Ansatz verwendet.

Dieser Ansatz erlaubt es denn auch, inhaltlich und graphisch neue Wege zu gehen, und zum Beispiel dynamische Konsensus, dynamische Alignment-Kompression oder graphische Darstellung zu integrieren. Diese Algorithmen hätten die Grundstruktur aller bekannten Sequenzeditoren bei weitem gesprengt, obwohl sie unbedingt notwendig für die Wartung und Wertung heutiger großer Alignments sind.

VIII Phylogenie

VIII.1 Wissenschaftliche Ziele

Die Möglichkeit genetische Sequenzen zu analysieren, eröffnete in der Mikrobiologie ungeahnte neue Wege, objektiv Verwandtschaftsgrade zwischen Organismen zu ermitteln und einen Stammbaum (Phylogenie) zu berechnen.

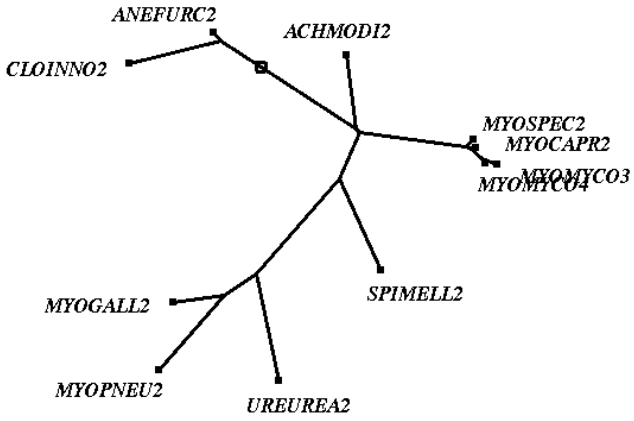


Abb. 37 Beispiel eines kleinen Stammbaumes

Schon in der Anfangszeit der sequenzbasierten phylogenetischen Rekonstruktion entwickelte sich eine immer noch andauernde Diskussion um den Sinn und die Zuverlässigkeit dieser neuen Techniken. Dabei waren die Anforderungen der verschiedenen Biologen an eine phylogenetische Baumrekonstruktion äußerst unterschiedlich:

- Biologen, die mit Organismen aus Umweltproben arbeiten, sind häufig nur daran interessiert, 'ihren' neu gefundenen Organismus phylogenetisch einzuordnen. Das Interesse liegt mehr darin, die nächsten bekannten Verwandten zu finden, als eine perfekte Phylogenie zu bestimmen. Aus diesem Grund werden auf diesem Gebiet auch meist nur ein kleiner Teil einer Sequenz bestimmt, gerade soviele Basen, daß eine phylogenetische Einordnung möglich ist.
- Eine andere Gruppe versucht, den tatsächlichen Stammbaum einer ausgewählten Organismengruppe zu finden. Mit Hilfe langer Sequenzen wird versucht, durch Einsatz verschiedenster Verfahren sich dem tatsächlichen Stammbaum zu 'nähern' und gleichzeitig die Qualität dieser Stammbaumschätzung zu ermitteln.
- Das erfahrungsgemäß schwierigste Ziel streben Forscher an, die versuchen, den universellen Stammbaum aller bekannten Sequenzen zu ermitteln. Ziel ist es nicht nur, einzelne Organismen in einem Baum anzuordnen, sondern auch, eine allgemeine Ordnung für alle Sequenzen zu finden. Zur Erreichung dieses Zieles finden regelmäßig Treffen mit inteRNStionaler Beteiligung statt (z.B. RDP, DeWachter

> und TU-München). Beim letzten derartigen Treffen wurde zum Beispiel versucht, sich auf einen universellen 16S-rRNS-basierten Stammbaum festzulegen, der dann als Grundlage für die Neuauflage von 'Bergey's Manual' dienen soll [Bergey's Manual]. Da dieses Standardwerk die Basis für die Forschungsarbeiten für die nächsten 10 Jahre dienen soll, wird dieser Stammbaum in die Werke vieler Forscher auf diesem Gebiet einfließen. Fehler in diesem Nachschlagewerk werden nicht oder nur schwer korrigierbar sein.
> Dabei stellt die Größe der Datenmenge und die Qualitätsanforderungen ein praktisch unlösbares Problem dar. Daher müssen sich alle Biologen darüber im klaren sein, daß dieser Baum nur eine Möglichkeit aus einer fast unendlichen Menge von sinnvollen Stammbäumen darstellt.

Im Laufe der Zeit entwickelten Bioinformatiker eine unglaubliche Vielzahl unterschiedlichster Methoden und Algorithmen zur phylogenetischen Analyse, von denen jedoch nur zwei Untergruppen sinnvoll eingesetzt werden: Die Distanzverfahren und Parsimony/Maximum-Likelihood-Methoden. Da eine vollständige Analyse beider Verfahren äußerst komplex und detailreich wäre, soll auf den folgenden Seiten nur eine äußerst knappe, minimale Einführung in die Basis-Algorithmen gegeben werden.

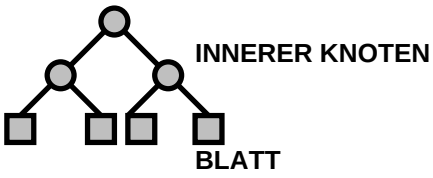
VIII.2 Theoretische Grundlagen

Man folgende grundlegende **Definitionen.**

- Gegeben ist eine Menge **M** von Organismen **B**
- Eine Sequenz **S** bestehe aus einer Abfolge von Teilmengen der Buchstabenmenge *'ACGT-'*, die repräsentativ für die Basen Adenin, Cytosin, Guanin, Thymin und das Füllzeichens GAP stehen. Teilmengen werden verwendet, um mehrere Möglichkeiten einer Base an einer bestimmten Sequenzposition zuzulassen.
- Zu jedem Organismus *B* gehört die Sequenz S_B.
- Alle Sequenzen *S* der Organismen *B* aus *M* sind gleich lang und haben die Länge **n** (Alignmentlänge).
- **S[i]** bezeichnet den Wert einer Sequenz an der Alignmentposition *i*

Des weiteren betrachten wir vollständige **Binärbäume**:

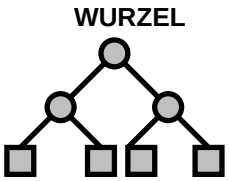
Alle Knoten des Baumes sind entweder Blatt oder innerer Knoten.



Jedes Blatt hat genau einen Vater.

Jeder innerer Knoten hat genau zwei Söhne (linker und rechter Sohn) und maximal einen Vater.

Ist der Baum gewurzelt, gibt es genau einen inneren Knoten, der keinen Vater hat.



Des weiteren seien folgende **Operatoren** auf Sequenzpositionen definiert:

- Sei ø die leere Menge
- $S_C[i] = S_A[i] \cup S_B[i]$ ist die Vereinigungsmenge zweier Teilmengen, dabei werden Duplikate entfernt.
- $S_C[i] = S_A[i] \cap S_B[i]$ definiert die Schnittmenge zweier Teilmengen.

VIII.3 Einführung in den Parsimony-Algorithmus

VIII.3.1 Parsimony als bewertendes Kriterium

Die Grundüberlegung hinter allen Parsimony-Methoden ist, daß eine Mutation unwahrscheinlicher ist als eine Nicht-Mutation. Wäre diese Annahme in der Natur nicht erfüllt, dürfte es keinerlei Ähnlichkeiten zwischen zwei beliebigen Sequenzen geben, da alle Basen mit hoher Geschwindigkeit mutieren, sodaß keine Verwandtschaftsgrade über Ähnlichkeiten in den RNS-Sequenzen festgestellt werden könnten.

Ausgehend von dieser Grundüberlegung versucht man nun, einen Stammbaum zu finden, bei dem die minimale Anzahl von Mutationen einen minimalen Wert annimmt. Ein kurzes Beispiel soll dies verdeutlichen:

Gegeben seien 4 Sequenzen *S* der Länge *n*:

Bakterium	Sequenz
1	’A’
2	’A’
3	’C’
4	’C’

Mit diesen vier Organismen können dann 3 verschiedene Bäume aufgebaut werden:

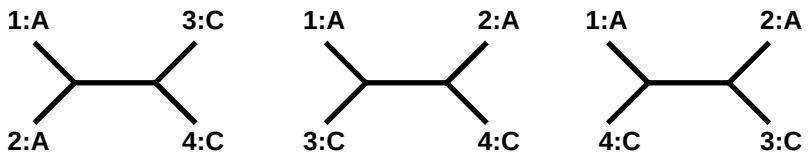


Abb. 38 Die drei möglichen Bäume mit vier Blättern

Es ist leicht ersichtlich, daß für den linken Baum minimal eine Mutation notwendig ist, für die beiden anderen Bäume minimal 2 Mutationen. Aus diesem Grund wird der linke Baum als der wahrscheinlichste eingestuft.

Es gibt eine ganze Reihe von verschiedenen Parsimony-Variationen. Im weiteren Verlauf wird exemplarisch die von **Wagner und Fitch** entwickelte Methode vorgestellt. Andere Verfahren weichen jedoch von dem hier vorgestellten Grundalgorithmus nur in Details ab.

Der allgemeine Algorithmus zur Berechnung der minimal nötigen Mutationen eines gegebenen Baumes mit Sequenzen geht folgendermaßen vor:

- Die minimale Mutationsanzahl wird für jede Spalte des Alignments getrennt berechnet und die Gesamtmutationsanzahl als ihre Summe berechnet.
- In den ungewurzelten Baum wird willkürlich eine Wurzel gesetzt. Die Position der Wurzel hat keinen Einfluß auf das Endergebnis.
- Jedem Knoten *K* in dem Baum wird ein Attribut *M* zugeordnet, dessen Wert eine Teilmenge der Menge aller möglichen Basen *ACGT*- ist.
- Für alle Spalten *i* des Alignments werden die Attribute *M* der Blätter *B* des Baumes auf den Wert der Sequenz *S* des entsprechenden Organismuses an der Alignmentposition *i* gesetzt: $M_B=S_B[i]$, wobei S_B die Sequenz des Organismus/Blattes *B* ist.
- Ausgehend von den Blättern werden alle inneren Attribute M_K eines Knotens *K* folgendermaßen aus den Attributen seiner beiden Söhne *L* und *J* gesetzt:

$$M_K = \left\{ \begin{array}{ll} M_J \cap M_L & falls \;\; M_J \cap M_L \neq \emptyset \\ M_J \cup M_L & falls \;\; M_J \cap M_L = \emptyset \end{array} \right\}$$

  Jedesmal, wenn die Vereinigungsmenge gebildet wird, ist dies ein Zeichen dafür, daß mindestens eine Mutation stattgefunden hat.

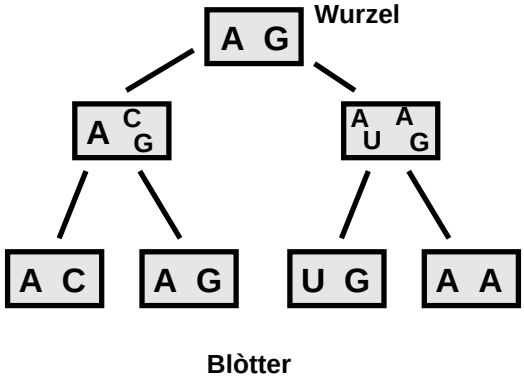


Abb. 39 Beispiel der Attributbestimmung der inneren Knoten bei einer Sequenzlänge von 2.

- Für alle Spalten zählt man die Anzahl der benötigten Schritte, bei denen eine Vereinigungsmenge gebildet wurde.
- Die gesamte minimale Mutationsanzahl errechnet sich aus der Summe der minimalen Spaltenmutationen.

VIII.3.2 Suchheuristiken

Der Parsimony-Algorithmus liefert nur eine Bewertung eines gegebenen Baumes, jedoch keine Vorschrift, wie dieser Baum zu generieren ist. Eigentlich müßte man alle möglichen Bäume generieren und diese bewerten, um sicher den besten Baum zu finden. Aber bereits bei kleinen Bäumen gibt es so viele verschiedene Baumtopologien, daß es schon

bei mittelgroßen Bäumen rechentechnisch unmöglich wird, alle möglichen Topologien zu bewerten:

Anzahl der Blätter	Anzahl der Topologien
4	3
5	3 * 5 = 15
6	3 * 5 * 7 = 105
k	$\prod_{k=2}^{n} (2k-3)$

Abb. 40 Abhängigkeit der Anzahl möglicher Topologien von der Anzahl der Blätter [Swofford, Olsen 90]

Das Problem der Baumrekonstruktion ist höchstwahrscheinlich NP-vollständig, für dessen Lösung keine polynominial-zeitbeschränkten deterministischen Algorithmen verwendet werden können. Zur Lösung solcher Probleme verwendet man Strategien, welche auf plausiblen Vermutungen und bisher gesammelten Erfahrungen beruhen. Solche Strategien bezeichnet man als **Heuristiken**. Eine gute Heuristik erzeugt mit hoher Wahrscheinlichkeit innerhalb vernünftiger Zeit eine relativ gute Lösung. Sie liefert jedoch keine Garantie für das Finden der optimalen Lösung.

Prinzipiell arbeiten alle bisher entwickelten Heuristiken zur Baumrekonstruktion nach einem zweistufigen Verfahren:

I. Stufe: **Inkrementeller Aufbau der Bäume:** Der Baum wird stückweise aufgebaut. Zu einer guten Teillösung wird ein weiterer Organismus an allen möglichen Positionen im Baum eingefügt und die so entstandenen Bäume bewertet. Die beste gefundene Position wird schließlich realisiert:

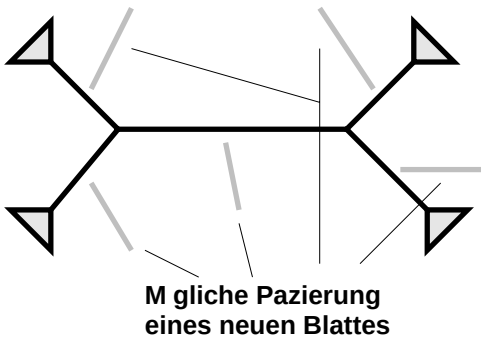


Abb. 41 Mögliche Positionen eines einzufügenden Organismus in einen gegebenen Baum

II. Stufe: **Nachoptimierung der Bäume:** In einem gegebenen Baum werden testweise zwei Äste miteinander vertauscht. Erhält man durch diese Vertauschung einen besseren Baum, wird diese nicht rückgängig gemacht. Wird der Baum schlechter, hat man entweder die Möglichkeit, durch eine weitere Vertauschung

einen besseren Baum zu erhalten, oder man macht die Vertauschung rückgängig:

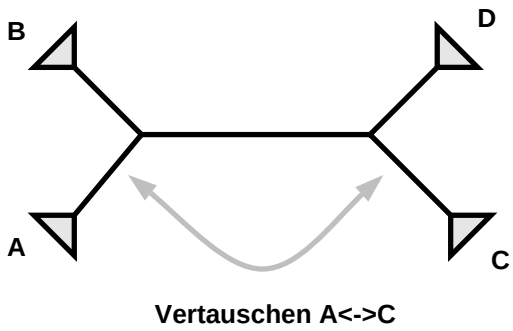


Abb. 42 Vertauschen zweier Unterbäume in einem Baum

Je nach Anzahl der inneren Knoten auf dem Pfad der vertauschten Äste unterscheidet man folgende Fälle:

a. Sind die beiden vertauschten Kanten benachbart, spricht man von Nearest Neighbour Interchange (**NNI**)
b. andernfalls handelt es sich um eine allgemeine Vertauschung. Da es sehr viele mögliche Tauschpartner gibt, ist es nicht trivial, vielversprechende Partner auszuwählen. Das allgemein eingesetzte Programm **fastDNSml** bietet hierzu die Möglichkeit, einzustellen, wieviele innere Knoten maximal auf dem Pfad zwischen den Tauschpartnern liegen dürfen.

Jede Art von Baumveränderung kann mit einer endlichen Anzahl von wiederholten **NNI**–Vertauschungen nachgebildet werden. Aus diesem Grund soll im weiteren hauptsächlich auf diese Art der Baummanipulation eingegangen werden.

Alle heute eingesetzten Heuristiken zur Berechnung eines Stammbaumes mit Hilfe des Parsimony-Kriteriums versuchen, **einen** gegebenen Baum zu verbessern. Da diese Heuristiken nur lokal begrenzte Baumveränderungen auf einem einzigen Baum durchführen, wird bei größeren Bäumen mit hoher Wahrscheinlichkeit der Fall eintreten, daß ein zwar stabiles lokales Optimum erreicht wird, das nicht mehr verlassen wird, obwohl es global optimalere Lösungen gäbe. Aus diesem Grund wurden in dieser Arbeit grundlegende Konzepte entwickelt, genetische Algorithmen auf Stammbaumberechnungen anzuwenden.

VIII.4 Astlängen und Signifikanz-Schätzung

VIII.4.1 Theorie

Oft ist man nicht nur an der Topologie eines Stammbaumes interessiert, sondern auch an dem Verwandtschaftsgrad zweier Organismengruppen. Dieser Verwandtschaftsgrad wird durch eine Distanz angegeben, die die relative Anzahl von Mutationen zwischen den Sequenzen beider Gruppen angibt.

Diese Astlänge korreliert in den meisten Fällen mit der Korrektheitswahrscheinlichkeit einer bestimmten Baumtopology: Lange Äste sind im allgemeinen wahrscheinlich korrekter als kurze.

Folgendes Beispiel soll dies verdeutlichen:

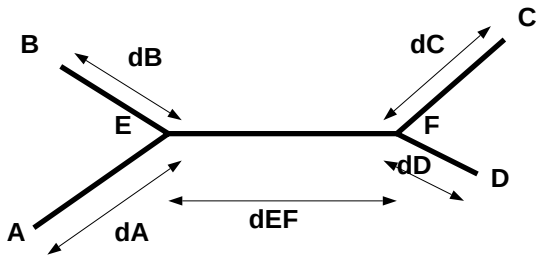


Abb. 43 Astlängen in einem Baum

Die Distanzen im Baum sollen die tatsächlichen Verhältnisse in der Natur widerspiegeln: Lange Distanzen zwischen Knoten entsprechen einem langem Evolutionszeitraum, während kurze einen kurzen Zeitraum wiederspiegeln. Außerdem sind diese Astlängen häufig ein Maß für die Güte des Baumes: Je länger ein Ast ist, desto wahrscheinlicher hat ein Baumberechnungsalgorithmus diesen auch korrekt gefunden, einfach deshalb, da die vielen Mutationen, die diesen Ast ausmachen, auch statistisch signifikant erkannt werden können.

Zusammengefaßt können zwei Vorgehensweisen formuliert werden:

I. Die Astlängen sollen die Mutationen repräsentieren, die tatsächlich stattgefunden haben. Das Problem dabei ist, daß an einem bestimmten Platz in der Sequenz mehrere Mutationen stattgefunden haben können, jedoch pro Base einer Sequenz nur eine Mutation festgestellt werden kann. Deswegen muß man diese nicht mehr sichtbaren Mutationen anhand der beobachteten schätzen. Je weiter dabei in die Vergangenheit geschätzt weden muß, desto ungenauer ist das Ergebnis.
II. Konzentriert man sich dagegen nur auf darauf die Korrektheitswahrscheinlichkeiten in einem Baum darzustellen, sollte die Astlänge nur jede meßbare Mutation wiederspiegeln.

Biologisch korrekt wäre es, nach Methode **I** vorzugehen, allerdings werden dann lange Astlängen noch länger, was dazu führt, daß die Abstände zwischen den inneren Knoten im Bereich der Wurzel eines Stammbaumes auch sehr lang werden. Diese langen Astlängen können allzuleicht als sichere (weil ja lange) Topologien mißinterpretiert werden.

Da das Programmpaket auch Biologen eine phylogenetische Analyse ermöglichen soll, die keine Experten auf diesem Gebiet sind, wurde bewußt darauf verzichtet, die korrekte Astlänge zu schätzen, sondern vielmehr die Signifikanz der topologischen Korrektheit existierender Bäume aufzuzeigen.

Eine mögliche Lösung zur Bestimmung der Astlängen wäre, eine Distanzmatrix aller Sequenzen zu berechnen und dann mit Hilfe dieser Matrix die Astlängen einer gegebenen Baumtopologie zu berechnen. Dieses Verfahren wurde auch in der Anfangszeit des Programpaketes ARB implementiert, allerdings stellt die Berechnen dieser Matrix mit steigender Sequenzanzahl ein Rechenzeitproblem dar, ab 1500 Sequenzen ist sie so gut wie unmöglich. Infolgedessen wurde folgender Algorithmus entwickelt:

Die Länge des Astes zwischen einem Blatt und seinem Vater kann aus der Anzahl der Mutationen berechnet werden, die für dieses Blatt mit Hilfe des Parsimony Algorithmus gemessen wurden. Entfernt man dieses Blatt aus dem Baum, entspricht die Senkung der benötigten Mutationen korrigiert um eventuell unsichtbare Mutationen gerade dieser Astlänge. Daraus ergibt sich ein äußerst einfacher Algorithmus:

- Berechne die minimale Anzahl von Mutationen M_{total} des Baumes.
- Für alle Blätter B: Entferne dieses Blatt, messe die Anzahl der Mutationen M_B und mache die Änderung wieder rückgängig. Die Länge des Astes von B zu seinem Vaterknoten berechnet sich zu: $l_{Bfather(B)} = f\left(\frac{M_{total}-M_B}{Anzahl\ der\ Basen\ von\ S_B}\right)$ wobei f die gemessenen Mutationen in Astlängen umwandelt.

Es wurde versucht, auch die Astlänge zwischen inneren Knoten zu berechnen, indem ganze Teilbäume entfernt und aus der gemessenen Mutationssenkung die Astlänge berechnet wurde. Allerdings erzeugte dieses Vorgehen äußerst ungenaue Astlängen. Wird nämlich ein Teilbaum entfernt, kann man nur alle Mutationen messen, für die der Teilbaum verantwortlich war, nicht nur die Mutationen auf dem Ast zwischen Teilbaum und 'Restbaum'. Um diese Länge zu berechnen, müssen alle inneren Astlängen in diesem Teilbaum abgezogen werden. Statistische Fehler in den Astlängen des Teilbaumes summieren sich und das Ergebnis wird ungenau.

Die Lösung dieses Problems wurde mittels eines dritten Ansatzes zur Astlängenbestimmung angegangen.

Gegeben sei folgender kleiner Baum:

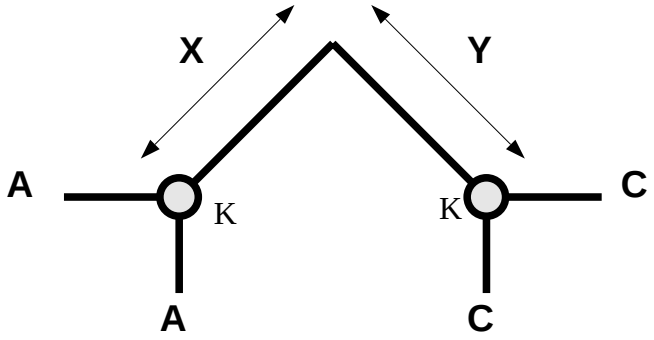


Abb. 44 Berechnen der Astlängen durch NNI

Es soll die Länge *X+Y* des Astes, der die beiden inneren Knoten **K** verbindet, bestimmt werden, d.h. es soll die Anzahl der Mutationen gemessen werden, die auf diesem Ast stattgefunden haben. Vertauscht man einen der beiden rechten Äste mit einem der beiden linken Äste, mißt man diese Mutationen doppelt:

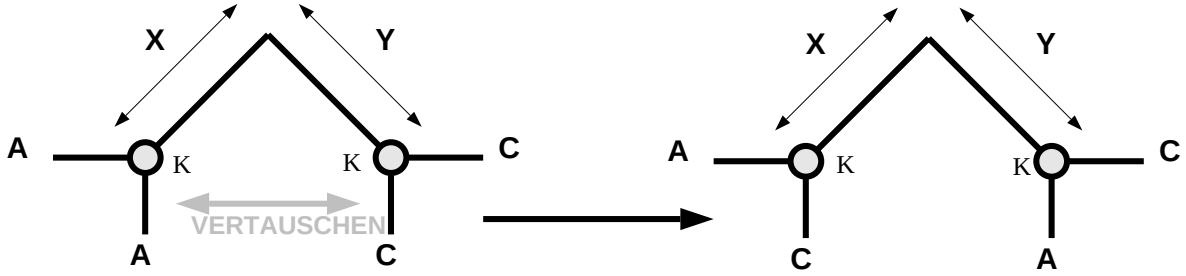


Abb. 45 Berechnen der Astlängen durch NNI

Da es zwei verschiedene Möglichkeiten gibt, einen linken mit einem rechten Ast zu vertauschen, sollten auch beide Möglichkeiten durchgeführt werden, das Gesamtergebnis wird dann aus den Einzelergebnissen arithmetisch gemittelt.

Die Länge des Astes *X+Y* errechnet sich aus der relativen Zunahme der Mutationen bei der Vertauschung zweier an den Ast angrenzenden Unterbäume.

Nun kommt es in der Praxis jedoch häufig vor, daß durch weitere Mutationen in den angrenzenden Ästen die Mutationen in dem zu messenden Ast unentdeckt bleiben:

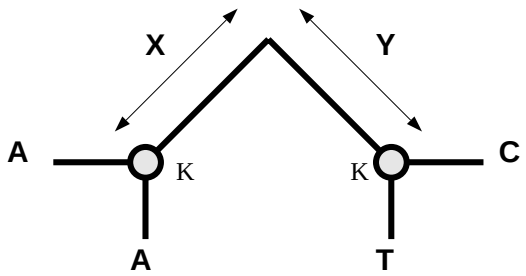


Abb. 46 Unentdeckte Mutation im Ast *X/Y*

Durch Vertauschen der Äste ändert sich die minimale Mutationsanzahl von *2* nicht, und die Mutation auf dem Ast *X* oder *Y* bleibt unentdeckt.

Ein mathematisches Modell, daß diese unsichtbaren Mutationen berücksichtigt, müßte folgende biologische Eigenheiten berücksichtigen:

- unterschiedliche Variabilität einzelner Spalten
- unterschiedliches GC-Verhältnis einzelner Spalten

Eine hinreichend genaue mathematische Lösung steht noch aus, war jedoch nicht Ziel dieser Arbeit.[1]

Um dennoch den Einfluß dieser unsichtbaren Mutationen abzuschätzen, wurde eine praktische Untersuchung an realen Bäumen durchgeführt. Dabei wurde die durch dieses Verfahren gemessene Astlänge einfach in Relation mit den mit traditionellen Methoden errechneten gesetzt.

- Die Astlängen des offiziellen Baums der RDP, berechnet nach dem Maximum Likelihood-Algorithmus, wurden verglichen mit den Astlängen, die der oben vorgestellte Algorithmus für denselben Baum errechnete.
- Für einen Ausschnitt aus diesem Baum wurden mit einem Distanzverfahren (Neighbour-Joining mit Jukes-Cantor Transformation) die Astlängen neu berechnet und diese wiederum mit denen des oben vorgestellten Algorithmus verglichen.

Dabei ergab sich:

Grundbaum	Verhältnis der Astlängen von Parsimony:Original		
	ca. von	ca. bis	Durchschnitt
Max.Lik	0.25	1.1	0.550211
Nei.Join.	0.11	1.8	0.570306

Abb. 47 Relation parsimonybasierter Astlängen zu traditionell berechneten Astlängen

In der gegenwärtigen Implementierung werden aufgrund dieser Untersuchungen die inneren, d.h. die mit Hilfe von NNI berechneten Astlängen auf 2.0 * 'mittlere Mutationszahlsteigerung durch NNI' gesetzt.

VIII.4.2 Praktische Durchführung

Während die Idee zu dieser Theorie sehr schnell erdacht und entwickelt worden war, gestaltete sich die Durchführung überaus zeitintensiv, bis durch konsequenten Einsatz von Informatikmethoden eine effiziente und elegant programmierte Implementierung erreicht werden konnte. Da diese Implementierung auch die Grundlage der noch folgenden Algorithmen ist, sollen hier einige Ideen auszugsweise angedeutet werden:

[1] Das Problem dabei ist, daß Parsimony ein diskreter Algorithmus ist, d.h. zu einem gegeben Baum müssen alle möglichen Sequenzen durchprobiert werden und das Gesamtergebnis als gewichtetes Mittel aller Sequenzmöglichkeiten berechnet werden. Dabei entspricht das Gewicht der Wahrscheinlichkeit, mit der eine Sequenzmöglichkeit auftritt.

Zwischenspeicherung der berechneten inneren Baumattribute Die Berechnung der minimalen Anzahl von Mutationen erfolgt nicht spaltenweise, sondern knotenorientiert. Dazu wird jedem Knoten für jede Spalte *i* das Attribut *A[i]* zugeordnet. Das Array dieser Attribute wird in den Knoten dauerhaft gespeichert.

Abhängigkeitsgraph der Attribute Mit Hilfe eines Abhängigkeitsgraphen der inneren Attribute kann erreicht werden, daß bei lokalen Baumänderungen nur die Attribute zwischen allen Änderungspositionen und der Wurzel des Baumes neu berechnet werden müssen. Da die Höhe des Baumes ungefähr *log(Anzahl der Blätter)* mißt, ist eine deutliche Steigerung der Rechengeschwindigkeit meßbar.

Dynamische Wurzelplazierung Die meisten Baumoperationen werden rekursiv auf alle Äste des Baumes angewandt. Verschiebt man die Wurzel des Baumes immer in die Nähe des Ortes der Veränderung, kann der Pfad zur Wurzel minimal gehalten werden. Auf diese Weise wird erreicht, daß ein NNI mit nur *3*Sequenzlänge* Attributberechnungen durchgeführt werden kann.

Stackorientierte Undo Operationen Sehr häufig werden Baumoperationen nur testweise durchgeführt. Ein Rückgängigmachen dieser Baumoperation bedeutet normalerweise, daß veränderte Attribute *A* der inneren Knoten wiederhergestellt werden müssen. Dazu wurde ein effizienter 'UNDO'-Mechanismus entwickelt, der es erlaubt, alte Baumzustände mit unbedeutendem Rechenaufwand wiederherzustellen. Die Bedienung dieses Mechanismus wurde stackartig durchgeführt, das heißt, daß der Zustand eines Baumes auf einem virtuellen Stack gespeichert werden kann. Eine NNI Operation sieht also im Programmtext folgendermaßen aus: (C++ ähnliche Syntax)

```
input:
tree:          The tree
branch:        the branch


int old_sum_mutations = tree.count_mutations();   // Berechne alte Mutationsmenge
tree.push();                                      // Alten Baum merken
tree.NNI(branch);                                 // Äste vertauschen
int new_sum_mutations = tree.count_mutations();   // neu testen
if (new_sum_mutations < old_sum_mutations){       // besser geworden ???
       tree.clear_stack();                        // Zustand beibehalten
                                                  // gemerkten Zustand löschen
}else{
       tree.pop();                                // alten Zustand herstellen
}
```

Dabei ist zu bemerken, daß die Operation *push* nicht den ganzen Baum zwischenspeichert, sondern nur ein Flag setzt, daß alle Änderungsprozeduren an einzelnen Knoten dafür sorgen müssen, daß der alte Zustand dieses Knotens gespeichert wird.

Duale Sicht der Binärbäume Man kann einen Baum sowohl knotenorientiert als auch kantenorientiert betrachten:

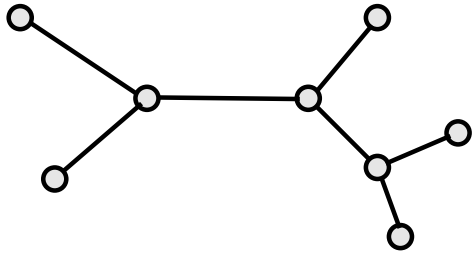


Abb. 48 Knotenorientierte Sicht eines Baumes

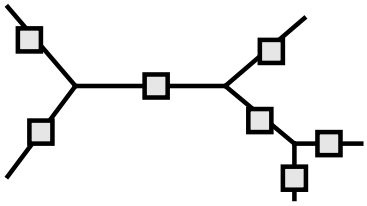


Abb. 49 Kantenorientierte Baumsicht

Für die meisten Algorithmen reicht es aus, mit einer knotenorientierten Sicht zu arbeiten. Knoten haben allerdings den Nachteil, daß sie nicht dazu verwendet werden können, Kanten zu identifizieren. Auch die Beschreibung einer Kante durch ihre beiden Endknoten führt nicht zum Ziel: Möchte man zum Beispiel alle Kanten eines Baumes mit NNI bearbeiten, sorgen Baumänderungen dafür, daß die Nachbarschaftsbeziehung zwischen Knoten sich ändert. So wurde zusätzlich zur Knotenrepräsentation eine Kantenrepräsentation entwickelt. Beide Betrachtungsweisen beschreiben denselben Sachverhalt, allerdings ist es je nach Anwendungsfall einfacher, mit der einer oder mit der anderen oder beiden gleichzeitig zu arbeiten.

VIII.4.3 Ergebnisse

Ziel dieses Abschnittes war es, einen Grundalgorithmus zu entwickeln, der es erlaubt, basierend auf dem Parsimonykriterium in kurzer Zeit Astlängen für einen gegebenen Binärbaum zu entwickeln. Dabei sollten sich die Längen der Äste mehr an der Wahrscheinlichkeit der topologischen Korrektheit dieser Äste orientieren als an der evolutorischen Zeit, die diese Äste repräsentieren. Da eine wissenschaftlich korrekte Analyse der Astlängen den Rahmen dieser Arbeit sprengen würde, soll hier ein rein optischer Vergleich der mit dem vorgestellten Algorithmus ermittelten Astlängen mit denen zweier Referenzalgorithmen durchgeführt werden. Dieser Vergleich soll hier ausreichend sein, zum einen, da die Bäume von den Biologen optisch betrachtet werden, zum anderen, weil eben ein mathematisch korrekter Vergleich den Rahmen dieser Arbeit sprengen würde.

- Distanzmatrixverfahren: Obwohl die Distanzmatrixverfahren Probleme haben, große Bäume topologisch korrekt zu berechnen, zeichnen sie sich doch durch wirklichkeitsnahe Astlängen aus:

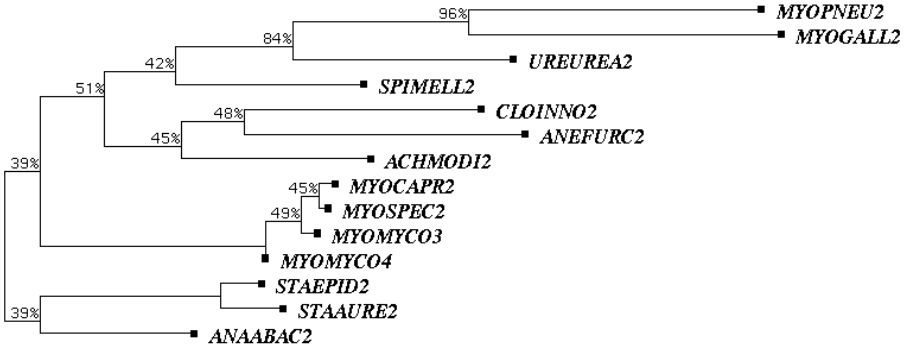


Abb. 50 Mit Hilfe eines Distanzverfahrens berechneter Baum (Die Zahlen an den Ästen geben die Wahrscheinlichkeit der Korrektheit dieses Astes an).

- Maximum Likelihood-Verfahren: Diese Verfahren stellen eine Erweiterung des Parsimonyverfahrens dar, und berechnen sowohl Topologie als auch Astlängen. Von vielen Biologen wird dieses Verfahren als das mit Abstand beste Verfahren zur Bestimmung von Baumtopologien angesehen (insbesondere seitens der RDP):

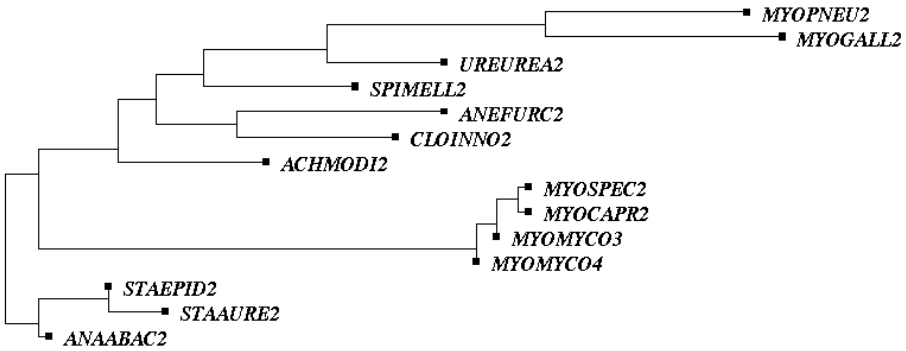


Abb. 51 Ein Maximum-Likelihood-Baum

- Das in dieser Arbeit entwickelte Verfahren:

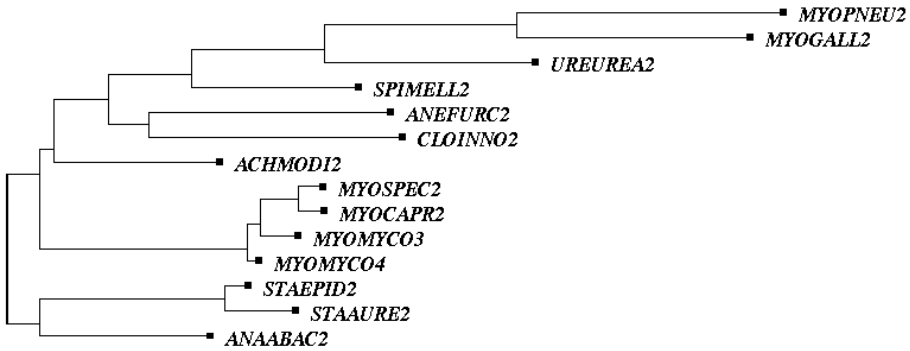


Abb. 52 Mit Hilfe von Parsimony berechnete Astlängen

Dieser einfache optische Vergleich zeigt, daß die Astlängen dieser drei Verfahren doch erheblich voneinander abweichen. Insbesondere fällt auf, daß das Maximum Likelihood-Verfahren dazu tendiert, unnatürlich kurze Äste zu generieren. Mehrere Vergleiche dieser Art haben gezeigt, daß die mit Parsimony berechneten Astlängen im wesentlichen mit denen von Referenzverfahren übereinstimmen.

Aufgrund der programmiertechnisch aufwendigen Implementierung ist auch die benötigte Rechenzeit zu dieser Berechnung praktikabel:

Rechner	Anzahl der Bakterien/ Sequenzlänge	benötigte Rechenzeit
Sparc 2	100/150	0.5 sec
Digital Alpha 400 Mhz	10000/1500	20 sec
Pentium Pro 180 Mhz	10000/1500	30 sec

Da die benötigte Rechenzeit linear zu der Anzahl der zu berechnenden Äste ist, können auch die Astlängen der in Zukunft existierenden sehr großen Bäume berechnet werden.

VIII.5 Bootstrap bei großen Bäumen

VIII.5.1 Einführung

Da der Wunsch der Biologen nach einem ’perfekten’ Baum nicht zu erfüllen ist, geht man dazu über, die statistische Signifikanz einzelner Äste zu bewerten. In der Praxis hat sich dazu das sogenannte **Bootstrap**–Verfahren durchgesetzt: Ausgehend von einem Alignment der Länge *n* werden zufällig *n* Spalten dieses Alignments ausgewählt (Wiederholungen möglich). Diese Spalten werden in beliebiger Reihenfolge zu einem neuen Alignment zusammengefügt. Mit diesem wird ein Baum berechnet und gespeichert. Dieses Vorgehen wiederholt man mehrmals (je nach geforderter Genauigkeit einige hundert oder tausend mal) und berechnet aus den einzelnen Bäumen einen Durchschnittsbaum:

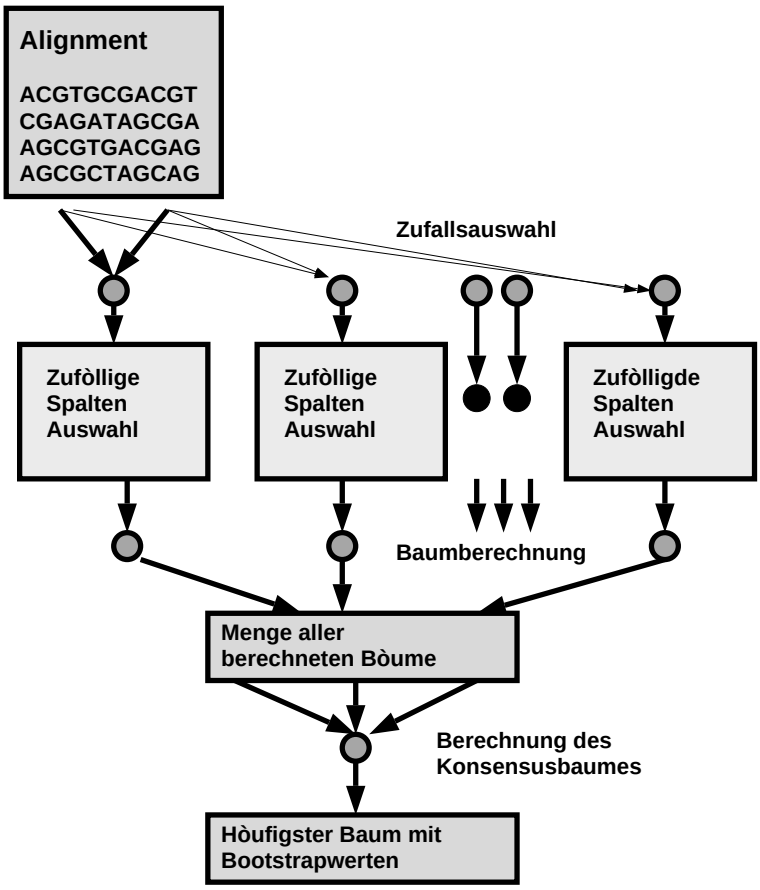


Abb. 53 Prinzip des Bootstrap-Verfahrens

Dieses Vorgehen basiert auf zwei Voraussetzungen:

I. Es ist möglich, automatisch innerhalb kurzer Zeit einen Baum zu berechnen.

II. Ändert man die Alignment-Daten künstlich um kleine Werte, ändert sich auch die Baumtopologie nur wenig (kein chaotisches Verhalten).

Zu **I**: Benötigt eine einzelne Baumberechnung schon mehrere Wochen Rechenleistung, was bei den heutigen Baumgrößen bei Einsatz eines Maximum Likelihood-Verfahrens durchaus die Regel ist, wird man kaum Bootstrapberechnungen mit mehr als tausend Zyklen durchführen. Hinzu kommt, daß heute die großen Bäume von Hand aus Teilbäumen zusammengesetzt sind, also nicht vollständig automatisch berechnet wurden und demnach für dieses Vorgehen nicht geeignet sind[1]. Es kommen also nur Bäume kleineren Ausmaßes (bis 50–100 Spezies) in Betracht.

Zu **II.**: Wie praktische Erfahrungen gezeigt haben, kann gerade bei großen Bäumen die Hinzunahme einer weiteren zu analysierenden Sequenz große Änderungen in der Baumtopologie verursachen. So kann es zum Beispiel vorkommen, daß es zwar eine stabile Grundtopologie gibt, eine einzelne Sequenz sich aber nicht stabil im Baum plazieren läßt[2]. Diese zufällig plazierte Spezies macht eine sinnvolle Bootstrap-Analyse schwierig, wenn nicht sogar unmöglich.

VIII.5.2 Algorithmus

Im Rahmen dieser Arbeit wurde ein neuer Weg entwickelt, Bootstrap-Werte unter bestimmten Vorraussetzungen korrekt zu berechnen oder aber wenigstens eine obere Schranke für diese zu bestimmen.

Zur Veranschaulichung dieses Vorgehens soll folgendes Beispiel dienen: In einem zu testenden Baum sei nur eine einzige Kante *K* unsicher, alle anderen haben nahezu 100%ige Bootstrap-Werte.

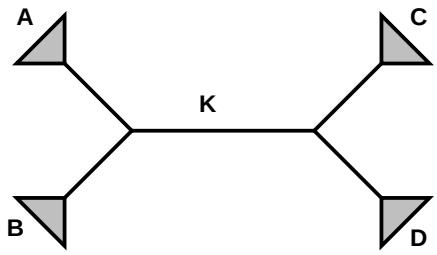


Abb. 54 Baum mit einer unsicheren Kannte

Die Unsicherheit der Kante *K* soll darin bestehen, daß in 20% aller aus dem Bootstrap-Alignment berechneten Bäume die Unterbäume *B* und *D* vertauscht sind, während die Unterbäume in ihrer Topologie unverändert bleiben. In genau diesem Fall kann der Bootstrap-Wert für *K* korrekt und exakt berechnet werden, indem man überlegt, mit welcher Wahrscheinlichkeit Alignmentpositionen so ausgewählt werden, daß der Baum, bei dem *B* und *D* vertauscht sind, mit weniger Mutationen auskommt als die Originaltopologie.

[1] Aber gerade bei diesen Bäumen wären Bootstrap-Werte äußerst hilfreich, um Fehler und Schwachstellen zu erkennen.

[2] siehe auch Untersuchungen zur Plazierung der Spezies *Archeoglobus fulgidus*. [Achenbach Richter 97]

VIII.5

Bestimmung der Anzahl der einer Kante unterstützenden/entgegenwirkenden Alignmentpositionen

Zunächst wird jede Spalte des Alignments getrennt analysiert. Dann werden die pro Spalte minimal notwendigen Mutationen für den originalen wie auch für den modifizierten Baum berechnet.

Das Vertauschen zweier benachbarter Äste in einem Baum (Nearest Neighbour Interchange = NNI) verändert die minimal notwendigen Mutationen in jeder Spalte um höchstens eins.

Dies kann durch Untersuchung aller möglichen Kombinationen bewiesen werden.

Hierzu wird zunächst die Wurzel des Baumes an den zu untersuchenden Ast *K* gesetzt, woraufhin eine NNI Operation durchgeführt wird (NNI-Baum). Bei beiden Bäume wird dann mit Hilfe des schon beschriebenen Parsimony-Algorithmus festgestellt, wo Mutationen stattgefunden haben.

Dabei sollen 12 Grundtypen unterschieden werden, die im folgenden in vier Gruppen aufgeteilt werden und untersucht werden sollen:

- Maximal eine Mutation im Original bzw. im NNI-Baum.
- Eine Mutation im Original und zwei Mutationen im NNI–Baum.
- Zwei Mutationen im Original und zwei oder drei Mutationen im NNI-Baum
- Eine Mutation im Original und drei Mutationen im NNI-Baum

Die Menge möglicher Zustände (=Attributwert *A*) an den Wurzeln der Unterbäume *A,B,C* und *D* soll mit *a,b,c* und *d* bezeichnet werden. Die grauen Balken in den folgenden Graphiken zeigen an, daß die Schnittmenge der Attributwerte zweier Söhne eines Knotens leer ist, d.h. daß eine Mutation stattgefunden hat. In einer kurzen Auswertung jedes Grundtyps soll nachgeprüft werden, ob eine derartige Kombination durch eine NNI-Operation erreicht werden kann. Dies soll entweder an einem konkreten Beispiel bewiesen oder die Unmöglichkeit eines derartigen Beispiels aufgezeigt werden.

Erste Gruppe: Maximal eine Mutation im Original bzw. im NNI-Baum:

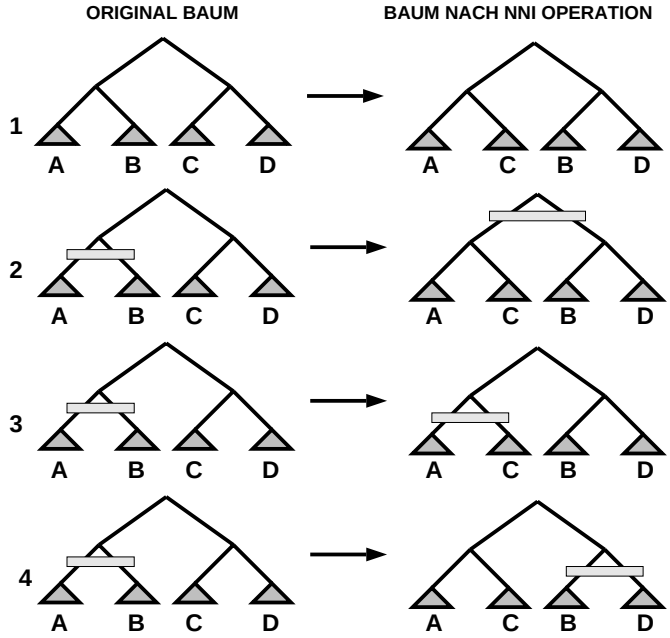


Abb. 55 Erste Gruppe: Maximal eine Mutation im Original bzw. im NNI-Baum

Auswertung der Grundtypen 1–4.

1. $$a \cap b \cap c \cap d \neq \emptyset$$

   Kann im Originalbaum keine Mutation festgestellt werden, so ist es plausibel, daß durch eine wie auch immer geartete NNI-Operation keine Mutationen induziert werden können. Durch Vertauschung von *A* und *D* ändert sich also nichts. Aus diesem Grunde ist es müßig zu untersuchen, ob sich derartige Kombinationen als möglich erweisen.
2. Der Grundtyp 2 läßt sich auch durch folgende Formel mathematisch korrekt beschreiben:

   $$a \cap b = \emptyset \ \wedge \ c \cap d \neq \emptyset \ \wedge \ (a \cup b) \cap c \cap d \neq \emptyset \wedge$$
   $$a \cap c \neq \emptyset \ \wedge \ b \cap d \neq \emptyset \ \wedge \ a \cap b \cap c \cap d = \emptyset$$

   Mögliche konkrete Werte für *a,b,c* und *d* wären zum Beispiel:

Baum	*a*	*b*	*c*	*d*
Attributwert	{A}	{C}	{A,C}	{C}

3. Mögliche Werte für Typ 3:

Baum	*a*	*b*	*c*	*d*
Attributwert	{A}	{C}	{C}	{C}

4. Mögliche Werte für Typ 4:

Baum	*a*	*b*	*c*	*d*
Attributwert	{C}	{A}	{C}	{C}

Gruppe 2: Eine Mutation im Original und zwei Mutationen im NNI–Baum:

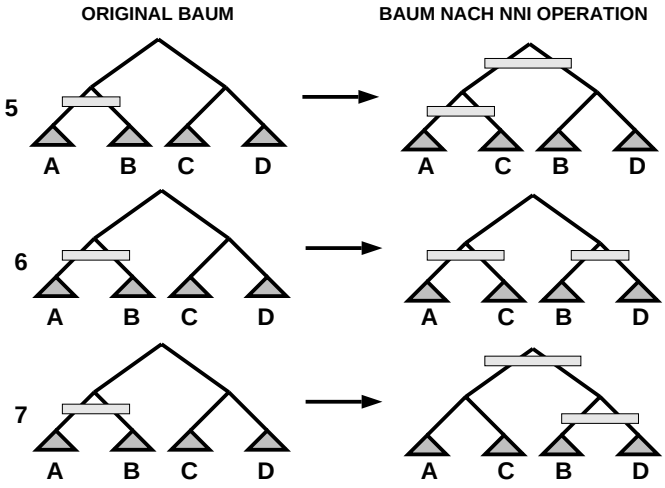


Abb. 56 Eine Mutation im Original und zwei Mutationen im NNI–Baum

Mögliche Werte für	*a*	*b*	*c*	*d*
5	{A}	{C}	{G}	{C,G}
6	{A}	{C}	{G}	{G}
7	{C}	{A}	{C,G}	{G}

Zwei Mutationen im Original und zwei oder drei Mutationen im NNI-Baum

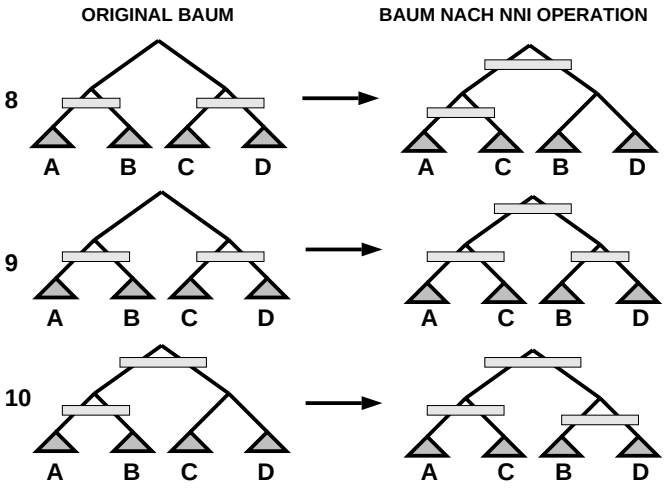


Abb. 57 Zwei Mutationen im Original und zwei oder drei Mutationen im NNI-Baum

VIII.5

Mögliche Werte für	*a*	*b*	*c*	*d*
8	{A}	{C}	{G}	{C}
9	{A}	{G}	{A,C}	{T}
10	Kombination nicht möglich: Beweis durch Widerspruch: Es gibt eine Base, die sowohl in *c* und *d* enthalten ist: $c \cap b \neq \emptyset$. Dann ist auch $(a \cup c) \cap (b \cap d) \neq \emptyset$ -> Wiederspruch			

Eine Mutation im Original und drei Mutationen im NNI-Baum

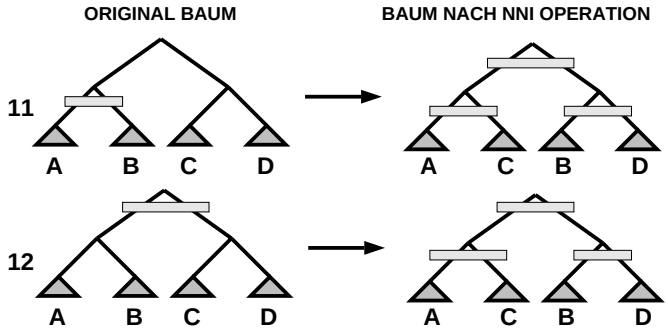


Abb. 58 Eine Mutation im Original und drei Mutationen im NNI-Baum

Situation 11 und 12 sind nicht möglich. Beweis:

11	Da $c \cap d \neq \emptyset$ setze $x = c \cap d$, Da $(a \cup (c \cap d)) \cup (b \cup (c \cap d)) \subset (a \cup c) \cup (b \cup d)$ und $x \subset (a \cup x) \cup (b \cup x)$ ist, folgt ein Widerspruch.
12	analog zu 11.

Somit ist der Beweis geführt, daß sich durch eine NNI–Operation die minimale Mutationsanzahl pro Spalte um höchstens eins ändert.

Diese NNI-Operation soll nun im einem realen Baum auf einer beliebigen zu untersuchenden Kante ausgeführt werden. Dann werden alle Spalten gezählt, bei denen die **Mutationsanzahl** durch diesen NNI sich

- **verringert,**
- **vergrößert,**
- **oder unverändert bleibt.**

Die Summe der gezählten Spalten ist logischerweise identisch mit der Länge des Alignments. Im weiteren sollen diese Zahlen als die Anzahl der Spalten bezeichnet werden,

- die einer gegebene Kante entgegen wirken,
- die eine gegebene Kante unterstützen
- oder die von einer gegebenen Kante unabhängig sind.

An jeder Kante eines Baumes mit minimalen ungewichteten Mutationen ist stets die Anzahl der unterstützenden Alignmentpositionen größer oder gleich der Anzahl der dieser Kante entgegen wirkenden Positionen

Gäbe es also zu einer beliebigen Kante mehr entgegenwirkende Alignmentpositionen als unterstützende, könnte man durch eine Vertauschung zweier benachbarter Kanten die Gesamtmutationsanzahl dieses Baumes reduzieren. Dies steht im Widerspruch zur Bedingung, daß der gegebene Baum einer mit minimaler Mutationsanzahl ist.

Wenn nun in einem Bootstrap-Prozeß durch eine zufällige Auswahl mehr einer Kante entgegen– als zu ihr unterstützend wirkende Alignmentpositionen ausgewählt würden, würde ein Parsimony-Algorithmus eine andere Baumtopologie berechnen.

Es reicht folglich aus, die Wahrscheinlichkeit zu berechnen, mit der die gegebene Topologie unterstützt wird.

Berechnung einer Bootstrap-Grenze aus der Anzahl der eine Kante unterstützenden / einer Kante entgegenwirkenden Alignmentpositionen

Aus der Zählung aus der Anzahl der Mutationen in den einzelnen Spalten nach dem NNI seien folgende Werte bekannt:

- **n_-:** Anzahl der entgegen wirkenden Positionen.
- **n_+:** Anzahl der unterstützenden Positionen.
- **n_0:** Anzahl der neutralen Positionen.

Daraus lassen sich folgende Werte berechnen:

- **n_A:** Länge des Alignments = n_- + n_+ + n_0
- **f_-:** $= \frac{n_-}{n_A}$ relative Anzahl der entgegenwirkenden Positionenen am Gesamtalignment
- **f_0**= $\frac{n_0}{n_A}$
- **f_+**= $\frac{n_+}{n_A}$

Die Gesamtwahrscheinlichkeit *P* errechnet sich aus der Summe der Einzelwahrscheinlichkeiten aller Kombinationen bei denen $n_+ > n_-$:

$$P = \sum_{i_+=1}^{n_A} \sum_{i_-=0}^{min(i_+-1, n_A-i_+)} p(i_+, n_A - i_+ - i_-, i_-)$$

mit

$$p(i_+, i_0, i_-) = (f_-)^{i_-} * (f_0)^{i_0} * (f_+)^{i_+} * \frac{n_A!}{(i_-!) * (i_0!) * (i_+!)}$$

Berechnung der Gesamtbootstrap-Abschätzung aus den beiden möglichen NNI-Operationen an einer Kante

Bis jetzt wurde davon ausgegangen, daß nur eine der beiden möglichen NNI-Operationen durchgeführt wird, es gibt jedoch zwei mögliche NNIs:

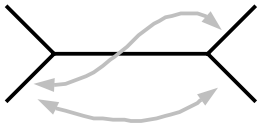


Abb. 59 Zwei mögliche Vertauschoperationen zweier an eine Kante angrenzenden Kanten

Für jede dieser Vertauschoperation kann ein Bootstrap-Wert B_i (*i=1,2)* berechnet werden. Jeder dieser Bootstrap-Werte entspricht der Wahrscheinlichkeit, daß in einem Bootstrap-Prozeß diese Vertauschoperation zu keiner Senkung der Gesamtmutationsanzahl des Baumes führt (unter der Bedingung, daß alle anderen Äste stabil sind). Zur Berechnung der Wahrscheinlichkeit, daß keine der Vertauschoperationen zu einer Verbesserung des Baumes (unter den bekannten Bedingungen) führt, gibt es zwei Möglichkeiten:

1. Die beiden Vertauschoperationen sind unabhängig, dann berechnet sich die Gesamtwahrscheinlichkeit *B* als Produkt der Einzelwahrscheinlichkeiten:

$$B = B_1 * B_2$$

2. Die beiden Operationen sind nicht unabhängig, dann kann nur eine obere Grenze für *B* abgeschätzt werden:

$$B = \min(B_1, B_2)$$

   In diesem Fall geht die Information über das größere Element von B_1, B_2 verloren.

Der tatsächliche Wert liegt irgendwo zwischen diesen beiden Extremen. Je nach persönlicher Vorstellung soll sich der Anwender selbst aussuchen, welche dieser beiden Formeln er bevorzugt:

- die sicherere Abschätzung mit der *min*–Funktion, bei der allerdings Information verloren geht.
- die informationsverlustfreie Formel, deren Ergebnis aber nicht mehr eine sichere obere Grenze für den Bootstrap-Wert darstellt.

Diskussion des Algorithmus unter realen Bedingungen

Alle bis jetzt durchgeführten Berechnungen wurden unter der Bedingung durchgeführt, daß der restliche Baum konstant ist. Gibt man diese Bedingung auf, gibt es im wesentlichen zwei mögliche Szenarien:

Szenario 1:

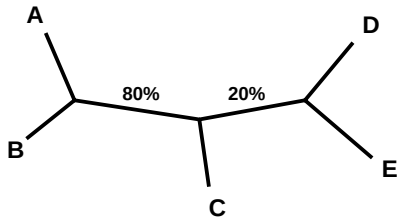


Abb. 60 Szenario 1

In diesem Szenario sei *E* ein Organismus, der sich schwer plazieren läßt. Daher resultiert auch der niedrige, nach dem neuen Verfahren berechnete Bootstrap-Wert *20%*. Werden während eines konventionellen Bootstrap-Verfahrens die Blätter *C* und *E* vertauscht, verringern sich auch die angrenzenden Bootstrap-Werte:

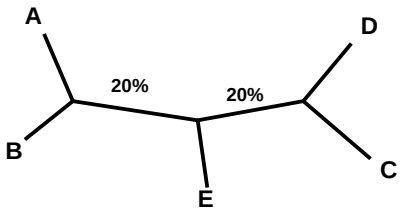


Abb. 61 Beeinflußung der angrenzenden Bootstrap-Werte

Der ursprünglich geschätzte Bootstrap-Wert von 80% war also zu hoch.

Zusammenfassend kann nur gesagt werden, daß die korrekt berechneten Bootstrapwerte in der Umgebung unsicherer Kanten deutlich niedriger ausfallen als in demselben Baum, sobald diese Kante entfernt wurde. Da der hier vorgestellte Algorithmus nur lokal arbeitet, wird er in derartigen Situationen immer zu hohe Werte liefern, nämlich die Werte eines Baumes, bei dem angrenzende instabile Kanten entfernt wurden. Ist einem Biologen dieser Sachverhalt bewußt, ist es sogar sinnvoll, die zu hohen Werte anzuzeigen, da sie unabhängig von einer möglicherweisen instabilen Teiltopologie innerhalb des Baumes sind. Anders ausgedrückt würde ein schwer zu plazierender Organismus in einem Baum alle Bootstrap-Werte auf sehr niedrige Werte reduzieren, so daß

1. dieser Problemorganismus nur schwer aufgespürt werden könnte und daß
2. die Bootstrap-Werte sich hauptsächlich auf diese Spezies beziehen würden.

Szenario 2: Bis jetzt wurde jede NNI-Operarion einzeln betrachtet. Im zweiten Szenario sollen nun gekoppelte NNI-Vertauschungen durchgeführt werden. Dazu wurden intensive Untersuchungen durchgeführt, inwieweit lokale Änderungen in einem Teilbaum Auswirkungen auf die restlichen Teilbäume haben. Dabei mußte leider festgestellt werden, daß die Reichweite des Einflusses lokaler Baumänderung ziemlich groß ist und sich oft über den ganzen Baum erstreckt. Dieser Sachverhalt macht es schwierig, wenn nicht gar praktisch unmöglich, komplexere Baumoperationen als den NNI in Bootstrap-Werte umzurechnen. Man könnte damit beginnen, jeweils zwei NNI-Operationen, die nicht zu weit im Baum voneinander entfernt liegen, durchzuführen. Durch den Einsatz leistungsfähiger (Parallel-)Rechner wird es in Zukunft vielleicht möglich sein, auch drei oder mehr NNI-Operationen bewerten zu können.

Zusammenfassend kann hierzu gesagt werden, daß eine exakte Analyse ein NP-vollständiges Problem darstellt und in praktischen Fällen niemals exakt gelöst werden kann. Es bleibt nur übrig, einfache Heuristiken zu entwickeln, die eine suboptimale Teillösung berechnen. Der in dieser Arbeit beschriebene Algorithmus, genau eine NNI-Operation auszuführen und zu bewerten, ist die einfachste aller möglichen Heuristiken und liefert immerhin gute Aussagen über die Korrektheit einer Kante in einem Baum.

VIII.5.3 Implementierungs-Hinweise

Die derzeitige **Implementierung zur Berechnung der** $n_{-/+/0}$ folgt genau dem Schema, das oben im Kapitel Theorie beschrieben wurde:

- Verschieben der Wurzel auf die zu untersuchende Kante.
- Neuberechnung der Attribute der Wurzel und der daran angrenzenden Knoten. Dabei werden die pro Spalte benötigten Mutationen mitprotokolliert.
- Durchführung eines der beiden möglichen NNIs mit Neuberechnung der Attribute derselben drei Knoten mit Protokollgenerierung.
- Durchführung aller anderen möglichen NNIs.
- Differenzbildung der Protokolle des Originalbaumes mit denen eines veränderten Baumes.

Bei der **Berechnung der Wahrscheinlichkeit** gibt es zwei Implementierungsprobleme:

1. Die Zwischenergebnisse in der Formel für *p* können leicht den Zahlenraum von **Maschinenzahlen** übersteigen (Fakultät der Länge des Alignments >1000). Deshalb geht man zum Logaritmus über:

$$p = \exp\left(\lg\left(f_-\right) * i_- + \lg\left(f_0\right) * i_0 + \lg\left(f_+\right) * i_+ + \lg\left(n_A!\right) - \lg\left(i_-!\right) - \lg\left(i_0!\right) - \lg\left(i_+!\right)\right)$$

2. Theoretisch müssen für alle Werte von i_- und i_+ die Funktion *p(...)* ausgewertet werden, dazu sind $\frac{n_A*(n_A-1)}{2}$ Berechnungen von *p* notwendig. Untersucht man die Werte von *p* genauer, stellt man fest, daß sie meistens kleiner als die Maschinengenauigkeit (**eps**) der Gesamtlösung sind, sie also ohne Änderung des Ergebnisses einfach weggelassen werden können. Ohne Beweis soll angemerkt werden, daß die Funktion $p(i_-, i_0, i+)$ für

$$i_- = n_-$$
$$i_+ = n_+$$
$$i_0 = n_0$$

ein Maximum hat, und nach allen Seiten hin monoton abfällt.[1] Ausgehend von diesem Optimum wird der Bereich für i_-, i_+ vollständig für alle Werte von *p* > *eps* abgetastet. Praktische Versuche haben gezeigt, daß sich so der benötigte Rechenaufwand bis auf 1/1000 reduzieren ließ.

Abschließend soll noch die benötigte Rechenzeit für ein konkretes Beispiel dargestellt werden:

Rechner	Baumknoten	Alignmentlänge	Laufzeit
Digital 400MHz	5000	1500	8sec

[1] Für $n_{-/+/0} \rightarrow \infty$ nähern sich die Werte von *p* der Gaußschen Glockenkurve an.

VIII.5.4 Erweiterung des Algorithmus auf gewichtete Alignmentspalten

Bis jetzt wurde davon ausgegangen, daß jede Spalte gleich gewichtet ist. Werden Alignmentspalten Gewichte zugeordnet, d.h. alle benötigten Mutationen dieser Spalte werden mit dem entsprechenden Gewicht multipliziert, muß zwischen zwei Situationen unterschieden werden:

1. Die Gewichte nehmen nur eine **kleine Anzahl unterschiedlicher diskreter Werte** an. Sie lassen sich also in eine kleine Menge von Gewichtsklassen einteilen (n_+). Dann wird zu jeder Gewichtsklasse eine negative Gewichtsklasse (n_-) erstellt. Aus dieser Gesamt-Gewichtsklassen-Menge zieht man nun alle möglichen Kombinationen von maximal n_A Klassen, deren Werte-Summe größer als Null ist, und summiert die Wahrscheinlichkeiten, daß eine Kombination bei einer NNI-Operation im Baum auch angenommen wird. Bei fester Anzahl von Gewichtsklassen läßt sich diese Berechnung in polynominialer Zeit durchführen.
2. Die Gewichte lassen sich **nicht in Klassen** einteilen. In diesem Fall ist es nicht möglich, eine korrekte Berechnung der Bootstrap-Werte in polynominialer Rechenzeit, d.h. relativ effizient, durchzuführen. In diesem Fall bleibt nur die Möglichkeit, durch Einsatz einer modifizierten Monte-Carlo-Methode das Ergebnis abzuschätzen.

Eine sehr wichtige Anwendung hierfür ist die Möglichkeit, für Maximum-Likelihood Baumberechnungs-Algorithmen nach demselben Verfahren vorzugehen, allerdings verändert sich anstelle der Mutationsanzahl pro Spalte die Likelihood, allerdings nicht um +1 oder –1, sondern um eine reelle Zahl. Diese reelle Zahl läßt sich auch als eine gewichtete 1 interpretieren, so daß der soeben beschriebene 2. Fall zutrifft.

Damit wäre ein allgemeines Verfahren entwickelt, das die Korrektheit einzelner Kanten in einem Baum abschätzt und dabei so wenig Rechenzeit benötigt, daß selbst große Bäume damit analysiert werden können.

VIII.5.5 Diskussion

Bis jetzt war es bei großen Bäumen nicht möglich, die Wahrscheinlichkeit der Korrektheit von Baumkanten zu berechnen. Man hat sich damit beholfen, die Astlänge als Maß dieser Wahrscheinlichkeit zu interpretieren[1]. Allerdings führte dieses Vorgehen nur zu einer sehr ungenauen Näherung, die dem Anwender nicht selten eine zu hohe Wahrscheinlichkeit der Korrektheit seines Baumes suggerierte.

Auch das hier vorgestelle neue Verfahren kann keine korrekten Bootstrap-Werte berechnen, doch sind die Berechnungen um ein Vielfaches genauer als die mit der traditionellen Vorgehensweise ermittelten. Da die Implementierung dieses Algorithmus noch sehr neu ist, müssen erst noch weitere Erfahrungen gesammelt werden, bevor eindeutigere Aussagen dazu

[1] Untersuchungen von W. Ludwig haben gezeigt, daß lange Äste hohe Bootstrapwerte erhalten und umgekehrt.

getroffen werden können. Erste Versuche haben jedoch einen unerwarteten, aber äußerst wichtigen Nebeneffekt dieser Methode festgestellt: Da die Bootstrapwert-Schätzungen nicht wie bei traditionellen Methoden mit Hilfe der Monte-Carlo-Methode berechnet wurden, können verschiedene Bootstrapwerte, die unter unterschiedlichen Bedingungen berechnet wurden, miteinander verglichen werden. Damit ist es zum Beispiel erstmals möglich, Filtersätze für eine Vorauswahl an Alignmentpositionen zu optimieren. Angesichts des enormen Bedarfs der Benutzer für eine solche Funktionalität ist es verwunderlich, daß bisher noch keine entsprechenden Programme veröffentlicht wurden.

Eines der wichtigsten Ziele, wenn nicht sogar das wichtigste Ziel überhaupt, muß es für jeden Bioinformatiker im Bereich Phylogenie sein, bei den Anwendern den Irrglauben an die Möglichkeit eines perfekten Baumalgorithmus auszutreiben. Dieses erreicht man nicht zuletzt dadurch, daß man die Kanten der Bäume mit Wahrscheinlichkeitswerten (=Bootstrap-Werten) versieht. Diese Anforderung wird durch den hier entwickelter Algorithmus schnell auch für größte Bäume zufriedenstellend gelöst.

VIII.6 Genetische Algorithmen

VIII.6.1 Das Grundproblem

Mit steigender Anzahl von Sequenzen wird es immer schwieriger, einen phylogenetischen Baum zu rekonstruieren. Während die Distanzverfahren aufgrund prinzipieller Probleme schon bei mittelgroßen Bäumen zu keinen brauchbaren Ergebnissen mehr kommen, können die Parsimony– bzw. Maximum Likelihood-Algorithmen theoretisch noch brauchbare Bäume konstruieren, aus praktischen Rechenzeitgründen muß jedoch auf Heuristiken zurückgegriffen werden. Die heute bekannten und eingesetzten Heuristiken wurden für kleine Bäume entwickelt, sie versagen jedoch bei den heute üblichen Datenmengen.

In anderen Bereichen der kombinatorischen Optimierung wird in der Forschung schon länger erfolgreich mit genetischen Algorithmen [Mühlenbein 88] experimentiert. Außerdem wurden bereits erste erfolgreiche Versuche [Gerrits] durchgeführt, dieses Prinzip auch auf phylogenetische Bäume anzuwenden.

Daher sollen hier nun Verfahren vorgestellt werden, die entwickelt wurden, um die effiziente Anwendung von genetischen Algorithmen auf die phylogenetische Analyse zu erlauben. Dabei soll die Entwicklung und Optimierung sowohl eines sogenannten Mutations- als auch eines sogenannten Crossoveralgorithmus behandelt werden. Die hier entwickelten Grundalgorithmen sind in ihren Grundzügen schon länger bekannt, die Arbeit mit ihnen konzentrierte sich daher darauf, sie für den praktischen Einsatz zu modifizieren und zu optimieren. So hat sich herausgestellt, daß durch eine geschickte Umsetzung die Komplexität deutlich veringert werden konnte, was in der Praxis eine Beschleunigung der Algorithmen um bis zu dem Faktor 1000 zu Folge hat.

VIII.6.2 Einführung

Als **genetische Algorithmen** bezeichnet man Methoden, die sich für die Problemlösung Strategien aus der Evolutionstheorie zunutze machen. Dabei versucht man nicht, eine einzige optimale Lösung zu berechnen, sondern aus einer Menge nicht optimaler Teillösungen durch Anwendung der Prinzipien **Mutation**, **Selektion** und **Crossover** schrittweise zu besseren Lösungen zu kommen, solange sich in praktikabler Rechenzeit Verbesserungen einstellen [Goldberg 89].

Ausgehend von einer Menge möglicher Startlösungen (**Startpopulation**) geht man nach folgendem Schema vor:

- Im Laufe des Algorithmus (der Evolution) überleben nur die am besten bewerteten Lösungen (**Selektion**). Die anderen sterben (werden gelöscht).
- Mit einer gewissen Wahrscheinlichkeit werden die einzelnen Lösungen zufällig oder gezielt verändert (**Mutation**).

- Nach einer bestimmten Zeit werden die verschiedenen Teillösungen wieder miteinander kombiniert (**Crossover**).

Dieses Vorgehen ist der Evolution von genetischen Sequenzen nachempfunden. Um Verwechslungen von algorithmischen Genen mit biologischen Genen zu vermeiden, wird im folgenden das Wort **Gene** bewußt nicht verwendet.

VIII.6.3 Existierende Heuristiken und ihre algorithmische Beschränkung

Die heute vielversprechendsten Algorithmen zur Berechnung großer Stammbäume arbeiten nach dem Parsimony- bzw. Maximum Likelihood-Algorithmus. Mit Hilfe dieser Algorithmen kann allerdings nur ein gegebener Baum bewertet werden, sie liefern keine Vorschrift zur algorithmischen Berechnung von Bäumen. Um nicht alle Baumtopologien testen zu müssen, werden Heuristiken eingesetzt, die innerhalb kurzer Zeit eine gute Lösung finden sollen. Durch Kombination verschiedener Heuristiken versucht man, mit möglichst hoher Wahrscheinlichkeit die optimale Lösung zu finden.

Funktionale Betrachtung der Heuristiken

Das Prinzip aller in der Stammbaumrekonstruktion eingesetzten Algorithmen beruht darauf, ausgehend von einer relativ guten Lösung durch lokale Modifikationen der Topologie eine bessere Lösung zu finden. Die dabei auftretenden Schwierigkeiten können mit Hilfe des folgenden Schaubildes erklärt werden:

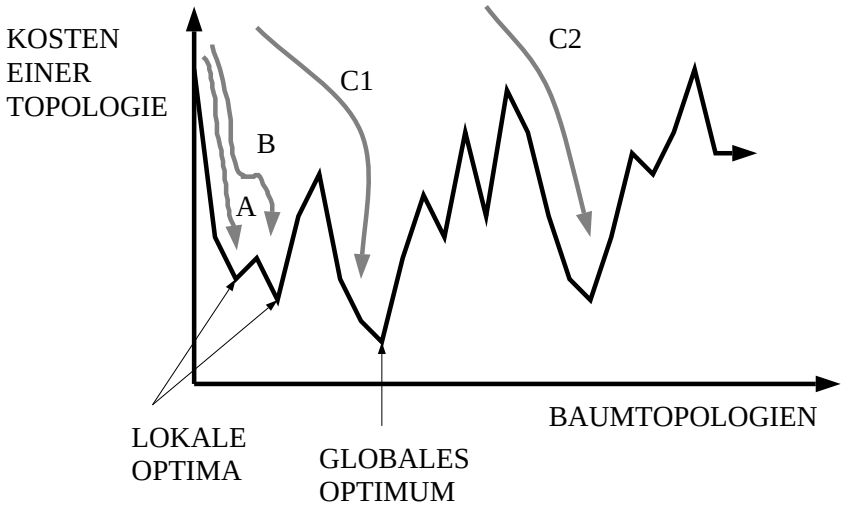


Abb. 62 Arbeitsweise der Heuristiken

Die Heuristik *A* läuft streng monoton zum nächsten lokalen Optimum (**Hill Climbing** Strategie). Sie kann jedoch dieses nicht wieder verlassen, da sie auf dem Weg zu einem besseren Optimum schlechter bewertete Zwischenstufen einnehmen müßte (typisches Beispiel: **NNI**).

Heuristik *B* hat die Eigenschaft, von einem lokalen Optimum kleinere Verschlechterungen auf der Suche nach einer besseren Lösung zu akzeptieren. Dadurch kann sie einen größeren Suchraum abdecken und erreicht eine höhere Wahrscheinlichkeit beim Finden der optimalen Lösung. Je nach Festlegung dieser in Kauf genommenen Verschlechterungen kommt man zu unterschiedlichen bekannten Verfahren:

- **Simulated Annealing** [Lundy 85]: Ist ausgehend von einer gegebenen Topologie die nächste schlechter bewertet, wird sie mit einer Wahrscheinlichkeit abgelehnt, die umgekehrt proportional zur Verschlechterung ist. Die Idee dieser Heuristik wurde der Physik entlehnt: kühlt man eine normale Schmelze langsam ab, kristallisiert diese (lokales Optimum). Die langsame Abkühlung garantiert, daß solche Teilkristalle mit einer gewissen Wahrscheinlichkeit wieder in den flüssigen Zustand übergehen. Mit fallender Temperatur kristallisiert sich im Idealfall nach und nach ein Einkristall heraus (globales Optimum).
- **Threshold Accepting** [Dueck 88]: Legt man einen dynamischen (im Laufe der Zeit fallenden) Schwellwert einer gerade noch akzeptierten Verschlechterung fest, kann man die maximal in Kauf genommene Verschlechterung kontrollieren und erhält die Heuristik des Threshold Accepting.

Erhöht man die in Kauf genommene Verschlechterung, erhöht sich zwangsläufig die Wahrscheinlichkeit, mit der das globale Optimum gefunden wird. Dabei steigt jedoch auch der Suchraum entsprechend an, im Extremfall werden alle Topologien abgesucht, eine Vorgehen, das man durch den Einsatz von Heuristiken gerade vermeiden wollte.

Im Bereich der Stammbaum-Berechnung hat sich auch die Heuristik des zufälligen Startpunktes als effektiv herausgestellt. Dabei wird der Gesamtalgorithmus mehrfach durchgeführt, allerdings jeweils von einem unterschiedlichen Startpunkt aus. In obiger Graphik sind diese Startpunkte mit *C1* und *C2* bezeichnet.

In der Vergangenheit hat sich gezeigt, daß, je größer das zu lösende Problem ist, desto weniger führen Simulated Annealing und Threshold Accepting zu einer brauchbaren Lösung.

Die in den heutigen Programmen fastDNSml und phylip eingesetzten Heuristiken beruhen im wesentlichen darauf, zwei Kanten in einem Baum miteinander zu vertauschen [Swoffort 1990]:

NNI

Wie bereits mehrfach erwähnt werden bei einem Nearest Neighbour Interchange zwei benachbarte Kanten in einem Baum ausgetauscht. Es gibt zwei verschiedene Möglichkeiten dies durchzuführen:

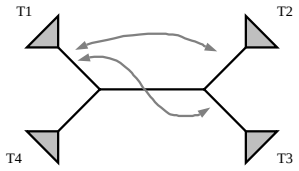


Abb. 63 NNI Operationen in einem Baum

Durch NNI erreichbare Topologien:

Abb. 64 Durch NNI erreichbare Topologien

Nach der Vertauschung wird der Baum neu bewertet. Kommt man dabei zu einer besseren Lösung, wird der modifizierte Baum beibehalten, andernfalls geht man zum Ursprungsbaum zurück.

Verschiebung einer Kante

Im wesentlichen erreicht man durch einen NNI, daß eine Kante eine andere 'überspringt'. Man kann dieses Vorgehen auch verallgemeinern und kommt damit zu einem Algorithmus, der einen Unterbaum entfernt und diesen maximal *n* Kanten von seiner ursprünglichen Position entfernt wieder einfügt[1]:

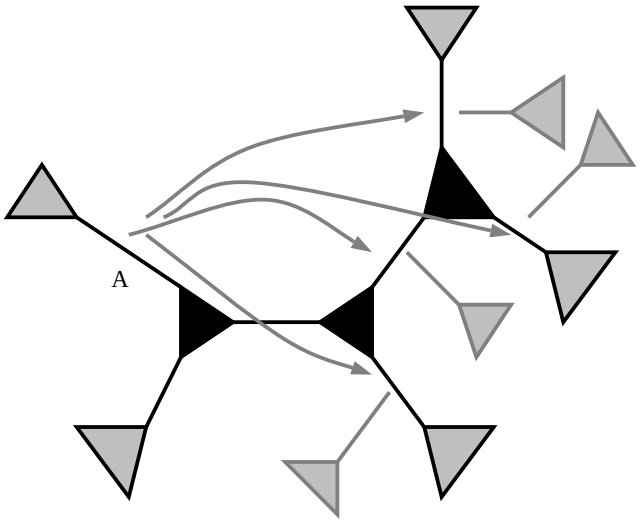


Abb. 65 Überspringen von Kanten

Diskussion

Beide hier vorgestellten Algorithmen haben die Eigenschaft, daß sie nur eine einzige Kante entfernen und neu zusammensetzen. Sie unterliegen somit zwei grundsätzlichen Beschränkungen, die an zwei Beispielen exemplarisch gezeigt werden sollen:

- **Kombinierte Vertauschungen:**

[1] Bei allen nun folgenden Graphiken werden die inneren Knoten durch ein schwarzes Dreieck dargestellt. Diese Veränderung in der Darstellungsform ist in einer inneren Orientierung der Knoten begründet, die jedoch erst später eingeführt wird.

VIII.6

Es sei folgender Ausgangsbaum gegeben:

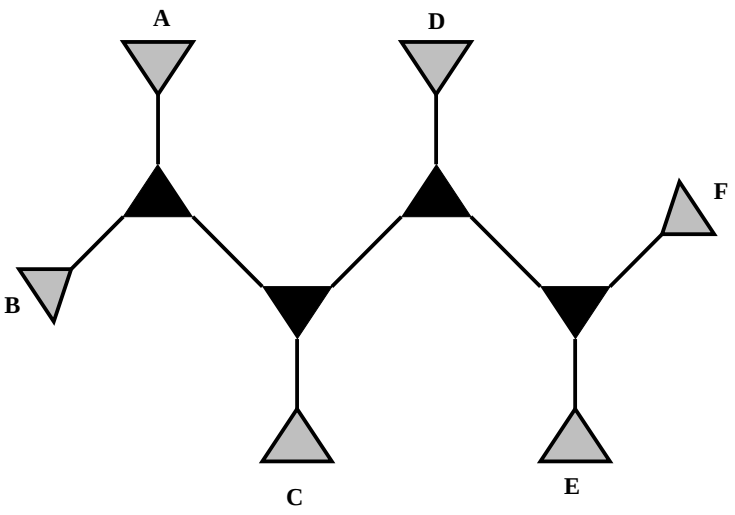


Abb. 66 Ausgangsbaum

Die optimale Lösung dieses Baumes habe folgende Gestalt:

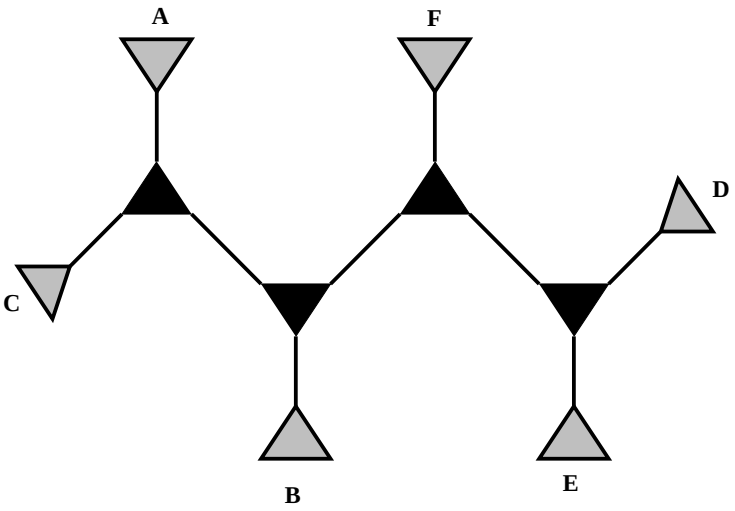


Abb. 67 Angenommene optimale Lösung des Beispielbaumes

Da alle nur durch eine Verschiebung erreichbaren Topologien schlechtere Bewertungen erhalten als der Ausgangsbaum, kann mit einer einzigen Verschiebung nicht die optimale Lösung erreicht werden. Die mögliche Lösung dieses Problems, nämlich der Übergang zu Simulated Annealing oder Threshold Accepting führt in der Praxis zu einem nicht mehr tolerablen Rechenleistungsbedarf.

- **Unabhängige Teilbäume:**

Bereits bei Bäumen mittlerer Größe (20–150 Blätter) kommt man mit Hilfe lokaler Änderung nicht mehr zum Ziel, ein nicht nur lokales Optimum zu finden. Erst der

wiederholte Durchlauf des Gesamtalgorithmus mit unterschiedlichen Startwerten läßt die Wahrscheinlichkeit, die optimale Lösung zu finden, über die 50%-Marke steigen.

Geht man zu großen Bäumen über, muß man feststellen, daß es relativ unabhängige Teilbäume gibt, die praktisch unabhängig von einander optimiert werden können. Arbeitet man nun mit zufälligen Startpunkten, werden mit einer gewissen Wahrscheinlichkeit jeweils unterschiedliche Teilbäume optimal sein, die Gesamtwahrscheinlichkeit der Optimalität aller Teilbäume errechnet sich dann als das Produkt der Einzelwahrscheinlichkeiten.

Liegt also die Wahrscheinlichkeit, einen optimalen Teilbaum zu finden bei einem gegebenen Startpunkt bei etwa 1% und somit mit mehreren hundert Durchläufen deutlich über der 90% Marke, liegt die Wahrscheinlichkeit zur Findung einer optimalen Gesamtlösung bei 50 Teilbäumen pro Durchlauf bei $0.01^{50} = 10^{-100}$. Somit hat man selbst bei mehreren Milliarden Durchläufen keine Chance, die optimale Lösung zufällig zu treffen.

Um dieses Problem zu lösen, wurden in der Vergangenheit die Bäume von Hand in Unterbäume zerlegt, einzeln optimiert und wieder zusammengefügt. Einen allgemeineren Ansatz zu finden, der auf dieses aufwendige Prozedere verzichten kann, soll Gegenstand dieses Kapitels sein. Mit Hilfe von genetischen Algorithmen soll erreicht werden, daß die Teilbäume unterschiedlicher lokaler Optima miteinander ausgetauscht werden können.

VIII.6.4 Der Mutationsalgorithmus

Übersicht

Basierend auf den Ideen des Kernigham-Lin-Algorithmus [Lin 73] entwickelte Bandelt eine weitere Heuristik zur Optimierung phylogenetischer Bäume. Diese sollte hier umgesetzt und optimiert werden. Die Idee dieses Algorithmus ist es, jeweils ausgehend von allen Kanten, einen NNI durchzuführen, jeweils beide neuen Topologien zu bewerten und, ausgehend von der besseren der beiden, an allen maximal 4 angrenzenden Nachbarkanten rekursiv fortzufahren.

In Laufe der Vorarbeiten konnte schon auf einschlägige Vorversuche mit dieser Vorgehensweise in einem leicht geänderten Kontext zurückgegriffen werden. Dabei hatte sich jedoch herausgestellt, daß die Verfolgung nur des optimalen Pfades oft zu unbefriedigenden Ergebnissen führte. Zur Verbesserung dieser Vorgehensweise wurde das Konzept des Bereiches um den optimalen Pfad entwickelt, eine Heuristik, die zwar mehr Rechenleistung benötigt, doch die Wahrscheinlichkeit der Findung der optimalen Lösung von 5% auf über 80% anhebt.

Der Algorithmus

Ausgehend von einer gegebenen Topologie werden an einer Kante alle möglichen Vertauschungen durchgeführt und bewertet. Die sich jeweils ergebenden besten Topologien werden

als Startpunkt für weitere Vertauschungen gewählt. Um zirkuläre Rekursionen zu vermeiden, wird eine geänderte Kante für alle weiteren Rekursionsstufen festgefroren, d.h. es gibt keine weitere Vertauschung von an diese Kante angrenzenden Kanten.

Stößt man im Laufe dieses Vorgehens auf eine besser bewertete Topologie als die Ausgangsituation, wird diese realisiert und die ursprüngliche verworfen. In allen anderen Fällen verbleibt man bei der am besten bewerteten Ausgangstopologie.

Sei eine Kante *K* gegeben. Ausgehend von dieser können durch NNI zwei unterschiedliche Topologien erzeugt werden. Bei jeder dieser beiden Topologien grenzen nun an die Kante *K* maximal 4 innere Kanten an, von denen wieder jeweils zwei Kanten weiterführen. Auch an diesen sind wieder jeweils zwei verschiedene NNI Operationen möglich. Somit erhält man maximal 8 weitere Vertauschungsmöglichkeiten:

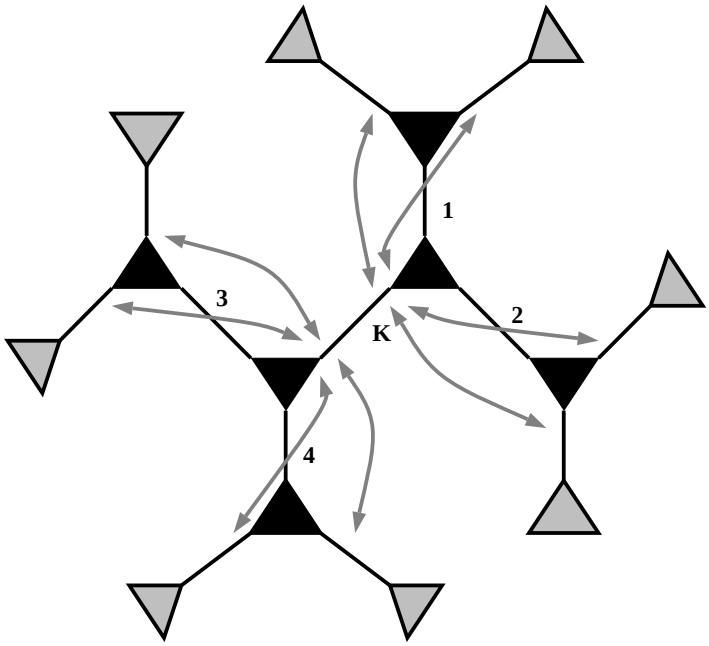


Abb. 68 Basisoperation des modifizierten Kernigham-Lin Algorithmus

Mit steigender Rekursionstiefe *deep* ergeben sich theoretisch exponentiell viele NNI Möglichkeiten: $2 * 8 * 8 * 8 * ... * 8 = 2 * 8^{deep-1}$, eine Menge, die wegen des großen Rechenaufwandes durch eine sinnvolle Begrenzung reduziert werden muß. Dazu wurden zwei Strategien entwickelt:

- **Statische Begrenzung** der verfolgten Topologien: In einer benutzerdefinierbaren Tabelle werden für jede Rekursionstiefe *deep* die maximal rekursiv weiterzuverfolgenden Topologien B_{deep} angegeben. So kann z.B. durch eine Begrenzung auf **2,2,2,2,1,1,1,1,1,1,1,0** die Anzahl der zu bewertenden Bäume auf 16 begrenzt werden (2*2*2*2*1*...*1). Allgemein gilt

$$\max_{d=0}^{d<BlaetterDesBaumes} \prod_{r=0}^{r<d} B_r$$

- **Dynamische Begrenzung**: Sei eine Ausgangstopologie *T* mit dem Wert *w* bewertet. Verschlechtert sich die Bewertung bei einem NNI wesentlich, sinkt die Wahrscheinlichkeit, daß diese schlechtere Topologie durch weitere Rekursionsstufen zu einer insgesamt verbesserten Topologie führen wird. So sollen nur solche Topologien weiterverfolgt werden, deren Bewertung nicht schlechter ist als die Bewertung der bisher optimale Topologie plus einen rekursionstiefenabhängigen Schwellwert *S(deep)*.

Anschaulich kann man sich die dynamische Suchbegrenzungen folgendermaßen vorstellen:

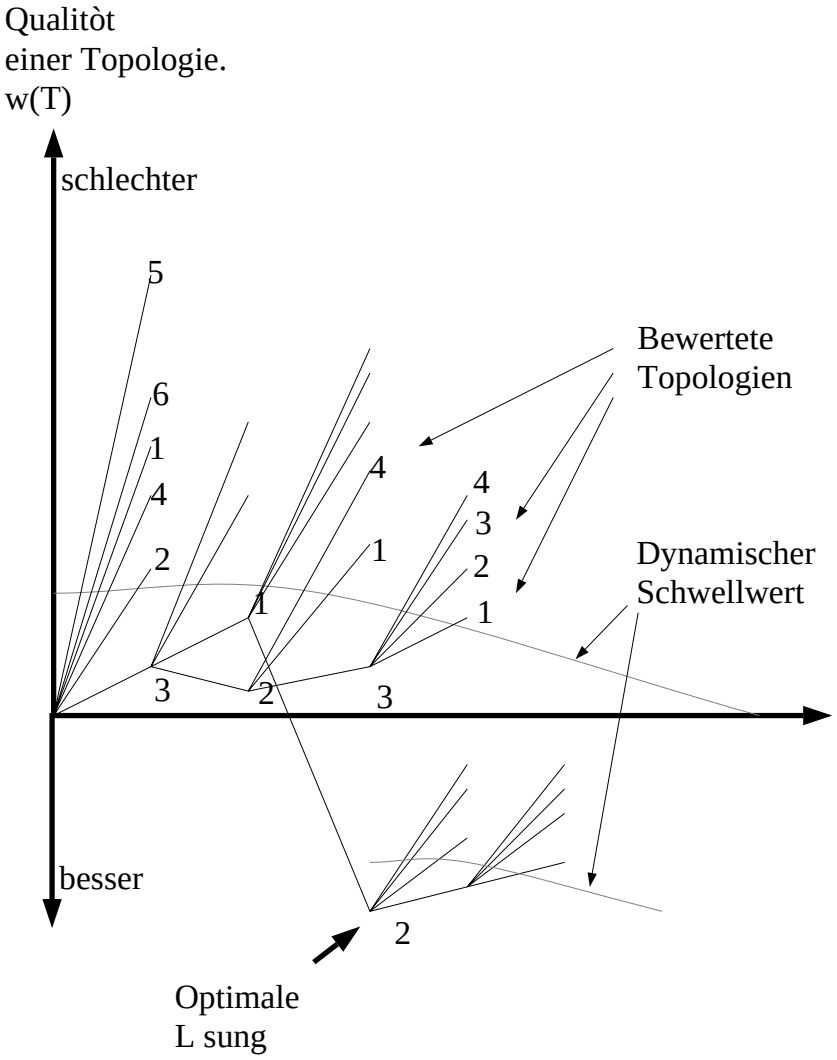


Abb. 69 Reduzierung des Suchraumes im modifizierten Kernigham Lin-Algorithmus

Eine effiziente Implementierung

Die für diesen Algorithmus notwendigen Basisoperationen sind:

- Vertauschen zweier benachbarter Kanten (NNI)
- Bewerten einer durch NNI veränderten Topologie.

- Rückgängigmachen nicht optimaler Topologien.

Das Vertauschen zweier benachbarter Kanten in einem Baum ist eine einfache Operation, die mit konstantem Rechenaufwand durchgeführt werden kann, zudem ist die benötigte Rechenleistung, verglichen mit der Bewertung eines Baumes, verschwindend gering.

Bei der Bewertung einer geänderten Topologie müssen im wesentlichen alle inneren Sequenzen auf den Pfaden der Veränderung und die der Wurzel neu berechnet werden. Verschiebt man die Wurzel vorher auf die mit NNI zu bearbeitende Kante, müssen exakt drei innere Sequenzen neu berechnet werden:

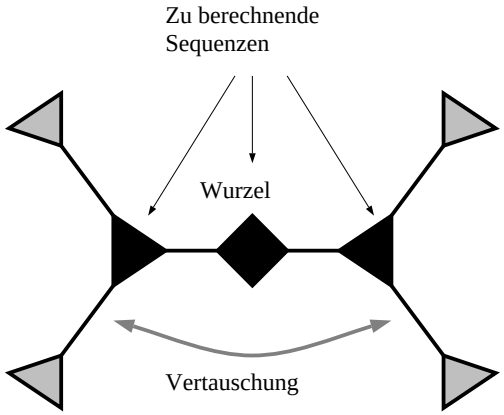


Abb. 70 Berechnung der inneren Sequenzen nach einem NNI

Das ergibt bei maximal 8 möglichen NNI Operationen bei einem rekursiven Kernigham-Lin-Schritt 24 innere Sequenzberechnungen.

Durch eine geschickte Implementierung kann erreicht werden, daß man mit Hilfe eines *Undo*–Stacks bereits berechnete und bewertete Topologien zwischenspeichert. Mit Hilfe dieses Konzeptes kann ein Rücksprung auf eine ältere Topologie in konstanter Zeit und mit minimalem Rechenaufwand realisiert werden.

Diskussion

In mehreren Testläufen an realen Daten wurde der modifizierte Kernigham-Lin Algorithmus testweise mit der Algorithmenkombination, die im Programm phylip eingesetzt wurde, verglichen. Es hat sich bei der Anwendung bei kleinen Bäumen gezeigt,

daß in etwa bei 50% der Versuche phylip, in den anderen
50% Kernigham-Lin zu einer jeweils besseren Lösung kam.

Dies zeigt, daß der Kernigham-Lin-Algorithmus einer guten Kombination aus Heuristiken nicht überlegen ist. Sein Vorteil besteht jedoch darin, daß er ein einziger Algorithmus, keine Kombination und somit leichter parametrisierbar ist. Für den weiteren Einsatz wurde aus diesem Grund dem Kernigham-Lin-Algorithmus für eine Testimplementierung der Vorzug gegeben. Erst mit genügend Erfahrung auf dem Gebiet der genetischen Algorithmen sollte hier zu einer Kombination verschiedener Heuristiken zurückgekehrt werden.

VIII.6.5 Entwicklung eines Crossover-Algorithmus

Einführung

Die zentrale Idee genetischer Algorithmen ist, daß suboptimale Teillösungen miteinander kombiniert werden, um effizient zu einer besseren Lösung zu gelangen. In den klassischen Anwendungsfällen dieser Algorithmen lassen sich diese Teillösungen in einer Zeichenkette kodieren[1].

Zwei derart in Zeichenketten kodierte Teillösungen können miteinander kombiniert werden, indem diese beiden Zeichenketten an einer meist zufällig gewählten Position aufgetrennt und Teilzeichenketten zwischen den Lösungen ausgetauscht werden (Crossover):

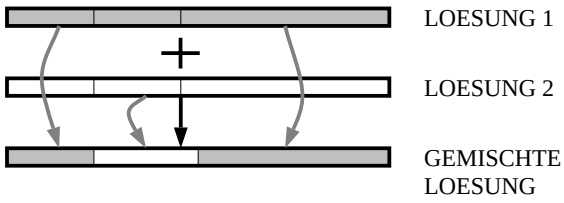


Abb. 71 Prinzip des Crossovers bei einfachen Zeichenketten

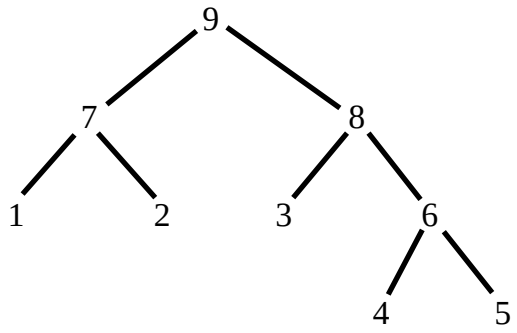
Auch Binärbäume lassen sich leicht in Zeichenketten kodieren. Dabei hat sich in der Biologie die Darstellung als wohlgeformter Klammerausdruck durchgesetzt. Ein als wohlgeformter Klammerausdruck kodierter Binärbaum ist

- entweder eine Blatt *B* (**Terminal**)
- oder ein Paar (*X*,*Y*), wobei *X* und *Y* wohlgeformte Klammerausdrücke sind. *X* wird als linker und *Y* als rechter Unterbaum des Knotens (*X*,*Y*) bezeichnet.

Ein einfaches Beispiel verdeutlicht dies:

---

[1] Betrachtet man eine Katze als eine Teillösung der Evolution, stellt man fest, daß sich diese vereinfacht gesehen auch in einer Zeichenkette kodieren läßt, ihrer Zellkern-DNS. Diese Analogie zwischen genetischen Algorithmen und biologischer Evolution hat dazu geführt, diese Zeichenketten als Gene zu bezeichnen.



kann ausgedrückt werden durch:

$$((1,2),(3,(4,5)))$$

Abb. 72 Beispiel einer wohlgeformten Klammerdarstellung

Nicht jeder Teilbaum eines in einer Klammerdarstellung kodierten ungewurzelten Baumes ist als wohlgeformter Klammerausdruck kodiert.
Der Beweis hierfür soll durch ein Gegenbeispiel geführt werden:

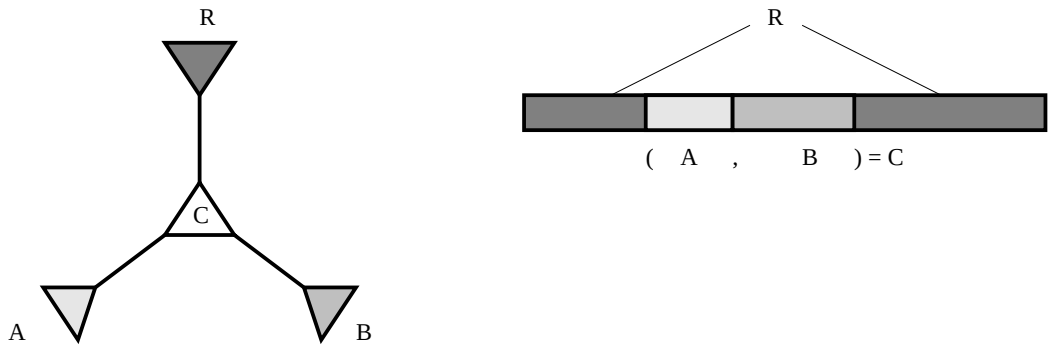


Abb. 73 Darstellung eines Baumes als wohlgeformter Klammerausdruck

Der Teilbaum *R* zerfällt in zwei Zeichenketten, er ist also nicht als wohlgeformter Klammerausdruck kodiert.

Jeder auf diese Weise wohlgeformte Klammerausdruck läßt sich als Binärbaum interpretieren und umgekehrt. Dies bedeutet, daß die Kombination zweier Bäume sich wieder als wohlgeformter Klammerausdruck schreiben läßt, oder, anders ausgedrückt, daß eine Kombination zweier wohlgeformter Klammerausdrücke wieder einen solchen ergeben muß. Aus diesem Grund muß ein wesentlich komplexerer Algorithmus als eine einfache Vertauschung von Teilzeichenketten gefunden werden, um einen Crossover-Algorithmus zu realisieren.

Insgesamt stellte sich heraus, daß direktes Arbeiten auf einer in einer Zeichenkette kodierten Lösung zu ineffizient ist, um damit sinnvoll Binärbäume verarbeiten zu können. Um dieses Problem zu lösen wurde zum Beispiel von Gerrits der Versuch unternommen, Bäume in einer redundanten Matrix zu speichern [Gerrits]. Das Ziel der vorliegenden Arbeit war jedoch eine möglichst effiziente, d.h. redundanzarme, Bearbeitung von Binärbäumen, die mit dem Matrixverfahren nicht erreichbar wäre. Daher arbeitet der hier entwickelte Algorithmus

direkt auf ungewurzelten Binärbäumen. Deren Kodierung ist für die Algorithmen nicht von Bedeutung und soll somit hier nicht diskutiert werden.

Während in den traditionellen Anwendungsgebieten genetischer Algorithmen ein Crossover einfach durch Vertauschen von Teilzeichenketten möglich ist, stößt man bei den hier behandelten Bäumen auf nicht unbedeutende Probleme. Deren Lösung konnte auf der Basis des von Prof. Bandelt[1]entwickelten Grundschemas Kombination zweier Bäume angegangen werden. Für dieses Schema wurde eine nicht nur einfache sondern auch höchst effiziente Implementierung entwickelt, die nicht nur für eine zufällige Kombination eine effiziente Bewertung liefert, sondern die auch in kurzer Zeit alle existierenden Kombinationen aus zwei Bäumen findet und berechnet.

Der Algorithmus

Die dem Crossover-Algorithmus zugrundeliegende Idee ist, einen (evtl. sehr gut bewerteten) Unterbaum *U* eines Baumes in einen Zielbaum einzufügen. Dazu müssen im Zielbaum *Ziel* alle Organismen entfernt werden, die auch in dem Baum *U* enthalten sind. Der Unterbaum *U*, bestehend aus den Blättern U1 bis U4, wird nun testweise an allen den Stellen in dem reduzierten Baum *Ziel* eingefügt, an denen zuvor die entfernten Organismen plaziert waren. An einem einfachen Beispiel kann dies verdeutlicht werden:

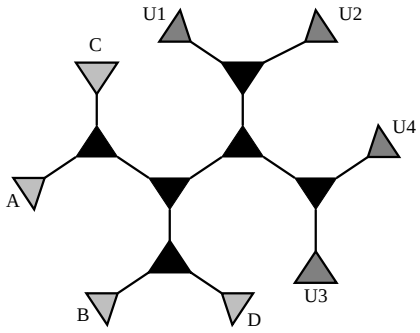


Abb. 74 Baum *Quelle* mit Unterbaum *U*, bestehend aus den Blättern *U1 – U4*

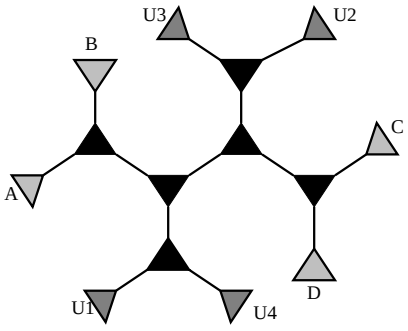


Abb. 75 Baum *Ziel* mit den markierten Blättern *U1 – U4* des Unterbaumes *U*

[1] Der Autor dieser Arbeit erhielt diese Ideen in persönlichen Gesprächen mit Prof. Bandelt.

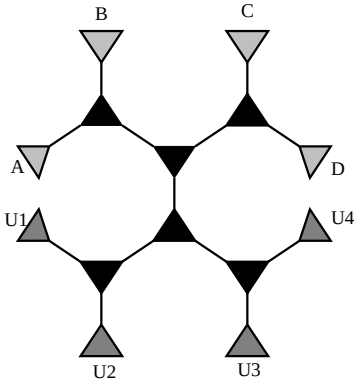


Abb. 76 Der einzig mögliche, durch Crossover von *Quelle* und *Ziel* mit diesem Algorithmus zu berechnende Baum: Im Baum Ziel wurden alle Blätter *U1 – U4* entfernt und an deren alter Position der Unterbaum *U* des Baumes *Quelle* eingefügt

Zusammengefaßt läßt sich dieser Algorithmus folgendermaßen formulieren:

1. Für zwei gegebenen Bäume *Quelle* und *Ziel* suche einen Unterbaum *U* von *Quelle*.
2. Für alle Blätter *L*, die in dem Unterbaum *U* vorkommen, führe folgendes aus:
   - Entferne *L* aus *Ziel* und markiere die veränderte Kante in *Ziel*
3. Füge den Baum *U* an alle Stellen in *Ziel* ein, an denen einen Kante markiert ist.
4. Bei Bedarf verbessere den so erhaltenen Baum mit Hilfe lokaler Optimierungen (wie zum Beispiel NNI) bis zu einem lokalen Optimum.
5. Bewerte alle auf diese Weise erhaltenen Bäume und speichere den am besten bewerteten. Dieser gespeicherte Baum ist dann die Lösung.

Dieser Algorithmus beschreibt die allgemeine Vorgehensweise des Crossover bei Bäumen. Allerdings beinhaltet er keine effiziente Lösung für die Schritte 2 und 3. Diese soll im folgenden vorgestellt werden.

Datenstruktur zur Darstellung von Bäumen

Bei ungewurzelten Bäumen gibt es primär die Begriffe des Vater- bzw. des Sohnknotens nicht. Bei einer gewurzelten Darstellung stellt jeder innere Knoten die Wurzel eines Unterbaumes dar, in der ungewurzelten Form grenzen an jeden inneren Knoten drei Unterbäume an. Um dennoch mit Hilfe von inneren Knoten Unterbäume beschreiben zu können, werden die inneren Knoten durch eine einfache Ringstruktur aus drei Untereinheiten (**Quarks**) ersetzt:

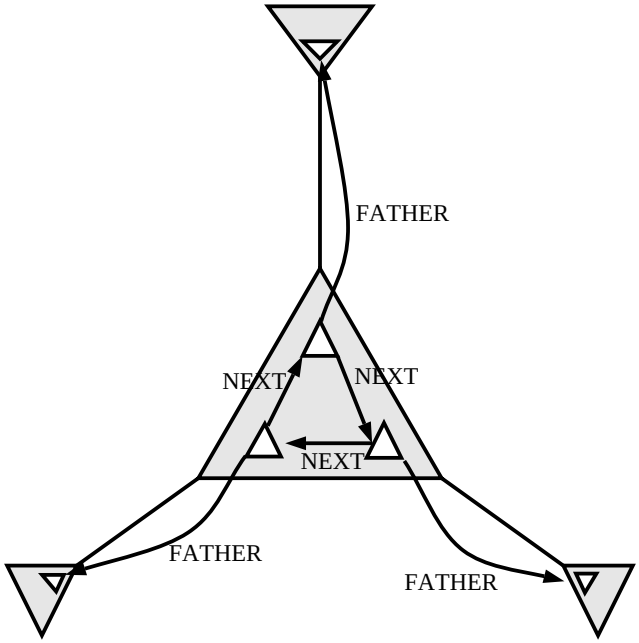


Abb. 77 Innere Ringstruktur innerer Baumknoten

Jeder Kante, die an einem inneren Knoten endet ist ein Quark **q** zugeordnet. Jeder dieser Quarks *q* besitzt sowohl einen Zeiger (**Father**) auf den Quark auf der anderen Seite seiner zugehörigen Kante, als auch einen Zeiger (**next**) auf einen weiteren Quark desselben inneren Knotens. Alle drei Quarks sind daher im Urzeigersinn ringförmig verbunden.

Es gilt: Father(Father(q)) = q

Eine Kante spaltet eine Baum in zwei Unterbäume. Die an diese Kante angrenzenden Quarks identifizieren (**repräsentieren**) exakt diese Unterbäume, und zwar jeder Quark jeweils den Baum, dessen Wurzelelement er zugeordnet ist. Die Vereinigungsmenge aller Blätter der Bäume Q und Father(Q) ist die Menge aller Blätter. Da die Begriffe Unterbaum und Quark semantisch beliebig austauschbar sind, sollen sie auch im weiteren Verlauf syntaktisch beliebig austauschbar sein.

Blätter *L* eines Baumes besitzen logischerweise nur einen Quark q_L, der keinen *next* Zeiger besitzt, d.h. *Next(q_L)* ist undefiniert. *q* wird in diesem Fall als **Blattquark** bezeichnet.

Bei einem Binärbaum gilt: $Next(q) \neq q$ und *Next(Next(Next(q))) = q*.

Vereinigung zweier benachbarter Unterbäume

Zwei Unterbäume *U* und *V* lassen sich zu einem echten Unterbaum, also einen Baum, der nicht identisch mit dem Gesamtbaum ist, vereinigen, wenn gilt:

- Sei q_U, der den Unterbaum *U* identifizierenden Quark
- Sei q_V, der den Unterbaum *V* identifizierenden Quark
- *Next(Father(q_U)) = Father(q_V)* oder
  Next(Father(q_V)) = Father(q_U)

Der den Vereinigungsunterbaum *S* repräsentierende Quark q_S wird mit folgender Rechenvorschrift berechnet:

$$q_S = Next(Father(q_U))\ falls\ Next(Father(q_V)) = Father(q_U)$$
$$q_S = Next(Father(q_V))\ sonst$$

Vereinigung zweier Mengen von Unterbäumen

Seien **Q** und **P** zwei disjunkte, ungeordnete Untermengen aller Quarks **B** eines Baumes. Dabei sind

$$Q = \{q_1, q_2, q_3, ..., q_n\}$$
$$P = \{p_1, p_2, ..., p_m\}$$

q_i und p_i seien Quarks. *Q* und *P* sollen disjunkt sein, also muß gelten:

$$\forall i, j \rightarrow q_i \in Q \wedge p_j \in P \rightarrow q_i \neq p_j$$

Q soll als **minimal** bezeichnet werden, wenn sich zwei beliebige Teilbäume in *Q* nicht zu einem zusammenfassen lassen (*Next(q)* wird durch *N(q)* und *Father(q)* durch *F(q)* abgekürzt):

$$\forall i, j : q_i \in Q \wedge p_j \in Q \rightarrow$$
$$(\forall b \in B \rightarrow (N(F(q_i)) \neq F(p_j) \wedge N(F(p_j)) \neq N(q_i)))$$

Ausgehend von zwei minimalen Mengen *Q* und *P* von Teilbäumen (=Quarks) soll eine minimale Vereinigungsmenge berechnet werden. Ein möglicher Algorithmus arbeitet nach folgendem Schema:

1. Der Menge *Q* wird ein Quark *q* entnommen.
2. Für alle Quarks *p* der Menge *P* wird getestet, ob sich *q* mit *p* vereinigen läßt.
   - ☐ Nein-> *q* wird in die Menge *P* eingefügt.
   - ☐ Ja-> *p* wird der Menge *P* entnommen und mit dem sich ergebenen Vereinigungsbaum (*q* = *p* vereinigt *q*) rekursiv Schritt 2 durchgeführt.
3. Solange die Menge *Q* nicht leer ist, fahre mit Schritt 1 fort.
4. Das Ergebnis steht in der Menge *P*

Mit folgenden Erweiterungen kann obiger Algorithmus unter Einsatz minimaler Rechenzeit durchgeführt werden. Hierfür soll jedem Quark *q* zwei weitere Hilfsattribute zugeordnet werden:

- ein Verweis V_q auf einen weiteren Quark.
- ein Zeitstempel T_q , der die Gültigkeit von *V* angibt. Dieser Zeitstempel wird mit dem Wert 0 initialisiert.

Es gibt eine globale algorithmische Zeit *Gt*, die von dem Algorithmus beliebig weitergestellt werden kann. Diese Zeit wird am Beginn mit dem Wert 1 initialisiert. Der Verweis *V* soll nur dann gültig sein, wenn der Wert seines Zeitstempels identisch ist mit dem Wert der globalen Uhr. Auf diese Weise können durch einfaches Weiterstellen der Uhr alle Verweise ungültig gemacht werden.

Vor Beginn des obigen Algorithmus wird für alle Elemente *p* aus *P* folgender Rechenschritt durchgeführt:

- Setze $a = N(F(p))$ und $b = N(N(F(p)))$
- Für $x=a$ und für $x=b$ setze $V_x = p$ und $T_x=Gt$

Mit diesen zusätzlichen Attributen kann der Test in Schritt 2 des obigen Algorithmus in konstanter Zeit durchgeführt werden. Dazu muß nur für den Quark *q* getestet werden, ob

- $T_{N(F(q))}=Gt$ oder $T(N(N(F(q)))=Gt$ ist.

Falls ja, läßt sich der zu vereinigende Unterbaum aus dem Attribut *V* des Quarks ablesen, dessen Zeitstempel gültig war.

Eine Ordnung auf Quarks

Es soll eine partielle paarweise Ordnung '<' auf Quarks definiert werden:

Seien q, u zwei Quarks in einem Baum, dann soll

q < u genau dann, wenn

Father(q) = Next(u) oder Father(q) = Next(Next(u))

Aus dieser Definition ergibt sich, daß für alle Quarks *i* , bei denen die *Next*–Operation definiert ist (d.h. alle nicht-Blattquarks), genau zwei Quarks *x* und *y* existieren, für die gilt: $x<i$ und $y<i$.

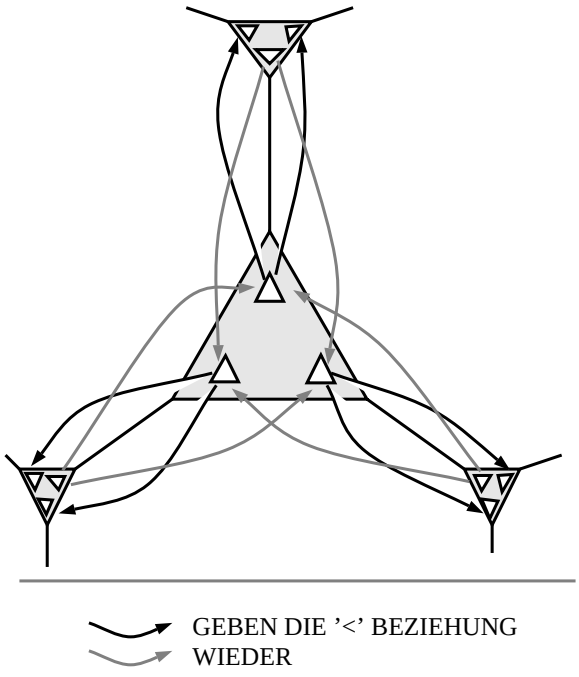


Abb. 78 Veranschaulichung einer Ordnung von Quarks

Mit Hilfe dieser paarweisen Ordnung läßt sich ihre transitive Hülle bilden. Die so enstandene partielle Ordnung '<<' sei wie folgt definiert:

$a << b$	falls $a < b$
$a << b$	falls es ein c gibt, mit $a << c$ und $c << b$

Mit Hilfe der << Ordnung kann man sogenannte **innere Quarks** definieren:

Ein Quark q heißt innerer Quark, genau dann wenn es einen Quark b gibt mit $b<<q$.

Identifizierung identischer Unterbäume

Um nun Unterbäume in unterschiedlichen Bäumen vergleichen zu können, soll ein '=' Operator auf den Quarks der Baumblätter definiert werden:

Seien A und B zwei Bäume und L und M jeweils zwei Blätter der entsprechenden Bäume.
Das Blatt L besitzt den Quark q_L und M den Quark q_M
dann soll gelten:
$q_L=q_M$ genau dann wenn L und M dieselbe Spezies identifizieren[1].

Mit Hilfe von partiell geordneten Quarks läßt sich definieren, wann zwei durch Quarks identifizierte Unterbäume U und V in jeweils zwei unterschiedlichen Bäumen A und B identisch sind. Für die transitive Hülle '==' des '='-Operators soll gelten:

$U==V$	falls $U=V$
$U==V$	falls es Quarks q_{LU}, q_{MU}, q_{LV} und q_{MV} gibt, so daß $q_{LU}<U$ und $q_{MU}<U$ und $q_{LV}<V$ und $q_{MV}<V$ und $q_{LU}==q_{MU}$ und $q_{LV}==q_{MV}$

Mit anderen Worten: zwei Unterbäume sind identisch, wenn ihre Söhne jeweils paarweise identisch sind.

Eine effiziente Crossover-Rechenvorschrift

In der Praxis hat sich gezeigt, daß aus zwei unterschiedlichen Startsituationen entstandene Bäume großenteils identisch sind, vor allem stimmen ganze Unterbäume überein. Dies führt dazu, daß bei einem zufällig durchgeführten Crossover die kombinierte Lösung mit hoher Wahrscheinlichkeit identisch mit einem der Ursprungsbäume ist. So soll das traditionellen Crossover-Vorgehen, eine zufällige Verschmelzung zweier Teillösungen durchzuführen, bei dieser speziellen Anwendung durch die Berechnung aller möglichen Verschmelzungen zweier Bäume ersetzt werden. Dieses Vorgehen ist insoweit sinnvoll, als für die Bewertung mehrerer Verschmelzungen nur unwesentlich mehr Rechenarbeit investiert werden muß als für die Bewertung einer einzigen.

[1] Es läßt sich keine mathematisch exakte Definition dafür angeben, wann zwei Blätter in einem Baum identisch sind, dies hängt von der Betrachtungsweise des Anwenders ab.

Der folgende Algorithmus ist in der Lage, effizient alle derartigen Kombination aus zwei Bäumen zu berechnen, wobei er automatisch alle Kombinationen markiert, bei denen durch die Anwendung eines Crossover-Algorithmus mit Sicherheit keine weitere Verbesserung erreicht werden kann.

Gegeben sind zwei Bäume *A* und *B*. Gesucht ist für jeden Quark *a* aus *A* die minimale Menge *F(a)* der Quarks *b* aus *B*, die genau diejenigen Organismen identifizieren, die in dem Unterbaum *a* enthalten sind. Alle Teilbäume F(a) werden aus *B* entfernt und der Teilbaum *a* an alle Kanten testweise eingefügt, an denen ursprünglich Elemente aus *F(a)* eingehängt waren. *F(a)* wird im weiteren mit *F* abgekürzt.

1. Dazu wird eine << geordnete Liste aus allen Quarks *a* aus *A* aufgebaut:

$$M_A = \{a_1, a_2, ..., a_n\}, \quad \forall 0 < i < j < n \rightarrow \neg(a_i >> a_j)$$

2. Mit Hilfe einer Crossover-Funktion soll eine Menge M_B aus Mengen *F* von Quarks bestimmt werden. Diese Menge wird wie folgt initialisiert:

$$M_B = \{F_1, F_2, ..., F_n\}, mit \;\; F_i = F(a_i)$$

   Eine Menge F_i soll im folgenden dazu dienen, alle minimalen Unterbäume im Baum zu speichern
3. Identifizieren die Blattquarks a_i und F_i dieselbe Spezies, definieren wir uns eine '='-Operationen zwischen Quarks und Mengen von Quarks: F_i=a_i und F_i==a_i
4. Für alle *i* von 1 bis *n* werden folgende Schritte ausgeführt:
   a. Falls a_i ein Blattquark ist, setze F_i={b_i}, *mit* b_i==a_i. Markiere die Menge F_i als **identisch** zu einem Unterbaum aus *A*.
   b. Sonst gibt es genau zwei Quarks *x* und *y* mit *x*<a_i und *y*<a_i. Sei *X=F(x)* und *Y=F(y)*. Bilde die minimale Vereinigungsmenge *V* aus *X* und *Y* und setze F_i=*V*:
      - Falls *X* und *Y* als identisch markiert sind und sich zu einem Baum vereinigen lassen, markiere die Vereinigung als **identisch**.
      - Falls *X* als identisch markiert ist und *Y* nur aus einem Element besteht, markiere die Vereinigung als **redundant**.

5. Für alle *i* von *1* bis *n* führe aus:
   a. Falls F_i als identisch markiert ist, führe keine Aktion aus.
   b. Falls F_i als redundant markiert ist, führe ebenfalls keine Aktion aus.
   c. Falls F_i als redundant und als identisch markiert ist, hat das Programm einen Fehler.
   d. sonst entferne alle Unterbäume von F_i aus *B* und füge an allen ehemaligen Positionen von F_i den Baum a_i testweise ein und bewerte sie. Die am besten bewertete Position wird in $\mathbf{T_{global}}$ global gespeichert.

In der globalen Variablen T_{global} läßt sich die beste Kombination der beiden Bäume *A* und *B* auslesen.

Diskussion

Um den vorgestellten Algorithmus zu bewerten, sollen Komplexitätsbetrachtungen durchgeführt werden. Bei einem mit Parsimony oder Maximum Likelihood bewerteten Baum müssen bei einer Topologieänderung alle inneren Knoten, die zwischen den Änderungen und der Wurzel liegen, neu berechnet werden. Verschiebt man die Wurzel in das Gebiet mit den häufigsten Änderungen, kann die zu investierende Rechenzeit optimiert werden. Wird also der Baum mehrfach topologisch geändert, wird die zu investierende Rechenleistung zur Bewertung um so niedriger ausfallen, je kürzer der Pfad zwischen den durchgeführten Änderungen ist.

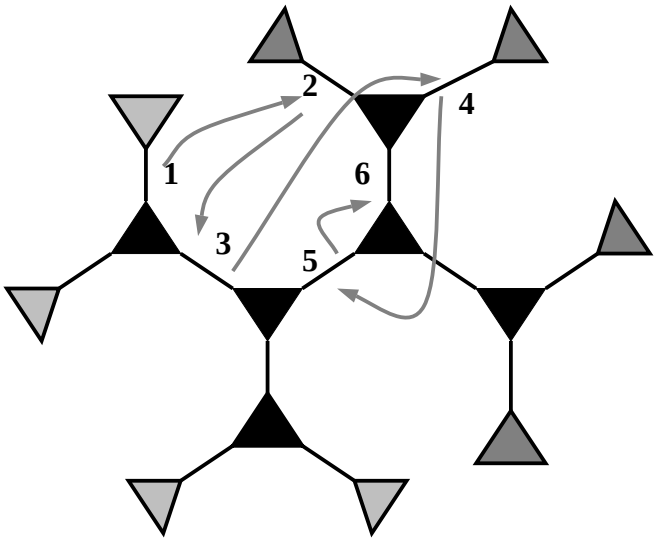


Abb. 79 Beispiel für eine schlechte Wahl der Reihenfolge von NNI-Operationen

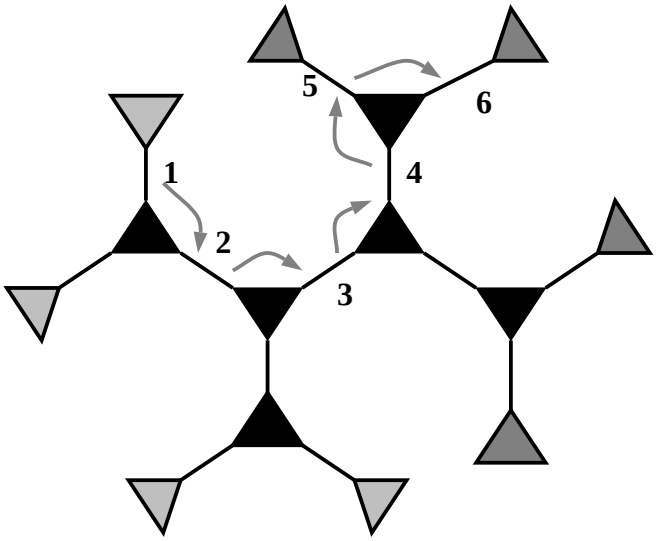


Abb. 80 Beispiel für eine gute Wahl der Reihenfolge von NNI-Operationen

Da als Lösung des vorgestellten Algorithmus der phylogenetische Baum gesucht wird, werden alle guten Zwischenlösungen mehr oder weniger die Grundstruktur des gesuchten

Baumes realisieren. Für diesen konkreten Anwendungsfall heißt das, daß alle Positionen, an denen ein Unterbaum des Baumes *A* in einen Baum *B* eingebaut werden muß, im Baum *B* relativ dicht beieinander stehen und sich somit bei der Abarbeitung zwangsläufig eine effizient zu bearbeitende Positionsfolge ergibt.

Durch geschickte Wahl der Ordnung der Quarks in der Bearbeitungsliste wird erreicht, daß auch beim Übergang zu einem neuen Teilbaum der Bereich in Baum *B*, der topologisch verändert wird, nicht allzu weit von dem am letzten benutzten Bereich liegt. Diese geschickte Ordnung der Quarks erreicht man zwangsläufig durch einen rekursiven Aufbau dieser Liste.

Insgesamt konnte durch die Entwicklung dieses Algorithmus erreicht werden, daß alle Kombinationen zweier Bäume in der theoretisch optimalen Zeit berechnet und bewertet werden können. Sind die beiden Bäume identisch, so beträgt der Rechenaufwand *O(Anzahl der Knoten)*, sind die beide Bäume maximal unterschiedlich, so benötigt man *O(Anzahl der Knoten^2 * Länge der Sequenzen)*. Daher wird der Algorithmus mit Annäherung ans globale Optimum immer schneller.

VIII.6.6 Zusammenfassung genetische Algorithmen

Mit steigender Sequenzanzahl stoßen die meisten Verfahren zur Rekonstruktion des Stammbaumes von Organismen an ihre Grenzen. Im wesentlichen bleiben zwei Verfahren übrig, die eine Lösung in vernünftiger Zeit berechnen können: Der schnelle aber einfache Parsimony-Algorithmus, der nicht frei von systematischen Fehlern ist, und das etwa hundertmal langsamere Maximum-Likelihood-Verfahren. Beide Methoden liefern aber nur die Bewertung einer gegebenen Topologie. Daher muß eine möglichst große Anzahl verschiedener Topologien mit Hilfe effizienter Algorithmen systematisch getestet werden, um eine zufriedenstellende Lösung zu finden.

Mit weiter steigender Sequenzanzahl wird es unmöglich, eine einzige Lösung systematisch so zu optimieren, daß das Ergebnis brauchbar ist. Erst durch das wiederholte Durchführen der Algorithmen mit verschiedenen Startsituationen steigt die Wahrscheinlichkeit, daß komplette Unterbäume identisch mit einem Unterbaum der tatsächlichen Lösung sind. Eine zufällige Korrektheit aller Unterbäume bleibt aber weiterhin sehr unwahrscheinlich. Als einziger Ausweg bietet sich an, einen Zwischenlösungs-Baum in Unterbäume zu zerlegen, diese einzeln zu optimieren und dann wieder zusammenzuführen.

Dieses Vorgehen wurde in der Vergangenheit von Hand durchgeführt, erst die Entwicklung eines effizienten Algorithmus, der alle Kombinationen zweier Bäume berechnet, erlaubt diesen Vorgang zu automatisieren.

Die so erhaltene, neue Heuristik erlaubt, den Suchraum der Verfahren zur Optimierung einer Lösung so zu verringern, daß diese um ein Vielfaches schneller werden. Die dadurch in Kauf genommenen lokalen Optima werden dann mit Hilfe des Crossover-Algorithmus verlassen.

Insgesamt erhält man einen in der Informatik gut bekannten genetischen Algorithmus. Allerdings lassen sich die Forschungsergebnisse über genetische Algorithmen aus anderen Kontexten nur bedingt in diesem speziellen biologischen Anwendungsfall übertragen.

VIII.7 Diskussion und Ausblick

Es gibt viele Einsatzbereiche der Phylogenie. Die größte Herausforderung jedoch ist die Bestimmung eines umfassenden Stammbaumes aller Lebewesen, berechnet auf Grundlage aller Sequenzen des am häufigsten sequenzierten Gens, der ribosomalen RNS. Die exponenentiell steigende Datenmenge stellt sowohl die Forscher wie auch die Betreiber von Rechenanlagen vor große Herausforderungen. So müssen nicht nur die Algorithmen immer weiter optimiert und verbessert werden, auch die Anforderungen an die Computer, mit denen diese Analysen durchgeführt werden können, steigen exponentiell.

Die Komplexität des Grundproblems und die beschränkten Rechnerkapazitäten machen den Einsatz von Heuristiken notwendig, die mit an Sicherheit grenzender Wahrscheinlichkeit nicht den korrekten Baum berechnen. So sagte 1997 G. Olsen im Rahmen eines Phylogenie-Workshops vor einer Expertenkommission, daß nur eines wirklich sicher sei, nämlich daß der von ihm berechnete Baum falsch sei.

Aber selbst eine nicht optimale Lösung stellt oft einen entscheidenden Fortschritt gegenüber der vorherigen Situation dar. Entscheidend ist also nicht, einen Stammbaum zu finden, der eine optimale Bewertung erhält, sondern die topologischen Unsicherheiten in einer gefundenen Lösung zu akzeptieren. Selbst wenn es möglich wäre, die optimale Topologie zu finden, ist damit nicht sicher gestellt, daß diese Topologie auch stabil ist, d.h. daß sie sich durch leichte, zufällige Änderung der Eingabedaten (wie Sequenzierfehler, neue Sequenzen oder leicht geändertes Alignment) nicht ändert. Oft liegt sogar der Fall vor, daß sich durch das Hinzufügen neuer Sequenzen zu einem gegebenen Aligment der daraus berechnete Baum entscheident ändert[1].

Es sollte daher nicht die korrekte Stammbaum-Topologie gefordert werden, sondern vielmehr eine gute Näherungslösung, bei der mit Hilfe graphischer Symbole die Stabilität der einzelnen Kanten dargestellt wird. Diese Stabilität einer Topologie kann im Normalfall an den Längen der Äste abgelesen werden: Je länger eine innere Kante in einem Baum, desto topologisch stabiler ist sie. Während Maximum-Likelihood-Algorithmen die Astlängen sofort als 'Nebenprodukt' mitberechnen, ist dies beim Parsimony-Algorithmus nicht möglich. Mit Hilfe eines einfachen Verfahrens kann jedoch die Stabilität einer Topologie mit Hilfe des Parsimony-Algorithmus berechnet werden. Dieser Wert läßt sich dann in einer optisch sichtbaren Astlänge darstellen.

Astlängen können jedoch nur eine relative Auskunft über die Stabilität einzelner Kanten geben, gefordert werden eigentlich konkrete Wahrscheinlichkeiten. Bei kleinen Bäumen lassen sich diese mit Hilfe des Bootstrap-Verfahrens leicht berechnen [Felsenstein 85a]. Die Ideen dieser Methode lassen sich aber auch auf einen mit Parsimony bewerteten Baum übertragen. So können zwar nicht die exakten Bootstrapwerte berechnet werden, sie können jedoch gut mit einer oberen Schranke abgeschätzt werden.

Die Neuberechnung eines Stammbaumes erfordert ernorme Rechenkapazitäten. Es stellt sich die Frage, ob es sinnvoller ist, diese Neuberechnung weltweit zentral durchzuführen oder de-

[1] Aufgrund dieser Instabilität verzichtete die RDP darauf, ihren veröffentlichten Baum nach dem Einfügen neuer Sequenzen zu optimieren, da sich die Topologie im Bereich des Zentrums dabei scheinbar zufällig veränderte. Diese häufigen Veränderungen hätten die Anwender des Baumes nur unnötig verwirrt.

zentral auf den einzelnen Rechnern der Anwender. Ist nur die phylogenetische Einordnung einer noch unbekannten Sequenz gefordert, ist keine Neuberechnung des Stammbaumes notwendig, diese Operation kann ohne Einschränkung dezentral in den einzelnen Labors durchgeführt werden. Diese hätten davon den Vorteil, daß sie noch unveröffentlichte Sequenzen nicht über das Internet versenden müßten und daß sie diese Sequenz in ihre lokale Datenbank integrieren könnten. Jedoch selbst das relative schnelle, innerhalb von 10 Sekunden durchführbare Einrechnen einer Sequenz in einen gegebenen Baum benötigt bei einer klassischen Implementierung des Parsimony-Algorithmus nicht unerhebliche Hauptspeicher-Ressourcen. Es gibt jedoch die Möglichkeit, durch geschickte Implementierung eines Verfahrens, das dynamisch die inneren, rekonstruierten Sequenzen bei der Parsimony-Bewertung zufällig löscht, die benötigten Hauptspeicherressource um den Faktor 10 zu reduzieren bei gleichzeitiger geringer Erhöhung der einzusetzenden Rechenleistung um den Faktor 1.5.

Langfristig kann und wird die Berechnung der Grundtopologie des Baumes nur an wenigen weltweit zentralen Stellen erfolgen. Der dort veröffentlichte Baum kann von den Anwendern unverändert übernommen werden und dient dann als Grundgerüst für das Einfügen ihrer privaten Sequenzen.

Die RDP und der Lehrstuhl für Mikrobiologie TU-München stellt diesen Service bereits im Internet bereit; Biologen, die keine eigene komplette Sequenzdatenbank aufbauen wollen, können diese Dienste nutzen.

IX Berechnung von Gen-Sonden

IX.1 Einführung und Stand der Technik

Im Laufe der Zeit der biologische Analyse setzte sich die ribosomale RNS als phylogenetischer Marker durch [Stackebrandt 92]. Infolgedessen wuchs die Datenmenge exponentiell an[1].

Es stellte sich zudem auch noch heraus, daß die Wahl, rRNS-Sequenzen zu verwenden, besonders glücklich war, da die hohe Kopienzahl, mit der die rRNS in den Zellen vorkommt, eine wichtige Voraussetzung für eine neue Technik darstellte: Durch Ausnutzung der Eigenschaft von RNS, sich unter bestimmten Bedingungen zu stabilen doppelsträngigen Hybriden zusammenzulagern [Schleifer et al., 93], konnte eine neue 'Sonden'-Technik entwickelt und eingesetzt werden. Dabei wird für eine gegebene Sequenzmenge ein kurzes Sequenzstück gesucht (Oligogonukleotid), das für dieses Sequenzgruppe charakteristisch ist, d.h. in allen diesen Sequenzen vorkommt und in möglichst keiner anderen. Zu diesem charakterisierenden Sequenzstück wird eine komplementäre Sonde synthetisiert, die üblicherweise mit einem Radioisotop, einem Fluoreszenzfarbstoff oder einem Enzym markiert wird[2]. Unter gewissen biochemischen Bedingungen lagert sich die so entstandene Sonde genau an die ursprünglich ausgewählte Sequenzmenge an und läßt die zugehörigen Bakterien im Falle einer Fluoreszenzmarkierung im Epifluoreszenzmikroskop aufleuchten.

---

[1] 50 Sequenzen 87 [Woese 87]
450 im Jahre 91 [Olsen et al. 91]
1700 im Jahre 1993 [Gutell et al., 93]
4300 im Jahre [Van de Peer et al., 96]
11000 Mitte 97 am Lehrstuhl für Mikrobiologie TU-München

[2] Giovannoni et al. (88) gelang mit radioaktiv markierten Sonden erstmals der Nachweis einzelner Bakterienzellen. Diese Ganzzellhybridisierung wurde aber erst durch den Einsatz von fluoreszenzmarkierten Sonden entscheidend vereinfacht [DeLong et al., 89] [Amann et al., 90a]

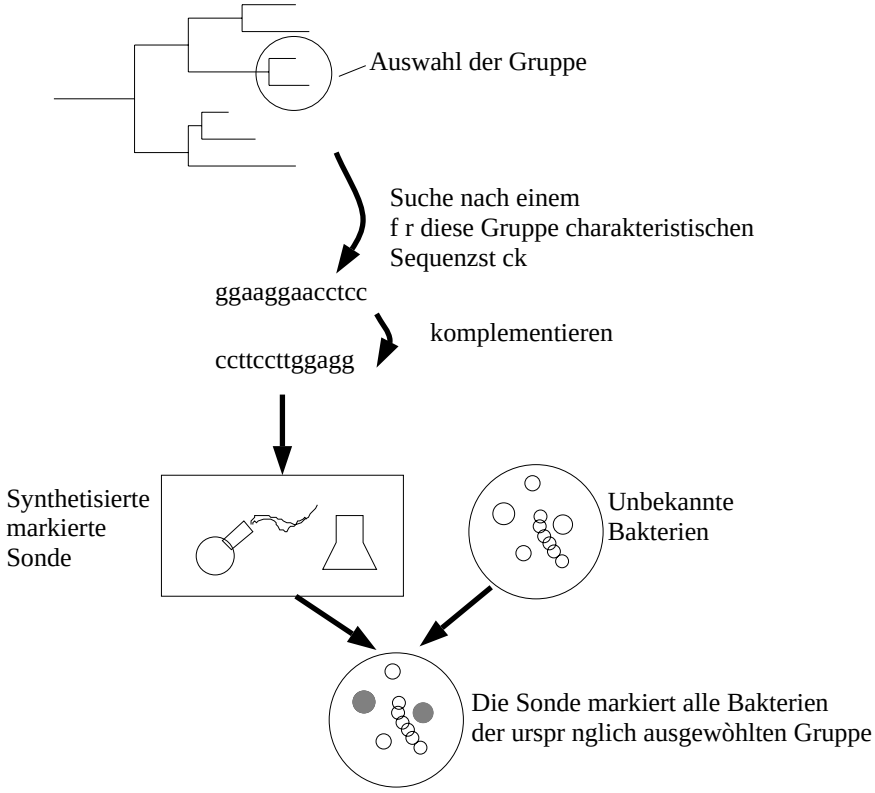


Abb. 81 Übersichtsschema der Sondentechnik

Mit Hilfe dieser Technik gelang es Norman Pace und seinen Mitarbeitern Anfang der 80er Jahre erstmals, Populationsanalysen von Mikroorganismen durchzuführen, ohne diese vorher kultivieren zu müssen [Stahl et al., 84+85] [Pace et al., 86] [Olsen et al., 86]. Mit Hilfe fluoreszenzmarkierter Oligonukleotide können nun Populationsstrukturen komplexer Lebensgemeinschaften **in situ** untersucht werden [Amann et al., 95]. Nachdem die Einsatzfähigkeit dieser neuen Technik unter Beweis gestellt worden war [Spring et al., 92], wurde für die verschiedensten Bakteriengruppen auf allen taxonomischen Ebenen charakteristische Sequenzabschnitte gesucht und gefunden [Amann et al., 95].

IX.2 Aufgaben

In der Vergangenheit konnten charakteristische Sequenzstelle für kleine Gruppen noch mühsam per Hand gesucht werden, allerdings war dies schon bald mit steigender Sequenzdatenmenge nicht mehr möglich. Daraus ergab sich für diese Arbeit als **Teilziel**, Algorithmen zu entwickelen, die es nicht nur erlauben, die Arbeitszeit entscheidend zu verkürzen, sondern auch die Möglichkeit bieten, Sonden für größere Gruppen zu entwickeln.

Dabei wurden hauptsächlich **drei Basis-Algorithmen** im Rahmen dieser Arbeit entwickelt:

- Zu Anfang wurden auf der Basis weniger ausgewählter Sequenzen mögliche Sonden von Hand entwickelt. Bevor diese Sonden mit großen Aufwand in der Praxis getestet werden, kann man sie theoretisch auf dem Computer simulieren. Dazu muß im wesentlichen diese Sonde mit allen Sequenzen verglichen werden. Es gibt nur wenige Programme (z.B. blast, fasta), die diesen Vergleich durchführen können und auch dies auf eine für diesen Anwendungsfall nicht zufriedenstellende Weise. So wurde für ARB eine möglichst praktische, schnelle und einfach zu bedienende Funktion (**Probe-Match**) entwickelt, die die Bindungseigenschaften einer gegebenen Sonde theoretisch abschätzt.
- Aufbauend auf einer solchen schnellen Sondenbindungs-Bewertung konnte dann ein Verfahren (**Probe-Design**) entwickelt werden, das automatisch eine solche Sonde erzeugt. Dazu wird eine große Anzahl (>10000) möglicher Sondenkandidaten generiert und mit dem Algorithmus bewertet. Die am besten bewerteten Kandidaten werden dem Benutzer als mögliches Ergebnis präsentiert.
- Mit steigender Sequenzmenge wird es immer schwieriger, eine einzige gute Sonde für eine ganze Gruppe von Sequenzen zu finden. Daher geht man in vielen Fällen dazu über, eine Kombination von Sonden einzusetzen [Amann et. al. 90b]. Um eine solche optimale Kombination zu finden, wurde ein Bewertungsschema und –algorithmus (**Multi-Probe**) für Sondenkombinationen entwickelt und darauf aufbauend ein Verfahren, das gute Kombinationen aus einer Menge von Einzelsonden erzeugt.

IX.3 Evaluierung von Sonden (Probe-Match)

IX.3.1 Bewertung der Bindung einer Sonde

Ziel der praktischen Anwendung einer Sonde ist es, eine Bakteriengruppe möglichst eindeutig von allen übrigen Bakterien abzugrenzen. Dies erreicht man dadurch, daß die gesuchte Sonde sich möglichst gut an die Sequenzen dieser Gruppe anlagert und an allen anderen Sequenzen möglichst schlecht. Diese Anlagerungs- bzw. Bindungsstärke drückt sich in der maximalen Temperatur aus, die diese Bindung chemisch übersteht. Vereinfacht läßt sich also die Qualität einer Sonde für eine Gruppe ermitteln, indem die folgende Temperaturdifferenz **dT** gemessen oder errechnet wird:

$$dT = \max\left(T_{Sonde\,bindet\,Gruppe}\right) - \max\left(T_{Sonde\,bindet\,fremden\,Organismus}\right)$$

Im Idealfall bindet eine Sonde an die komplementäre Zielregion perfekt:

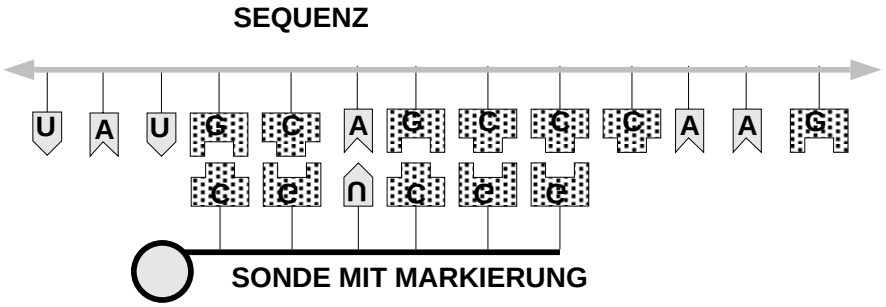


Abb. 82 Ideale Sondenhybridisierung

Fehler in der idealen Bindung (im weiteren als **Mismatches** bezeichnet) können unter Umständen dazu führen, daß die Bindungskraft der restlichen Basen nicht ausreicht, um die Sonde dauerhaft an die Zielregion zu binden:

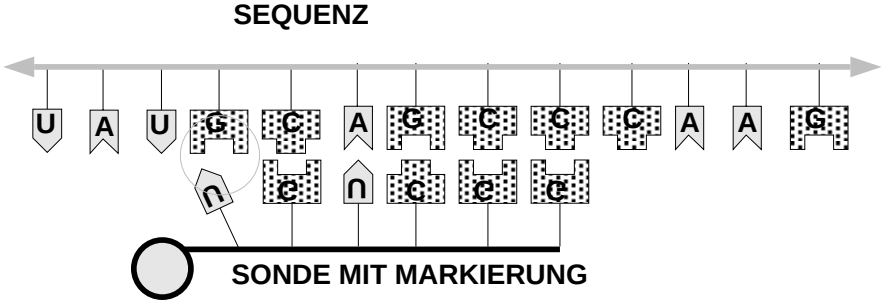


Abb. 83 Sondenbindung mit einem Mismatch am Rand

Bei allen weiteren Untersuchungen soll ausschließlich diese Temperaturdifferenz **dT** untersucht werden.

Es gibt eine ganze Reihe von Formeln, die die absolute Temperatur einer solchen Bindung berechnen[1]. Allerdings sind diese Formeln für den Einsatz bei langen Sonden-Sequenz-Bindungsbereichen optimiert worden, da bei den heute typischerweise eingesetzten Sonden mit der Länge 15 bis 20 Basen diese Näherungen zu ungenau sind. Der größte Schwachpunkt liegt im wesentlichen darin, daß meist nur die relative Anzahl der Mismatches berücksichtigt wird, aber weder deren Position noch Art. Es macht jedoch wohl einen nicht nur marginalen Unterschied, ob eine GC-Paarung durch einen GU-Mismatch oder einen GA-Mismatch ersetzt wird.

Je stärkere und je mehr Mismatches in einer Bindung vorkommen, desto unwahrscheinlicher ist eine physikalisch-chemische Bindung zwischen Sonde und Zielregion. Im speziellen Anwendungsfall ist man jedoch nicht an der Stärke einer Sondenbindung interessiert, sondern an der Abnahme der Bindungsenergie durch einzelne Mismatches. Da es hier nicht auf eine korrekte Formel ankommt, wurde ein einfacher benutzerparametrisierbarer Ansatz gewählt. Dazu können die relativen Bindungsstärken zwischen den verschiedenen Basen in Form einer Matrix eingegeben werden:

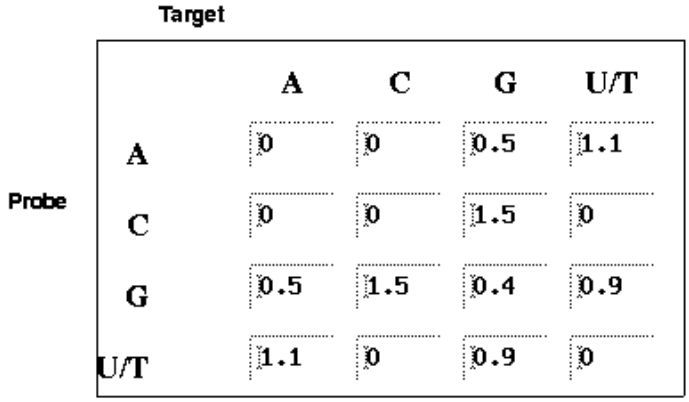


Abb. 84 Benutzermaske für relative Bindungsstärke

[1]

$$DNA/DNA: T_m = 81.5 + 16.6 * \ln\left(\frac{Na^+}{1.0 + 0.7[Na]}\right) + 0.41 * (\%(G+C)) - \frac{500}{n} - P$$

$$RNA/RNA: T_m = 78 + 16.6 * \ln\left(\frac{Na^+}{1.0 + 0.7[Na]}\right) + 0.7 * (\%(G+C)) - \frac{500}{n} - P$$

$$DNA/RNA: T_m = 67 + 16.6 * \ln\left(\frac{Na^+}{1.0 + 0.7[Na]}\right) + 0.8 * (\%(G+C)) - \frac{500}{n} - P$$

mit

- n Länge der Sequenz
- P % der Mismatches
- %(G+C) das Verhältnis (G+C) zu (A+C+G+T)
- Na^+ die Konzentration von Natrium Ionen

Hinzu kommt, daß die Stärke der Bindung am Rand der Sonde schwächer als in der Mitte ausfällt. So werden mit folgender empirischen Formel die Positionen der Mismatches innerhalb der Sonde gewichtet:

$$g[i] = MAX - \left(\frac{2i-n}{n}\right)^2 * (MAX - MIN)$$
$$n : Laenge\ der\ Sonde$$
$$MAX, MIN : Benutzerdefinierte\ Werte$$

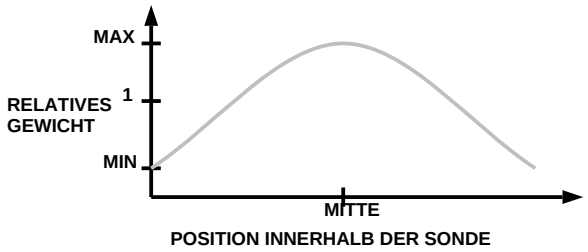


Abb. 85 Gewichtung der Sondenpositionen

Um nun eine Sonde bewerten zu können, muß diese für alle in Frage kommenden Zielregionen bewertet werden. Die Zielregion mit der höchsten Temperatur innerhalb einer Sequenz, also diejenige mit dem besten Bindungsverhalten, bestimmt dann das Gesamtbindungsverhalten dieser Sonde mit der Gesamtsequenz. Um dieses Bindungsverhalten einer Sonde für alle Sequenzen zu berechnen, muß mit Hilfe eines geeigneten Algorithmus die Menge aller Sequenzen effizient durchsucht werden (**Stringsuche in Sequenzen)**. Dieser Algorithmus läßt sich allerdings nicht nur für die Bewertung von Sonden einsetzen, sondern löst eine ganze Reihe weiterer Aufgaben wie zum Beispiel die Bestimmung der nächsten Verwandten eines Organismus. Diese Suche kann mit Hilfe eines Index effizient implementiert werden.

IX.3.2 Stringsuche in Sequenzen

Grundalgorithmen

Im weiteren soll nun ein praktikabler Algorithmus entworfen werden, der es erlaubt, mit möglichst wenig Rechenaufwand alle Positionen in einem gegebenen Alignment zu finden, die zu gegebenen Suchstrings[1] einen **Hammingabstand** aufweisen[2], der eine bestimmte Grenze nicht überschreitet. Diese Grenze soll im weiteren als **maximale Suchtiefe** bezeichnet werden.

[1] Dieser Suchstring ist im Anwendungsfall das komplementäre Gegenstück zur Sonde.

[2] Den Hammingabstand zwischen zwei Zeichenketten erhält man, in dem man positionsweise die Unterschiede zwischen diesen aufsummiert. Die Zeichenketten *hallo* und *Tallo* haben den Hammingabstand 1, da das *T* != *h*.

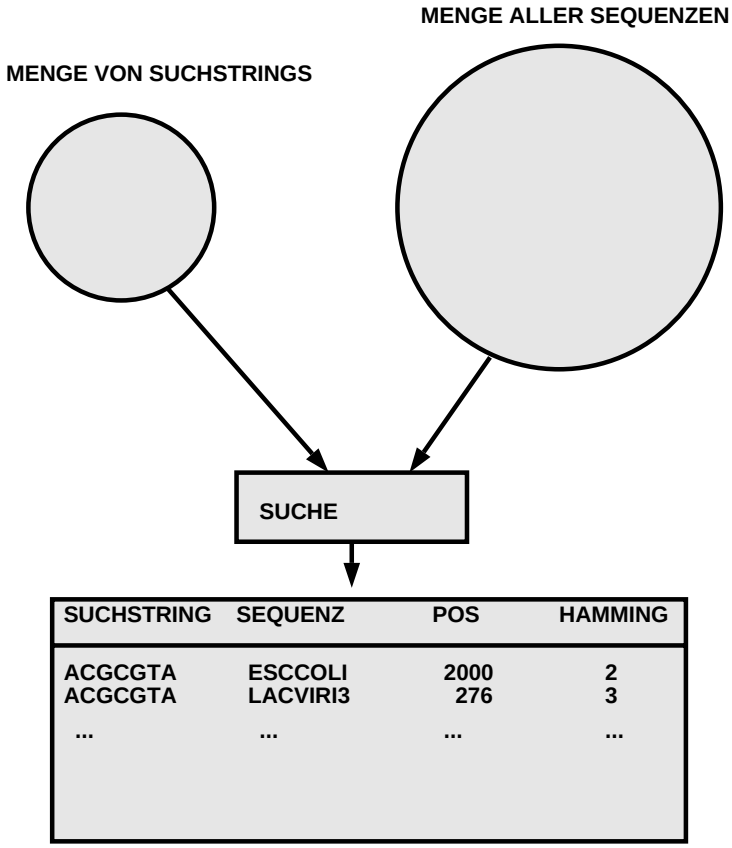


Abb. 86 Übersichtsschema Grundalgorithmus

Da ein trivialer Algorithmus nicht in Frage kommt, der alle Suchstrings mit allen Sequenzen in Alignment positionsweise vergleicht (Komplexität O(Anzahl_Suchstrings * Anzahl_der_Basen_im_Alignment)), gibt es prinzipiell zwei Vorgehensweisen:

I. Man generiert aus allen Suchstrings eine Art Schnellsuchindex. Mit Hilfe dieser optimierten Suchstruktur müssen nun alle Sequenzen bearbeitet werden, die sich ergebende Komplexität ist O(ld(Anzahl_Suchstrings) * Anzahl_der_Basen_im_Alignment). Der große Vorteil dieser Methode ist, daß auch für große Sequenzmengen kein Indexbaum generiert werden muß und somit wertvoller Festplattenspeicher eingespart werden kann. Besitzt der verwendete Rechner mehr Hauptspeicher als die Menge aller Sequenzen, so ist dieses Vorgehen bei wiederholter Anwendung relativ schnell. Aber selbst im optimalen Fall müssen immer noch alle Sequenzen des Alignments linear verarbeitet werden.
II. Aus allen Sequenzen des Alignments wird ein Schnellsuchindex generiert. Mit Hilfe dieses Index kann ein beliebiger Suchstring in kürzester Zeit gefunden werden: O(ld(Anzahl_der_Basen_im_Alignment)).

Für ARB wurde der zweite Weg gewählt. Dieser erlaubt, bei einer Bewertung von nur einer einzigen Sonde in kürzester Zeit Resultate zu erhalten.

In der Informatik gibt es eine ganze Reihe von Standardalgorithmen zur Berechnung und Anwendung eines derartigen Suchindex. Eine kurze Übersicht über die wichtigsten dieser Algorithmen soll deren Verwendbarkeit im konkreten Anwendungsfall aufzeigen. Für

einen sinnvollen Vergleich solcher Algorithmen sind konkrete, der Praxis entnommene Kontextinformationen erforderlich:

Beschreibung	Wert (Mitte 1997)
Anzahl der Sequenzen	10000
Länge des Alignments	10000 Zeichen
Mittlere Basenanzahl der Sequenzen	1000 Zeichen
Hauptspeicher der verwendeten Computer in Megabyte	64 - 256
Rechenleistung der Computer	100-500 Mips
Anzahl der Suchstrings	1-100
Anzahl der unterschiedlichen Symbole in dem Alignment	7 (ACGU Gap, N, '.')
Maximale Hammingdistanz bei der Suche	0-5
Festplattenpreise	10 Pfennig/Megabyte -> unbedeutend

Minimal benötigter Speicher zur Speicherung von	
einem Bakteriennamen	2 Bytes
1 Base	1 Byte
einer Position im Alignment	2 Bytes
einem C-Zeiger	4-8 Bytes
dem Speicherverschnitt des Betriebssystems pro Speichereinheit	8 Bytes

Abb. 87 Übersicht über Kontextinformationen zur Bewertung von Algorithmen:

Hashing

Das Grundprinzip des Hashing ist eine Abbildung, die jeder Zeichenkette ein natürliche Zahl in einem gegebenen Zahlenbereich zuordnet. Dieser **Hashindex** zeigt in ein Array, dessen Elemente Listen von Verweisen auf alle Zeichenketten sind, bei denen die *Hash-funktion* denselben Wert berechnet. Leider scheiden im vorliegenden Anwendungsfall *Has-hing*–Algorithmen aus, da diese nur exakte Suche in einem Datensatz erlauben. Man kann also nur unter hohem rechnerischem Aufwand nach allen Stellen suchen, die zur gegebenen Suchzeichenkette einen Hammingabstand kleiner der maximalen Suchtiefe aufweisen.

Modifiziertes Hashing, Indexlisten

Während man beim traditionellen Hashing für alle Teilzeichenketten eines Datensatzes die Hashfunktion *h* auswertet und dessen Ergebnis speichert, geht man bei den Indexlisten dazu über, nur für eine feste, kurze Länge von Teilzeichenketten diesen Vorgang durchzuführen.

Bei 4 verschiedenen Basen gibt es bei einer Zeichenkette der Länge 4 $4^4 = 256$ verschiedene Werte. Die Funktion *h* kann nun so konstruiert werden, daß sie injektiv wird:

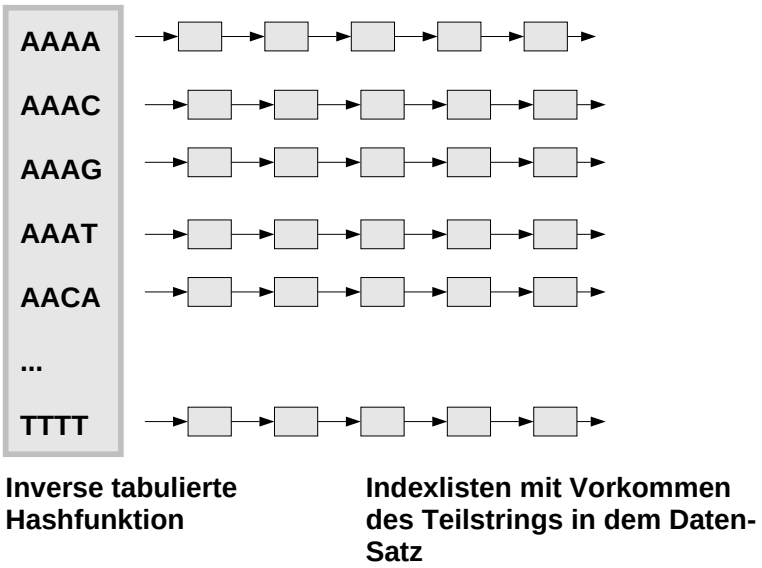


Abb. 88 Indexlisten zur schnellen Suche

Kommt nun im Suchstring der Teilstring *ACGT* vor, sucht man den Teil der Indexliste durch, der auf die Vorkommen von *ACGT* verweist. Ausgehend von dieser Position werden die rechten und linken angrenzenden Teilstrings der Suchsequenz auf den Hammingabstand hin untersucht:

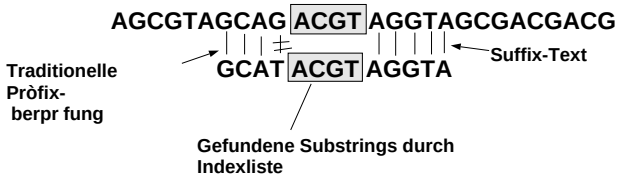


Abb. 89 Berechnung des Hammingabstandes bei gefundenem Substring

Durch geschickte Implementierung kann erreicht werden, daß pro Indexlistenelement nur 2 Byte Speicherplatz benötigt werden.

Diskussion: Ist die maximale Suchtiefe so gering, daß es bei allen Treffern immer vier aufeinanderliegende Positionen mit Hammingabstand 0 gibt, führt dieser Algorithmus zu erheblichen Rechenzeiteinsparungen. Bei einer Sondenlänge von 19 kann so immerhin bis zu einer Suchtiefe von 4 Mismatches gearbeitet werden. Aber selbst in diesem Falle muß das ganze Alignment durchgearbeitet werden, da es im Mittel alle 256 Positionen einen möglichen Treffer gibt. Ist dieses Alignment im Hauptspeicher des Computers zwischengespeichert, erhält man eine signifikante Beschleunigung gegenüber einem linear arbeitenden Programm. Andernfalls wird die Laufzeit durch den Flaschenhals *Rechner-Festplatte* begrenzt und es findet keine Beschleunigung der Rechenzeit statt. Der Speicherbedarf der gesamten Indexlisten ist genau doppelt so hoch wie der des zugrundeliegenden Datensatzes.

IX.3

Bäume

Erst durch den Einsatz von Baumalgorithmen kann auf das Laden des gesamten Alignments verzichtet werden. Dabei wird ein baumartiger Index für das Alignment aufgebaut. Jedem Knoten des Indexbaumes wird eine Base zugewiesen, so daß jedem Weg von der Wurzel zu einem Blatt ein charakteristischer String aus Basen zugeordnet wird. Jedes Blatt enthält die Information, an welchen Stellen im Alignment dieser charakteristische String auftritt. Dies soll an einem Beispiel erläutert werden:

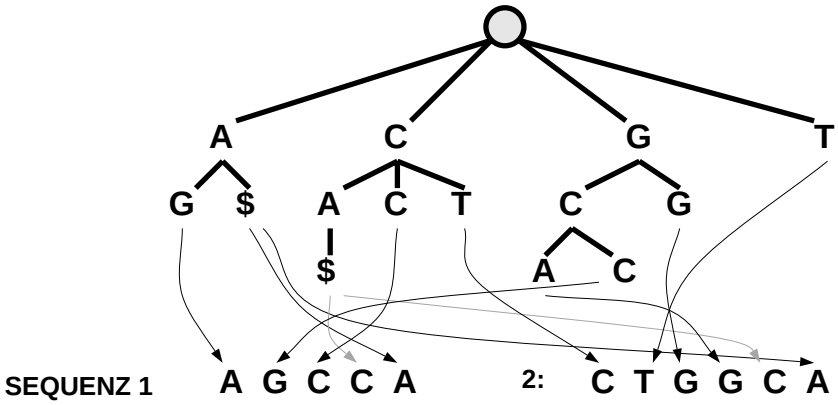


Abb. 90 Positionsbaum für zwei gegebene Sequenzen:

Für die beiden Sequenzen *AGCCA* und *CTGGCA* soll ein baumartiger Index aufgebaut werden. Dazu werden an die Sequenzen die eindeutigen Endmarkierungen *$* angehängt. Soll nun in diesem Baum zum Beispiel nach dem String *CTG* gesucht werden, geht man von der Wurzel zunächst zum Sohn *C*, dann zu dem Untersohn *T*, der als terminaler Knoten auf diejenige Sequenz verweist, die als einzige den String *CT* aufweist. Jetzt muß nur noch überprüft werden, ob dort auf *T* auch *G* folgt.

Diskussion: Bäume bieten ebenso wie das Hashing die Möglichkeit, schnell beliebige Teilzeichenketten zu finden. Außerdem erlauben sie die Anwendung eines einfachen Algorithmus, der eine Suche mit Mismatches zuläßt. Allerdings erhöht sich dabei die benötigte Rechenleistung mit der Anzahl der Mismatches. Hinzu kommt, daß Bäume eine Suche nach einer beliebig langen Zeichenketten erlauben. Aus diesen Gründen wurde für eine Implementierung im Rahmen des ARB-Projektes dieser Ansatz gewählt. Allerdings führt eine Implementierung, die ohne weitere Vorüberlegungen standardmäßig durchgeführt wird, aus mehreren Gründen zu einem Mißerfolg. So mußte der vorgestellt Basisalgorithmus noch entscheidend optimiert werden, um sinnvoll zur Anwendung gebracht werden zu können.

Da ein Alignment, an dem gearbeitet wird, nicht statisch ist, muß bei Bedarf der Indexbaum angepasst werden. Dabei gibt es grundsätzlich die beiden Möglichkeiten, den Index entweder laufend anzupassen oder ihn von Zeit zu Zeit vollkommen neu zu berechnen. Dieses zweite Vorgehen ist in der Praxis nicht nur schneller, es ermöglicht auch eine optimierte Speicherung des Baumes auf der Festplatte.

IX.3

Realisierung

Theoretische Abschätzungen

Die Abschätzung des Aufwands soll theoretisch für einen Indexbaum für ein Alignment durchgeführt werden. In einer klassischen Implementierung besitzt jeder innere Knoten des Indexbaumes:

- 7 möglicherweise leere Zeiger auf 7 Söhne = 28 Bytes.
- 8 Bytes Overhead für die Speicherverwaltung des Betriebssystems.

Jedes Blatt dieses Indexbaumes hat

- 8 Bytes Overhead für die Speicherverwaltung des Betriebssystems.
- Eine Referenz auf einen Organismus: 2 Bytes
- Ein Verweis in ein Alignment: 2 Bytes

Nimmt man einen durchschnittlichen Verzweigungsgrad des Baumes von 2 an[1], so besitzt der Baum ebensoviele innere Knoten wie Blätter. Bei 10 Millionen Blättern ergibt dies einen Gesamtspeicherbedarf von 480 MByte für den Index. Verzichtet man auf eine automatische Speicherverwaltung des Betriebsystems, so kann diese Größe auf 320 MByte reduziert werden. Dies entspricht aber immer noch etwa 32 mal der ursprünglichen Sequenzmenge und 128 mal dem Speicherbedarf, den das Programm 'blast' für eine komprimierte Darstellung der Sequenzen benötigt.

Bei einer testweisen Implementierung des Indexbaum-Algorithmus mußte leider festgestellt werden, daß der Speicherbedarf diese Abschätzung sogar noch übertraf. Bei genauerer Betrachtung stellte sich heraus, daß innerhalb eines Alignments oft Paare nahezu identischer Sequenzen vorkommen. Ein solches Paar resultiert in einer extremen Höhe dieses Indexbaumes, was zu unnötigem Speicherverbrauch führt.

Speicherbedarfs-Optimierung

Durch eine sinnvolle Kombination verschiedener Optimierungsstrategien konnte der Platzbedarf des Index immerhin um den Faktor 8 reduziert werden.

Begrenzung der Baumhöhe Durch eine Begrenzung der Baumhöhe auf 20 (dies entspricht in etwa der Länge der zu untersuchenden Oligonukleotidsonden) konnte das Entarten des Baumes im Sinne der oben genannten Paarigkeit verhindert werden. Dazu mußte ein neue Art von Blatt eingeführt werden, das eine Menge von Sequenz- und Alignmentpositionen speichern kann:

[1] Praktische Untersuchungen haben einen deutlich niedrigeren Verzweigungsgrad gezeigt

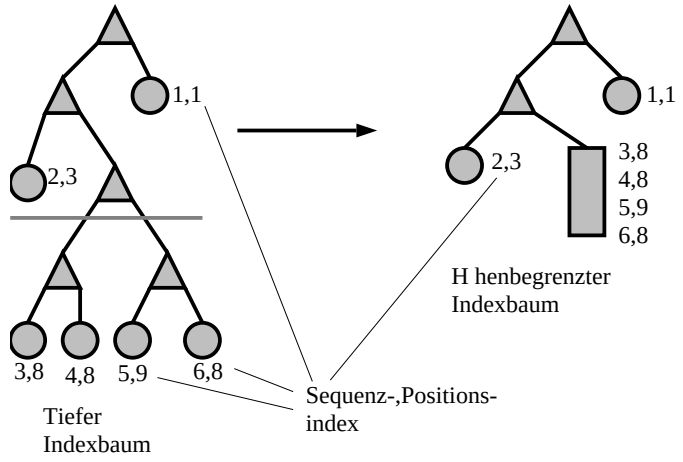


Abb. 91 Zusammenfassung eines Unterbaumes in ein Gruppenblatt

Die Begrenzung der Höhe sorgt dafür, daß der Baum nicht entarten kann. Die Begrenzung wird so gewählt, daß sie höher als die Länge der zu suchenden Zeichenketten ist. In diesem Fall hat diese Begrenzung keinen negativen Einfluß auf die Laufzeit, die Antwortzeiten verkürzen sich eher, wenn auch geringfügig.

Komprimierte Speicherung der Knoten Bei einer maximalen Verzweigungsbreite von 7 und einer durchschnittlichen von 2 sind im Mittel 5 Zeiger je Knoten unbenutzt. Mittels eines kurzen Bitfeldes, das jedem Knoten vorangestellt wird, kann angegeben werden, ob Speicherbedarf für einen Zeiger reserviert wurde. So muß nur Speicher für die benutzten Verweise bereitgestellt werden.

Zusätzlich besitzt jeder Knoten ein vorangestelltes **Tag-**Byte, das den Typ dieses Knotens bestimmt.

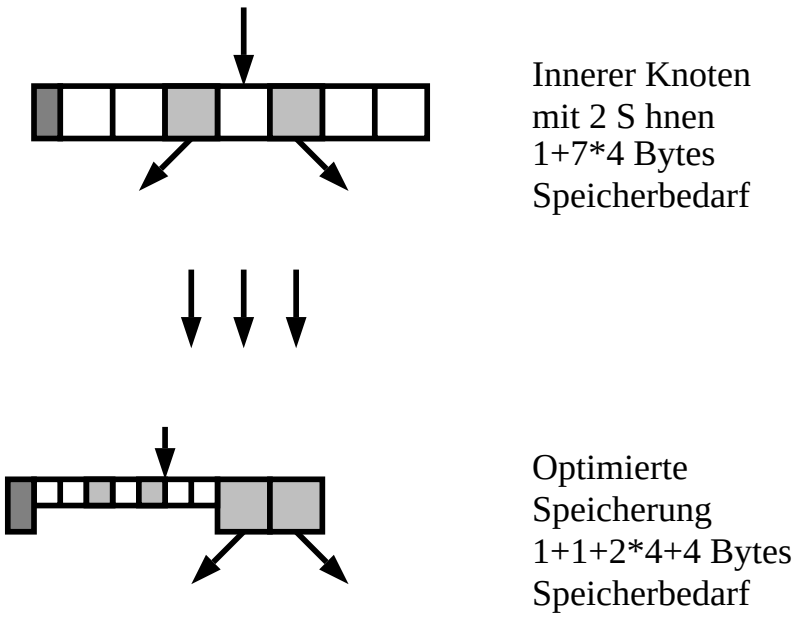


Abb. 92 Komprimierte Speicherung von inneren Knoten mit Hilfe eines Speicherbelegungsbitfeldes

Während der Erstellung des Index werden die Knoten und Blätter aus algorithmischen Gründen in beliebiger Reihenfolge im Speicher verteilt. Diese scheinbar zufällige Reihenfolge kann dergestalt umsortiert werden, daß ein innerer Knoten und dessen Söhne im Speicher des Computers nahe beieinander stehen. Werden nun Verweise auf Söhne nicht als absolute Zeiger, sondern relativ zur eigenen Position vermerkt, erhält man auf diese Weise mit hoher Wahrscheinlichkeit kleine Werte für die relativen Zeiger. Für die Speicherung dieser kleinen Zahlen werden prinzipiell nicht mehr 4 Byte, sondern nur noch 1 oder 2 Byte benötigt.

Da jedoch immer noch die Möglichkeit besteht, daß diese Zahlen nicht in 4 Byte gespeichert werden können, muß mit Hilfe eines weiteren Bitfeldes der Platzbedarf dieser Zahlen festgelegt werden. Dazu werden 8 Bit benötigt:

- Das erste Bit *B* gibt an, ob die höchste vorkommende zu speichernde Referenz sich in 2 Byte speichern läßt.
- Das *i*=2..8te Bit *I* bestimmt, ob die Referenz *i-1* als 1,2 oder 4 Byte Zahl gespeichert ist:

B	*I*	Anzahl der Bytes zur Speicherung einer Referenz
0	0	1
0	1	2
1	0	2
1	1	4

In der Praxis hat sich gezeigt, daß es sehr viele Knoten gibt, die nur einen Sohn haben und daß dieser im Speicher in unmittelbarer Folge des Knotens abgelegt ist. Für diesen Spezialfall wurde eine Speicherform entwickelt, die alle benötigten Informationen in dem noch teilweise ungenutzten Tag-Byte ablegt.

Eine Analyse des Indexbaumes für einen 16S-rRNS-Datensatz ergab folgende absolute Zahlen für die verschiedenen benötigten Speicherbedarfklassen:

Bedarf an Speicher für eine Referenz	Absolute Anzahl der Referenzen	Relative Anzahl der Referenzen
0 (im Tag Byte gespeichert)	1866785	50%
1	1724876	46%
2	106316	3%
4	45850	1%

Abb. 93 Typischer benötigter Speicherplatz für eine Referenz in einem Indexbaum

Der ursprüngliche Bedarf von 4 Bytes je Referenz konnte also auf unter 1 Byte pro Referenz gedrückt werden.

IX.3

Zugriffs-Optimierungen

Aufbau der Bäume Eine einfache Implementierung des Algorithmus zum Aufbau des Index-Baumes greift wahlfrei auf den schon aufgebauten Index zu. Aus Sicht der Speicherverwaltung geschehen diese Zugriffe in zufälliger Reihenfolge. Kann der gesamte Index-Baum im Hauptspeicher gehalten werden, können diese wahlfreien Zugriffe schnell, d.h. im Mikrosekunden-Bereich, durchgeführt werden. Reicht der Hauptspeicher jedoch nicht mehr aus und müssen Teile des Indexbaumes auf eine (langsame) Festplatte ausgelagert werden, sinkt die Zugriffsgeschwindigkeit um den Faktor 1000. Der Aufbau eines Indexbaumes unter Berücksichtigung einer solch verlangsamten Zugriffsgeschwindigkeit dauert für den praktischen Betrieb zu lange (mehrere Stunden!).

Infolgedessen wurde beim Aufbau des Index folgende Strategie gewählt: Der Indexbaum wird in mehreren Schritten aufgebaut. Jeder dieser Schritte generiert einen Teilbaum, der vom Speicherbedarf her in den Hauptspeicher des Rechners paßt. Die einzelnen Teilbäume werden dann in einem externen Speicher zusammengefügt. Jeder Schritt generiert den Teilbaum für ein bestimmtes Zeichenkettenpräfix:

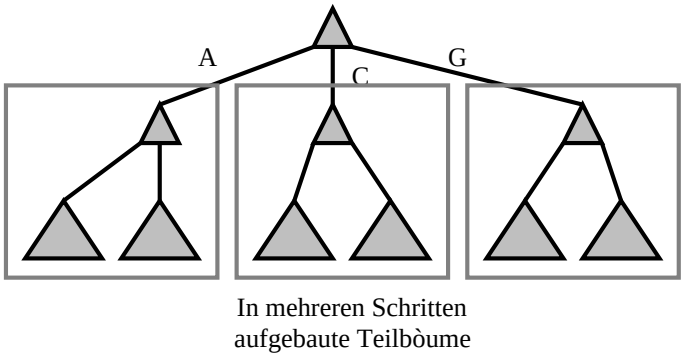


Abb. 94 Generierung des Index in mehreren Schritten

Algorithmus:

- Schätze die Größe des zu berechneten Index ab: *G*.
- Messe die Größe des benutzbaren Hauptspeichers *H*.
- Berechne die Anzahl der Schritte $N=integer(0.5+log_4(H/G))$.
  Die *integer*–Funktion liefert dabei den gerundeten Ganzzahlenwert des Arguments. Es wird der Logarithmus zur Basis 4 eingesetzt, da in der Praxis nur die 4 Buchstaben A, C, G und T mengenmäßig relevant vorkommen.
- Erstelle eine Menge *M* aller möglichen Zeichenketten aus den Buchstaben A, C, G, T, -, . und N der Länge *N*
- Für alle Zeichenketten *s* aus *M*:
  - ☐ generiere den Indexbaum für alle Positionen im Alignment, an denen die Sequenzen mit dem Präfix *s* beginnen.
  - ☐ schreibe diesen Teilindexbaum auf die Festplatte.

- Alle auf der Festplatte befindlichen Teilindexbäume werden zusammengefaßt. Dazu muß ein Indexbaum für die Menge der Präfixe *M* erstellt werden. Die Blätter dieses Indexbaumes zeigen nicht auf Alignmentpositionen, sondern auf einen Teilindexbaum.

Cluster-Bildung In der Praxis ist der Indexbaum größer als der Hauptspeicher. Die Zugriffsgeschwindigkeit wird daher im wesentlichen durch die langsamen Festplattenzugriffe bestimmt. Je häufiger ein solcher Festplattenzugriff nötig ist, desto langsamer arbeitet das Programm. Bei einem solchen Festplattenzugriff werden vom Betriebssystem regelmäßig nicht nur die benötigten Daten geladen, sondern auch der ganze umgebende Bereich. Wenn man dafür sorgt, daß die nächsten Zugriffe auf den Indexbaum in genau diesem umgebenden Bereich des letzten Zugriffs liegen, kann die Abarbeitung des Programmes ganz erheblich beschleunigt werden.

Daher wird der Indexbaum auf der Festplatte so organisiert, daß die Teilbäume in einem zusammenhängenden Bereich gespeichert werden:

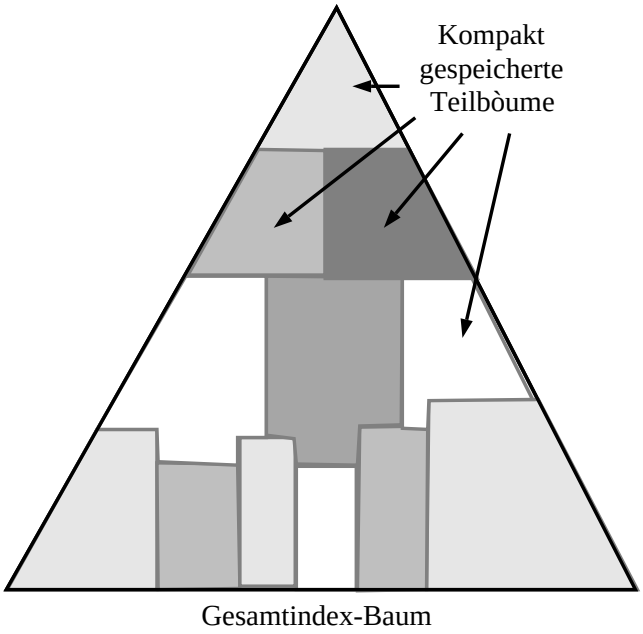


Abb. 95 Organisation des Indexbaumes in zusammenhängende Bereiche

Durch diese Bereichsbildung erhält man einen Algorithmus, der sich vom Zugriffsverhalten ähnlich wie ein *B*- oder *B**- Baum verhält, von der Programmierungsseite aber weiterhin eine einfache Baumschnittstelle zur Verfügung stellt.

Diskussion

Es gibt eine große Menge unterschiedlicher Sequenzsuchalgorithmen. Für große Datenbanken hat sich bis jetzt erstaunlicherweise eine reine lineare Suche durchgesetzt. Im speziellen Anwendungsfall jedoch wurde versucht, mit Hilfe eines Indexbaumes die Suche zu beschleunigen. Insbesondere dann, wenn eine einzelne Sonde im gesamten Datensatz gesucht worden soll, hat sich diese Methode als überaus schnell erwiesen. Sucht man diese einzelne

Sonde mit einer erlaubten Höchstmenge von 0 Mismatches, so ergeben sich Antwortzeiten, die im Bereich von 1 – 100 mSekunden liegen. Mit steigender Anzahl der gesuchten Sonden und mit steigender Anzahl möglicher Mismatches steigt die Antwortzeit jedoch auch bei der hier bevorzugten Lösung bis in den Sekunden-Bereich und damit in den Bereich linearer Suchprogramme.

IX.4 Generierung von Sonden (Probe-Design)

IX.4.1 Praktische Zielsetzungen

Für eine selektierte phylogenetische Gruppe von Sequenzen (**Zielgruppe**) soll eine diese Gruppe identifizierende Sonde theoretisch berechnet werden.

Aus praktischen Überlegungen heraus wurde nicht angestrebt, eine einzelne (einzige) Sonde zu berechnen, vielmehr werden dem Anwender möglichst viele sinnvolle Sonden präsentiert. Mit Hilfe eines leistungsfähigen Programms zur visuellen Kontrolle der Ergebnisse kann der Anwender dann selbst entscheiden, welches Ergebnis er bevorzugt. Dieses Vorgehen hat zu verschiedenen kontroversen Diskussionen geführt, da viele Anwender einen Algorithmus erhofft hatten, der ein korrektes Ergebnis berechnet. Der Verzicht auf die Forderung nach einem perfekten Ergebnis hat jedoch einige ganz entscheidende Vorteile:

- Oft stellen Mikrobiologen sehr komplexe Nebenbedingungen für eine Sonde. All diese Bedingungen in Form von Parametern zu erfassen wäre nicht nur aufwendig, sondern würde die Komplexität der Programmbedienung entscheidend erhöhen. Daher verzichtet man auf die Parametrisierung der Nebenbedingungen und überläßt dem Anwender die Aufgabe, die Ergebnisse nach seinen Wünschen zu interpretieren und weiter zu filtern.
- Oft gibt es keine ideale Sonde. In diesen Fällen ist es sinnvoller, wenigstens eine möglichst große Menge von Suboptima anzuzeigen, als gar kein Ergebnis zu liefern. So ermöglicht das Programm dem Anwender zumindest zu erkennen, wo die Probleme bei der Sondenberechnung liegen.
- Gerade Anwender, die noch wenig Erfahrung auf diesem Gebiet haben, stellen oft zu strenge Nebenbedingungen, die ein Algorithmus nicht erfüllen kann.

Das gewählte Vorgehen setzt allerdings voraus, daß es relativ einfach ist, die Sondenvorschläge des Programms graphisch zu untersuchen, d.h. daß das Bindungsverhalten einer Sonde der Lösungsmenge leicht getestet und phylogenetisch angezeigt werden kann. In der Implementierung von ARB sind aus diesem Grund die Programme Probe-Match und Probe-Design eng miteinander und mit dem phylogenetischen Baum vernetzt.

IX.4.2 Übersichtsschema

Ausgehend von der oben entwickelten Bewertungsvorschrift kann ein Algorithmus entwickelt werden, der mögliche Sonden berechnet. Dieser Algorithmus generiert eine Menge möglicher Sondenkandidaten, die mit Hilfe von Filterprogrammen bewertet und aussortiert werden.

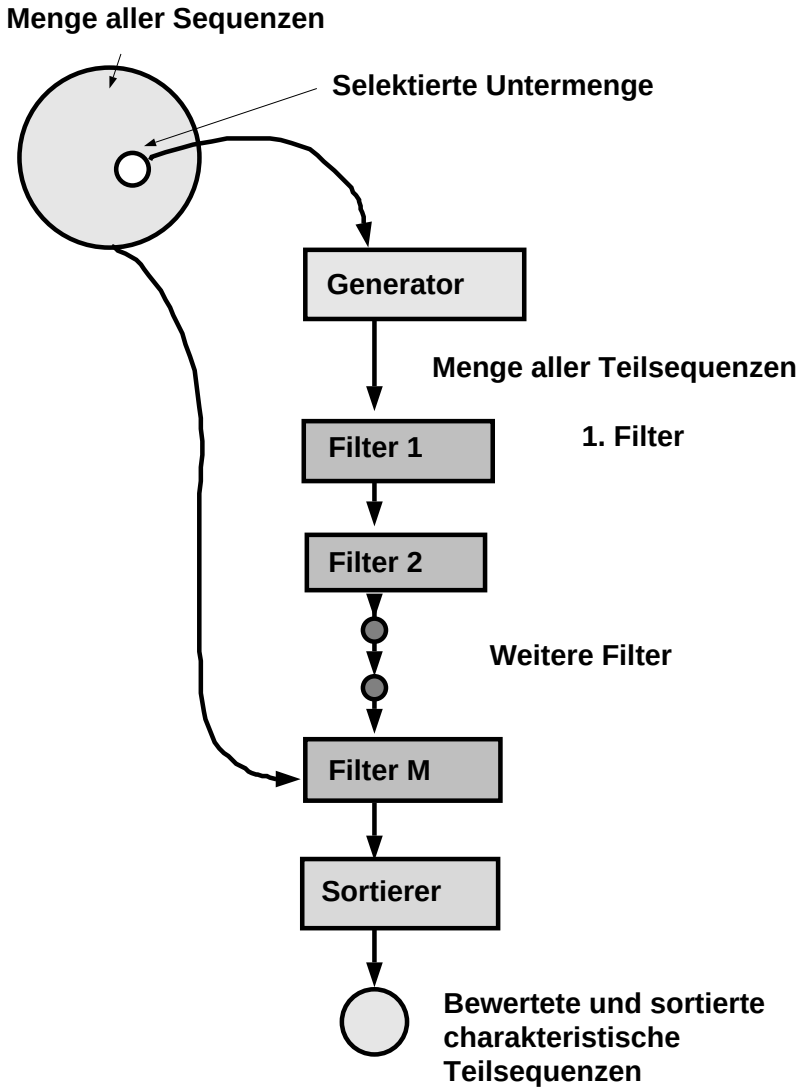


Abb. 96 Grundalgorithmus zur Bestimmung von charakteristischen Sequenzabschnitten

Aus der selektierten Untermenge von Sequenzen werden alle dort vorkommenden Teilsequenzen erzeugt. Diese sehr umfangreiche Menge wird im weiteren Verlauf durch eine Reihe von Filtern immer weiter reduziert, bis schließlich eine Reihe sinnvoller Sonden übrig bleibt, die so gut wie möglich die Anforderungen der Benutzer erfüllen. Sinnvollerweise werden **die rechenintensiveren Filter möglichst spät eingesetzt.** Insbesondere die Bewertung des Bindungsverhaltens einer Sonde mit allen Restsequenzen sollte so spät wie möglich in dieser Pipeline angeordnet werden.

Schließlich wird das Ergebnis mit Hilfe einer Qualitätsbewertung sortiert dargestellt.

Die oben dargestellte Pipeline kann auf zwei verschiedene Arten abgearbeitet werden:

1. Man nimmt jeweils eine Teilsequenz aus der Menge aller möglichen Teilsequenzen und testet der Reihe nach alle Filter durch. Dieses Vorgehen hat den Vorteil, daß der Speicherbedarf minimiert werden kann, da immer nur eine einzelne Teilsequenz zwischengespeichert werden muß.
2. Man generiert eine Menge aller möglichen Lösungen und wendet die Filter der Reihe nach auf diese Menge an. Dieses Vorgehen hat unter Umständen den Vorteil,

daß rechenintensive Filter durch Vorsortierung der Grundmenge optimiert werden können.

In der aktuellen Version von ARB werden die ersten 4 Filter nach der Vorgehensweise 1 durchlaufen, nur der fünfte Filter arbeitet nach der zweiten Methode. Der sich daraus ergebende Vorteil ist, daß der Speicherbedarf niedrig gehalten werden kann. Zudem haben diese Filter die Eigenschaft, unabhängig von der Reihenfolge der Daten zu arbeiten.

IX.4.3 Der Generator

Aufgabe des Generators ist es, aus einer gegebenen Sequenzmenge alle Teilsequenzen einer gegebenen Länge *l* herauszuschneiden. Mehrfachvorkommen einer Teilsequenz sollen nicht eliminiert werden:

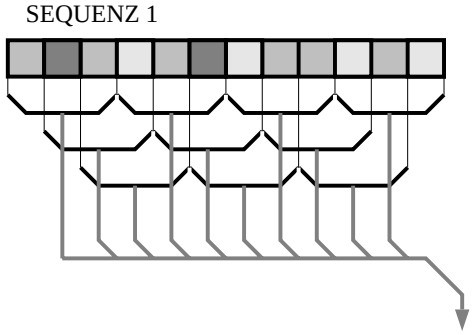


Abb. 97 Generierung aller Teilsequenzen der Länge *3*

IX.4.4 Die Filter

Temperatur– und GC-Filter

Aus der vom Generator erstellten Menge von Teilsequenzen sollen alle diejenigen Sondenvorschläge *V* herausgefiltert werden, deren Bindungstemperatur und GC-Verhältnis innerhalb benutzerdefinierter Grenzen liegt. In der zur Zeit eingesetzten Version von ARB wird eine einfache Formel zur Berechnung der Temperatur herangezogen:

Sei |A| die Anzahl der Basen Adenin in *V* und |C| die Anzahl der Basen Cytosin in *V* (G und T/U analog). Dann errechnet sich die geschätzte maximale Dissoziationstemperatur *T* der Sonde *V* nach folgender Formel:

$$T = 4|G| + 4|C| + 2|A| + 2|U|$$

Das sogenannte GC-Verhältnis berechnet sich aus der relativen Anzahl der Basen *G* und *C*:

$$GC - Verhaeltnis = \frac{|G| + |C|}{|G| + |C| + |A| + |U|}$$

Aus praktischen physikalisch-chemischen Gründen lassen sich nur Sonden einsetzen, deren Dissoziationstemperatur und GC-Verhältnis innerhalb gewisser Schranken liegt.

Min-Group-Hits-Filter

Dieser Filter sorgt dafür, daß eine Sonde mindestens eine minimale Anzahl der Sequenzen der Zielgruppe erfaßt. Idealerweise käme eine Sonde in allen selektierten Sequenzen vor. Allerdings ist dies oft nicht möglich, so daß auf eine untere Prozentschranke ausgewichen werden muß. Da der Generator keine Duplikate entfernt hat und es sehr unwahrscheinlich ist, daß eine Sonde in einer Sequenz zweimal enthalten ist[1], reicht es aus, die Anzahl zu bestimmen, wie oft eine Teilsequenz in der Menge der Teilsequenzen enthalten ist. Diese Anzahl wird in Relation zur Anzahl der Sequenzen der Zielgruppe gesetzt. Ist diese relative Anzahl geringer als ein benutzerdefinierter Wert, wird die getestete Sonde verworfen.

Die Implementierung dieses Filters kann mittels eines Hash-Algorithmus einfach und effizient durchgeführt werden.

Schneller Bindungsbewertungs-Filter

Nachdem mit Hilfe des Min-Group-Hits-Filters das Bindungsverhalten der Sondenvorschläge an die Zielgruppe getestet wurde, muß nun das Bindungsverhalten an alle restlichen Sequenzen getestet werden. Um wirkungsvoll die Zielgruppe möglichst gut gegen die restlichen Sequenzen abzugrenzen, sollte die Sonde möglichst schlecht an die übrigen Sequenzen binden. Um den benötigten Ressourcenverbrauch niedrig zu halten, wird dieser Test in zwei Schritten durchgeführt. Im ersten Schritt wird ein Schnelltest durchgeführt, der mit Hilfe des schon beschriebenen Indexbaumes alle exakten Treffer (d.h. 0 Mismatches) im Datensatz sucht. Eine solche exakte Suche kann sehr schnell durchgeführt werden. Sortiert man alle Sonden vor der Berechnung alphabetisch vor, steigt die Geschwindigkeit der Sondenbewertung, da die Wahrscheinlichkeit steigt, daß bereits auf in den Hauptspeicher geladene Teile des Indexbaumes zugegriffen wird. Der zweite Schritt besteht darin, daß der weiter unten beschriebene 'Bindungsbewertungsfilter' durchgeführt wird.

Positionsfilter bezüglich 16S-RNS von Escherichia coli

Aus physikalischen Gründen kann es vorkommen, daß nur bestimmte Bereiche der rRNS für eine Sonde zugänglich sind. Die anderen Bereiche sind durch Proteine oder andere Strukturen verdeckt. Durch eine Einschränkung der Alignmentpositionen für eine Sonde auf einen Bereich kann ein Anwender nicht verwendbare und somit uninteressante Sondenvorschläge ausfiltern.

Bindungsbewertungsfilter

Der Bindungsbewertungsfilter wird aufgrund seines Rechenleistungsbedarfs als letzte Stufe der Filterkette eingesetzt. Mit seiner Hilfe wird die theoretische Bindung der Sonde an alle Sequenzen, die nicht der Zielgruppe angehören, untersucht. Dabei soll die Suchtiefe möglichst tief gehalten werden (ungefähr 4 Mismatches Suchtiefe).

---

[1] Wahrscheinlichkeit, daß ein Teilstring der Länge 18 zweimal in einer Sequenz der Länge 2000 vorkommt: $\left(\frac{1}{4}\right)^{18} * 2000^2 * 0.5 = 2.91 * 10^{-5}$

IX.4.5 Bewertung eines Sondenvorschlags

Eine ideale Sonde erfüllt die Bedingungen der Filter optimal, wobei den Filtern ein unterschiedliches Gewicht zukommt. Da in den meisten Fällen die physikalischen Bedingungen, unter denen die am Ende ausgewählte Sonde später eingesetzt wird, in hohem Maße angepaßt werden können, sollen in die Bewertung der Sondenvorschläge nur Kriterien aufgenommen werden, die physikalisch und chemisch nicht beeinflußbar sind, nämlich

- die relative Anzahl der Sequenzen der Zielgruppe, die von dieser Sonde erfaßt werden,
- sowie die Anzahl der erfaßten Sequenzen außerhalb der Zielgruppe.

Die Frage, ob eine Sonde bindet, kann oft nicht mit einem eindeutigen ja oder nein beantwortet werden. Vielmehr gibt es einen fließenden Übergang zwischen perfekter Bindung und Nichtbindung.

Ziel ist es, die Sonde so zu berechnen, daß diese später unter Laborbedingungen das gewünschte Verhalten zeigt. Da unter Laborbedingungen mindestens eine gewisse Anzahl gewichteter Mismatches erforderlich ist, um eine eindeutige Nichtbindung zu erhalten (**Diskriminierung**), soll eine vorgeschlagene Sonde möglichst viele gewichtete Mismatches zu allen Sequenzen haben, die nicht der Zielgruppe angehören. Das Verhalten einer Sonde kann mit Hilfe der folgenden Kurve dargestellt werden:

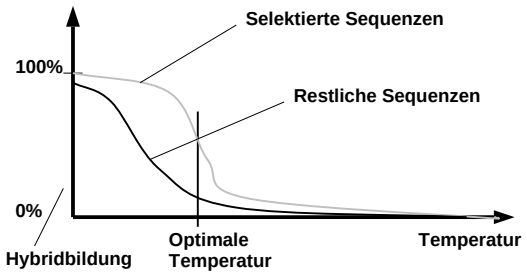


Abb. 98 Hybridisierungsverhalten einer Sonde bei Änderung der Temperatur

Je höher die Temperatur und je höher die Anzahl der gewichteten Mismatches, desto unwahrscheinlicher ist eine Bindung zwischen Sonde und Zielregion. Da im praktischen Experiment die Temperatur nicht exakt konstant gehalten werden kann, **sollte zwischen diesen beiden Kurven ein möglichst größer horizontaler Abstand liegen**, der dafür sorgt, daß ein genügend großer Spielraum für die chemisch-physikalische Durchführung bleibt. Dieser Abstand ist ein direktes Maß für die Qualtität einer Sonde.

IX.4.6 Anzeige der Daten

Die bewerteten Lösungsvorschläge werden nun sortiert, so daß die besten Sonden am Kopf der Ausgabetabelle plaziert werden. Die wichtigsten Informationen bzw. Attribute einer Sonde sollen für eine benutzerkontrollierte Bewertung in jeweils einer Zeile zusammengefaßt werden. Somit erhält der Anwender eine übersichtliche Tabelle als Ausgabe:

IX.4

```
Probe design Parameters:
Length of probe      18
Temperature        [30.0 -100.0]
GC-Content         [50.0 -100.0]
E.Coli Position    [   0 -100000]
Max Non Group Hits     0
Min Group Hits        50%
Target             le apos   ecol grps  G+C 4GC+2AT |Decrease T by n*.3C -> probe matches n non group species
GGAGCACACAACGCGUGG  18 A= 951  923   18  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;39;39; CCACGCGUUGUGUGCUCC
AGGGAAGCUGUUAAGUCC  18 B= 883  855   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; GGACUUAACAGCUUCCCU
UGGAACCUGAGACAAGGU  18 C= 335  311   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ACCUUGUCUCAGGUUCCA
GCCGAAGGGAAGCUGUUA  18 B-   5  850   17  55.6  56.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 8; UAACAGCUUCCCUUCGGC
GCACACAACGCGUGGAGC  18 A+   3  926   18  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 2; 2; GCUCCACGCGUUGUGUGC
GGGAGCACACAACGCGUG  18 A-   1  922   17  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;32;32; CACGCGUUGUGUGCUCCC
UGAAGCUGGAAUGCGUAG  18 D=1354 1325   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 4; CUACGCAUUCCAGCUUCA
GGAACCUGAGACAAGGUU  18 C+   1  312   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 7; 7; 7; AACCUUGUCUCAGGUUCC
GAACCUGAGACAAGGUUC  18 C+   2  313   18  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 3; 3; GAACCUUGUCUCAGGUUC
ACACAACGCGUGGAGCCU  18 A+   5  928   15  61.1  58.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; AGGCUCCACGCGUUGUGU
CACACAACGCGUGGAGCC  18 A+   4  927   15  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; GGCUCCACGCGUUGUGUG
GGAGUGGACGACGGUAGG  18 E=1205 1176   15  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; CCUACCGUCGUCCACUCC
CGAGACCCACGCCCUUAG  18 F=1130 1102   15  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; CUAAGGGCGUGGGUCUCG
AUGGAACCUGAGACAAGG  18 C-   1  310   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; CCUUGUCUCAGGUUCCAU
AACGAUGUGGACUUGGUG  18 G= 835  810   14  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; CACCAAGUCCACAUCGUU
AUGCGUAGUAAUCGCGUG  18 D+  10 1335   15  50.0  54.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; CACGCGAUUACUACGCAU
AACGCGUGGAGCCUGCGG  18 A+   9  932   14  72.2  62.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 2; CCGCAGGCUCCACGCGUU
CGACUGCCCCAGUGCCGA  18 H= 864  838   13  72.2  62.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; UCGGCACUGGGGCAGUCG
GACUGCCCCAGUGCCGAA  18 H+   1  839   13  66.7  60.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; UUCGGCACUGGGGCAGUC
UCACGCCACCCAAAAAGG  18 I=1431 1401   14  55.6  56.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 1;10;19; CCUUUUUGGGUGGCGUGA
GGGGAGCACACAACGCGU  18 A-   2  921   17  66.7  60.0  | 0; 0; 0; 0; 0; 0;32;32;32;32;67;67; ACGCGUUGUGUGCUCCCC
GCGUAGUAAUCGCGUGUC  18 D+  12 1337   15  55.6  56.0  | 0; 0; 0; 0; 0; 0; 0; 0; 7; 7; 7; 7;14; GACACGCGAUUACUACGC
UGCCCCAGUGCCGAAGGG  18 B-  14  841   14  72.2  62.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 3; CCCUUCGGCACUGGGGCA
AUCCUGGGAGGACCACCU  18 J= 724  699   13  61.1  58.0  | 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 2; 2; AGGUGGUCCUCCCAGGAU
GAAGCUGGAAUGCGUAGU  18 D+   1 1326   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 3; 7; 8;13;16;17;17; ACUACGCAUUCCAGCUUC
GAUGGAACCUGAGACAAG  18 C-   2  309   17  50.0  54.0  | 0; 0; 0; 0; 0; 0; 5; 5; 5;12;17;24;24; CUUGUCUCAGGUUCCAUC
GUGCCGAAGGGAAGCUGU  18 B-   7  848   17  61.1  58.0  | 0; 0; 0; 0; 0; 0; 8; 9; 9; 9;18;19;19; ACAGCUUCCCUUCGGCAC
AGACCCACGCCCUUAGUU  18 F+   2 1104   15  55.6  56.0  | 0; 0; 0; 0; 0; 0; 1; 1; 1;32;32;32; AACUAAGGGCGUGGGUCU
```

Abb. 99 Beispiel eines Probe-Design Ergebnisses:

- Die ersten Zeilen wiederholen die vom Benutzer eingegebenen Parameter.
- Die erste Spalte gibt die Zielregion der Sonde an,
- die nächste die Länge der Sonde.
- Darauf folgt die absolute Position der Sondenbindungsstelle im Alignment. Häufig wird jedoch dieselbe Zielregion häufiger als einmal verwendet. Um schnell erkennen zu können, daß es sich um keine vollständig neue Zielregion, sondern nur um eine leichte Abwandlung (+/- wenige Alignmentpositionen) einer bereit verwendeten Zielregion handelt, werden mittels zusätzlicher Zeichen derartige Verbindungen angezeigt. Dabei bedeutet:

Zeichenkette in der Ergebnistabelle	Bedeutung
X=####	Neue Zielregion *X* an der Alignmentposition ####
X+ ##	Leicht verschobene Zielregion *X* um den Offset ##
X- ##	Leicht verschobene Zielregion *X* um den Offset ##

Abb. 100 Bedeutung der Alignmentpositionsfeldes in der Sondenergebnisliste

- Die *ecol*–Spalte gibt die Position der Zielregion in der normierten Ecoli-Nummerierung an.
- Die *grps*–Spalte zählt die von der Sonde erfaßten Sequenzen der Zielgruppe.
- Die *G+C* Spalte zeigt das GC-Verhältnis der Sonde an.
- Die *2AT+4GC* Spalte gibt einen sehr groben Schätzwert für die Dissoziationstemperatur der Sonde an.

- Die darauf folgende Tabelle stellt den wichtigsten Ausschnitt aus dem Bindungsgraphen dar:

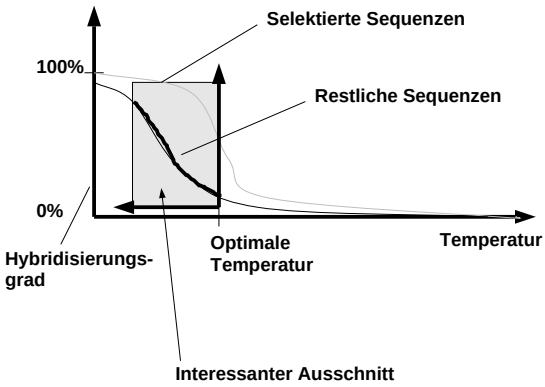


Abb. 101 Interessanter Ausschnitt aus der Hybridisierungskurve

Der Graph *g(x)* zeigt die Anzahl der erfaßten Sequenzen, die nicht der Zielgruppe angehören, falls die Temperatur im Experiment um *x* Grad ausgehend von der optimalen Temperatur gesenkt wird.

- Schließlich zeigt die letzte Spalte die eigentliche Sonde an. Da diese immer komplementär zur Zielsequenz ist, ist diese Angabe eigentlich redundant. Sie wurde jedoch in das Ergebnis aufgenommen, um die Fehlerwahrscheinlichkeit bei der Bildung des Komplements von Hand zu eliminieren.

IX.4.7 Diskussion

Die Berechnung einer praktisch verwendbaren Sonde ist kein triviales Problem. Die Fülle der einstellbaren Nebenbedingungen bei der Berechnung zwingen auch die Benutzer, die im Normalfall nur die grafische Programmoberfläche benutzen und sich nicht um die zugehörigen Programmierarbeiten bemühen müssen, sich intensiv mit den verwendeten Algorithmen auseinanderzusetzen. Im schlechtesten Fall kann der Algorithmus keine Lösung erzeugen, der Anwender kann dann nur versuchen, systematisch zu analysieren, wo das Problem liegt. In der Praxis haben sich folgende Situationen als die Hauptursache für ein schlechtes Ergebnis herausgestellt:[1]

- Es wurde vergessen, Mitglieder der phylogenetischen Gruppe, für die die Sonde berechnet werden soll, zu markieren. Diese Sequenzen sind mit den selektierten sehr nah verwandt, oder sogar identisch. Kein Algorithmus kann eine Sonde berechnen, die eine gegebene Sequenz eindeutig charakterisiert und zu einer identischen Sequenz inkompatibel ist. Um in solchen Fällen dennoch zu einem Ergebnis zu kommen, wurde in dem Probe-Design-Algorithmus ein Parameter implementiert, der es erlaubt, eine Höchstzahl von nicht selektierten Sequenzen festzulegen, an die die Sonde binden darf. Stellt man diesen Wert beispielsweise

[1] Der Grundalgorithmus wurde in Folge so modifiziert, daß er auf einige dieser Problemfälle flexibel reagieren kann.

auf 2 ein und erfassen alle Sondenvorschläge eine bestimmte nicht selektierte Sequenz mit, sollte man sich überlegen, ob diese Sequenz nicht zu der selektierten Gruppe dazugenommen werden soll oder ob es andere physikalisch-chemische Methoden gibt, mit deren Hilfe sich die betroffenen Bakterien unterscheiden lassen.

- Soll für eine einzelne Sequenz eine Sonde berechnet werden und gibt es im Alignment eine nahezu identische Sequenz, ist es oft unmöglich, eine Lösung zu finden. In diesem Fall bleibt nur die Möglichkeit, entweder ein anderes Molekül (z.B. 23S–rRNS statt 16S-rRNS) zu verwenden oder eine Gruppe der identischen Sequenzen zu markieren.
- Soll für eine Sequenz mit Sequenzierfehlern eine Sonde berechnet werden und gibt es zu dieser Sequenz (ohne Sequenzierfehler) eine nahezu identische Sequenz, so wird der Algorithmus eben diese Sequenzierfehler als mögliche Sonden vorschlagen. Um diesen Fall auszuschließen, sollte der Datensatz so weit wie möglich frei von Sequenzierfehlern sein.

Wie bereits erwähnt, liefert der verwendete Algorithmus nur Lösungsvorschläge. Eine Integration mit einem Probe-Match-Algorithmus ist unabdingbare Vorraussetzung für seinen effizienten Einsatz. Nur durch die schnelle phylogenetische Kontrolle der Einzellösungen kann ein Anwender die Vorschläge prüfen und vergleichen.

IX.5 Mehrfachsonden (Multi-Probe)

IX.5.1 Übersicht

Mit einer einzelnen Sonde kann eine der Umwelt entnommene Bakterien- bzw. DNS-Probe gewissermaßen nur eindimensional untersucht werden. Erst durch den Einsatz unterschiedlicher, jeweils farbig markierter Sonden kann eine differenzierte Analyse erfolgen. Werden drei Sonden mit den drei Grundfarben markiert und zu fixierten Bakterien zugegeben, können theoretisch bis zu 7 Bakterien in situ unterschieden werden (Snaider et. al. 1990).

Steht nur eine Einzelsonde zur Verfügung, so kann nur *eine* Untermenge von Organismen markiert werden. Schematisch in einem Mengendiagramm dargestellt, ergäbe sich somit das folgende Bild:

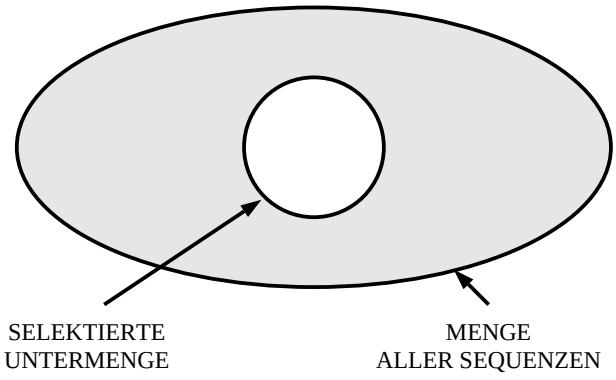


Abb. 102 Mengendiagramm: Die durch eine Einzelsonde markierte Untermenge.

Der multiple Einsatz von Sonden hingegen erlaubt drei grundsätzlich verschiedene Anwendungsfelder:

- Mit steigender Sequenzanzahl wird es immer schwieriger, wenn nicht gar unmöglich, eine Sonde für eine phylogenetische Gruppe von Bakterien zu konstruieren. Eine Begründung hierfür ist es, daß es selten Sonden gibt, die sich an die Sequenzen aller Mitglieder einer selektierten Gruppe anlagern, ohne dabei

auch Sequenzen außerhalb dieser Gruppe mitzudetektieren:

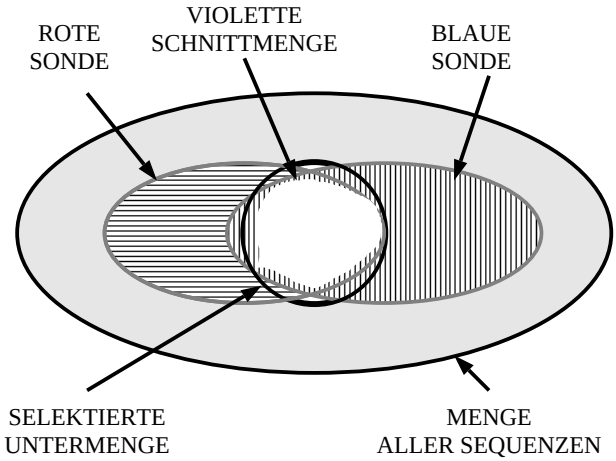


Abb. 103 Differenzierung einzelner Sequenzen, für die keine Einzelsonde konstruiert werden kann.

- Der Einsatz mehrfarbiger Sonden ermöglicht die systematische Suche nach noch unbekannten Bakterien. Dazu wird eine Sondenkombination errechnet, die bei allen bekannten Bakterien in einigen Farben aufleuchten würde, allerdings soll es keinen bekannten Organismus geben, bei dem eine Mischfarbe, wie zum Beispiel gelb, vorkommt. Gibt man diese Sondenkombination dann zu einer Umweltprobe mit verschiedensten Bakterien hinzu und leuchten einige Bakterien in einer Farbe, die vorher im Datensatz nicht vorkam, hat man mit großer Wahrscheinlichkeit ein neues Bakterium gefunden:

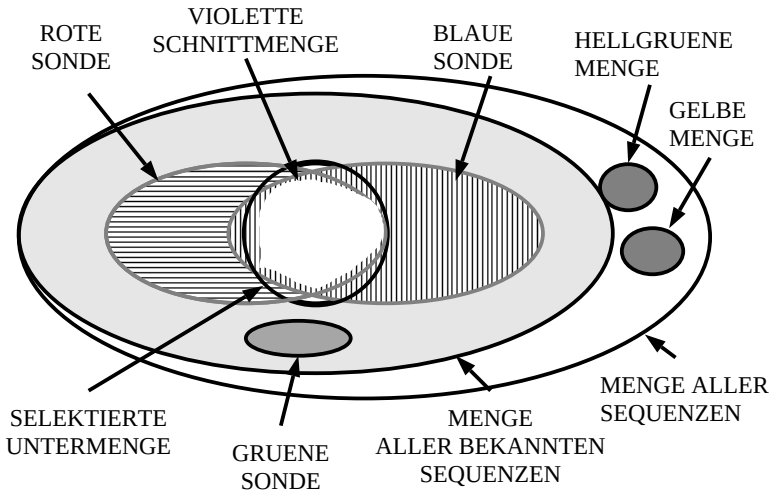


Abb. 104 Suche nach neuen, d.h. gelb und hellgrün leuchtenden Bakterien mit Hilfe von Mehrfachsonden

- Man möchte eine Umweltprobe von Bakterien möglichst schnell analysieren. Mehrere Farben erlauben theoretisch die Einteilung der Bakterien in

$2^{\text{Anzahl der Farben}}$ Klassen:

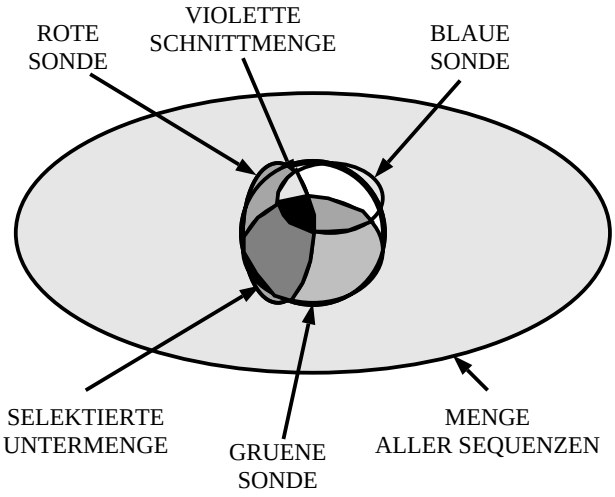


Abb. 105 Aufteilung einer Sequenzmenge in 7 verschiedenfarbige Untermengen:

Ausgehend von einem guten Sondensatz sind die hier vorgestellten Fälle 2 und 3 von Hand in kurzer Zeit zu lösen, wobei die Ergebnisse nicht schlechter sind als eine computerberechnete Lösung. Aus diesem Grunde soll im Rahmen dieser Arbeit ausschließlich der wichtigste und schwierigste Fall 1 angesprochen werden. Im Normalfall gibt es keine optimale Lösung für diesen Fall, die hier ermittelten Lösungen lassen sich dann für die Fälle 2 und 3 mitverwenden.

Der ideale Algorithmus zur Berechnung einer Sondenkombination ist ein erweiterter Probe-Design-Algorithmus. Anstatt eine einzelne Sonde zu berechnen, wird nun jede mögliche Kombination von Sonden getestet. Allerdings sind die hierfür benötigten Ressourcen so hoch, daß dieses Vorgehen nicht sinnvoll erscheint (Benötigt die Berechnung einer Einzelsonde etwa 10 Minuten, so würde die Berechnung einer Kombination aus drei Sonden 10 Minuten * Sequenzlänge * Sequenzlänge = 43 Jahre dauern).

Mit Blick auf diese utopischen Zeiten wurde ein anderer Ansatz gewählt: Man berechnet mit Hilfe des Probe-Design-Algorithmus eine möglichst große Menge von Sonden, wobei diese Sonden auch an eine große Anzahl von nicht selektierten Sequenzen binden dürfen. Diese als Einzelsonden untauglichen Vorschläge werden in einem zweiten Schritt kombiniert und bewertet. Die am besten bewerteten Kombinationen werden dem Anwender als mögliche Lösung präsentiert.

Für diese Bewertung muß ein Bewertungsschema gefunden werden. Ein kanonischer Ansatz, der jeder Sonde eine Menge erfaßter Sequenzen zuordnet und mit Hilfe Boolscher Algebra die Kombinationen der Sonden bewertet, führte nicht zu den gewünschten Ergebnissen. Es mußte ein neuer Algorithmus gefunden werden, der auf die spezielle Fragestellung eingeht und auch Bindungen einer Sonde an eine Zielregion mit Mismatches berücksichtigt.

Prinzipiell können alle möglichen Kombinationen aus Sonden bewertet werden. Bei einer Grundmenge von 100 Einzelsonden und einer gesuchten 3–fach-Sonde müssen also maximal 100*99*98 Kombinationen getestet werden. Man kann davon ausgehen, daß ein schneller Rechner im Mittel 100 Kombinationen pro Sekunde bewerten kann, so daß der Algorithmus in diesem Beispielfall nach 2.5 Stunden terminieren wird.

Um noch schneller an eine Lösung zu kommen, wurde für ARB ein genetischer Algorithmus eingesetzt: Aus einer Menge von Einzelsonden werden *n* Kombination gebildet und bewertet. Iterativ wird nun die Kombinationen dieser Menge geändert (mutiert), kombiniert und bewertet. Je besser eine Kombination bewertet wurde, desto größer ist ihr Einfluß auf die nachfolgenden Generationen. Dieser Algorithmus soll nun im folgenden vorgestellt werden.

IX.5.2 Bewertung einer Sondenkombination

Ziel ist es, eine Sondenkombination zu finden, die eine selektierte Gruppe möglichst gut von allen anderen Sequenzen differenziert. 'Möglichst gut' soll heißen, daß die selektierten Bakterien bei Anwendung der Sonde eine andere Farbe aufweisen als die nicht selektierten.

Es lassen sich drei grundlegende Bewertungskriterien formulieren:

- Die Sonde soll möglichst gut, d.h. ohne Mismatches an die selektierten Sequenzen binden.
- Möglichst viele der selektierten Sequenzen sollen mit der Sonde eine Bindung eingehen.
- Die selektierten Bakterien sollen möglichst gut gegenüber allen restlichen abgrenzbar sein. **Ziel der Bewertung ist es also, unter Berücksichtigung des Bindungsverhaltens der Einfachsonden jeden auftretenden Farbzustand der selektierten Sequenzen gegenüber jedem Farbzustand der übrigen Sequenzen abzugrenzen.**

Die Menge aller Sonden dieser Kombination, die an diese Sequenz binden, wird als **Farbe** diese Sondenkombination für diese Sequenz definiert. Unter einem **Farbabstand** zweier Farben einer Kombination soll im weiteren die Anzahl der Sonden dieser Kombination gemeint sein, die nur bei einer und genau einer der beiden Farben beteiligt ist. Verwendet man zum Beispiel eine Dreierkombination von Sonden, jeweils markiert mit Rot, Grün und Blau, so beträgt der Farbabstand zwischen Rot und Gelb eins (-Rot), zwischen Schwarz und Weiß drei (+Rot +Grün +Blau) und zwischen Rot und Blau zwei (-Rot +Blau).

Betrachtet man das typische Bindungsverhalten einer Sonde in Abhängigkeit der Mismatches, so kann man feststellen, daß es keine klare Grenze gibt, ab der man sagen könnte, eine Sonde bindet oder bindet nicht:

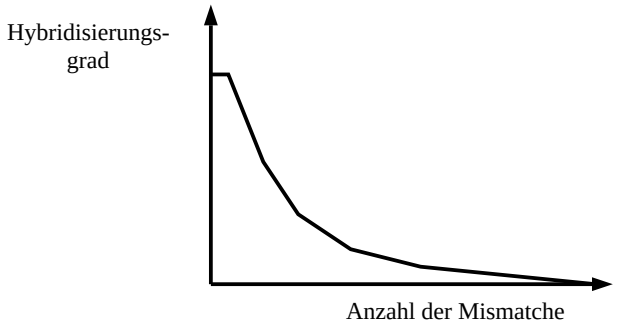


Abb. 106 Relative Anzahl der Bindungen einer Sonde in Abhängigkeit von gewichteten Mismatches

Da also keine exakte Grenze angegeben werden kann, sollen in Abhängigkeit von der gewichteten Anzahl der Mismatches drei Bereiche definiert werden[1]:

- Bereich **0**: Sonde bindet perfekt.
- Bereich **1**: Sonde bindet vielleicht.
- Bereich **2**: Sonde bindet sicher nicht

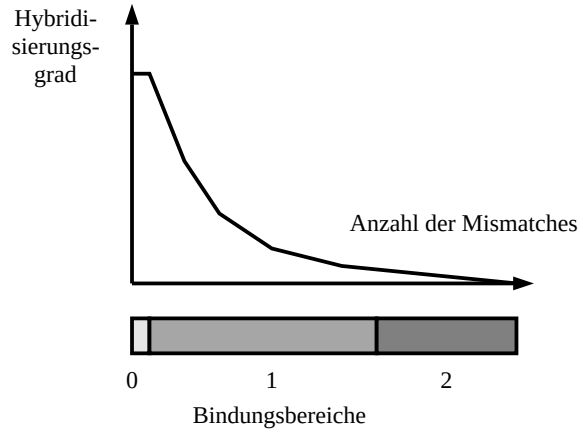


Abb. 107 Aufteilung des Bindungsverhaltens einer Sonde in 3 Bereiche

Für eine gegebene Sequenz läßt sich das Bindungverhalten eines Sondentupels nun mit einem Zahlentupel ausdrücken. Das Zahlentupel **210** soll folgende Aussage enthalten: Sonde 1 bindet nicht, Sonde 2 bindet vielleicht und Sonde 3 bindet sicher. Dieses Zahlentupel entspricht der oben eingeführten Definition der Bindungsfarbe und soll im weiteren synonym mit diesem Begriff verwendet werden.

Für eine gegebene Sonde kann nun eine Statistik über ein Alignment generiert werden. Dabei wird jeweils für die selektierten und für die restlichen Sequenzen ausgezählt, mit welcher absoluten Häufigkeit ein bestimmtes Zahlentupel (eine bestimmte Farbe) auftritt. Das Ergebnis könnte zum Beispiel wie folgt aussehen:

[1] Diese Bereiche gelten unter optimierten Laborbedingungen.

Bindungsfarbe der Sondenkombination	Anzahl der Treffer bei den selektierten Sequenzen	Anzahl der Treffer bei den restlichen Sequenzen
00	5	0
01	2	0
02	1	2
10	3	2
11	0	0
12	0	200
20	0	150
21	1	150
22	0	400

Abb. 108 Ausgangsituation für die Bewertung einer Sonde

Zusätzlich zu dieser Tabelle wird nun der Farbabstand aller auftretenden Farben verglichen. Eine Farbe gilt dann als unterscheidbar von einer anderen, wenn die beiden auch im Labor eindeutig unterscheidbar sind. Für die Farbe *00* ergibt sich die folgende Farbabstandstabelle:

Farbe 1	Farbe 2	Farbabstand
00	00	0
00	01	0
00	02	1
00	10	0
00	11	0
00	12	1
00	20	1
00	21	1
00	22	2

Abb. 109 Farbabstand zweier Farben

Für jede Farbe bei den selektierten Sequenzen muß nun ermittelt werden, welche Farben jeweils bei allen restlichen Sequenzen aufgetreten sind und wie groß der Abstand der Farben ist. (Frage: Kann man die selektierten von den nicht selektierten Sequenzen unterscheiden?) Dazu wird für jede Farbe 1 eine Tabelle erzeugt, in der der Farbabstand zu allen möglichen Farben 2 sowie die Anzahl der nicht selektierten Sequenzen, die die Farbe 2 annehmen, eingetragen wird Dies soll exemplarisch für die Farbe *00* durchgeführt werden:

Farbe 1	Farbe 2	Farbabstand 1-2	Anzahl der Farbe 2 bei den nicht selektierten Sequenzen
00	00	0	0
00	01	0	0
00	02	1	2
00	10	0	2
00	11	0	0
00	12	1	200
00	20	1	150
00	21	1	150
00	22	2	400

Abb. 110 Farbabstand und Anzahl des Auftretens der Farbe bei den nicht selektierten Sequenzen

Aus diesen Tabellen läßt sich für jede Farbe eine Summentabelle bilden, in der die Anzahlen der nicht selektierten Sequenzen, die zu den selektierten Sequenzen denselben Farbabstand aufweisen, durch Addition aggregiert werden:

Farbabstand zur Farbe 1 (im Beispiel *00*)	Anzahl der nicht selektierten Sequenzen
0	0 + 0 + 2 + 0 = 2
1	2 + 200 + 150 + 150 = 402
2	400

Abb. 111 Anzahl der Treffer je Farbabstand für die Ausgangsfarbe *00*

Aus praktischen Gründen kann es wie im vorliegenden Beispiel vorkommen, daß eine Sonde zufällig einige der nicht selektierten Sequenzen nicht eindeutig von den selektierten Sequenzen abgrenzen kann. Diese zufälligen Treffer können dazu führen, daß es praktisch unmöglich wird, eine eindeutige Lösung zu finden.

Um dennoch Lösungen angeben zu können, wurde der Parameter *ignorehits* eingeführt, der angibt, wie viele zufällig gebundene Sonden außerhalb der selektierten Sequenzen der Algorithmus ignorieren soll. Die besten Bindungen außerhalb der selektierten Sequenzen bleiben somit bis zur Anzahl *ignorehits* unberücksichtigt. Im Beispiel bedeutet dies bei *ignorehits=10*, daß folgende korrigierte Tabelle entsteht:

Farbabstand zur Farbe 1 (im Beispiel *00*)	Anzahl der nicht selektierten Sequenzen, wobei die 10 besten Treffer entfernt wurden.
0	0 + 0 + 2 + 0 = 2 - 2 = 0
1	2 + 200 + 150 + 150 = 402 - 8 = 394
2	400 -0 = 400

Abb. 112 Anzahl der um *ignorehits* korrigierten Treffer je Farbabstand für die Ausgangsfarbe *00*

Diese Tabelle besagt nun, wie gut die in der Farbe 1 aufleuchtenden selektierten Sequenzen gegen alle restlichen Sequenzen unterschieden werden können. Im Beispiel können 5 Bakterien, die in der Farbe *00* aufleuchten, gegen 394 Bakterien mit dem Farbabstand 1 und gegen 400 Bakterien mit dem Farbabstand 2 unterschieden werden. Insgesamt ergibt sich für 5 Bakterien ein minimaler Farbabstand von 1.

Die aggregierte Tabelle für alle Farben 1 der selektierten Sequenzen sieht folgendermaßen aus:

Farbe *F*	Anzahl der selektierten Sequenzen, die in der Farbe *F* aufleuchten	Minimaler Farbabstand zu allen nicht selektierten Sequenzen
00	5	1
01	2	1
02	1	1
10	3	0
11	1	0
12	0	0
20	0	0
21	1	0
22	0	0

Abb. 113 Gesamtbewertung einer Sondenkombination

Aufbauend auf diese Tabelle lassen sich einfache Formeln für eine mögliche Gesamtbewertung finden. In ARB wurde folgende Bewertungsformel verwendet:

$$Qualitaet = \sum_{F=Farben} Selektierte\ Sequenzen\ mit\ Farbe\ F * Farbabstand\ zum\ Rest$$

Die im Beispiel bewertete Sonde würde demnach die Bewertung 5*1 + 2*1 + 1*1 = 8 erhalten. Je nach Anwendungsfall läßt sich diese Formel vielfältig modifizieren:

- Ist es zum Beispiel wichtig, daß alle Sequenzen der selektierten Menge von einer Sondenkombination erfaßt werden, kann die Anzahl der Sequenzen mit einem höheren Gewicht (d.h. mit einer höheren Potenz) in die Formel eingehen.
- Ist es jedoch wichtiger, daß die Sequenzen mit möglichst großem Farbabstand unterschieden werden können, läßt sich in obiger Formel der Farbabstand zum Beispiel durch das Quadrat des Farbabstandes ersetzen.

Zusammenfassend kann gesagt werden, daß durch diesen Algorithmus eine einfache Tabelle erzeugt werden kann, mit deren Hilfe die Qualität oder Güte einer Sonde einfach bewertet werden kann.

IX.5.3 Generierung sinnvoller Sonden-Kombinationen

Aus einer Menge von Einzelsonden soll eine möglichst gute Kombination zur optischen Unterscheidung von Sequenzen gefunden werden. Da aus praktischen Gründen eine vollständige Bewertung aller möglichen Kombinationen ausscheidet, wird in ARB ein genetischer Algorithmus eingesetzt, der im folgenden kurz wiedergegeben wird.

Jeder Einzelsonde wird eine eindeutige Zahl zugeordnet, so daß sich eine Sondenkombination als ein einfaches ungeordnetes Zahlentupel codieren läßt.

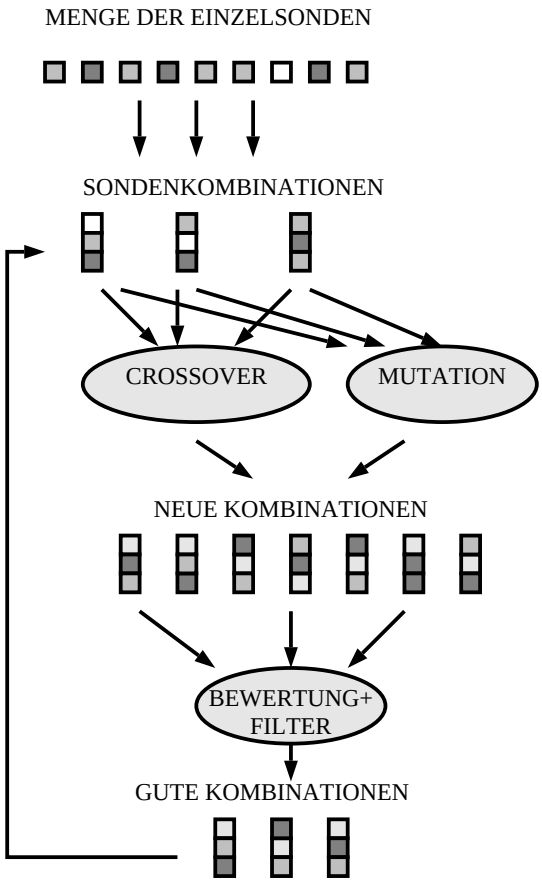


Abb. 114 Übersicht über den genetischen Algorithmus zur Findung einer guten Sondenkombination

Aus der Menge der bewerteten Einzelsonden wird eine Startmenge (Startpopulation) aus **n** Mehrfachsonden gebildet. Dies geschieht, indem zufällig Kombinationen aus den Einzelsonden zusammengestellt werden, wobei die Bewertung der Einzelsonde berücksichtigt wird,

indem gut bewertete Einzelsonden mit höherer Wahrscheinlichkeit in einer Kombination auftauchen.

Diese Menge von Sondentupeln wird nun wiederholt verändert, bewertet und gefiltert:

- Es gibt zwei Möglichkeit zu neuen Kombinationen von Sonden zu kommen: Zum einen kann eine bestehende Kombination dadurch zufällig geändert werden, daß eine Sonde durch eine zufällig gewählte andere Sonde ausgetauscht wird, zum anderen lassen sich zwei Sondenkombinationen zufällig kombinieren (crossover). Dazu werden aus den beiden Ausgangskombinationen ohne System Sonden ausgewählt und zu einem neuen Tupel zusammengefügt.
- Alle so erhaltenen Kombinationen werden bewertet.
- Durch Kombination und Mutation wird eine wesentlich größere Menge an Kombinationen generiert als die Ausgangsmenge. Da die Gesamtanzahl der Sondenkombinationen konstant gehalten werden soll, wird diese Menge durch einen Filter wieder auf ihre ursprüngliche Anzahl reduziert. Dazu werden die **n** besten Kombinationen ausgewählt und die restlichen verworfen.

Wurden nach genügend Iterationsschritten mehrere brauchbare Lösungen berechnet, kann der Vorgang abgebrochen werden. Die bis dahin erhaltenen Kombinationen werden dem Anwender als Lösung präsentiert. Diese lassen sich wie die Einzelsonden schnell und einfach mit Hilfe des phylogenetischen Baumes überprüfen.

IX.6 Sonden: Diskussion und Ausblick

Die in ARB implementierten Programme zur Generierung und Evaluation von Sonden wurden entwickelt als die Sondentechnik an sich noch neu war. Ziel der damaligen Entwicklungen war es, die Forschung im Bereich der computerunterstützten Sondenberechnung voranzutreiben. Ein solches Ziel kann um so besser erreicht werden, je flexibler und transparenter die eingesetzten Algorithmen und Programme sind. So wurden die entwickelten Algorithmen in eine interaktive Analyseumgebung eingebettet. Das rein interaktive Vorgehen besitzt jedoch auch einige gravierende Nachteile: In der jetzigen Implementierung müssen alle Vorgänge von Hand ausgelöst und überwacht werden. Langfristig gesehen sollte hingegen das Ziel sein, ein System zu schaffen, das mit Hilfe der bis dahin gesammelten Erfahrung den gesamten Prozeß der Sondenberechnung durchführt und optimiert. Da in diesen Prozeß eine Vielzahl unterschiedlichster Parameter mit einfließen, wird diese vollautomatische Sondenberechnung auf einem Expertensystem aufsetzen müssen.

Das Problem der Sondenberechnung umfaßt mehrere Unteraufgaben, deren Lösung zum Teil nicht mit Standardalgorithmen der Informatik erreicht werden kann. Daher entstanden für ARB eine Reihe speziell für diesen Anwendungsfall optimierte Algorithmen. Besonders der Algorithmus zur schnellen Sequenzsuche in einem Alignment ist in der derzeitigen Implementierung besonders im Hinblick auf die Anwendung auf ribosomale RNS-Sequenzen optimiert. Diese Sequenzen weisen die besondere Eigenschaft auf, untereinander größtenteils sehr ähnlich, teils sogar vollkommen identisch zu sein. Wird der derzeitige Algorithmus auf andere Alignments angewendet, ist mit einer deutlich schlechteren Performance zu rechnen. Es wäre in diesem Zusammenhang zu überlegen, ob man den für diesen speziellen Anwendungsfall optimierten Algorithmus nicht durch einen etwas breiter anwendbaren ersetzt, der bei anderen Datensätzen gleichbleibend gute Leistung zeigt, bei ribosomalen RNS-Sequenzen jedoch etwas schlechter abschneidet.

Eine Sondenberechnung ist von vielen Parametern abhängig, auf die ein Anwender nur teilweise Einfluß nehmen kann. Denkbar wären hier falsch berechnete phylogenetische Bäume, schlecht sequenzierte Alignments, schlechte Auswahl der phylogenetischen Gruppe, unvollständige Bäume usw. Sowohl in einem interaktiven wie auch in einem vollautomatischen System kann um so flexibler auf diese Probleme eingegangen werden, je mehr Information zur Verfügung steht. So wurde in der jetzigen Implementierung von ARB begonnnen, möglichst alle verfügbaren Informationen über Sonden, Phylogenie und Alignments in einem Gesamtsystem so zu integrieren, daß sie eine integrierte Einheit sowohl in Funktion als auch in Bedienung bilden. Der nächste logische Schritt wird die Verknüpfung der Sondenberechnung mit einer öffentlichen Sondendatenbank sein. Es gibt heute weltweit zwei öffentliche Sondendatenbanken, die der RDP und die des Lehrstuhls für Mikrobiologie der TU München. Noch sind diese Datenbanken allerdings reine Datenbanken und noch nicht mit den Algorithmen Probe-Match, Probe-Design und Multi-Probe verbunden. Erst durch diese Verbindung kann ein System geschaffen werden, in dem die gesamten Erfahrungen in Form einer Datenbank in den Berechnungsprozeß einfließen können.

X Das integrierte Softwaresystem

X.1 Was ist das Besondere an ARB ?

Warum hat sich das Projekt ARB als so erfolgreich im praktischen Einsatz erwiesen? Zur Beantwortung dieser Frage lassen sich drei entscheidende Ideen identifizieren, die so grundlegend sind, daß sie als die Säulen des Erfolges von ARB angesehen werden können.

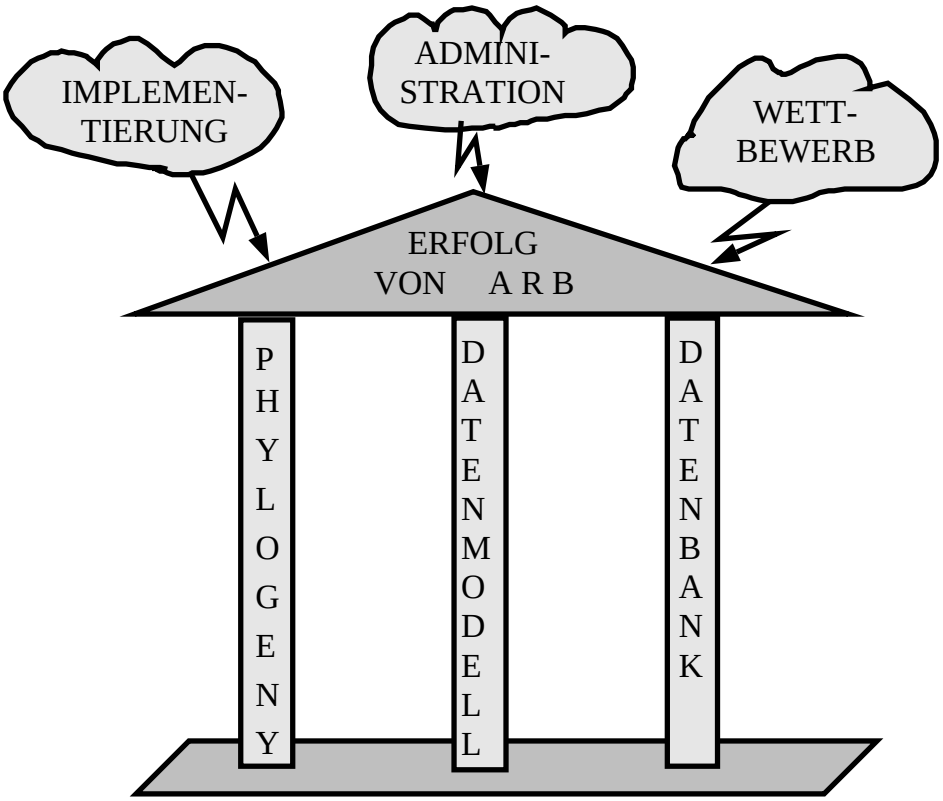


Abb. 115 Die Säulen der praktischen Einsetzbarkeit von ARB

Diese drei zentralen Ideen sind:

- die **Phylogenie** als zentrales Objekt der Datenbetrachung, –verarbeitung und –speicherung
- ein einfaches und flexibles **Datenmodell**
- die effiziente Organisation von Daten und Algorithmen mit Hilfe einer speziell entwickelten **Datenbankverwaltung**. Dazu wurde auf eine (vor dieser Arbeit schon) existierende Datenbank (aus Vereinfachungsgründen des weiteren mit ARB-Datenbank abgekürzt) aufgesetzt und leistungsfähige Kompressionsalgorithmen integriert.

Diese sollen nun kurz vorgestellt werden, wobei **bewußt auf den Einsatz von formalen Beschreibungsmethoden verzichtet wurde**, da diese keine neuen Erkenntnisse liefern würden.

X.1.1 Phylogenie als zentrales Objekt

In allen bekannten integrierten Sequenzanalyse-Paketen stehen die Sequenzen und somit die Grunddaten im Vordergrund. Dieses Vorgehen stößt aber mit steigender Sequenzanzahl schnell auf Grenzen. Deshalb wurde bei ARB der neuartige Ansatz gemacht, den phylogenetischen Baum und damit ein Strukturierungselement in den Mittelpunkt der Betrachtung zu stellen.

Die Idee

Betrachtet man existierende integrierte Programme, stellt man fest, daß diese meistens von Sequenzen ausgehen. Dies ist vom Anwender aus gesehen das, was intuitiv erwartet wird, da dieser mit seinen noch nicht alignten Sequenzen die Arbeit beginnt. So zeigt zum Beispiel der GDE-Editor immer die Arbeitsdaten auf dem Bildschirm. Es entspricht damit dem in der Textverarbeitung bekannten WYSIWYG-Prinzip[1]. Was jedoch bereits bei Textverarbeitung bei langen Texten zu Problemen führt, wird logischerweise auch bei dieser Anwendung nicht unproblematisch sein. Deshalb wurde für ARB bereits am Beginn der Arbeiten dazu übergegangen, den phylogenetischen Baum als den Ausgangspunkt aller Analysen zu betrachten. Statt also Sequenzen in einem Sequenzeditor für eine Analyse zu selektieren, muß man nun Blätter in einem Baum per Mausklick selektieren:

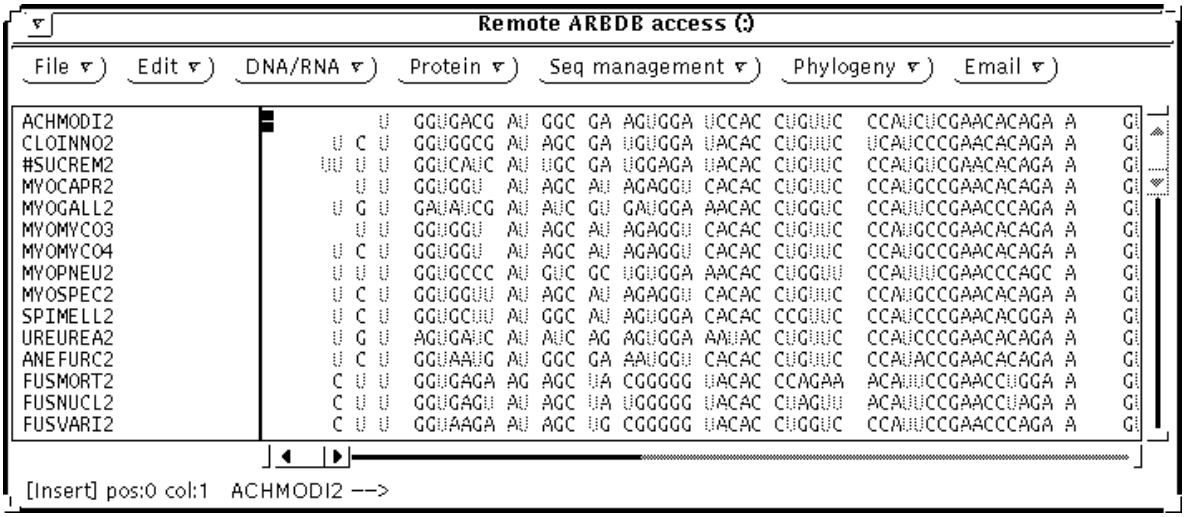


Abb. 116 Traditionelle Vorgehensweise am Beispiel des GDE-Editors

[1] What You See Is What You Get

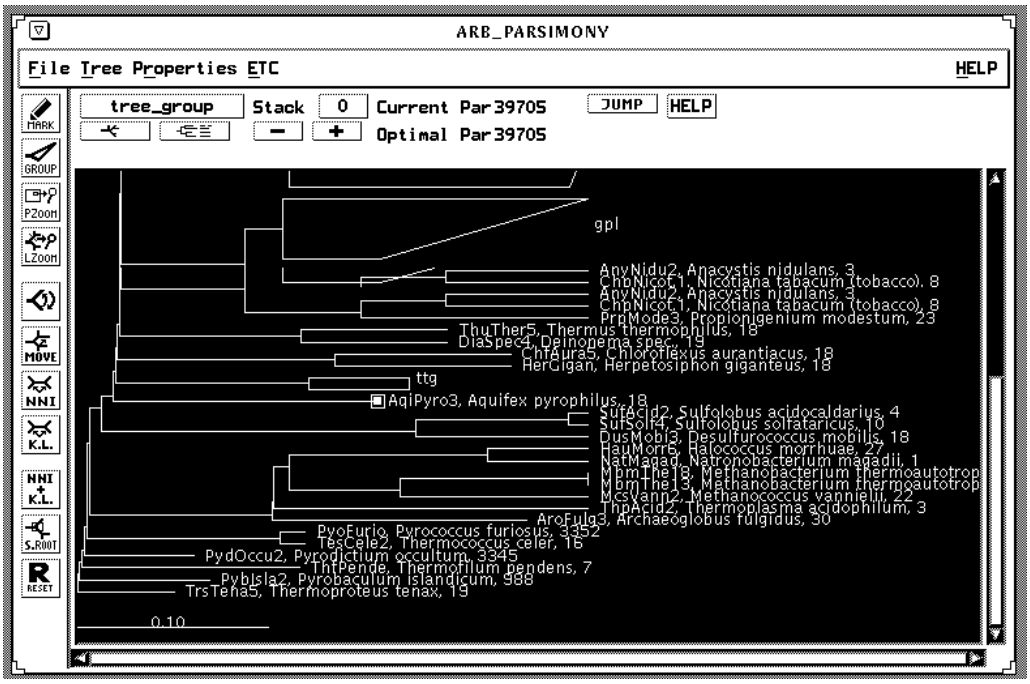


Abb. 117 Neue Vorgehensweise: Screenshot das ARB Hauptfensters.

Anwendungen

Die Hauptanwendung der Phylogenie als zentrales Objekt ist natürlich die Darstellung des Stammbaumes. Über diese kanonische Anwendung hinaus kann man einen Stammbaum zum einen als Ausgabe–, zum anderen als Eingabeobjekt sehen. Der Baum ist das Ausgabeobjekt, wenn sich Ergebnisse beliebiger Algorithmen mit Hilfe dieses Baumes im phylogenetischen Kontext betrachten lassen. Er ist aber das Eingabeobjekt, wenn ein Anwender phylogenetische Gruppen von Lebewesen selektieren möchte. Einige Möglichkeiten, mit dem Baum zu arbeiten, sind in der folgenden Abbildung schematisch dargestellt und sollen im folgenden kurz erläutert werden.

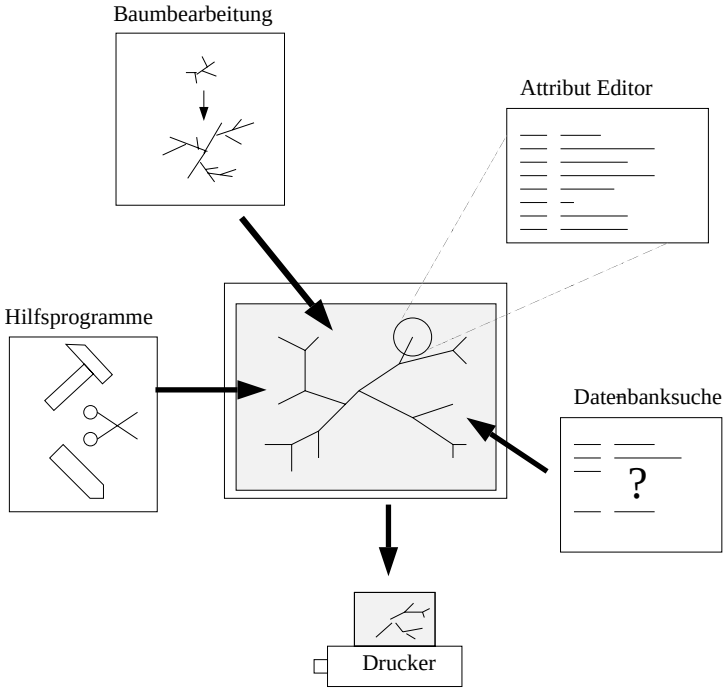


Abb. 118 Phylogenie als die zentrale Bearbeitungsstelle

Der Baum als Eingabeobjekt ist die Grundlage für die phylogenetische Selektion von Organismengruppen.

- Die Sequenzen einer Gruppe von Organismen, die phylogenetisch verwandt sind, sollen neu alignt werden. Mit Hilfe eines Baumes kann der Anwender sicher sein, keine Organismen innerhalb der Gruppe versehentlich zu vergessen, wie es sonst leicht passieren kann.
- Für eine Gruppe von phylogenetisch verwandten Organismen soll ein charakteristisches Sequenzmerkmal gefunden werden. In diesem Fall ist es unbedingt notwendig, genau eine vollständige phylogenetische Gruppe zu markieren. Das Vergessen von Gruppenmitgliedern würde dazu führen, daß der verwendete Berechnungsalgorithmus nicht in der Lage wäre, Lösungen zu finden.

Auch der Baum als Ausgabeobjekt wird häufig für die folgenden Anwendungen benötigt.

- Für eine vom Benutzer vorgegebene Sequenz sollen die nächsten Nachbarn angezeigt werden. Mit Hilfe eines Stammbaumes läßt sich die Sequenz phylogenetisch schnell einordnen und somit die nächsten Verwandten ausgemacht werden.
- Morphologische Merkmale, zum Beispiel die Eigenschaft der Sporenbildung, sollen im Stammbaum angezeigt werden. So kann man einen Eindruck erhalten, wann im Laufe der Evolution dieses Merkmal hervorgebracht wurde. Unter Umständen ist es somit auch möglich, bekannte Eigenschaften von Organismen auf noch unbekannte Organismen zu extrapolieren. Besitzen z.B. 20 aus einer Gruppe von 100 Organismen, eine bestimmte Eigenschaft und die restlichen 80 wurden noch nicht untersucht, so kann mit einer gewissen Wahrscheinlichkeit diese Eigenschaft bei allen Gruppenmitgliedern angenommen werden.

- Beliebige Ergebnisse von Analysealgorithmen lassen sich einfach im Stammbaum anzeigen. So erhält man schnell einen Eindruck des evolutorischen Spielraums innerhalb einer gegebenen Gruppe:

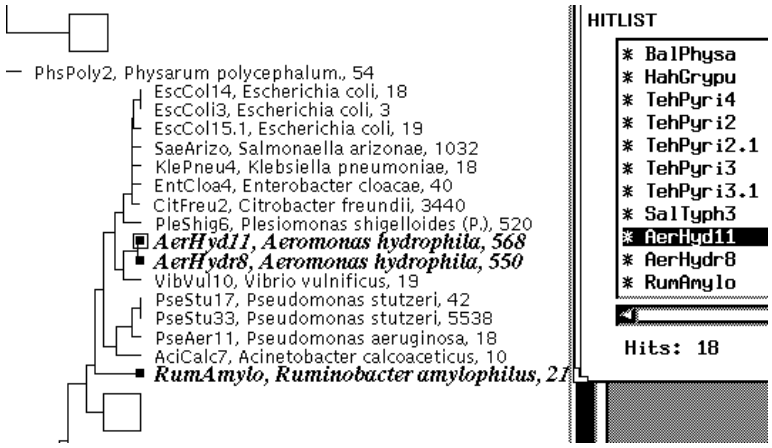


Abb. 119 Anzeigen des Ergebnisses (markierte Elemente) einer Datenbanksuche

Implementierung

Es gibt vielerlei Möglichkeiten, phylogenetische Bäume zu speichern. Für die Implementierung bei ARB war eine möglichst einfache und effiziente Durchführung vorrangig. Deshalb werden nur Binärbäume zugelassen, bei denen den Verbindungen zwischen den Knoten Längen zugeordnet werden. Durch Setzen einer solchen Länge auf den Wert 0 kann man optisch n-äre Bäume simulieren. Da in den meisten Fällen der gesamte Baum zur Anzeige benötigt wird, wird dieser als Einheit in der Datenbank gespeichert. Daraus folgt, daß lokale Änderungen im Baum immer dazu führen, daß der ganze Baum gespeichert werden muß. Aufgrund dieser einfachen Vorgehensweise konnten höchst effiziente Algorithmen zur Speicherung der Bäume entwickelt werden, das Laden eines 10000–Organismen großen Baumes dauert bei ARB weniger als eine 1/5 Sekunde (Pentium Pro 180).

Diskussion

Diese neuartige Vorgehensweise, nicht die Sequenzen, sondern den Baum in den Mittelpunkt der Betrachtung zu stellen, bietet ungeahnte Vorteile, für die jedoch leider auch einige gravierende Nachteile hingenommen werden müssen:

- Die Benutzer haben ihre Alignment-Daten nicht mehr während der gesamten Sitzung vor Augen. Sie sind daher gezwungen, sich auf eine höhere Abstraktionsebene einzustellen. Insbesondere für Personen, deren Abstraktionsvermögen nicht besonders hoch ausgebildet ist, stellt dies ein nicht zu unterschätzendes Problem dar.
- Da die Sequenzen zunächst einmal als Rohsequenzen vorliegen, sie also noch nicht alignt wurden, kann in diesem Rohzustand auch noch kein Stammbaum generiert werden. Infolgedessen erscheinen neue Sequenzen vorerst nicht auf dem Bildschirm. Der Anwender muß also hinnehmen, daß er seine Daten nicht sehen kann obwohl sie vorhanden sind. Um sie zu visualisieren, muß er sich einer traditionellen Datenbanksuche bedienen.

- Ohne Phylogenie gibt es bei ARB kein sinnvolles Arbeiten. Da die Phylogenie bei ARB im Mittelpunkt steht, wird jeder Anwender dazu gezwungen, sich damit auseinanderzusetzen, auch wenn sein Problem auf ganz anderen Themengebieten angesiedelt ist.
- Es kann durchaus dazu kommen, daß die Phylogenie bei der Analyse der Sequenzen überbewertet wird. Dies ist vor allem dann der Fall, wenn ein nicht unerheblicher Teil der gesamten Analysezeit für das Berechnen der Phylogenie gebraucht wird.

Diesen Nachteilen steht jedoch eine ganze Reihe gewichtiger Vorteile entgegen, die das gewählte Vorgehen mehr als rechtfertigen. Insbesondere mit steigender Sequenzanzahl, wie bei der ribosomalen RNS, fallen folgende Vorteile immer stärker ins Gewicht:

- Auch große Sequenzmengen lassen sich einfach verwalten. Wie bei einem Dateisystem, in dem Ordner angelegt werden, werden die Sequenzen mit Hilfe des Stammbaumes ihren natürlichen Gruppen zugeordnet. Da die Namen dieser Gruppen weltweit einheitlich gebraucht werden, können Sequenzen innerhalb kürzester Zeit auch in fremden Datenbanken wiedergefunden werden.
- Die Interpretation beliebiger Sequenzanalysen kann nun im Kontext der Phylogenie erfolgen. Erst so können neue Zusammenhänge bei der Analyse von Organismen gefunden werden. Die Fülle der Anwendungsfälle ist noch nicht überschaubar und die vollständige Verknüpfung aller Daten mit dem Stammbaum wird wohl eines der entscheidenden Themen der Zukunft sein. Auf verschiedenen inteRNStionalen Konferenzen und Workshops wird zur Zeit die Frage behandelt, wie es in kurzer Zeit bewerkstelligt werden könnte, existierende relationale Datenbanken um einen Stammbaum zu erweitern, um ein neues Instrument bei der Interpretation von Analysedaten zu bekommen.
- Die Editierung der Sequenzen wird effizienter. Dadurch, daß beim Starten eines Sequenzeditors die Sequenzen in der Reihenfolge erscheinen, in der sie auch im Stammbaum angeordnet sind, kann man mit einem Blick die Unterschiede zwischen einer neuen Sequenz und ihren nächsten Verwandten erkennen. Der Vorgang des Alignens, der Kontrolle des Alignments und die Suche nach Bereichen mit Sequenzierfehlern werden weitestgehend vereinfacht. Außerdem entfällt das mühsame Verschieben der Sequenzen von Hand.
- Durch konsequenten Verwendung des Stammbaumes wird der Einsatz einer ganzen Reihe von Algorithmen möglich. So werden zum Beispiel alle Algorithmen, die die Eigenschaften einer Organismengruppe mit den Eigenschaften einer anderen vergleichen, erst durch die Möglichkeit der einfachen Selektion phylogenetische Gruppen einsetzbar.

Zusammenfassend kann gesagt werden, daß durch dieses neue Konzept die Bedienung des Programmes auf den ersten Blick erschwert wurde, da sich die Anwender von Anfang an mit der Phylogenie beschäftigen müssen. Gerade dadurch jedoch, daß sie zu diesem methodischen Vorgehen gezwungen werden, arbeiten sie auf lange Sicht effizienter und strukturierter und sind mit den Ergebnissen zufriedener.

Ausblick

Als dieses Konzept 1993 implementiert wurde, war dies eine Entscheidung, die der weiteren Entwicklung weit voraus griff. Als Zielgruppe wurde damals auch nicht der inteRNStionale Markt anvisiert, es wurde ausschließlich für den an der TU München arbeitenden Experten der Phylogenie, W. Ludwig, entworfen. Die Anzahl der bekannten ribosomalen Sequenzen stieg jedoch seitdem exponentiell an, so daß seither alle Arbeitsgruppen gezwungen wurden, sich mit dem Problem auseinanderzusetzen, wie diese Datenmengen verarbeitet werden sollten. So versuchten die Arbeisgruppen des ALE Projektes zum Beispiel, die Phylogenie nachträglich in ihren Editor zu integrieren. Außerdem gab es in den USA Versuche, eine integrierte Datenbank zu schaffen, die die Phylogenie mit berücksichtigt. Bergeys Manual, ein wichtiges Standardwerk der Mikrobiologie in Buchform, sollte anhand der Phylogenie in Kapitel und Unterkapitel aufgeteilt werden.

Langfristig führt kein Weg an der Phylogenie als Klassifikationsinstrument vorbei. Gerade auf diesem Gebiet hat ARB einen bis jetzt unerreichten Vorsprung. Dieser Vorsprung drückt sich beispielsweise dadurch aus, daß das ALE Projekt gestoppt und durch ARB ersetzt wurde. Auch setzen weltweit immer mehr Labors ARB als das zentrale Programm zur Verwaltung der Sequenzen ein. Weltweite Resonanz und inteRNStionaler Zuspruch ist ein untrügerisches Indiz für die Akzeptanz von ARB.

In dieser Zwischenbilanz zum Projekt ARB kann heute im Rückblick eindeutig davon ausgegangen werden, daß die Entscheidung, die Phylogenie so hoch zu bewerten, der wichtigste Grundstein des Projektes war.

X.1.2 Das Datenmodell

Das Grundmodell von ARB

Mit ARB sollte ein Datenmodell etabliert werden, das die traditionellen Beschränkungen existierender Software vermied. Traditionell wurde für jeden Datentyp eine eigene Datei generiert. Jedes Softwaremodul, das Daten aus dieser Datei lesen wollte, mußte diese Datei in den Speicher lesen, die benötigten Informationen extrahieren und unter Umständen eine neue Datei schreiben. Alle Information, die nicht gelesen werden konnten, wurden somit auch nicht geschrieben und waren somit für die weitere Analyse verloren.

Dieses Viel-Dateien-System schien nicht besonders vielversprechend. Daher wurde ein Datenmodell generiert, das es erlaubte, alle Informationen in einer einzigen Datei abzuspeichern. Damit der Programmieraufwand bei der Extraktion der Daten niedrig bleibt, mußten diese Informationen in einer möglichst einfachen Struktur abgelegt werden, die dann aber noch so flexibel sein sollte, daß auch zukünftige Anforderungen keine grundlegende Änderung des Grundmodells zur Folge hätten.

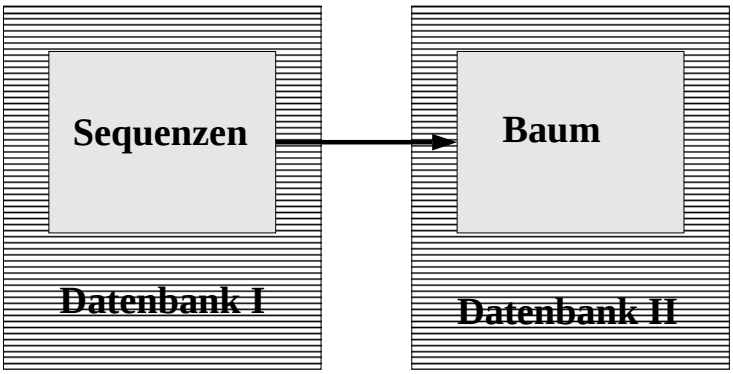


Abb. 120 Traditionelle Vorgehensweise mit unabhängigen Dateien

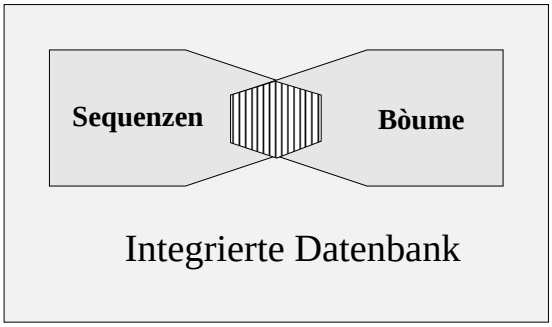


Abb. 121 Integrierte Datenbank für alle auftretenden Datentypen

Das für ARB entwickelte Datenmodell entstand evolutorisch; bei jeder neuen Anforderung wurde das jeweils aktuelle Modell unter Umständen entsprechend modifiziert. So gelang es, ein sowohl einfaches, als auch praktikables Modell zu entwickeln.

Der Grundstein dieses neuen Datenmodelles wurde bereits 1992 gelegt. Damals wurde als Ziel formuliert, eine Schnittstelle zwischen dem Prototypen von ARB und dem Sequenzeditor **edtu** [Diplomarbeit S. Grumann] zu schaffen. Da diese Schnittstelle möglichst flexibel gestalten werden sollte, wurde folgendes Datenmodell entwickelt:

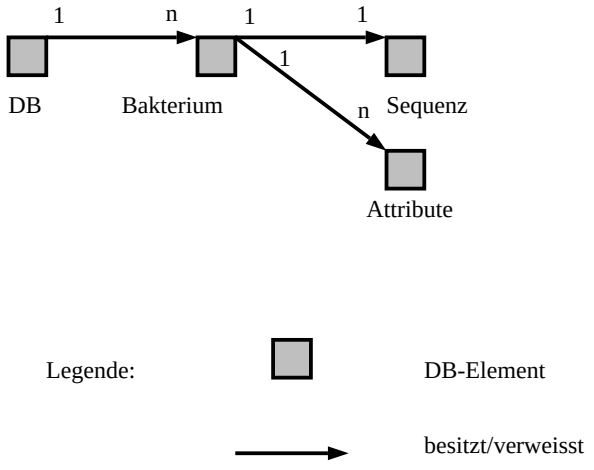


Abb. 122 Prototype des ARB Datenmodells

Die Grundidee dieses in obiger Abbildung schematisch dargestellten ersten Datenmodells war es, daß die gesamte zu speichernde Information als Attribute eines Organismus gesehen werden. Selbst die Datensätze jedes Organismuses können als Attribute der Datenbank angesehen werden. Es wurden also zwei Arten von Attributen generiert:

Attribute Type	Bemerkung
Terminal	Ein solches Objekt kann einen beliebigen Wert aufnehmen, sei es eine beliebige Zeichenkette, eine Zahl oder ein Vektor aus Zahlen.
Non-Terminal/ Ordner (Container)	Ein Objekt dieses Typs hat selber keinen Wert, es diente als Ordner für weitere Datenbankobjekte. Es ist mit einem Ordner eines Dateisystems vergleichbar.

Dies führte zu einer flexiblen hierarchischen Datenbank, die einem Dateisystem nicht unähnlich ist.

Diesem Grundprinzip ist ARB bis heute treu geblieben und ist dabei noch an keine ernstzunehmenden Grenzen gestoßen. Natürlich hat das Grundmodell diverse einfache Erweiterungen erhalten, im wesentlichen basiert das Datenmodell heute auf folgenden Grundobjekten:

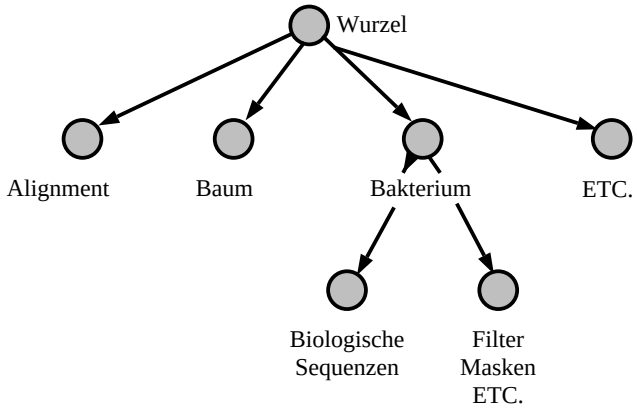


Abb. 123 Vereinfachte Übersicht über die Datenobjekte

In dem Datensatz müssen im wesentlichen Organismen mit ihren Sequenzen und Bäume gespeichert werden. Da viele Sequenzen in einem Datensatz keinen biologischen Ursprung haben, sondern mit Hilfe des Computers berechnet wurden (z.B. Konsensus, Filter, ...) soll zwischen echten Sequenzen und computergenerierten unterschieden werden. Jedem biologischen Organismus und jedem Filter sind eine oder mehrere Sequenzen verschiedener Gene zugeordnet. Alle Sequenzen eines Gens aller Organismen lassen sich virtuell zu einem Alignment zusammenfassen. Um für ein solches Alignment globale Informationen wie seine Länge, sein Typ und seine Herkunft zu speichern, wird ein globales Alignmentobjekt benötigt. Dieses speichert ausschließlich globale Information.

Jedem dieser Objekte können beliebig viele weitere Attribute zugeordnet werden, sei es Name eines Organismus, Entdecker, Publikation, Lebensraum o. ä.

Alle diese Objekte sind nun in einer Hierarchie angeordnet, wobei durchaus symbolische Verweise zu beliebig vielen anderen Objekten erlaubt sind:

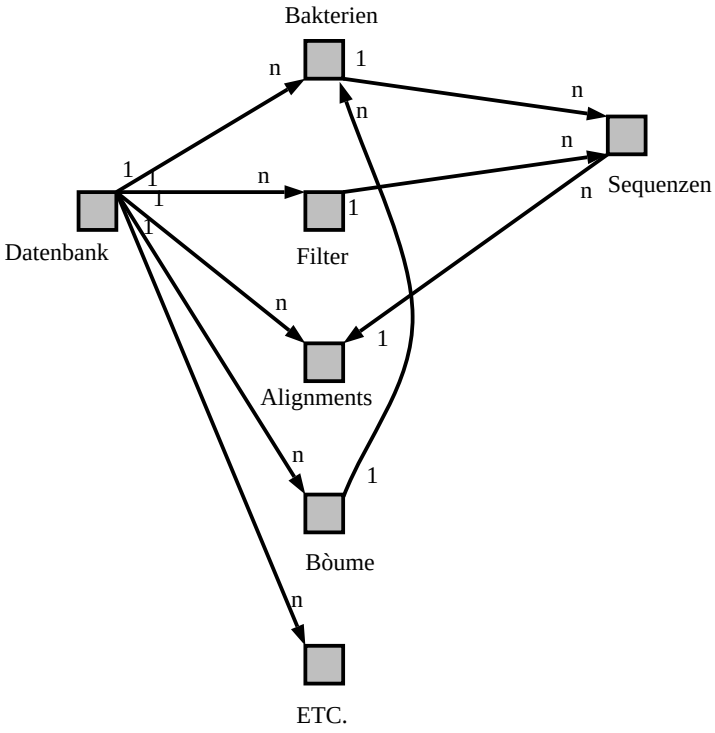


Abb. 124 Übersicht über das Datenmodell

In der Datenbank sind Sequenzen, Filter und Bäume gespeichert. Jeder Organismus oder Filter besitzt eine oder mehrere Sequenzen, die einem bestimmten Alignment angehören. Das Stammbaumobjekt beinhaltet nicht die zu seiner Berechnung verwendeten Sequenzen, sondern Verweise auf die entsprechenden Organismen. Um einen derartigen Verweis zu speichern, verwendet man den eindeutigen Kurznamen des Objektes, auf das verwiesen werden soll. Alle Informationen sind als Attribute zu den entsprechenden Objekten abgelegt. So besitzt zum Beispiel jeder Organismus ein Attribut **'name'**, das den eindeutigen Kurznamen beinhaltet sowie **'fullname'**, das den biologisch gültigen Namen aufnimmt. Hinzu kommen noch eine Vielzahl weiterer, keinem Objekt zugeordnete, Attribute, wie der Name des Datensatzes, der Name des gerade in Bearbeitung befindlichen Baumes, usw... .

Diskussion

Ein Datenmodell kann immer nur ein Kompromiß zwischen Einfachheit und Sicherheit sein. Je komplexer ein Datenmodell wird, desto besser lassen sich automatisierte Konsistenzüberwachungen einbauen. Für ARB lag die Priorität eindeutig bei Einfachheit und Flexibilität. Dies bedeutet zum Beispiel, daß von jedem beliebigen Modul aus, ungeachtet der Sinnhaftigkeit derartiger Aktionen, beliebige Attribute in der Datenbank generiert und verändert werden können. Es hat sich gezeigt, daß dieses System trotz dieser Offenheit, die auf den ersten Blick gefährlich scheint, von den Anwendern sehr sorgfältig und aufmerksam gehandhabt wurde. Im Normalfall wurde das Programmpaket zusammen mit allen

Datensätzen übernommen. Ausnahmslos wurde dabei die von W. Ludwig vorgegebenen Struktur akzeptiert.

Das Datenmodell hat bis heute seine Aufgabe voll und ganz erfüllt, insbesondere konnte durch seine Einfachheit eine neuartige Art der Datenbank realisiert werden, die zur dritten Säule des Erfolges von ARB wurde.

Vor einem halben Jahr konnte in den USA ein inteRNStionaler Expertenkreis gewonnen werden, um in intensiver Zusammenarbeit vor Ort ARB mit einer objektorientierten Datenbank zu verbinden (**ORDP**). Diese objektorientierte Datenbank befand sich zu jener Zeit gerade in Entwicklung; mit ihrer Hilfe sollten die gesammelten Daten der RDP verwaltet werden. Um ein neues Datenmodell zu entwickeln und zu implementieren, waren über eine halbe Million Dollar bereitgestellt worden. Das Arbeitspaket umfasste allerdings nicht die Programmierung der fehlenden Anwendungen. Die Entwicklung von ARB basierte auf dem Wunsch der Biologen, praktische Werkzeuge an die Hand zu bekommen, nicht darauf, daß Sequenzen mit Hilfe einer kommerziellen Datenbank gespeichert werden sollten. Gerade dadurch, daß sich beide Projekte komplementär ergänzten, war ihre Kombination ideal, zumal dabei beide Seiten ihre Modelle auf Unzulänglichkeiten testen konnten.

Die RDP hatte fast 2 Mannjahre in die Entwicklung eines Datenmodells investiert. Das Ergebnis war ein 5 seitiges Entity-Relationship-Diagramm, das zwar wohl alle bekannten Aspekte einer biologischen Datenbank berücksichtigte, jedoch schwer zu verstehen und für zukünftige Anforderungen zu unflexibel war. Insgesamt besaß dieses erste Modell einen zu hohen Komplexitätsgrad, um verwendbar zu sein. Daher wurde von der RDP eine stark vereinfachte Version entwickelt, die etwa nur ein Fünftel des ursprünglichen Umfanges ausmachte und wesentlichen flexibler war, jedoch keine Bäume berücksichtigte.

Um nun Daten zwischen beiden Datenbanken flexibel transportieren zu können, mußte eine neutrale Schnittstelle für die beiden Datenmodelle aus ARB und ORDP entwickelt werden:

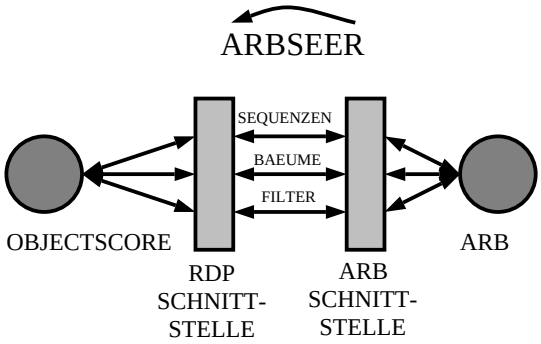


Abb. 125 Datenaustausch zwischen ORDP und ARB

Hierfür mußten die Daten dem einen Datenmodell entnommen, umstrukturiert und in das andere Datenmodell umgeschrieben werden. Diese Schnittstelle wurde ARBSEER getauft. Die notwendigen Änderungen und Anpassungen des Datenmodells fielen dabei auf Seiten von ARB deutlich geringer aus als auf der 'Gegenseite', allerdings wurden auch folgende Schwachstellen offengelegt:

- Jedem Organismus sind ausschließlich Attribute zugeordnet. Teilen sich zwei oder mehrere Organismen denselben Datenbankeintrag, z.B. den gleichen Autor,

muß dieser immer wieder als Kopie bei jedem Bakterium neu abgelegt werden. Diese unnötige Redundanz führt zu unnötigem Speicherverbrauch. Aus diesem Grunde wurde das ursprüngliche Datenmodell um einen weiteren Datenmodelltyp erweitert, den des symbolischen Links:

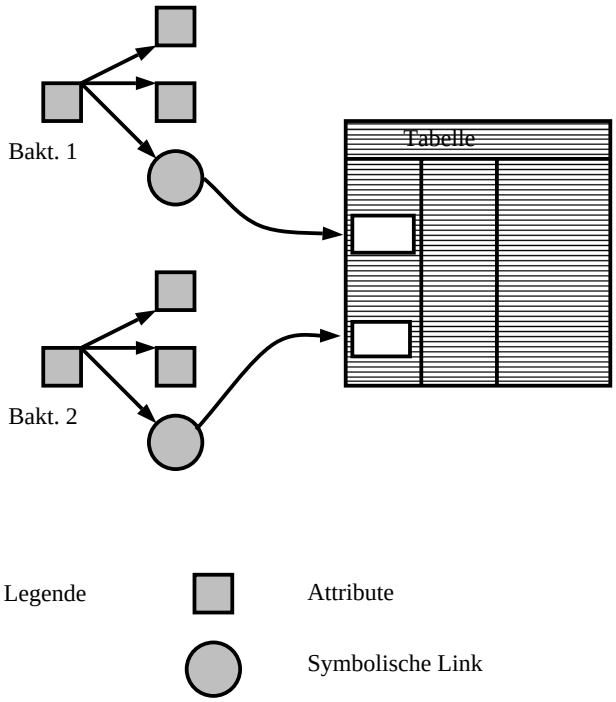


Abb. 126 Datenbank Link zur Vermeidung von Redundanz:

Ein Symbolischer Link hat selbst keinen Wert. Er zeigt auf eine bestimmte Zeile einer beliebigen Tabelle. Aus der Sicht des Anwenders ist er mit Datenbanktyp 'Ordner' vergleichbar. Es können dynamisch beliebige Typen von Links definiert werden, für die je ein eigenes Programm erstellt werden muß, das diesen Link auswertet. Die 'Tabelle' kann also sowohl Teil des Datenbestandes selbst sein wie auch eine räumlich entfernte Datenbank z.B im Ausland, die mit Hilfe des Internet abgefragt wird.

- In dem ARB Datenmodell wird jede Information als reale originäre Information interpretiert. Dies bedeutet, daß es komplexe Abhängigkeiten zwischen einzelnen Sequenzen gibt, die zwar gespeichert, aber nicht automatische getestet werden können. So kann zwar das Aminosäurealigment aus dem DNS Alignment berechnet, gespeichert und verarbeitet werden, bei Änderung des zugrundeliegenden DNS-Alignments wird jedoch das davon abhängige und redundante Proteinalignment nicht mitgeändert. Eine ideale biologische Datenbank sollte in der Lage sein, mit virtuellen Datensätzen zu arbeiten. Allerdings ist eine saubere Umsetzung dieses Konzeptes aufgrund fehlender Kapazitäten zur Zeit noch nicht in Angriff genommen. In den letzten Monaten wurden allerdings Schritte unternommen, deren Ziel es ist, die Finanzierung dieser Erweiterung des ARB Datenmodells sicherzustellen

Die Erfahrung hat gezeigt, daß Analyse-Algorithmen umso leichter implementiert werden können, je einfacher das Datenmodell ist. Jede biologische Datenbank sollte daher eine einfache Schnittstelle bieten, die jegliche Komplexität der dahinterliegenden Struktur versteckt. Je direkter die Umsetzung zwischen Schnittstelle und realem Datenmodell ist, desto effizienter wird die Datenbank arbeiten. So entstand für ARB ein einfaches, flexibles und effizientes Datenbankschema, das sein Ziel, beliebige Daten zu speichern und zu verwalten, fast optimal erreicht.

Zusammenfassung und Ausblick

Die ursprüngliche Aufgabe von ARB war es nicht, Daten zu verwalten, sondern Daten aus existierenden Datenbanken zu speichern und zu analysieren. So war auch ursprünglich der Aspekt der Konsistenzerhaltung der Daten nur von sekundärer Bedeutung. Da ein sehr flexibles Datenmodell eingesetzt wurde, begannen die Anwender mit der Zeit, ARB auch als primäre Datenbank zu nutzen, d.h. ihre Sequenzen im großen Umfang mit Hilfe von ARB zu verwalten. Per se stellt dieses Vorgehen kein Problem dar, allerdings gibt es keine Kontrollsysteme, die die Qualität der Datenbankeinträge garantierten. Eine Möglichkeit wäre, die ARB Datenbank entsprechend zu erweitern, allerdings zöge dies eine derartige Vielzahl von Änderungen im Gesamtsystem nach sich, so daß der Aufwand nicht unter 2 Mannjahren liegen würde. Ein anderer sinnvollerer Weg wäre es, ein Programm zu entwickeln, das regelmäßig die gesamte Datenbank durcharbeitet und Inkonsistenzen meldet bzw. korrigiert.

Das andere große Problem des Datenmodells von ARB ist es, nur unzureichend mit redundanten Daten umgehen zu können. Es gibt wohl noch keine mikrobiologische Datenbank, die dieses Problem auch nur ansatzweise löst. Da die Datenbank von ARB im wesentlichen aus den Datentypen Ordner, Link und Terminal besteht, dürfte es prinzipiell nicht allzu kompliziert sein, diese auf virtuelle Protein-, oder zusammengesetzte Sequenzen zu erweitern. Mittels dieser Vorgehensweise sollten innerhalb von 1.5 Jahren brauchbare Ergebnisse vorliegen.

X.1.3 Das Datenbanksystem

Der automatische, kontrollierte Zugriff auf Daten geschieht normalerweise mit Hilfe eines Datenbanksystems. Wir hatten als Ziel, das Datenbanksystem auch dafür zu nutzen, die verschiedenen Teilmodule zu vernetzen und zu synchronisieren. So entwickelte der Autor vor Beginn dieser Arbeit ein ARB-Datenbanksystem, das aufgrund seiner höchst effizienten inneren Struktur eine vollkommen neue Art von Datenbankapplikationen ermöglicht. Dieses Datenbanksystem wurde auch für diese Arbeit genutzt, dabei lesen entsprechende Programme die Datenbank nicht beim Programmstart, sondern immer dann und nur dann, wenn die Daten tatsächlich benötigt werden. So werden die in einem Sequenzeditor angezeigten Sequenzen nur dann aus der Datenbank gelesen, wenn diese auf dem Bildschirm gezeichnet werden.

X.1

Die Idee

In traditionellen Datenbanken werden mit Hilfe einer Datenbank-Abfragesprache die benötigten Informationen in der Datenbank gesucht und ausgelesen. Im Normalfall hat dann die gelesene Information keine Verbindung zur Datenbank mehr. Änderungen der gelesenen Datensätze, die währenddessen in der Datenbank von anderen Modulen durchgeführt werden, können nur sehr begrenzt mitverfolgt werden:

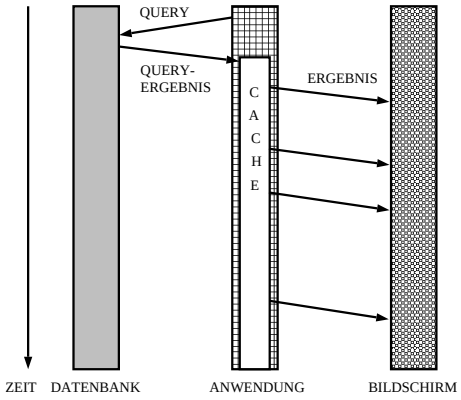


Abb. 127 Traditionelle Datenbankabfrage

Die Ergebnisse einer Datenbank-Abfrage (**Query**) werden bei herkömlichen Systemen in einem lokalen Speicher der Anwendung zwischengespeichert (**Cache)** und nur bei Bedarf auf dem Bildschirm dargestellt.

Die Idee bei der Entwicklung der Datenbank für ARB war, eine Datenbank zu schaffen, die so schnell arbeitet, daß es möglich wird, die Daten zwar mit Hilfe einer Datenbank-Abfragesprache zu suchen, als Ergebnis aber nur eine Referenz auf den gefundenen Datenbankeintrag auszugeben. Erst zum spätestmöglichen Zeitpunkt wird dann mit Hilfe dieser Referenz der Inhalt der Datenbank gelesen und am Bildschirm dargestellt:

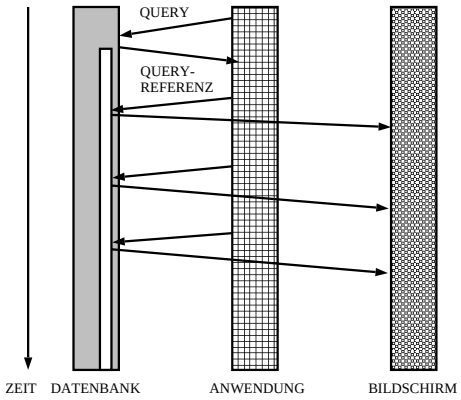


Abb. 128 Datenbankabfrage mit Referenz als Ergebnis

Zusätzlich kann nun jedes referenzierte Datenbankobjekt ’beobachtet’ werden. Dazu kann sich der Anwendungsprozeß als ’Listener’ des referenzierten Objektes eintragen. Ändert sich dieses Datenbankobjekt, werden alle angeschlossenen ’Listener’ davon informiert. Ist

der Typ des Datenbankfeldes ein Ordner, werden die Listener auch dann informiert, wenn eines der Inhalte (also Arbeitsmappen und Datenblätter bzw. 'Kinder', 'Enkel' ...) sich geändert hat.

Auf den ersten Blick scheint diese Art mit Datenbank-Abfragen umzugehen, nicht sehr innovativ zu sein. Tatsächlich bedeutet jedoch eine konsequente Umsetzung dieser Idee eine ganz neue Art der Programmierung. Viele Probleme, die für jede Anwendung einzeln gelöst werden müßten, lassen sich hier innerhalb der Datenbank zentral realisieren, seien es Transaktionen, verteilte Anwendungen, Undo-Funktionen oder Parallelisierung.

Anwendungen

Jedes Programmodul, mit dessen Hilfe eine Information gespeichert werden soll, arbeitet nun nicht in seinem lokalen Speicher, sondern schreibt diese Information in die Datenbank. Die Datenbank bietet in diesem Fall die Möglichkeit, temporäre Daten abzulegen, die bei Beendigung des Programmes gelöscht werden. Jedes Programmodul behält Referenzen auf die Datenbankeinträge, um später schnell wieder darauf zuzugreifen. Weiterhin beobachtet jedes Modul mit Hilfe sogenannter Triggerfunktionen Veränderungen der Datenbankeinträge. Dieses Vorgehen führt dazu, daß viele Probleme der einzelnen Module zentral gelöst werden können:

- **Atomität**: Die Datenbank garantiert, daß nur konsistente Zustände gespeichert werden. Um Datenbankeinträge lesen oder verändern zu können, muß jedes Programm eine Datenbanksitzung (**Transaktion**) öffnen, die Daten bearbeiten und die Transaktion wieder schließen. Das Gesamtsystem kann dann dafür sorgen, daß entweder alle oder keine Änderung in der Datenbank dauerhaft gespeichert werden.
- **Verteilte Module:** Da eine Datenbank auch über Prozeßgrenzen die Datenspeicherung und Organisation übernehmen kann, ist es möglich, das Programm in kleine, relativ unabhängige Module aufzuteilen. Die Datenbank übernimmt dann die **Synchronisation** zwischen diesen Modulen. Dies soll an einem Beispiel verdeutlicht werden: Ein Programmierer erstellt ein Programm, das für eine gegebene Spalte im Alignment eine Sequenzstatistik erstellt, ein anderer erstellt einen einfachen Sequenzeditor:

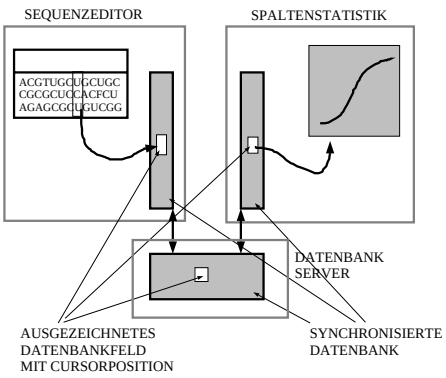


Abb. 129 Zusammenspiel eines Editors mit einer Spalten-Statistik

Dieser Sequenzeditor wird nun so programmiert, daß er die Position des Cursors in ein bestimmtes Datenbankfeld speichert. Das Modul 'Sequenzstatistik' beobachtet nun dieses Datenbankfeld und jedesmal, wenn der Benutzer den Cursor auf eine neue Spalte setzt, bekommt er für diese neue Spalte eine eben berechnete Statistik angezeigt. Die Implementierung dieses Verhaltens beschränkt sich auf etwa 20 Zeilen Programmcode, bei konventioneller Programmierung wäre diese Menge mindestens um den Faktor 50 überschritten worden.

- **Sichere Programmierung:** Läuft die Datenbank in einem anderen Prozeß als die Analysefunktionen, kann eine Fehlfunktion eines dieser Module nicht die Konsistenz der Daten gefährden. Dies ist besonders dann wichtig, wenn mehrere noch programmierunerfahrene Studenten des öfteren unsichere und häufig abstürzende Programme implementieren. Bei den unvermeidbaren Zusammenbrüchen des Sequenzeditors lag dank der gewählten Datenbankstruktur der aufgetretene Datenverlust jeweils unter einem Zeichen.

- **Skalierbarkeit:** Da die Daten nicht in einem lokalen Speicher, sondern effizient von einer Datenbank verwaltet werden, können die Anwendungen auch mit großen Datenmengen effizient arbeiten. So ist es zum Beispiel mit dem Sequenzeditor ARB_EDIT4 möglich, 180 MegaByte Sequenzdaten auf einmal auf einem leistungsfähigeren PC (64 megaByte Speicher) zu bearbeiten.

- **Parallelisierung:** Die Datenbank kann als eine Art virtueller verteilter Hauptspeicher gesehen werden, der dazu dienen kann, Algorithmen einfach zu parallelisieren. Spezielle Datenbankeinträge dienen dann der Synchronisation und Steuerung der auf verschiedenen Rechnern ablaufenden Rechenprozesse:

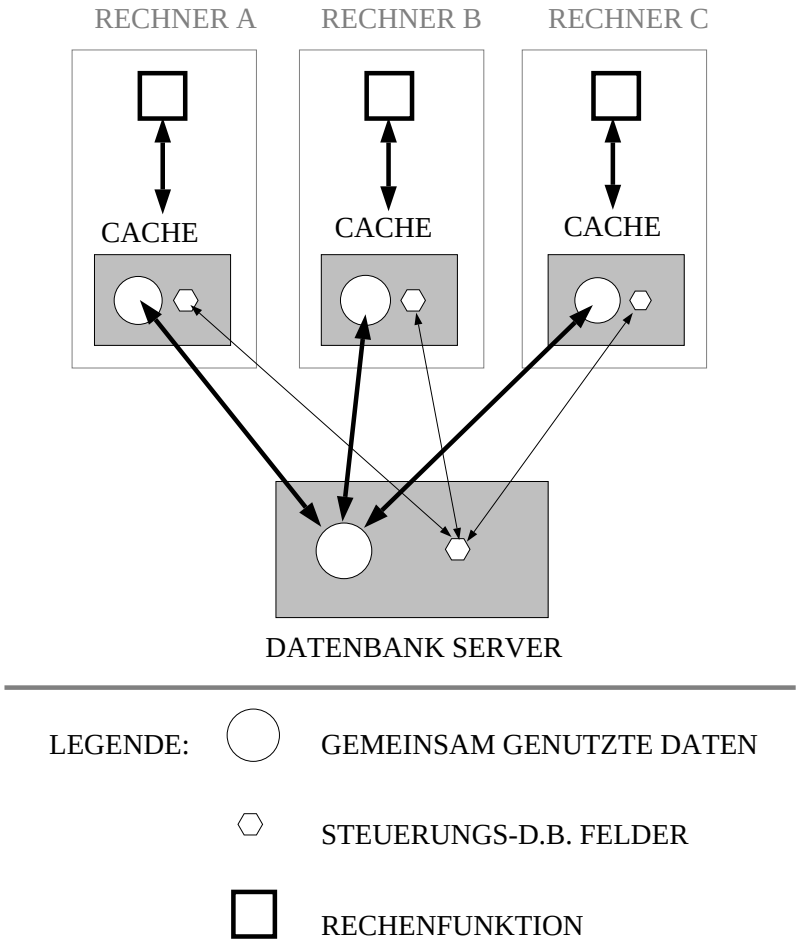


Abb. 130 Einfache Parallelisierung mit Hilfe einer effizienten Datenbank:

Ob es sinnvoll ist, eine Datenbank zur Parallelisierung von Algorithmen einzusetzen, hängt davon ab, wie effizient, oder, anders ausgedrückt, wieviel Computer-Ressourcen die Datenbank benötigt im Vergleich zum Ressourcenverbrauch der restlichen Algorithmen. Entsprechende Leistungsmessungen sind unter dem folgenden Unterkapitel ’Umsetzung’ aufgelistet.

- **UNDO:** Für ein professionelles Arbeiten ist für den Benutzer die Möglichkeit zur Rückgängigmachung möglichst aller Funktionen wünschenswert. Mit Hilfe der ARB-Datenbank kann schrittweise zu einem älteren Datenbankzuständen zurückgesprungen werden. Außerdem erlaubt die ARB-Datenbank das Rückgängigmachen des Rückgängigmachens (REDO).
- **Kompression:** Betrachtet man die Inhalte einer Sequenzdatenbank genauer, stellt man fest, daß die meisten Daten aus ’.’ und ’-’ Zeichen bestehen. Um diese Redundanz optimierter zu speichern, wurde in die ARB Datenbank eine Kompression auf Datenbankfeldebene eingebaut. Dies bedeutet, daß jeder Datenbankeintrag in komprimierter Form auf der Festplatte bzw. im Hauptspeicher liegt. Dadurch wird im Mittel ein Kompressionsfaktor von 1:3 erreicht. Durch Entwicklung einer speziellen und komplexen Kompressionstechnik konnte dieser Wert im letzten Jahr auf 1:20 verbessert werden. Hinzu kam, daß die Datenbankfelder aufgrund der eingesetzten Kompression so kompakt gespeichert werden können, daß der

Datenbank-Overhead etwa 70% des gesamten Speicherbedarfs ausmachte, so daß die Kompression an einem praktischen Optimum angelangt war.

Obwohl das Problemfeld Datenbanken nicht neu ist, ist diese Art der Verknüpfung von Datenbank und Algorithmen nicht nur im Bereich der Biologie revolutionär. Selbst einfache Algorithmen können somit ohne viel Aufwand zu einem Gesamtsystem integriert werden.

Typisches Anwenderverhalten

Im Laufe der Zeit entstand eine Datenbank, deren Implementierung mit der Implementierung traditioneller Datenbanksysteme kaum mehr Gemeinsamkeiten aufweist. Sie orientiert sich weniger an conventionellen Datenbankalgorithmen als vielmehr daran, möglichst effizient und für einen Anwender transparent die benötigten Informationen zusammenzustellen. Es hatten sich im Laufe der Zeit folgende typische Anwendungsfälle herauskristallisiert:

- Anwender, denen nur daran gelegen ist, ihre eigenen Sequenzen zu verrechnen, veränderen in der Datenbank im Mittel etwa 10000–30000 Datenbankfelder. Dabei ändern sie bei jeder Arbeitssitzung im wesentlichen die gleichen. Die Gesamtmenge aller Änderungen übersteigt selten die Grenze von 1 MegaByte.
- Administratoren hingegen veränderen häufig nahezu 10% aller Datenbankfelder.

Daher arbeitet jeder Einzelbenutzer einer zentralen Datenbank mit seiner privaten Kopie der Datenbankinhalte, die regelmäßig auf den aktuellen Stand gebracht wird.

Organisation

Die oben erläuterte Vorgehensweise bei der Strukturierung der ARB-Datenbank führte zu folgendem Vorgehen: Zu Beginn jeder Sitzung wird die gesamte Datenbank in den Hauptspeicher geladen. Bei einer Datenbankgröße von 180 MegaByte (10 MegaByte komprimiert) dauert der Ladevorgang selten mehr als 3–4 Sekunden, eine Wartezeit, die einem Anwender durchaus zugemutet werden kann. Dieses Vorgehen wird selbst bei steigendem Datenaufkommen nicht an seine Grenzen stoßen, da die Größe des Hauptspeichers, der für die wichtigsten Analysealgorithmen benötigt wird, in etwa dem 2–5 fachen der Datenbank entspricht.[1] Dieses Vorgehen führt dazu, daß die Datenbank in einem wesentlich effizienteren Format gespeichert werden kann, da auf die Speicherung von Strukturinformationen weitestgehend verzichtet werden konnte.

Aufgrund der sehr hohen Hauptspeicheranforderung des gesamten Systems kann man davon ausgehen, daß selten benötigte Teile des Speichers vom Betriebssystem auf Festplatte ausgelagert (ausgeswapped) werden. Um nun auch bei ausgelagertem Hauptspeicher hohe Geschwindigkeiten zu erhalten, wird die Datenbank nicht einfach unstruktiert im Hauptspeicher organisiert, sondern in sogenannte Segmente unterteilt. Das Ziel dabei ist, Informationen, auf die mit hoher Wahrscheinlichkeit hintereinander zugegriffen werden wird, im selben Segment zu speichern. Um die Zugriffe auf den Hauptspeicher zu optimieren, wird dieser in Einheiten (Kacheln) aufgeteilt, und zwar so, daß jede Kachel vom Betriebssystem genau auf einer physikalischen Festplatteneinheit (Sektor) gespeichert wird:

[1] Für Parsimony beträgt die nötige Größe des Hauptspeichers mindestens Sequenzlänge mal Sequenzanzahl, d.h. bei 180 MegaByte Sequenzdaten werden 180 MegaByte RAM benötigt, während die dazugehörige optimierte Datenbank lediglich 20 MegaByte beansprucht.

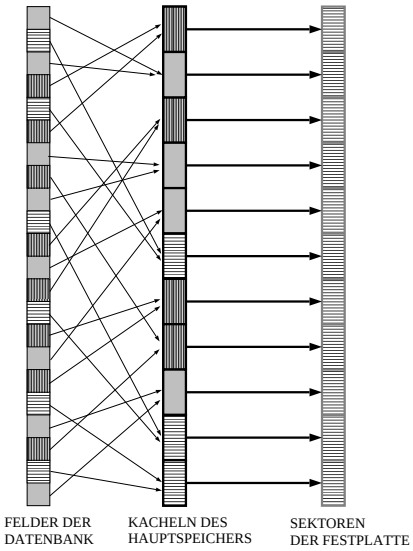


Abb. 131 Segmentierung des Hauptspeichers für effizienten Zugriff

Die Datenbankfelder, die zu einem Segment gehören, werden in für dieses Segment reservierten Kacheln gespeichert.

Speicherformat

Ausgehend von den beiden typischen oben erwähnten Anwendungsfällen wurde folgende Strategie bei der Speicherung der Datenbank entwickelt:

- Zur Speicherung weniger Änderungen, d.h. bis maximal 30000 geänderte Datenbankfelder, wird eine zusätzliche Datei angelegt, in der nur die Änderungen gespeichert werden. Beim Start einer Datenbanksitzung wird die Grunddatenbank geladen und nach der neuesten Änderungsdatei gesucht, dann werden diese Änderungen durchgeführt. Dieses Vorgehen erlaubt, mehr als eine Änderungsdatei zu verwenden. In der jetzigen (1.1.99) Implementierung werden die letzten 5 Änderungsdateien nicht gelöscht. Der Benutzer hat somit die Möglichkeit, die Datenbankzustände zur Zeit der letzten fünf Speichervorgänge zu inspizieren.
- Bei Änderungen größerer Bereiche der Datenbank erscheint es sinnvoll, die gesamte Datenbank komplett neu zu schreiben.

Es gibt durchaus Anwendungen, bei denen eine traditionelle Datenbank der ARB-Datenbank in Bezug auf Skalierbarkeit bei großen Datenmengen überlegen ist. Um dieses Problem etwas zu reduzieren, wurde die ARB Datenbank um das Konzept der **FastLoad**–Datei erweitert. In diesem Falle kann aus einer im Hauptspeicher gehaltenen Datenbank eine traditionelle Datenbank-Datei erstellt werden, auf die zum Lesen von Datensätzen zugegriffen wirden, ohne diese vollständig in den Hauptspeicher laden zu müssen. Diese Datei ist im wesentlichen genauso aufgebaut wie eine in den Hauptspeicher geladene Datenbank, so daß mit denselben Algorithmen darauf zugegriffen werden kann. Da sie alle Strukturinformationen der Datenbank enthält, ist diese Datei jedoch etwa um den Faktor 1.8 größer als die ursprüngliche Datenbank-Datei. Insbesondere auf Rechnern mit wenig Speicher bringt dieses Vorgehen deutlich beschleunigte Ladezeiten mit sich.

X.1

Cache-Strategien

In einer typischen Datensitzung werden am Anfang einige Datenbankelemente gesucht, auf die dann im weiteren Verlauf wiederholt lesend zugegriffen wird. Um die Zugriffszeit auf bereits einmal gelesene Datensätze zu reduzieren, wurde ein Cache-Algorithmus implementiert. Dieser hat die Aufgabe, zuletzt gelesene Daten zwischenzuspeichern, in der Annahme, daß diese in naher Zukunft nochmals verwendet werden und dann schnell aus diesem Zwischenspeicher gelesen werden können. Bei der ARB-Datenbank wurde im wesentlichen eine drei-stufige Cache-Strategie implementiert:

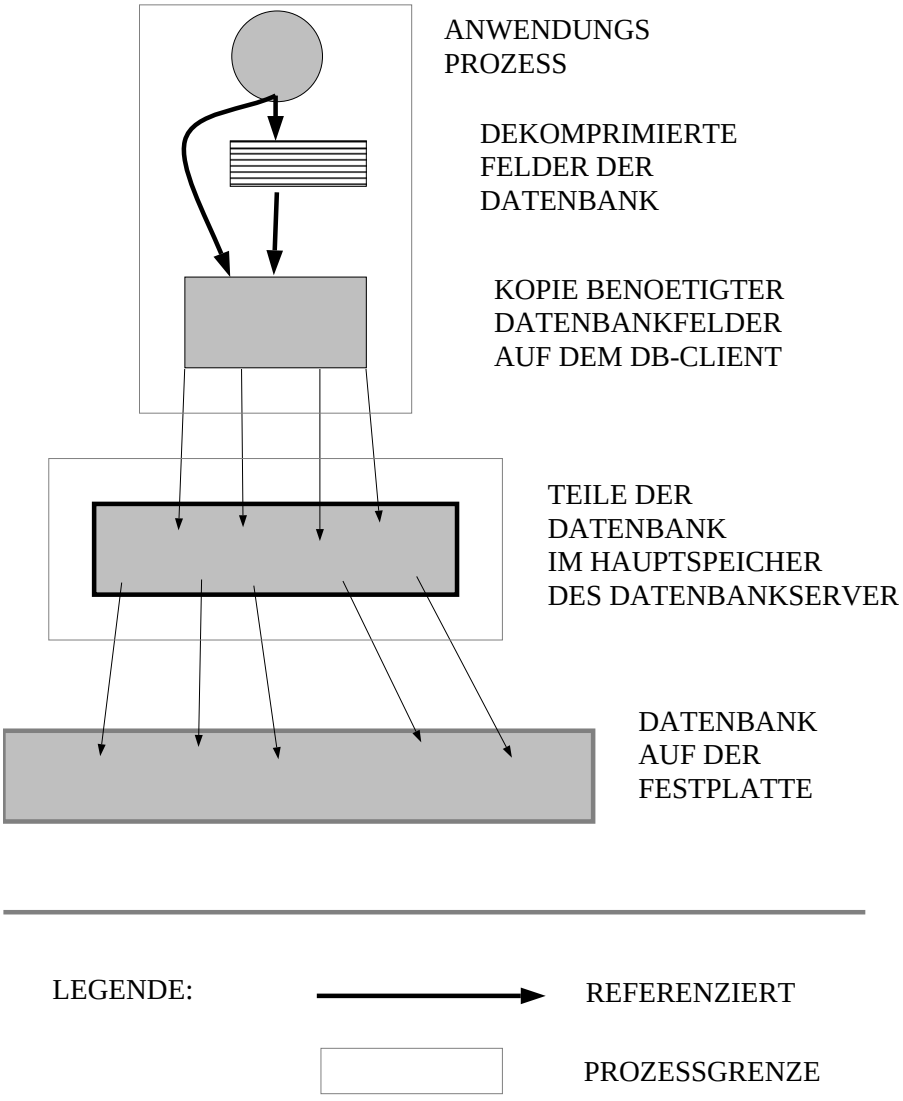


Abb. 132 Dreistufige Cache-Strategie der ARB Datenbank

1. Die von dem Datenbank-Server benötigten Felder werden in den Hauptspeicher zwischengespeichert. Dazu wird kein eigener Zwischenspeicherbereich definiert, sondern das Programm nutzt die Funktionen des Betriebssystems, das wesentlich effizienter den Hauptspeicher an die einzelnen Prozesse verteilen kann (siehe Funktion **mmap** des Unix-Betriebssystems).

2. Alle von einer Datenbank-Anwendung (Datenbank-Client) benötigten Datenbank-Felder werden im Prozeßraum der Anwendung zwischengespeichert. Dies hat zudem den Vorteil, daß ein Datenbankelement mit Hilfe eines *C*—Zeigers identifiziert werden kann. Startet eine Anwendung eine Transaktion, sendet der Datenbankserver Prozeß dem Client alle seit der letzten Clienttransaktion geänderten Daten. Auf diese Weise kann der lokale Zwischenspeicher auf den neuesten Stand gebracht werden. Alle Änderungen innerhalb einer Transaktion werden nur auf der Client-Seite durchgeführt, eine Beendigung dieser Transaktion sendet alle geänderten Daten dann zum Server. Der Server startet, bevor er Daten eines Clients empfängt, eine interne Transaktion, die beendet wird, sobald der Transport der Daten ohne Fehler beendet wurde.
3. Die meisten Daten müssen, bevor sie verwendet werden, dekomprimiert werden. Da in vielen Fällen auf dieselben Daten in einer Sekunde mehrere hundertmal zugegriffen wird, werden die dekomprimierten Daten zwischengespeichert. Für diesen Vorgang wird ein fest definierbarer Speicherbereich reserviert, nach dem LRU (Least Recently Used) Verfahren werden die am längsten nicht mehr benötigten zwischengespeicherte Werte gelöscht.

Die Implementierung der Datenbank ist so gestaltet, daß Server- und Clientfunktionalitäten symmetrisch aufgebaut sind. Dies erlaubt, den Datenbankserverprozeß mit dem einer beliebigen Datenbankanwendung zu verschmelzen. In diesem Fall wird kein Client-seitiger Zwischenspeicher benötigt, aufwendige Prozeßwechsel entfallen und die Datenbankperformance dieser einen Anwendung steigt um ein Vielfaches. Daher wurde das datenbankintensivste Programm, **ARB_NT**, mit der Serverfunktionalität verschmolzen. Das dadurch entstande Sicherheitsrisiko (ein Programmabsturz der Anwendung hat logischerweise auch eine gewaltsame Beendigung des Datenbankservers zur Folge) ist in der Praxis kein Problem, da das Programm ARB_NT äußerst stabil arbeitet (weniger als 1 Absturz/Monat).

Diskussion

Der Hauptvorteil bei der Entwicklung eines eigenen Datenbanksystems lag nicht darin, neue Funktionalitäten für den Endanwender zu schaffen, vielmehr sollte der Prozeß der Softwareentwicklung entscheidend beschleunigt werden, indem viele der zu lösenden Probleme zentral in der Datenbank bearbeitet werden konnten. Vergleicht man diese Eigenschaften mit dem am Anfang aufgestellten Anforderungsprofil, stellt man fest, daß die meisten Vorgaben in den Bereich der praktischen Nebenbedingungen fallen und mit Hilfe dieses Datenbanksystems gelöst werden konnten:

- Die Möglichkeit, Funktionen in kleine Module aufzubrechen, gestattete es, programmierunerfahrene Studenten effizient einzusetzen. Selbst im ungünstigsten Fall, falls die Teilprogramme der Studenten häufig abstürzten, war in keiner Weise die Konsistenz der Benutzerdaten gefährdet. Auf der anderen Seite konnten mit Hilfe der Datenbank diese Algorithmen so mit dem Rest des Systems vernetzt werden, daß für einen Anwender keine Trennung zwischen den Modulen sichtbar ist.
- Die Datenbank läßt sich nicht nur mit der Programmiersprache C, sondern auch mit der interpretierten Sprache Perl ansprechen. Dadurch ist es möglich, daß das

Gesamtsystem relativ offen gehalten ist. Ein Programmierer muß im wesentlichen nur wissen, wie die Daten organisiert sind, um direkt in medias res zu gehen und sofort praktisch mit allen Algorithmen zusammenarbeiten. Dieses Vorgehen wurde soweit perfektioniert, daß Perl als Makro-Sprache verwendet werden kann. Möchte der Benutzer zum Beispiel mehrere Menüfunktionen zusammenfassen, startet er den Makro-Rekorder, bedient mit der Maus entsprechende Menüs und stoppt den Rekorder wieder. Aus dieser Aufzeichnung wird ein kleines Perl-Programm generiert, das dieselbe Funktionalität besitzt wie die mit der Maus ausgeführten Aktionen.

- Dadurch, daß die Algorithmen auf einem einheitlichen Datenbestand arbeiten, ist es möglich, beliebige Algorithmen zur Zusammenarbeit zu bewegen. Dieses Vorgehen ist bis jetzt in der Biologie unübertroffen.
- Dadurch, daß auf den Einsatz einer kommerziellen Datenbank verzichtet wurde, ist das Gesamtergebnis frei von Rechten Dritter, beliebig portabel und im Quelltext (**Source-Code**) vorhanden. Dies ist die Grundvoraussetzung dafür, daß dieses Programm auch anderen Forschungseinrichtungen unentgeltlich zur Verfügung gestellt werden kann.
- Dadurch, daß der Source-Codes der Datenbank jederzeit geändert werden konnte, war es möglich, diese für die biologische Anwendung zu optimieren. Insbesondere der Einsatz einer effektiven Kompression erwies sich dabei als essentiell.

Alle Eigenschaften der Datenbank können jedoch nur dann sinnvoll in die Praxis umgesetzt werden, wenn diese auch die nötige Performance besitzt. Aus dem Grund sollen hier noch Ergebnisse von praktischen Geschwindigkeitsversuchen angegeben werden. Die Testläufe wurden auf einer ’Digital Alpha 400’-Workstation durchgeführt. Jeder Testlauf wurde unter zwei verschiedenen Versuchsumgebungen gestartet: Im ersten Fall war der Prozeß der Testanwendung mit dem des Datenbankservers identisch. Somit entfielen langsame Prozeßwechsel zwischen Datenbankserver und Datenbankanwendung. Im zweiten Fall war die Testanwendung in einem eigenen Prozeß untergebracht. Gemessen wurde, wie oft eine bestimmte Funktion in einer Sekunde ausgeführt werden konnte:

Funktion [Durchlaufe/Sekunde]	Serverprozess == Clientprozess	Serverprozess != Clientprozess
1. Transaktionen	1.280.000	4.507
2. Datenbankänderungen	1.607.000	1.600.000
3. Änderung + Commit	177.000	2790
4. Änderung +Rollback	46.200	1460
5. Zahl lesen	7.000.000	7.000.000
6. Sequenz lesen	1.900.000	1.900.000
7. Bakterium suchen (index)	240.000	6680
8. Bakterium suchen (linear)	36	36

Abb. 133 Performance-Ergebnisse der Datenbank (7000 Organismen, 10k Sequenzlänge, Computer: Alpha 400 MHz)

X.1

Dabei wurde folgendes gemessen:

1. **Transaktion:** Die Anzahl der Transaktionen pro Sekunde (Beginn einer Teilsitzung – Aktion – Ende der Teilsitzung), wobei keine Aktionen innerhalb einer Transaktion durchgeführt wurden.
2. **Datenbankänderungen:** Die Anzahl der möglichen Schreibvorgänge pro Sekunde. Für den gesamten Testlauf wurde nur eine Transaktion geöffnet. In der Testanwendung wurde ein und dasselbe Datenbankfeld wiederholt beschrieben.
3. **Änderung und Commit:** Es wurde wiederholt eine Transaktion geöffnet, ein Datenbankelement geschrieben und die Transaktion ordnungsgemäß beendet (committed).
4. **Änderung und Rollback:** Es wurde wiederholt eine Transaktion geöffnet, ein Datenbankelement geschrieben und die Transaktion für ungültig erklärt (Rollback).
5. **Zahlfeld lesen:** Innerhalb einer einzigen Transaktion wurde wiederholt ein Datenbankfeld, das eine Zahl beinhaltet, gelesen.
6. **Sequenz lesen:** Es wurde wiederholt innerhalb einer einzigen Transaktion die Sequenz eines Organismus gelesen. Die Sequenzlänge betrug dabei 10000 Zeichen.
7. **Organismus suchen (index):** Es wurde wiederholt ein Organismus anhand seines Namens gesucht. Diese Suche ist besonders schnell, da die Datenbank für dieses Suchfeld einen Index verwaltet.
8. **Organismus suchen (linear):** Ohne die Hilfe eines Index müssen alle Daten aller Organismen linear durchsucht werden. Im schlechtesten Fall müssen dabei alle Felder der Datenbank durchsucht werden. Die angegebene Zahl gibt an, wie oft dieser Vorgang in einer Sekunde durchgeführt werden kann, wenn ein nicht vorhandener Organismus anhand des biologischen Namens gesucht werden. Dabei mußten je Suche mehr als 7150 Datensätze durchgearbeitet werden, wobei jeder Datensatz einzeln dekomprimiert werden mußte. Diese Art der Suche kommt relativ selten im Anwendungfall vor, und zwar nur dann, wenn ein Anwender von Hand Organismen anhand beliebiger Suchkriterien aufspüren möchte.

Auffallend ist der große Unterschied der Werte, je nachdem, ob Datenbank-Server und Client sich denselben Prozeß teilen oder nicht. So wurde versucht, alle datenbankintensiven Anwendungen in das Hauptprogramm ARB_NT zu integrieren. Diese Integration fiel nicht besonders schwierig aus, da die Kommunikation der Module über die Datenbank abgewickelt wird und im wesentlichen kein Unterschied besteht, ob ein Teilmodul nun in einem eigenen Prozeß läuft oder nicht. Weiterhin wird der große Einfluß des clientseitigen Cache deutlich. Jedesmal, wenn ein Prozeßwechsel durchgeführt werden muß, erreicht die Performance nur in Ausnahmefällen mehr als fünftausend Zugriffe pro Sekunde.

Ausblick

Im Laufe der letzten Jahre entstanden über 25.000 Zeilen spärlich dokumentierter Programmcode allein für die Implementierung des Datenbanksystems. Dabei wurde vor allem Wert auf die Performance der Datenbank gelegt und weniger auf ein perfekt durchstrukturiertes Programm. Die heutige Implementierung der Datenbank ist so gut wie fehlerfrei, für die mikrobiologischen Anwendungen nahezu optimal und äußerst schnell.

Wollte man diese Datenbank um nicht triviale Erweiterungen modifizieren, stieße man wahrscheinlich schnell an Grenzen, denn der Programmcode ist soweit optimiert, daß er nur schwer zu ändern ist. Hinzu kommt, daß die Datenbank in der Programmiersprache C geschrieben wurde, viele Funktionalitäten jedoch der Programmiersprache C++ nachgebildet wurden. Wesentlich eleganter wäre es gewesen, die Datenbank von vornherein in C++ zu entwickeln, dies war jedoch aus Kompatibilitätsgründen nicht möglich.

Im Laufe der Zeit entwickelte sich, insbesondere in Zusammenarbeit mit der RDP, eine Vorstellung vom idealen Anwendungsfall für die ARB-Datenbank:

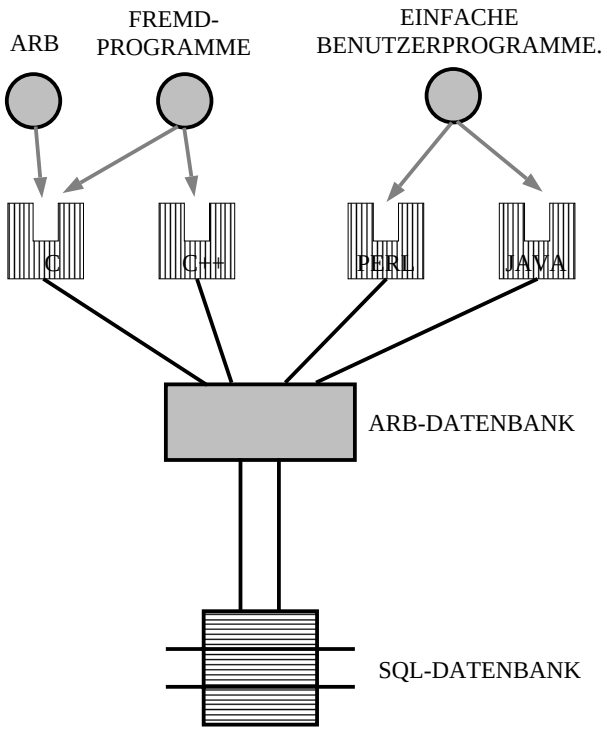


Abb. 134 Ideale mikrobiologische Analyse-Umgebung

Die Datenbank sollte eine Reihe von Schnittstellen zur Verfügung stellen, eine C oder C++ Schnittstelle für komplexe Programme, wie zum Beispiel das restliche ARB Programm, sowie eine Java– bzw. Perl-Schnittstelle, um Anwendern eine einfache, sichere und portable Schnittstelle für kleinere, nicht rechenzeitintensive Anwendungen an die Hand zu geben. Im Laufe der Diskussionen stellte sich heraus, daß Java aufgrund seines 'Listener'-Konzeptes wohl die ideale Programiersprache für die Datenbank darstellt.

Der wirkliche Schwachpunkt der ARB Datenbank ist, daß alle Daten im Hauptspeicher gehalten werden müssen. Dies macht bei Anwendungen Sinn, bei denen tatsächlich alle Daten bearbeitet werden. Enthält die Datenbank jedoch Daten, die nur selten benötigt werden, ist dieses Vorgehen unzweckmäßig. Als einfachen Ausweg könnte man das ARB-Datenbanksystem allein dazu verwenden, den Anwendungen Information zur Verfügung zu stellen, ohne sie zu speichern. Die Datenbank dient in diesem Fall nur als Cache, Kompressionsmodul und Synchronisationseinheit. Auf diese Weise könnte man die Vorteile einer traditionellen SQL-basierten Datenbank mit denen der ARB-Datenbank vereinigen. Diese Vorgehensweise wurde in dem schon erwähnten Prototypen ARBSEER implementiert.

X.2 Integration von Algorithmen

In den letzten Jahren wurde weltweit eine Anzahl von Algorithmen entwickelt, die für die Integration in Teilmodule in ARB interessant schienen. Zum großen Teil wurden für diese auch bereits von dritter Seite entsprechende Computerprogramme erstellt. Um nun nicht alle diese Programme neu zu schreiben, mußte ein Konzept entwickelt werden, auf welche Weise diese Algorithmen bzw. Programme zu einem Gesamtsystem integriert werden können. Im wesentlichen wurden zur Erreichung dieses Ziels drei Wege beschritten:

- Funktionen auf Zeichenströmen
- Verwendung eines existierenden Integrationspaketes: GDE
- Integration auf Datenbankebene

Im folgendenen sollen diese Integrationswege kurz erläutert werden.

X.2.1 Funktionen auf Zeichenströmen

Prinzip

Es existiert eine ganze Reihe von Algorithmen, die nur auf jeweils einer Sequenz oder Zeichenkette arbeiten. Beispielsweise könnte ein Algorithmus das Verhältnis der Anzahl Basen *G und C* zu der aller Basen *A, C, G und T* berechnen (GC-Verhältnis). Man kann die Arbeitsweise dieser Algorithmen in folgendem Schaubild zusammenfassen:

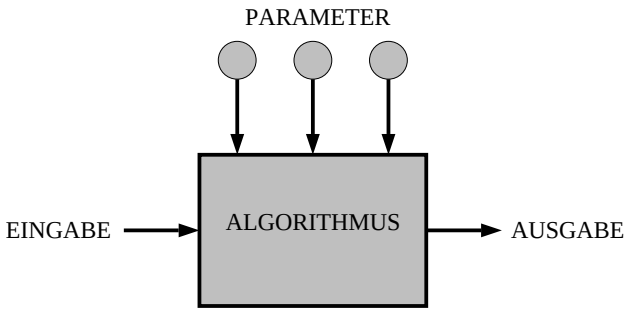


Abb. 135 Arbeitsweise von einfachen Zeichenketten-Analyse-Algorithmen

Ein solcher Algorithmus bearbeitet die Eingabe und gibt Ergebnisse aus. Zudem kann er durch Parameter konfiguriert werden. Der Vorteil solcher Algorithmen liegt in der Möglichkeit, sie hintereinanderzuschalten (Pipeline), wobei die Ausgabe des einen Algorithmus die Eingabe des nächsten darstellt.

X.2

Umsetzung

In ARB wurde ein Funktionsverwalter (**ACI**[1]) implementiert, bei dem jedes Modul eine endliche Anzahl solcher Algorithmen zur Bearbeitung von Zeichenketten unter einem eindeutigen Namen anmelden kann. Jeder, der nun eine solche registrierte Funktion verwenden möchte, stellt einen **ACI**–Befehl in Form einer Kommandozeichenkette zusammen, der die Funktion mit ihren Parameterwerten beschreibt:

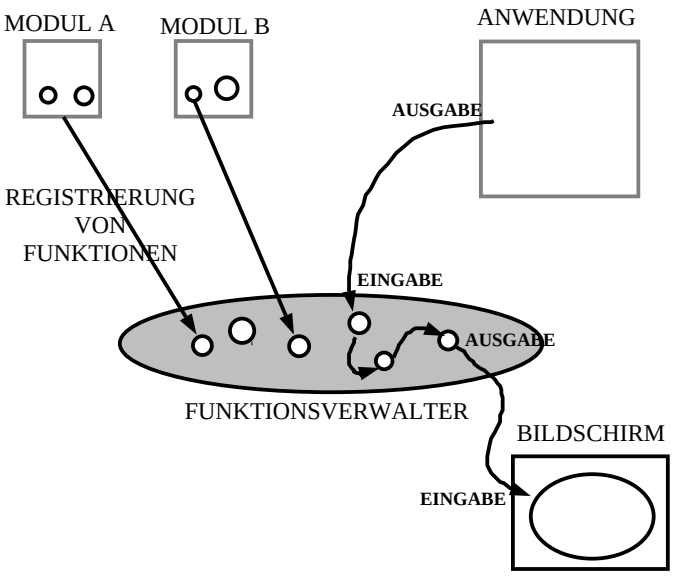


Abb. 136 Funktionsverwalter

Eine solcher Befehl für das Zählen aller ’A’s in einer Zeichenkette könnte zum Beispiel so aussehen:

```
count("A")
```

Um alle beim Funktionsverwalter registrierten Funktionen dem Anwender in einfacher Form zur Verfügung zu stellen, wurde das **NDS–System**[2] entwickelt. Diesem System wird als Eingabe ein Datensatz eines Organismuses vorgegeben. In einem Fenster kann der Benutzer eine Reihe von Datenbankfeldern selektieren, die aus dem Datensatz gelesen werden, wobei der Inhalt jedes einzelnen Feldes noch durch einen **ACI** Befehl modifiziert werden kann.

Die folgene Abbildung zeigt einen Screenshot des Konfigurationsfensters:

---

1 ACI steht für **A**rb **C**ommand **I**nterpreter

2 NDS steht für: Node Display Settings

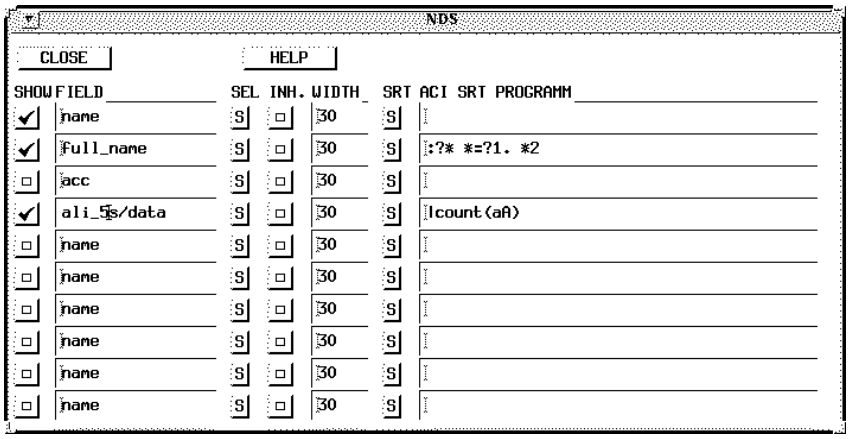


Abb. 137 Typische Einstellungen des NDS-Systems

Die Struktur des NDS Systems läßt sich mit folgendem Schema darstellen. Die Ausgabe kann dabei sowohl direkt auf dem Bildschirm erfolgen, oder in Form eines Datenbankfeldes gespeichert werden.

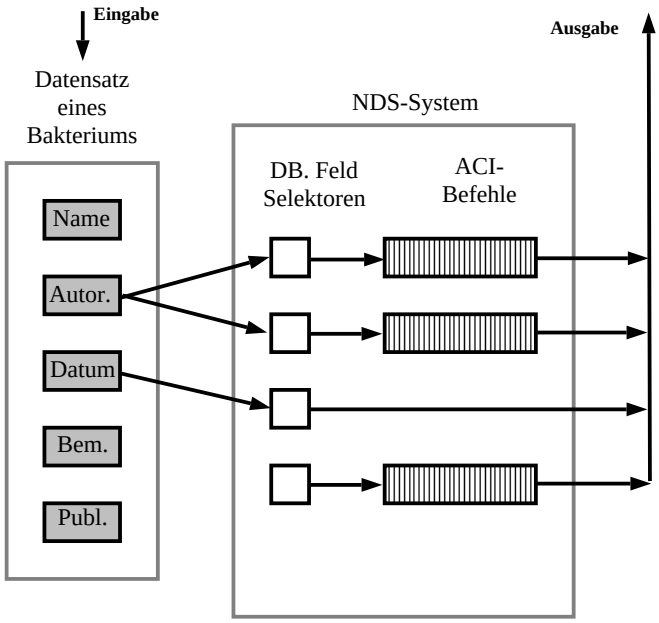


Abb. 138 Schema des NDS-Systems

Diskussion

Aufgrund der einfachen Anwendung des **NDS**—Systems war es ein leichtes, dieses in die verschiedensten Programme einzubauen. Als interessante Anwendung ist zum Beispiel zu nennen, daß der Benutzer sich mit Hilfe des **NDS** beliebige Daten aus der Datenbank am Bildschirm anzeigen lassen kann, und zwar nicht nur in tabellarischer Form, sondern auch an den Blättern der phylogenetischen Bäume. Somit kann zum Beispiel das *GC*–Verhältnis der Sequenzen in Relation zur Phylogenie gesetzt werden. Dieses System erlaubt es, auf flexible Weise an verschiedensten Stellen im System Datenbankinformationen effizient zusammenzustellen.

Sicherlich ist es gerade für Nicht-Computerfachleute nicht besonders einfach, eine Programmiersprache zu erlernen. Hat man aber das Grundprinzip verstanden, kann dieses Wissen an den verschiedensten Stellen in ARB äußerst effektiv eingesetzt werden.

X.2.2 GDE Schnittstelle

Das Paket GDE

Wie bereits oben erwähnt, wurde an der RDP (zur Zeit Michigan State University in East Lansing) von S. Smith [smith@bioimage.millipore.com] das Programmpaket **GDE** (**G**enetic **D**ata **E**nvironment) entwickelt. Dieses hatte zum Ziel, die vielen bereits vorhandenen frei verfügbaren Analyseprogramme auf eine einfache Art und Weise in einen Sequenzeditor zu integrieren. Der folgende kurze Auszug aus der Originalanleitung bestätigt dies:

> The Genetic Data Environment is part of a growing set of programs for manipulating and analyzing "genetic" data. It differs in design from other analysis programs in that it is intended to be an expandable and customizable system, while still being easy to use.
>
> There are a tremendous number of publicly available programs for sequence analysis. Many of these programs have found their way into commercial packages which incorporate them into integrated, easy to use systems. The goal of the GDE is to minimize the amount of effort required to integrate sequence analysis functions into a common environment. The GDE takes care of the user interface issues, and allows the programmer to concentrate on the analysis itself. Existing programs can be tied into the GDE in a matter of hours (or minutes) as apposed to days or weeks. Programs may be written in any language, and still seamlessly be incorporated into the GDE. These programs are, and will continue to be, available at no charge. It is the hope that this system will grow in functionality as more and more people see the benefits of a modular analysis environment. Users are encouraged to make modifications to the system, and forward all changes and additions to Steven Smith at smith@bioimage.millipore.com.

Abb. 139 Originalauszug aus der GDE Anleitung

Das Paket GDE ist im Source-Code frei verfügbar und erfreut sich großer Beliebtheit bei einer großen Benutzergemeinde.

Das Prinzip von GDE ist, daß mit Hilfe einer Beschreibungsdatei die für ein externes Programm notwendigen Parameter beschrieben werden. GDE sorgt dann dafür, daß der Benutzer diese Parameter nach seinen Wünschen verändern kann. Zur Ausführung dieses Sachverhaltes soll hier nur kurz an einem exemplarischen Beispiel das Zusammenspiel von Beschreibungsdatei (Abb. 140) und Benutzeroberfläche (Abb. 141) gezeigt werden:

X.2

```
item:Phylip 3.5

itemmethod:(/bin/rm -f outfile infile treefile ;\
readseq -a -f12 in1 > infile;\
$PREEDIT $BOOTSTRP ${ARB_XCMD:-cmdtool} $PROGRAM;\
$CONS ${ARB_TEXTEDIT:-textedit} outfile $DISPLAY_FUNC; rm in1 )&

arg:PROGRAM
argtype:choice_menu
arglabel:Which program to run?
argchoice:DNSPARS:DNSpars
argchoice:DNSML:DNSml
argchoice:DNSMLK:DNSmlk
argchoice:DNSCOMP:DNScomp
argchoice:DNSMOVE:DNSmove
argchoice:DNSINVAR:DNSinvar
argchoice:PROTPARS:protpars
#argchoice:PROTML:protml

arg:BOOTSTRP
arglabel:Bootstrap data?
argtype:chooser
argchoice:No:
argchoice:Yes: ${ARB_XCMD:-cmdtool} seqboot ; /bin/mv -f outfile infile;

arg:CONS
arglabel:Consensus tree?
argtype:chooser
argchoice:No:
argchoice:Yes: /bin/mv -f treefile infile; ${ARB_XCMD:-cmdtool} consense ;

arg:DISPLAY_FUNC
argtype:chooser
arglabel:View tree using treetool?
argchoice:Yes:& treetool treefile
argchoice:No:

arg:PREEDIT
argtype:chooser
arglabel:Edit input before running?
argchoice:No:
argchoice:Yes:${ARB_TEXTEDIT:-textedit} infile;

in:in1
informat:genbank
inmask:
insave:
```

Abb. 140 Ausschnitt aus der Konfigurationsdatei für den GDE Editor

Die oben dargestellte Beschreibungsdatei führt zu folgender Eingabemaske:

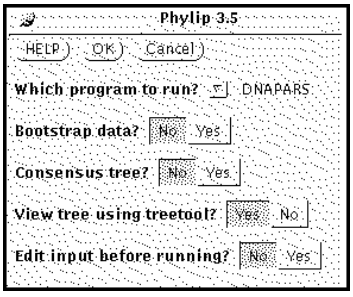


Abb. 141 Eingabemaske zur Berechnung der Phylogenie

X.2

Integration von GDE und ARB

Die Ideen, die hinter dem GDE-Programm stecken, sind von den Ansätzen her schlecht zu überbieten. Daher lag der Gedanke nahe, die Funktionalitäten von GDE mit denen von ARB zu verbinden. Für diese Integration stehen generell zwei Wege zur Verfügung:

- Anpassung und Integration von ARB in GDE
- Anpassung und Integration von GDE in ARB

Erweiterung des GDE Editors um ARB Funktionalitäten

Um das GDE-System möglichst flexibel einsetzen zu können, entwickelte S. Smith eine ganze Reihe unterschiedlicher Dateilese- und –schreibfunktionen, die mit den gebräuchlichsten Sequenzdateiformaten umgehen konnten. Diese Liste der benutzbaren Formate mußte nur um eines erweitert werden, nämlich das für die Anbindung an die ARB Datenbank. Da die Software-Schnittstelle dieser Datenbank bewußt offen und leicht ansprechbar gehalten wurde, konnte bereits nach nur einem Tag Arbeit eine lauffähige Version von GDE mit ARB-Anbindung vorgestellt werden.

Es ergaben sich jedoch eine ganze Reihe von Schwierigkeiten, insbesondere da GDE und ARB eine unterschiedliche Vorstellung davon haben, was eine Sequenz ist. In GDE entspricht jede Sequenz einem Datensatz, während in ARB jedem bakteriellen Datensatz eine Sequenz zugeordnet ist. So kann man im GDE-Editor Sequenzen löschen, duplizieren oder verschieben, beim Zurückspeichern der Daten nach ARB tritt dann jedoch ein Problem auf: Wird zum Beispiel im GDE-Editor eine Sequenz gelöscht, muß beim Zurückschreiben entschieden werden, ob die Sequenz fehlt, weil sie gelöscht oder weil sie nicht markiert und somit auch nicht geladen wurde. Aus diesem Grunde sollte GDE nicht dazu verwendet werden, Datensätze zu verwalten, sondern ausschließlich dafür, Sequenzen zu editieren.

Erweiterung von ARB um GDE Funktionalitäten

Als zweiter Weg wurde die Verbindung von GDE und ARB von der ARB Seite her angegangen. Es gelang, aus dem GDE-System die Ansteuerung externer Funktionen zu entnehmen, diese mit einer ’Motif’-kompatible Benutzeroberfläche auszustatten und in ARB zu integrieren. Diese Integration war so erfolgreich, daß nun fast alle von GDE angesteuerten Programme auch mit ARB zusammen arbeiten. Allerdings mußten folgende Konzepte des GDE-Systems entfernt und durch neue Konzepte ersetzt werden:

- GDE verwaltet nur Sequenzen, keine Bäume. Möchte man nun einen Baum berechnen lassen, wird diese Berechnung in einem eigenen Prozeß durchgeführt. Da GDE nicht in der Lage ist, mit Bäumen umzugehen, wird nach Beendigung dieser Berechnung ein externes Baum-Zeichenprogramm aufgerufen. ARB hingegen ist sehr wohl in der Lage, Bäume zu verwalten. Anstatt das Baum-Zeichenprogramm aufzurufen, wird ein Programm (arb_readtree) gestartet, das den Baum aus dem Dateisystem liest, eine Verbindung zum Datenbankserver aufbaut, den Baum dort speichert und eine Meldung an den Benutzer schickt.
- Wenn GDE Sequenzänderungen vornimmt, z.B. die DNS in Aminosäure übersetzt, erzeugt GDE hierfür neue Sequenzen. Dieses Vorgehen ist in ARB nicht sinnvoll, da die Zuordnung der neue berechneten Sequenzen zu den ursprünglichen nicht

mehr möglich ist. Aus diesem Grunde mußten alle Funktionen dieser Kategorie abgeschaltet werden.

Nachdem die nicht kompatiblen Funktionen eliminiert waren, ließ sich der große Rest problemlos von ARB aus ansprechen und benutzen. Im Laufe der Zeit wurde des zugrundeliegende System mehrfach optimiert.

Diskussion

Während die Erweiterung des GDE-Editors um eine ARB-Datenbankanbindung nur dazu führte, daß für ARB ein weiterer Editor zur Verfügung stand, war die Erweiterung von ARB um GDE-Funktionalitäten wesentlich interessanter. Dem ARB-Benutzer stand nun mit einem Schlag fast die ganze Bandbreite existierender Software zur Verfügung, inklusive der oben vorgestellten Pakete phylip, fastDNSml sowie das japanische molphy-System.

Allerdings wurde bei dieser Art der Integration die Software nicht geändert, d.h. es wurden die Schwachstellen existierender Programme nicht behoben. Insbesondere die steigende Datenmenge sprengt aber die Grenzen der meisten Applikationen, so daß man insgesamt sagen kann:

Für kleinere Datenmengen (bis zu 500 Sequenzen) ist die in ARB integrierte GDE Funktionalität sinnvoll einsetzbar, für größere Datenmengen muß auf die speziell für ARB optimierten Algorithmen zurückgegriffen werden.

X.2.3 Datenbank Schnittstelle

Im Prinzip stellt das ARB-Datenbanksystem allen Prozessen des ARB-Systems einen virtuellen gemeinsamen Datenpool zur Verfügung. Die Daten(felder) in diesem Datenpool sind nach festen Regeln angeordnet[1]. Jedes Modul hat uneingeschränkten Lese– und Schreibzugriff auf alle Daten, d.h. beliebige Module können einfach dadurch zusammengeschaltet werden, daß sie gemeinsam genutzte Datenfelder in der Datenbank lesen und schreiben:

[1] Diese Regeln sind teilweise im Source-Code (/usr/arb/WINDOW/aw_awars.hxx) wie auch in der Datenbank — Dokumentation beschrieben.

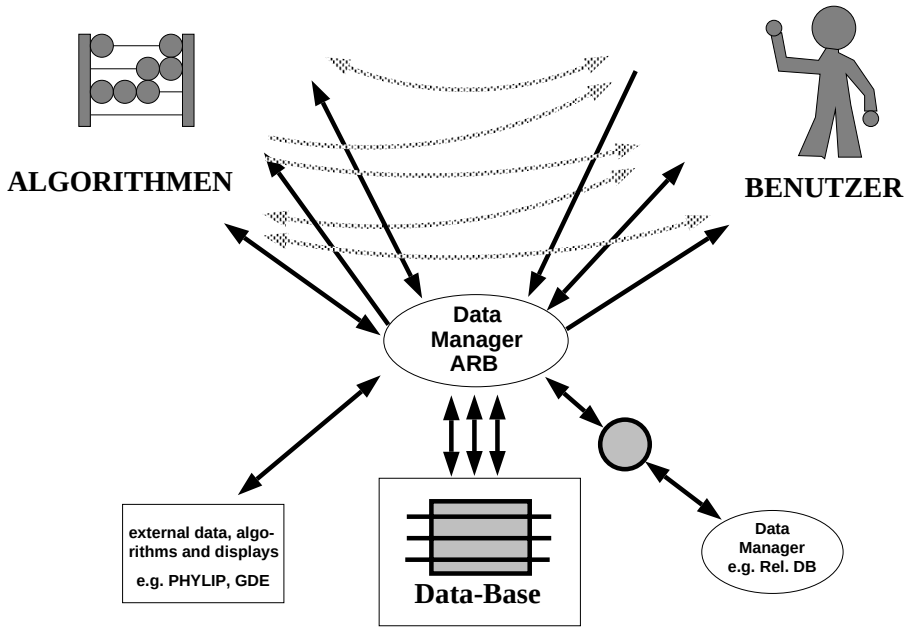


Abb. 142 Virtuelle Verbindungen der Module

Für einen Anwender scheinen so die einzelnen Programmteile miteinander verflochten zu sein, er arbeitet mit virtuellen Verbindungen.

So wird die Integration der Algorithmen zu einem Gesamtsystem mit Hilfe der ARB Datenbank trivial. Die Module müssen nur so programmiert sein, daß sie auf Änderungen der für sie relevanten Datenbankfelder flexibel reagieren.

X.3 Struktur des Software Paketes ARB

Weltweit arbeiten eine Vielzahl von Labors an der Sequenzierung und Analyse von ribosomalen RNS Sequenzen. Ziel eines neuen Softwarepakets muß es daher sein, den Labors alle notwendigen Informationen und Werkzeuge in höchst optimierter Form an die Hand zu geben, um eine weltweite Vernetzung herbeizuführen. Dadurch sollen Doppelarbeiten in den verschiedenen Labors vermieden werden. Das Publizieren eines neuen Programms alleine führt diese Globalisierung nicht herbei, daher sollte für ARB ein Weg gefunden werden, sich in die globale Umgebung effektiv zu integrieren. Die Art, wie ARB sich an die globalen Strukturen anbinden läßt, wie auch seine Art, mit Analysemodulen umzugehen, hat dazu geführt, daß viele Labors ARB verwenden.

X.3.1 Weltweite Datenflußanalyse

Publizierte DNS- und Aminosäure-Sequenzen werden von großen Instituten, wie Genbank und EMBL[1], gesammelt und der Öffentlichkeit per Internet zur Verfügung gestellt.

[1] EMBL ist ein von der EU finanziertes mikrobiologiesches Forschungszentrum in Heidelberg.

X.3

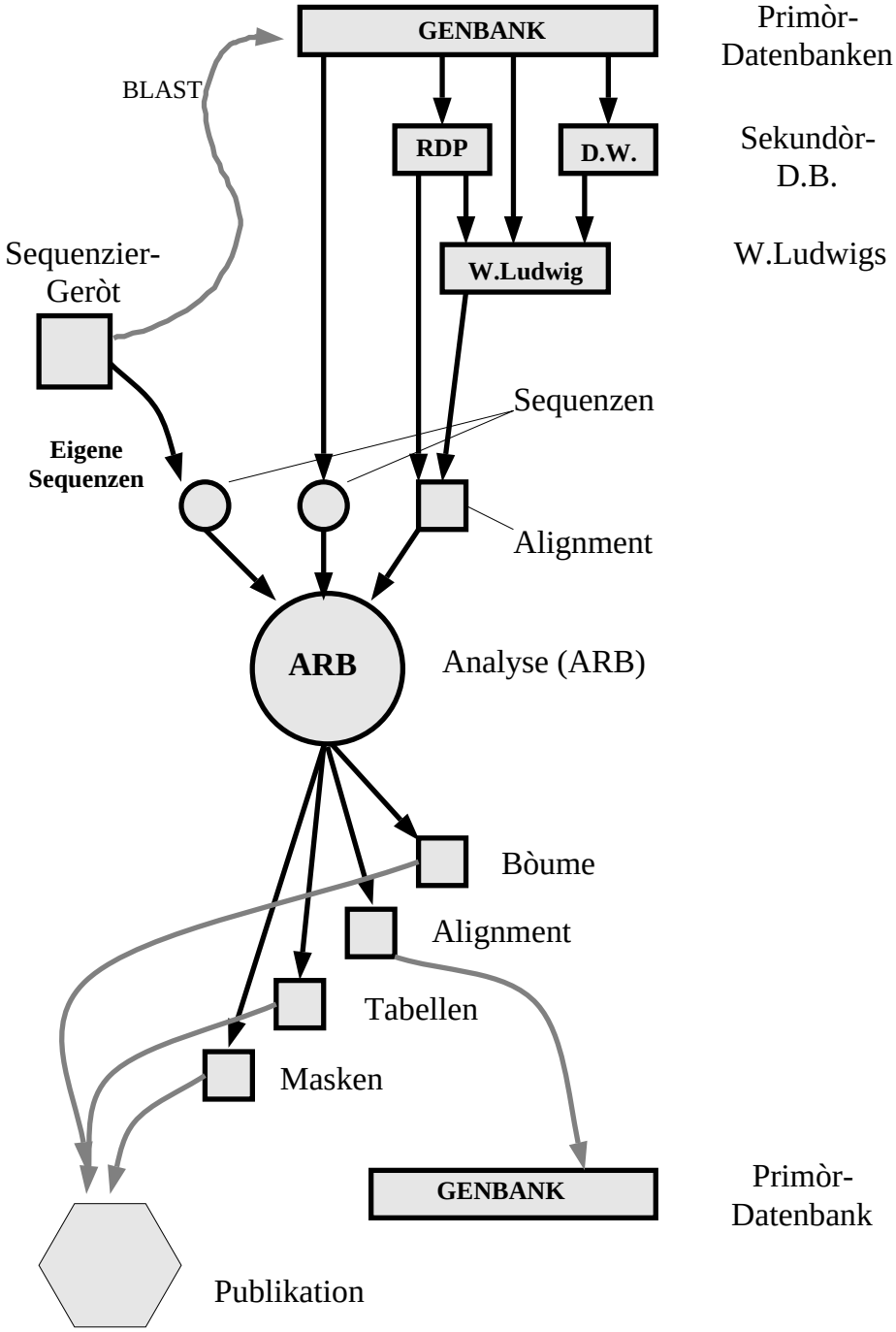


Abb. 143 Globale Datenströme

Ausgehend von den Primärdatenbanken, gibt es einige wenige Institutionen, nämlich die RDP und die DeWachter Gruppe, die sich darauf spezialisiert haben, Sequenzen der ribosomalen RNS (rRNS) zu sammeln und in alignter Form der Öffentlichkeit anzubieten. Allerdings benötigten diese Gruppen in der Vergangenheit etwa 1 Jahr, bis eine neue Sequenz den Weg von der Primärdatenbank bis auf deren Internet-Server gefunden hatte. Da dieser Zeitdifferenz bei dem momentan zu beobachtenden exponentiellen Datenwachstum entscheidende Bedeutung zukommt, wurde auch an der TU-München eine Datenbank aufgebaut, deren Hauptziel es zum einen war, die Datenbank der RDP um die neuesten Sequenzen

erweitert dem Laborbetrieb zur Verfügung zu stellen, zum anderen sollte die schlechte Qualität der Alignments der RDP verbessert werden. Hinzu kam, daß hier mit der integrierten Datenbank von ARB gearbeitetet wird und somit die Möglichkeit besteht, Bäume, Filter, Sequenzen und Statistiken auf einmal zu bearbeiten.

Der globale Datenfluß stellt sich nun folgendermaßen dar:

Ein beliebiges Labor, das mit ARB arbeitet, analysiert die ribosomale RNS-Sequenz eines Organismus. Daraufhin wird versucht, möglichst alle verwandten Sequenzen zu sammeln und mit dieser neuen Sequenz zu vergleichen. Dazu werden Suchanfragen an die Primärdatenbanken gestellt (blast / fasta). Das Ergebnis dieser Suche, zusammen mit den neuen Sequenzen sowie einem möglichst aktuellen Datensatz, werden in einer ARB-Datenbank zusammengefaßt und analysiert. Die dabei erhaltenen Ergebnisse werden publiziert und die neuen Sequenzen in nicht alignter Form wieder den Primärdatenbanken zur Verfügung gestellt.

X.3.2 ARBs interner Datenfluß

Die meisten Analyseprogramme für genetische Sequenzen arbeiten nach dem Pipeline-Prinzip. Dabei werden die Module so hintereinandergeschaltet, daß die Ausgaben des einen Programms die Eingabe des nächsten sind. Mit Hilfe des neuen Datenbanksystems gelang es, eine neue neue Vorgehensweise anzuwenden. Dabei wird die Datenbank als ein Pool von typisierten Objekten betrachtet. Typisiert heißt in diesem Zusammenhang, daß jedes Objekt in der Datenbank einem Typ zugeordnet ist, z.B. dem Baum-Typ, dem Sequenz-Typ, dem Filter-Typ usw. Jedes Modul hat eine Reihe möglicher Eingabeobjekttypen und einen Ausgabetyp. Auf diese Weise können die Module kreisförmig um die Datenbank angeordnet werden. Dabei können sie flexibel die benötigten Daten aus der Datenbank herausgreifen:

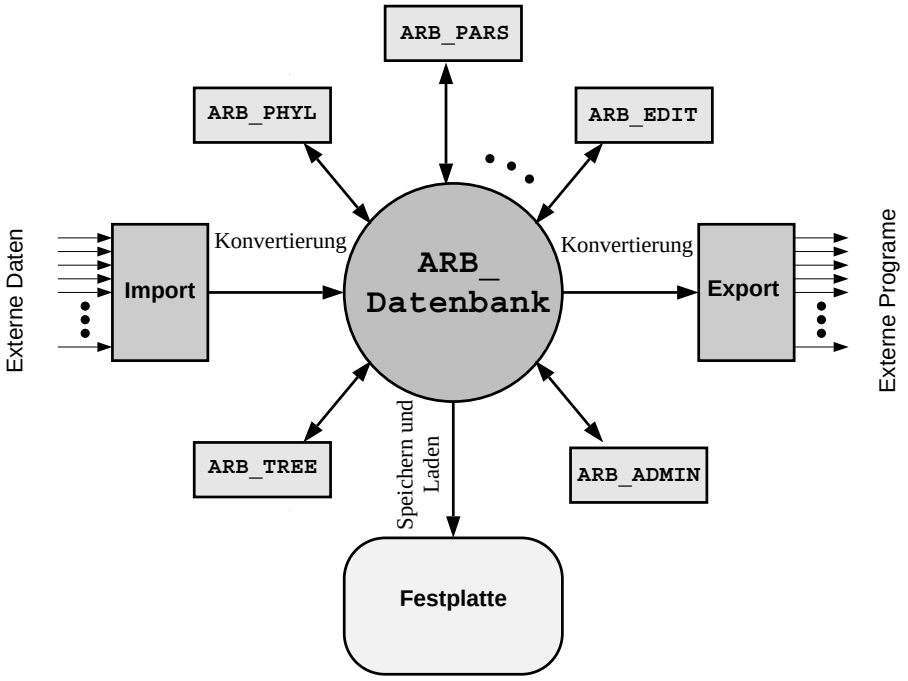


Abb. 144 Die Organisation der ARB-Module

Um mit der 'Außenwelt' in Kontakt zu treten, gibt es zwei spezielle Module, die Import– bzw. Export-Funktionen, die sich aus jeweils mehreren Einzelmodulen zusammensetzen.

X.3.3 Kurze Übersicht über die wichtigsten ARB-Module

Wie bereits erwähnt, benötigt jedes Teilmodul als Eingabe Objekte eines bestimmten Typs. Dabei kann es sich im wesentlich um folgende Objekttypen handeln:

Typ	Beschreibung
unalignte **Sequenz**	Eine Sequenz, wie sie aus einer Sequenziermaschine kommt, ohne Lücken.
Alignment	Eine Menge von Sequenzen, bei denen durch Einfügung eines Lücken-Symbols homologe Bereiche in der selben Spalte stehen.
kleiner **Baum**	Ein Baum, der in praktikabler Zeit neu berechnet werden kann, hat meist nicht mehr als 100 Blätter.
großer **Baum**	Ein Baum, dessen Neuberechnung nicht ohne weiteres mit einem Programm durchgeführt werden kann, umfaßt meistens die Sequenzen aller Organismen.

Abb. 145 Übersicht über die in ARB verwendeten wichtigsten Objekt-Typen (Continued . . .)

X.3

Typ	Beschreibung
Baum mit **Bootstrap** Werten	Ein Baum, bei dem den Kanten die Wahrscheinlichkeit ihrer Korrektheit zugeordnet ist.
Filter	Eine spezielle Sequenz, die bestimmte Spalten in einem Alignment markiert (oder filtert).
Column-**Statistic**	Eine spaltenweise Statistik über ein Alignment.
Rates	Angabe über die durchschnittliche Mutationsrate einer Alignmentspalte.
PT Index File	Ein Sequenzindex, der es erlaubt, effizient bestimmte Teilsequenzen im Datensatz zu suchen
Helix Template	Eine spezielle Sequenz, die die Struktur der Helices aller Sequenzen beschreibt.
Postscript	Darunter sind alle diejenigen Ergebnistypen von Algorithmen zusammengefaßt, die nur für die visuelle Kontrolle verwendet werden können, und die normalerweise nicht wieder in weitere Analysen eingehen.

Abb. 145 Übersicht über die in ARB verwendeten wichtigsten Objekt-Typen

Die nun folgende Übersicht über ausgewählte in ARB implementierte Funktionen erlaubt auf einfache und schnelle Weise die Zusammenhänge zwischen den verschiedenen Objekttypen zu erkennen:

Modul name	Eingabetypen [optional]	Ausgabetyp	Beschreibung
ARB_EDIT4	Sequenzen [Alignment] [Helix] [Baum] [C.Statistic]	Alignment [Helix]	Editieren von Sequenzen, wobei eine ganze Reihe von zusätzlichen Informationen eingeblendet werden können
CLUSTALW	Sequenzen	Alignment	Alignen einer kleinen Menge von Sequenzen.
PT_SERVER	Sequenzen	PT Index	Erzeugung einer Indexdatei für schnelle Sequenzsuche
ARB_ALIGN	Sequenzen Alignment [PT Index]	Alignment	Alignen einzelner Sequenzen zu einem gegebenen Alignment
phylip-Module	Alignment [Filter]	k. Baum [k. B. Bootstrap]	Berechnung kleiner Bäume mit optionalen Bootstrapwerten.

Abb. 146 Übersicht über die wichtigsten ARB-Module (Continued . . .)

Modul name	Eingabetypen [optional]	Ausgabetyp	Beschreibung
fastdnaml	Alignment [Filter] [Rates]	k. Baum	Berechnung kleiner Bäume
ARB_POSVARBYPARS	Alignment g. Baum	C. Statistik	Berechnung einer Spaltenstatistik aufgrund eines Baumes und Alignments.
DNAmlrates	Alignment Baum	Rates	Berechnung der spaltenweisen Mutationsrate
ARB_PHYL	Alignment	Filter	Berechnung von Filtern
Consens	Alignment	Filter	Berechnung des Konsensus
ARB_PROBE	Baum PT Index	Sonden	Berechnung von Oligonukleotiden, die eine Gruppe von Sequenzen charakterisieren.
ARB_DIST	Alignment C. Statistic	k. Baum mit Bootstrap	Berechnung von Bäumen nach dem Distanzverfahren mit optionalen Bootstrapwerten.
ARB_PARS	Alignment g. Baum	g. Baum	Hinzufügen neu alignter Sequenzen zu einem gegebenen Baum nach der Parsimony-Methode
ARB_PARS	Alignment g.Baum	g. Baum mit Bootstrap	Schätzung von Bootstrapwerten für große Bäume.
ARB_STAT	Alignment g. Baum	Postscript	Berechnung von statistischen Informationen zur Bewertung der Qualität eines Alignments
ARB_PRIMER	Alignment	Postscript	Berechnung möglicher Primer
MULTI-PROBE	Baum Sonden	Sondentupel	Berechnung von optimalen Kombinationen von Sonden.

Abb. 146 Übersicht über die wichtigsten ARB-Module

X.3.4 Ausblick

Durch den Einsatz von Modulen mit typisierten Eingabeströmen konnte eine sehr flexibel einsetzbare Analyseumgebung geschaffen werden. Nahezu jede Ausgabe eines Programms läßt sich als Eingabe anderer Programme nutzen, wobei durch die hohe Anzahl der Kombinationsmöglichkeiten systematische Untersuchungen an Algorithmen und Daten möglich werden. Dies ist aber auch zugleich der Nachteil dieser Vorgehensweise. Anwender, die nicht an komplexen Berechnungen interessiert sind, sondern nur ihre Sequenzen verrechnen wollen, sind oft überfordert, und selbst diejenigen, die bereits mehrere tausend Stunden mit

ARB gearbeitet haben, kennen zum Teil viele Möglichkeiten des Paketes noch nicht. Allerdings sollte an dieser Stelle auch nochmals daran erinnert werden, daß ARB nicht mit dem Ziel entwickelt wurde, Routineangelegenheiten zu optimieren, vielmehr sollte mit seiner Hilfe die Sequenz- und Phylogenieforschung effizienter gestaltet werden.

Vergleicht man die interne Struktur von ARB mit der globalen Datenflußanalyse, so muß man feststellen, daß viele Objekttypen, die von ARB verwaltet werden können, nicht global ausgetauscht werden. Dies ist ein großer Nachteil für die Benutzergemeinde, unter Umständen müssen viele Arbeiten nochmals durchgeführt werden, die bereits an einer zentralen Stelle erledigt wurden. Dieses Problem war auch Gegenstand intensiver Diskussionen mit Vertretern der RDP. Dabei wurde gemeinsam an einer globalen Lösung gearbeitet:

Die RDP geht in Zukunft mit hoher Wahrscheinlichkeit dazu über, ARB als einen Bestandteil ihrer Administrationstools zu nutzen. Dies bedeutet, daß sie ohne Mehraufwand ihre Datenbank im ARB-Format auf ihrem Internet-Server für die Benutzergemeinde zur Verfügung stellen kann. So kommen nicht nur die größeren Labors in den Genuß eines integrierten Datensatzes, der alle relevanten Informationen enthält, seien es Alignments, Bäume, Filter, oder Helix-Beschreibungen.

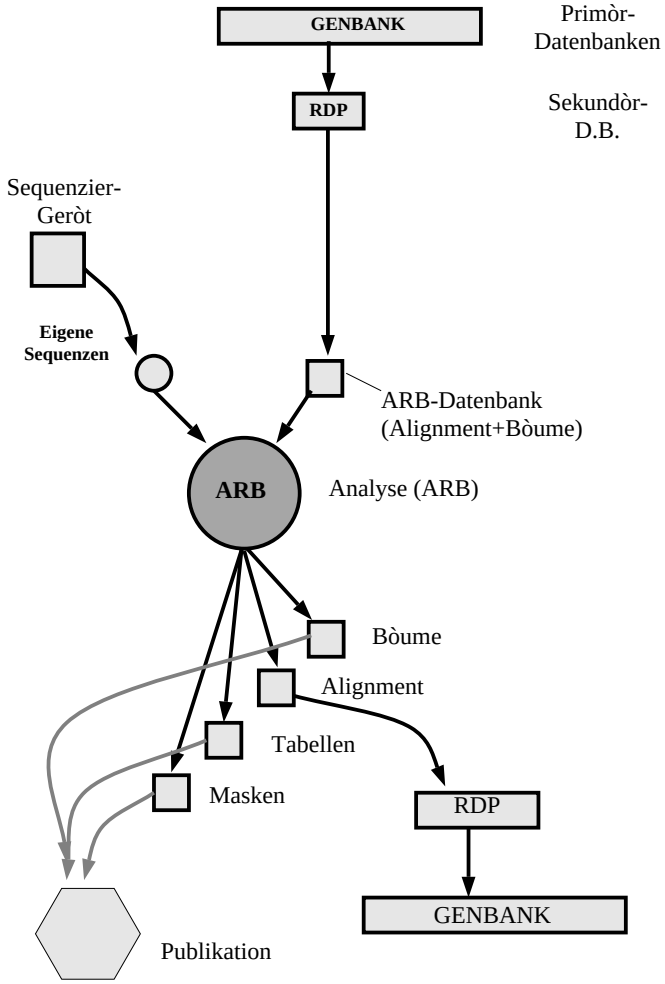


Abb. 147 Idealer globaler Datenfluß

X.3

Sollte sich ARB in Fortsetzung des bisherigen inteRNStionalen Erfolges weltweit durchsetzen, könnte der inteRNStionale Austausch von Daten derart gehandhabt werden, daß man, anstatt die nicht alignten Sequenzen zum Beispiel an Genbank zu senden, diese in alignter Form an die RDP weiterleitet. Die RDP könnte dann die Sequenzen an Genbank weiterreichen. Mehrere komplexe und zeitaufwendige Datenbankextraktionen und Zwischenschritte könnten dadurch eingespart werden. Zudem würde das Alignment der Sequenzen nicht verloren gehen, doppelte Arbeit könnte also entfallen, viele unnötige Datenströme könnten einfach weggelassen werden.

X.4 Kompression als Voraussetzung für praktische Anwendbarkeit

X.4.1 Übersicht

Mit der Weiterentwicklung und Verbesserung der Methoden der Sequenzanalyse setzte eine Flut von ribosomalen RNS Sequenzen ein. Da dieses **exponentielle Wachstum** der Datenmenge mit der exponentiellen Rechenleistungs-Steigerung heutiger Rechner Hand in Hand geht, waren viele Anwender gezwungen, sich immer wieder die neuesten Rechnergenerationen zuzulegen, um überhaupt noch arbeiten zu können. Es wäre also höchst wünschenswert, aus diesem **Teufelskreis** von Rechnerentwicklung und Datenwachstum auszubrechen, um mit günstigeren Computern arbeiten zu können. Nur wenn dies gelingt, kann eine große Anzahl von Anwendern von den jeweils neuesten Erkenntnissen auf diesem Gebiet profitieren.

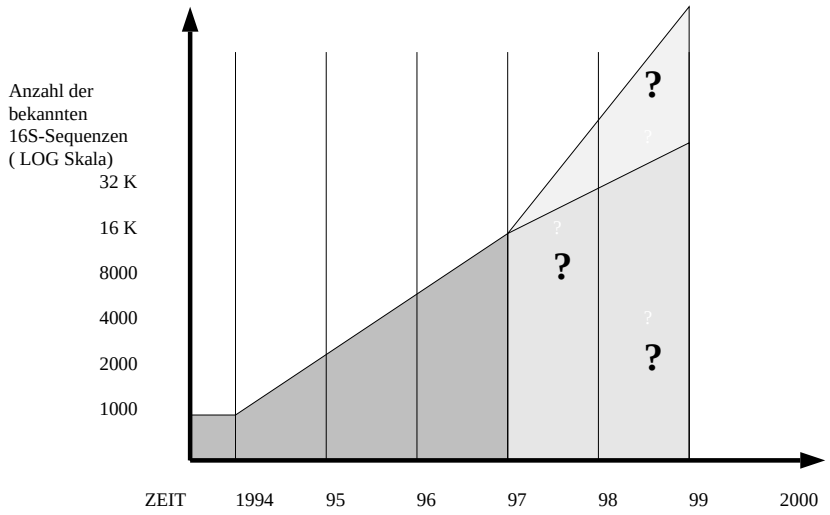


Abb. 148 Exponentielles Wachstum der 16S-rRNS Sequenze

Allerdings ist dieses Problem nicht neu: bereits seit der Erfindung des Computers sind Programmierer immer wieder gezwungen, ihre Daten auf preiswerte Speichermedien, also Festplatten, auszulagern. Im Bereich der Biologie kann diese Auslagerung nur bedingt Anwendung finden: Sollen 10000 Sequenzen der Länge 10000 Basen gleichzeitig editiert werden, so benötigt man minimal 100 MegaByte Hauptspeicher, unabhängig von dem zugrundeliegenden Datenbanksystem[1]; allerdings wächst dieser Wert noch um das zwei– bis

[1] Dasselbe gilt für alle bekannten Baumberechnungsalgorithmen: die Vorgehensweise dieser Algorithmen erfordert es, auf die Sequenzen oft und in beliebiger Reihenfolge zuzugreifen, so daß diese nur bedingt auf Festplatte ausgelagert werden können.

dreifache, wenn man eine ’undo’ Funktion implementieren möchte. So war es unumgänglich, nach neuen Lösungen und Wegen zu suchen. Dabei stellte sich heraus, daß eine triviale Lösung den größten Erfolg versprach: **Datenreduktion**. Damit ist zum einen gemeint, daß alle phylogenetisch unwichtigen Daten, wie zum Bespiel **Bilder**, nicht in der Sequenz-, sondern in einer herkömmlichen **relationalen Datenbank** gespeichert werden, zum anderen, daß man die Daten soweit wie möglich komprimiert.

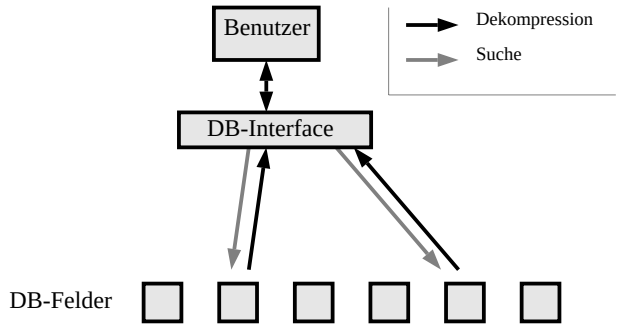
Durch Einsatz von bekannten und im Rahmen dieses Projektes neu entwickelten **Kompressionsverfahren** konnte der Speicherbedarf der Sequenzen stark reduziert werden. Kompressionsraten von bis zu **1:20** komprimierten selbst die größten Datensätze soweit, daß sie sich selbst mit normalen Pentium PCs bearbeiten ließen. Durch eine geschickte Implementierung gelang es zudem, die Dekompression so transparent in die Anwenderprogramme zu integrieren, daß diese ohne Code-Änderung direkt mit komprimierten Daten arbeiten können, d.h. daß zum Beispiel beim Sequenzeditor nur noch die Sequenzen dekomprimiert werden, die gerade auf dem Bildschirm sichtbar sind. Somit werden theoretisch nur noch **5 Megabyte** Hauptspeicher für die oben angegeben Anwendung benötigt, eine Größe die selbst normale PCs spielend bewältigen.

Als angenehmer Nebeneffekt wurden die einzelnen Datensätze so klein, daß es bei weitem schneller war, **alle Datensätze** in den Hauptspeicher zu laden und danach die einzelnen benötigten Sätze zu suchen, als nur einen Teil von ihnen einzeln von Platte zu holen.[1]

X.4.2 Anforderungen

Da Kompression schon immer eine wichtige Rolle in der Informatik spielte, wurde eine Vielzahl verschiedener Algorithmen entwickelt. Allerdings arbeiten diese meist auf einem Eingabestrom von Zeichen, der (de)-komprimiert wieder ausgegeben wird. In diesem speziellen Anwendungsfall jedoch sollte jeder einzelne Datensatz einer ganzen Datenbank mit einer Kompression versehen werden:

Traditionelle Datenkompression



Geforderte Kompression/Dekompression

Leider stellte sich heraus, daß fast alle bekannten Verfahren deutlich hinter den Erwartungen zurückfielen. Aus diesem Grund wurde zunächst mit Hilfe zweier typischer Anwendungs-

[1] Gesamte Datenbank (10mByte) laden + 15000 Zugriffe: 3 sec
15000 einzelne Zugriffe * 10% (Festplattencache Misses) * 10msec = 15sec

beispiele ein Anforderungsprofil erstellt werden, woraufhin existierende Verfahren an diesem gemessen, verbessert und erweitert wurden. Dazu soll von folgender Situation ausgegangen werden:

Größe der 16S-rRNS Datenbank (Stand Januar 1997)

Ein Sequenz-Datensatz beinhaltet typischerweise ein Feld, in dem eine Sequenz gespeichert ist, sowie 5 bis 20 die Sequenz beschreibende Textfelder, in denen zum Beispiel die Sequenz, der Autor und anderes gespeichert sind.

- Anzahl der Datensätze: 10000
- Länge einer Sequenz: 10000 Basen
- Anzahl der Sequenzen pro Datensatz: 1
- Anzahl der Datensatz-Textfelder pro Datensatz: 10
- Durchschnittliche Länge eines Textfeldes: 100 Zeichen

Anwendungsbeispiel 1: Editieren aller Sequenzen, die einem bestimmten Autor zugeordnet werden.

Die Aufgabe gliedert sich in zwei Schritte:

- Dekompromieren des Datenfeldes 'author' aller Datensätze und anschließende lineare Suche. (1 MegaByte unkomprimierte Zusatzdaten)
- Laden, Dekomprimieren und Anzeigen weniger Sequenzen.

Anwendungsbeispiel 2: Editieren aller Sequenzen

Obwohl es auf den ersten Blick unsinnig scheint, 10000 Sequenzen auf einmal zu editieren, hat sich doch in der Praxis gezeigt, daß dies sehr wohl wünschenswert ist, und zwar insbesondere dann, wenn die Sequenzen verschiedener Spezies-Gruppen gegeneinander alignt werden sollen.

Um den Programmieraufwand für ARB-Module gering zu halten, sollten die einzelnen Module jeweils möglichst unabhängig von den verwendeten Kompressionsverfahren programmiert werden können. Dies setzte voraus, daß die Kompression noch unterhalb der Datenbankschnittstelle implementiert wurde. Daraus ergab sich folgendes Anforderungsprofil.

	Anforderung	Kommentar
I	**hohe Kompressionsraten**	Der Algorithmus darf und soll die speziellen Eigenschaften von Sequenzdaten ausnutzen
II	**schneller wahlfreier Zugriff**	Es solle nicht immer die gesamte Datenbank dekomprimiert werden, wenn nur einzelne Felder benötigt werden.
III	**schnelle Dekompression**	Oft wird eine lineare Datenbanksuche die Dekompression großer Teile des Datenbestandes erfordern.

	Anforderung	Kommentar
IV	**Unempfindlichkeit gegen Daten-übertragungsfehler.**	Werden bei einer Übertragung der Datenbank Fehler mitübertragen oder sind einzelnde Sektoren einer Festplatte nicht mehr lesbar, soll bis auf die betroffenen Datensätze die Gesamtdatenbank gerettet werden.[1]
V	**Transparenz**	Die Anwendungsalgorithmen sollen von der Kompression/Dekompression abgeschirmt werden. Für den Anwender der Datenbank soll diese unkomprimiert erscheinen.

X.4.3 Existierende Algorithmen und Diskussion

Mathematische Grundlagen[2]

Definition:

Information ist die Bedeutung, die durch eine Nachricht übermittelt wird. Der Begriff Information ist in diesem Sinne subjektiv.

Zur Übertragung von Information bedienen wir uns einer **Sprache**. Jede Sprache hat ein System bestimmter Regeln, Grammatik genannt, nach denen eine Nachricht aufgebaut sein muß.

Eine in einer Sprache abgefaßte Nachricht ist meist aus **Zeichen** zusammengesetzt. Die Menge aller unterschiedlichen Zeichen wird als **Alphabet** bezeichnet.

Unter **Codierung** versteht man die Abbildung einer Sprache auf eine andere Sprache. Die Codierung ist definiert als Abbildung einzelner Zeichen eines Quellalphabets auf einzelne Zeichen oder Zeichenfolgen des Zielalphabets. Die so für ein einzelnes Zeichen des Quellalphabets erhaltene Zeichenfolge nennt man **Wort**. Die Menge aller Wörter der Zielsprache, die durch diese Abbildung erzeugt werden können, nennt man **Code**.

Die von Shannon 1948 entwickelte Informationstheorie versucht, ein Maß für den **Informationsgehalt** für ein Zeichen einer Nachricht zu finden, d.h. ein Maß dafür, wieviel Information eine diskrete, d.h. wohl unterscheidbare Nachricht enthält. Eine Nachricht besteht aus einer Zeichenfolge, in der die Zeichen mit bestimmten Wahrscheinlichkeiten

[1] Bei dem public domain Program ’gzip’ zum Beispiel ist in diesem Fall die ganze Datenbank verloren

[2] Die hier vorgestellte Einführung in die Informations– und Codierungstheorie stellt nur minimal die nötigen Vorraussetzungen für die nächsten Kapitel zusammen. Interessierte seien auf die weiterführenden Literatur verwiesen: [bielb 90@] [nels 92]

p auftreten. Der Informationsgehalt eines Zeichens oder einer Zeichenfolge soll folgende Eigenschaften haben:

I. Der Informationsgehalt soll nur von der Wahrscheinlichkeit abhängen, mit der das Zeichen gesendet wird. Je wahrscheinlicher ein Zeichen auftritt, je niedriger soll sein Informationsgehalt sein.
II. Der Informationsgehalt unabhängiger Zeichen einer Nachricht soll gleich der Summe der einzelnen Informationsgehalte sein:

$$f(x) + f(y) = f(x + y)$$

Definitionen

Der **Informationsgehalt h eines Zeichens** mit der Auftrittswahrscheinlichkeit **p** beträgt

$$h = ld\left(\frac{1}{p}\right) = -ld(p)$$

Sei $\mathbf{h_i}$ der Informationsgehalt des i-ten Zeichens und $\mathbf{p_i}$ die Wahrscheinlichkeit, mit der dieses Zeichen auftritt. Der **mittlere Informationsgehalt H**, auch **Entropie** genannt, ist definiert durch:

$$H = \sum_i p_i * h_i = -\sum_i p_i * ld(p_i)$$

Sei l_i die Länge des dem i-ten Zeichen entsprechenden Codes/Codewortes. Die **mittlere Wortlänge L** eines Codes ist definiert als die mit den Auftrittswahrscheinlichkeiten gewichtete Summe der Längen der den einzelnen Zeichen entsprechenden Codewörtern:

$$L = \sum_i p_i * l_i$$

Die **Redundanz R** ist definiert als die Differenz zwischen der mittleren Wortlänge des vorliegenden Codes und dem mittleren Informationsgehalt der Zeichen des Quellalphabets:

$$R = L - H, \quad mit\, R \geq 0$$

Ziel der **Kompression** ist es, Codes mit möglichst wenig Redundanz zu konstruieren um somit die Daten in möglichst kompakter Form zu speichern. Dabei gilt es allerdings zwei Arten von Redundanzen zu berücksichtigen:

I. Redundanzen auf **syntaktischer** Ebenen
II. Redundanzen auf **semantischer** Ebene.

Für das Auffinden und Entfernen von Redundanzen im syntaktischen Bereich existieren **etablierte Techniken** und Vorgehensweisen. Diese sollen im folgenden auf ihre Eignung in diesem speziellen Anwendungsfall hin überprüft werden. Algorithmen, die eine semantische Kompression durchführen, sind meist speziell für **bestimmte Datentypen** konstruiert. Es wird sich im weiteren Verlauf jedoch zeigen, daß es nicht trivial ist, die gängigen Algorithmen zur semantischen Komprimierung auf Sequenzdaten anzuwenden, und daß eine neue semantische Kompression speziell für den vorliegenden Anwendungfall gefunden werden muß.

Zu Beginn soll kurz ein typischer Datensatz vorgestellt werden:

```
LOCUS       LACVIRI3 117  bp RNS RNS
DEFINITION  LACTOBACILLUS VIRIDESCENS
ACCESSION   ARB_C57D21E1
SOURCE
  ORGANISM  LACTOBACILLUS VIRIDESCENS
REFERENCE   1
  AUTHORS
  TITLE
  JOURNSL
ORIGIN
0           ------U-G- U--UGUGAUG -AU-GGC-AU -UGAGGU-CA CAC-CUGUUC --CCAUACCG
60          AACACAGA-A ----GUUAAG CUCAAU-AGC G-CC--GAA- -AGU-AGUUG GA-GG-AUCU
120         -CUUCCUGCG -AGGAUA--- -G---G-ACG UCGCA-A-U- GC-------- ......
//
```

Abb. 149 Typischer Datensatz einer Sequenz

Huffmann Kompression

Theorie Die Huffmann Codierung ist der **syntaktischen Kompression** zuzuordnen und komprimiert eine Nachricht in ein Binäralphabet. Dabei werden die einzelnen Zeichen des Quellalphabets als unabhängig betrachtet und in Wörter variabler Länge kodiert, wobei die **Länge eines solchen Wortes** dem **Informationsgehalt** des zu kodierenden Zeichens **entspricht**. Die Grenzen dieser Codierung liegen darin, daß eine ganzzahlige Anzahl von Bits für jeden Code verwendet wird. Beträgt die Entropie eines Zeichens beispielsweise 2.5 Bit, dann umfaßt der Huffman-Code entweder 2 oder 3 Bit aber nicht 2.5 Bit. Dieser “Nachteil” kann mit der **arithmetischen Codierung** umgangen werden, indem man nicht jedes Zeichen, sondern die gesamte Nachricht kodiert. Dadurch können dann auch Codewörter entstehen, die aus einem Bruchteil eines Bits bestehen. Allerdings ist der **notwendige Rechenaufwand** so hoch und der in der Praxis zu erwartende Gewinn so niedrig, daß dieses Verfahren nur **wenig Beachtung** findet.

Beispiel für eine mögliche Kodierung einer DNS Sequenz:

Zeichen	Häufigkeit %	-ld(p)	Code
A	10%	3.32	0100
C	20%	2.32	10
G	20%	2.32	11
T	15%	2.737	011

Abb. 150 Codierungstabelle für die Zeichen ’ACGT-.’, berechnet mit Hilfe des Huffmann Algorithmus (Continued . . .)

X.4

Zeichen	Häufigkeit %	-ld(p)	Code
-	30%	1.73	00
.	5%	4.322	0101

Abb. 150 Codierungstabelle für die Zeichen ’ACGT-.’,
berechnet mit Hilfe des Huffmann Algorithmus

Die Codes lassen sich auch als Baum darstellen, Zeichen entsprechen dabei den Blättern und die Pfade von der Wurzel zu den Zeichen ihrer Codierung. Dieser Baum wird auch als **Codebaum** bezeichnet:

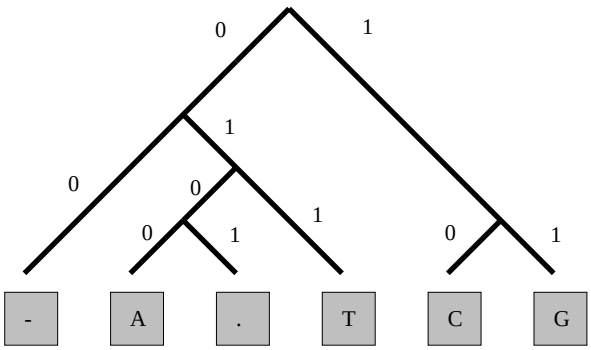


Abb. 151 Codebaum zur Codierung der Zeichen ’ACGT-.’

Für eine Nachricht, die nur aus den Zeichen besteht, die dieser Baum beinhaltet, errechnet sich hieraus eine mittlere Wortlänge von L = 2.45, eine Entropie H = 2.41 und somit eine Redundanz von 0.04. Wurde bisher für jedes Zeichen ein Byte verwendet, ergibt sich daraus eine theoretische Kompressionsrate von 2.45 zu 8, dies entspricht 1 zu 3.265. Allerdings wird dieses Verhältnis in der Praxis niedriger ausfallen, da zusätzlich zur komprimiertern Nachricht der Codebaum für die Codierung gespeichert werden muß.

Durchführung Für die Erstellung des Codebaumes gibt es mehrere Möglichkeiten:

- Es wird eine Wahrscheinlichkeitstabelle für die gesamte Nachricht erstellt und nur ein Codebaum erzeugt.
- Die Nachricht wird in UnteRNSchrichten aufgebrochen, für die dann jeweils eine Häufigkeits- und Codetabelle erstellt wird.

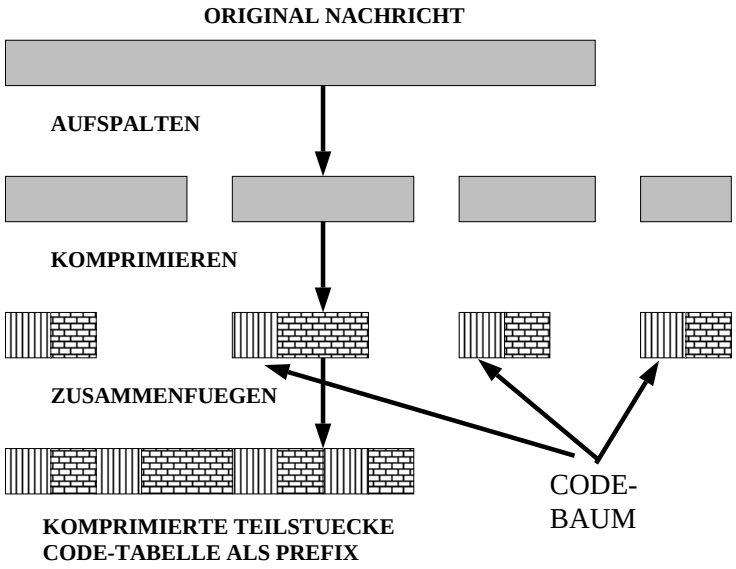


Abb. 152 Aufteilen einer Nachricht in Teilnachrichten

Untersucht man typische Datensätze einer Sequenzdatenbank, stellt man leicht fest, daß sich die Daten aus ASCII-Beschreibungen des Organismus und der eigentlichen Sequenz zusammensetzen. Dabei ist die Häufigkeitsverteilung der Buchstaben zwischen diesen verschiedenen Datentypen sehr unterschiedlich. Während die Sequenzen aus wenig verschiedenen Zeichen bestehen, dafür aber sehr lang sind, sind die 'Beschreibungsdaten' kurz und umfassen alle Zeichen des Alphabets. Daraus ergibt sich:

- Für Sequenzen ist es ohne großen Verlust möglich, eine Codetabelle für jede Sequenz einzeln zu ermitteln und speichern. Dadurch können auch Sequenzen mit jeweils unterschiedlicher Buchstabenhäufigkeitsverteilung optimal komprimiert werden.
- Beschreibungsdaten sind in der Regel so kurz, daß ein pro Feld gespeicherter Codebaum eine Kompression des Feldes wett machen würde. Als Ausweg bietet sich an, einen solchen Baum für alle Datenfelder eines Types aller Datensätze zu berechnen. Allerdings ergibt sich dadurch die Problematik, daß nach Änderung einzelner Datenfelder die Häufigkeitstabelle nicht mehr gültig ist. Dies ist vor allem dann ein Problem, wenn Zeichen gespeichert werden sollen, für die noch kein Codewort reserviert wurde. Eine Neuberechnung eines solchen Codebaumes bei Änderung einzelnder Datenfelder kommt aus praktischen Gründen nicht in Frage, da dann der gesamte Datenbestand neu codiert werden müßte. Man könnte jedoch selbst für Zeichen, die noch nicht aufgetreten sind, eine minimale Häufigkeit annehmen. Dadurch könnten diese auch ohne Änderung des Codebaumes codiert werden.

Da die Huffmann-Kodierung einfach zu implementieren ist, unabhängig vom Typ der Daten mittlere Kompressionsraten liefert und vor allem gewinnbringend in Kombination zu anderen Verfahren eingesetzt werden kann (**Orthogonalität**), wurde diese Technik in ARB eingesetzt. Um jedoch die Größe der Codebäume klein zu halten wurde sie noch um **folgendes Konzept erweitert**:

Alle 8–Bit Zeichen *s*, die mit einer sehr geringen Häufigkeit $p_s < p_{min}$ auftreten, werden nach folgendem Verfahren codiert: Es wird ein virtuelles Zeichen *v* erzeugt, und dessen Häufigkeit mit $p_v = \sum_{s \in \{s | p_s < p_{min} |\}} p_s$ angenommen. Für dieses Zeichen *v* und alle restlichen Zeichen *t* mit $p_t >= p_{min}$ wird ein Codebaum *C* erstellt. Alle Zeichen, die jetzt nicht als Blätter im Codebaum vorkommen, werden dadurch codiert, indem man ihnen den Code für *v* voranstellt.

Zeichen	Codierung
s mit $p_s >= p_{min}$	C(s)
s mit $p_s < p_{min}$	C(v)+s

Abb. 153 Modifizierter Huffmann Komprimierungs Algorithmus

Dadurch erreicht man zum einen, daß die Codetabelle relativ klein bleibt, zum anderen, daß man mit relativ geringer Wortlänge auch Zeichen codieren kann, für die die Codetabelle noch keine Einträge hat.

Ergebnisse Bei der Integration der Huffmann-Algorithmen stellte sich heraus, daß sich alle Sequenzen im ungünstigsten Fall um den Faktor 3 komprimieren ließen, selbst dann, wenn jede Sequenz einzeln für sich behandelt wurde.

Andererseits ließen sich häufig die Beschreibungsdaten nur so gering komprimieren, daß sie im Verband mit der Codetabelle länger als die Ausgangsdaten wurden. Aus diesem Grund wird in der derzeitigen Implementierung zunächst eine Komprimierung getestet, und nur bei positivem Ergebnis wird diese auch beibehalten.

Der große Vorteil dieses Verfahrens ist es zudem, daß es anderen Kompressionsverfahren nachgeschaltet werden kann, ohne an Leistungsfähigkeit einzubüßen. So multiplizieren sich die Kompressionsraten der Huffmann-Codierung mit denen der noch zu betrachtenden semantischen Komprimierungs-Verfahren.

RunLength-Komprimierung[1]

Analog zur Huffmann-Kompressions operiert auch die RunLength-Komprimierung auf der Ebene der Zeichenketten und wird daher der Gruppe der syntaktischen Kompressions-Algorithmen zugeordnet.

Betrachtet man alignte Sequenzen genauer, so stellt man fest, daß es immer wieder größere Bereiche gibt, die aus '-' Zeichen bestehen. Diese Bereiche, die aus gleichen Zeichen bestehen, werden als im folgenden als **RUN**s bezeichnet. Betrachtet man zum Beispiel die folgende typische Teil-Sequenz,

==========ACGT=========C=A=C======= :36 Zeichen

so erkennt man schnell, daß diese in eine kompaktere Form gebracht werden kann, in dem jeder RUN durch seine Länge und das in ihm auftretende Zeichen kodiert wird:

10=1A1C1G1T10=1C1=1A1C7= :24 Zeichen

[1] Da dieses Verfahren allgemein bekannt ist, soll es hier möglichst 'informal' behandelt werden, um die eigentliche Idee klar herauszustellen.

Kommen allerdings in der Sequenz selber die Zeichen ’0–9’ vor, ist die Dekodierung nicht mehr eindeutig. Eine einfache Lösung wäre es, RUNs, die länger als 9 Zeichen sind, in solche mit weniger als 9 Zeichen aufzuspalten. Somit ist die Zeichenanzahl, die die RUN-Länge angibt, auf einen festen Wert begrenzt:

9=1=1A1C1G1T9=1=1C1=1A1C7= :26 Zeichen

Insbesondere bei Sequenzen wechseln sich lange RUNs mit längeren Abschnitten kurzer RUNs, so daß es vorteilhaft erscheint, sich sogenannte ’Nicht-RUNs’ zu definieren und diese mit negativen Zahlen anzudeuten:

9=1=-4ACGT9=1=-5C=A=C7= :23 Zeichen

Allerdings kann durch geschickte Wahl der RUNs noch besser komprimiert werden:

9=-5=ACGT9=-6=C=A=C7= :21 Zeichen

Schließlich ließe sich die 0 dazu verwenden, sehr lange RUNs einzuleiten:

100 * A wird codiert zu ’001A’

4000 * A wird codiert zu ’0004A’

und 4567 * A wird codiert zu ’0004A005A06A7A’

Sollte eine Sequenz hauptsächlich aus sehr langen RUNs bestehen, wird eine eventuell nachgeschaltete Huffmann-Codierung dafür sorgen, daß die häufig auftretende ’0’ durch ein kurzes Bit-Wort codiert wird.

Man kann sich leicht selbst überlegen, daß Nachrichten, die so codiert wurden, eindeutig decodiert werden können[1].

Ergebnisse

Die Kombination aus RunLength-Verfahren und nachgeschaltetem Huffmann-Algorithmus war bis Mitte 1996 in dem Programmpaket ARB im Einsatz. Die dabei durchschnittlich erreichten Kompressionsraten lagen im Bereich von 1:3 bis 1:8. So ließen sich Mitte 1996

60.47 Mega-Byte alignte Sequenzdaten auf 8.44 Mega-Byte komprimieren.

Da bei der RunLength-Dekomprimierung nur 8–Bit Zeichen gelesen und umgewandelt werden, ist es möglich, diese äußerst effizient zu implementieren; Dekompressionsraten von **10 MByte/Sekunde**[2]sind hier normal.

Tabellengestützte Verfahren

Theorie Bei der Huffmann-Kodierung und verwandten Verfahren wird im allgemeinen jeweils ein einzelnes Symbol codiert. Bei tabellengestützten Komprimierungsschemata hingegen wird im Eingabestrom nach Gruppen von Symbolen gesucht, die in einer Tabelle aufgeführt sind. Wird eine Übereinstimmung gefunden, kann anstelle der ganzen Symbolgruppe ein Verweis auf den entsprechenden Tabelleneintrag ausgegeben werden. Dabei unterscheidet man zwischen statischen und adaptiven Tabellenschemata:

Bei einer **statischen Tabelle** wird zunächst der gesamte Eingabestrom durchgearbeitet und eine Referenztabelle erstellt. Diese unter Umständen später noch modifizierte Tabelle wird

[1] Auf einen formalen Beweis wird hier bewußt verzichtet.

[2] gemessen auf einer Digital 500/400 Workstation

bei der eigentlichen Komprimierung als erstes ausgegeben, danach folgen die durch Verweise ersetzten Symbolfolgen.

Ein **adaptives Tabellenschema** vermeidet das Problem des Tabellenoverheads dadurch, daß die Tabelle sukzessiv aufgebaut wird. Dies geschieht, indem alle schon gelesenen Teilstücke des Eingabestroms als Tabelle interpretiert werden.

Diskussion Betrachtet man jede Sequenz einzeln für sich, stellt man schnell fest, daß es keine längeren, d.h. statistisch signifikanten[1], Teilsequenzen gibt, die in einer Sequenz mehrfach auftreten. Jedes tabellengestützte Verfahren, das die Sequenz einzeln betrachtet, wird keine besseren Kompressionsrate als die Verfahren RunLength und Huffmann erzielen.

Vergleicht man allerdings homologe Sequenzen verschiedener Organismen miteinander, so stellt man fest, daß diese in weiten Bereichen identisch sind, und zwar in der Regel umso mehr, je näher verwandt beide Organismen sind. Dies ist keine überaschende Feststellung, baut doch die gesamte Technik der molekularen Phylogeniebestimmung auf dieser Tatsache auf:

```
     101        111        121        131        141       150
     |          |          |          |          |         |
73   -GGU-AGUAC GU----UUAA ---UCGUGCG -AGAGUA--- -G---G-AAG  104   ANAABAC2
73   -GGU-AGUUG GG-AC-UUU- -GUCCCUGKG -AGAGUA--- -G---G-ACG  107   BACMEGA2
73   -GGU-AGUUG GG-GU-GUUA -GCCCCUGCA -AGAGUA--- -G---G-ACG  108   BACPAST2
73   -GGU-AGUUG GG-GG-UCU- -CCCCCUGCG -AGAGUA--- -G---G-ACG  107   CLOTYRO5
73   -GGU-AGUUG GG-GG-UCU- -CCCCCUGCG -AGAGUA--- -G---G-ACG  107   CLOTYRO6
73   -GGU-AGUUG GG-GG-UUU- -CCCCCUGCG -AGAGUA--- -G---G-ACG  107   CLOTYRO7
73   -GGU-AGUUG GG-GG-UUU- -CCCCCUGCG -AGAGUA--- -G---G-ACG  107   CLOTYRO8
73   -UGU-AGUGA GG-GG-GUUG -CCCCUUGUG -AGAGUA--- -G---G-ACG  108   ENTFAEC2
73   -AGU-AGUUG GG-GG-AUCG -CCCCCUGCG -AGGAUA--- -G---G-ACG  108   LACBREV3
73   -AGT-AGTTG GT-GG-GAAA -CTGCCTGCG -AGGATA--- -G---G-AAG  108   LACBULG2
73   -AGU-AGUUG GG-GG-AUCG -CUCCCUGCG -AGGGUA--- -G---G-ACG  108   LACPLAN2
73   -AGU-AGUUG GA-GG-AUCU -CUUCCUGCG -AGGAUA--- -G---G-ACG  108   LACVIRI3
73   -AGU-AGUUG GA-GG-AUCU -CUUCCUGCG -AGGAUA--- -G---G-ACG  108   LACVIRI4
73   -GGU-AGUCG AA-C--UUAC --GUUCCGCU -AGAGUA--- -G---A-ACG  106   STAAURE2
73   -GGU-AGUCG GA-C--UUAC --GUUCCGCU -AGAGUA--- -G---G-ACG  106   STAEPID2
```

Abb. 154 Typischer Auschnitt aus Sequenz-Alignment

Komprimiert man nun die Sequenzen (Abb. 154) mit Hilfe eines tabellengestützten Verfahrens so, erhält man eine Kompressionsrate von 1301:335, das entspricht ungefähr 4:1, während eine einfache Huffman-Kodierung nur um den Faktor 2 komprimiert hätte.

Dadurch, daß bei gemeinsamen Codierung von **n** alignten Sequenzen immer **n** Sequenzen auf einmal komprimiert/dekomprimiert werden, ergeben sich einige gravierende Nachteile:

- die Dekompression einer einzelnen Sequenz dauert im Mittel **(n+1)/2** mal so lang wie die Dekompression bei unabhängig codierten Sequenzen
- eine Änderung einer Sequenz erfordert die Dekompression und anschließende Kompression von **n** Sequenzen.

[1] Wird die Teilsequenz sehr kurz, treten mit einer bestimmten Häufigkeit Teilsequenzen auf, die mehrfach vorkommen. Allerdings ist der Informationsgehalt einer solchen Teilsequenz identisch mit der Summe des Informationsgehaltes ihrer Zeichen, so daß keine Kompression zu erwarten ist.

Die **schlechte Performance** traditioneller tabellengestützter Verfahren[1] im Bereich Sequenzkomprimierung ließ den Wunsch aufkommen, neue und effizientere Verfahren auf Basis dieser Technik zu entwickeln.

Betrachtet man nicht die Sequenz, sondern die Beschreibungsdatenfelder, so wird man zu ähnlichen Schlüssen wie bei den Sequenzdaten kommen. Allerdings besteht der Inhalt dieser Felder meistens aus englischen Wörtern, eine Datenart, die dafür prädestiniert scheint, mit tabellengestützten Verfahren komprimiert zu werden.

X.4.4 Komprimierung von Sequenzen

Um die Rate der Kompression der Sequenzen noch zu steigern, wurde Anfang 1996 begonnen, ein neues semantisches Komprimierungsverfahren für Sequenzen zu entwickeln. Dabei sollten insbesondere die speziellen Eigenschaften typischer Sequenzdatensätze ausgenutzt werden:

- Die Sequenzen sind homolog, d.h. sie stammen alle von derselben Ursequenz ab.
- Mit Hilfe der Sequenzen läßt sich ein hypothetischer Stammbaum erzeugen, der die Entwicklung der einzelnen Sequenzen widerspiegelt.
- Die Sequenzen sind (oft) alignt.[2]

Die **Idee** des neuen Verfahrens ist denkbar einfach: Mit Hilfe eines **Stammbaumes** werden jeweils **nah verwandte** Sequenzen zu einer Gruppe zusammengefaßt. Für jede Gruppe berechnet man eine **virtuelle Sequenz**, die im wesentlichen dem Konsensus der Gruppensequenzen entspricht. Danach speichert man nur noch die **Unterschiede** zwischen einem Gruppenmitglied und ihrem Gruppenkonsensus. Diese Unterschiede lassen sich relativ gut mit RUN-Length und Huffman komprimieren. Auf dieses Weise lassen sich Kompressionsraten von bis zu 1:50 erzeugen.

Theorie

Nahezu die gesamte Theorie der Phylogenie baut auf der Tatsache auf, daß Sequenzen innerhalb gewisser Grenzen konservativ sind, d.h. daß sie sich im Laufe einer gewissen Zeit kaum und nur gering verändern. Da sich außerdem die meisten Organismen baumartig entwickeln, ist dieser Stammbaum anhand homologer Sequenzen nachvollziehbar. Um nun zu einem guten Kompressionsverfahren zu kommen, betrachtet man die Sequenzen nicht einzeln, sondern als Blätter dieses Baumes. Auch den Knoten dieses Baumes werden Sequenzen zugeordnet, diese lassen sich allerdings nicht exakt rekonstruieren, sondern nur mit Hilfe verschiedenster Algorithmen[3] erraten. Da jedoch kein Interesse besteht, die tatsächlichen Vatersequenzen zu berechnen, sondern nur gute Kompression gefordert wird, besteht eine relativ große Freiheit beim Generieren dieser Väter. Da phylogenetische Bäume sich immer als binäre Bäume darstellen lassen, soll auch im weiteren Vorgehen von binären

[1] **Anforderung 'schneller wahlfreier Zugriff' und 'schnelle Dekompressionsrate' nicht erfüllt**

[2] Alle zur Zeit am Lehrstuhl für Mikrobiologie TU-München verarbeiteten Datensätze liegen in alignter Form vor.

[3] Konkrete Rekonstruktion mit Parsimony-Methoden, statistische mit Hilfe des Maximum-Likelihood Ansatzes.

Bäumen ausgegangen werden. Der gesamte Kompressions-Algorithmus gliedert sich somit in mehrere Schritte:

- **Suche nach einem möglichen Stammbaum**: Anforderung an diesem Stammbaum sind bei dieser Problemstellung nicht biologische Korrektheit, sondern ausschließlich effiziente und gute Kompression, wobei man nur an einer guten Näherung, nicht an einem korrekten Stammbaum interessiert ist.[1]
- **Generierung von Sequenzen für Stammbaumknoten:** Es ist nicht notwendig und sinnvoll, für jeden Knoten eines binären Stammbaumes eine Sequenz zu berechnen, denn im Falle einer Dekomprimierung müssen alle Sequenzen entlang des Pfades von der Blattsequenz bis zur Wurzel dekomprimiert werden, so daß man die Tiefe des Baumes möglichst gering halten sollte. Daher erscheint es sinnvoll, benachbarte Knoten rekursiv im Stammbaum zusammenzufassen, bis jeder innere Knoten mehr als **n/2** und weniger als **n** Söhne hat.
  Prinzipiell werden die inneren Sequenzen so generiert, daß die Summe der Differenzen der Sequenzen ihrer Söhne minimal wird. Im weiteren soll die Sequenz eines Sohnes Sequenz und die seines Vaters **Mastersequenz** genannt werden. Jeder Mastersequenz wird eine eindeutige Nummer zugeordnet.
- **Berechnung der Unterschiede zwischen Sequenz und Mastersequenz:** Ausgehend vom Vergleich zwischen Sequenz und Mastersequenz wird nun eine Differenzsequenz generiert, die Abweichungen zwischen diesen beiden Sequenzen beschreibt.
- **Komprimierung der Differenzsequenz:** Die entstandene Sequenz wird schließlich mit Hilfe der schon besprochenen RunLength und Huffmann-Verfahren komprimiert. Um nun eindeutig dekodieren zu können, wird dem Ergebnis noch die entsprechende eindeutige Nummer ihres Masters vorangestellt.

Generierung der Master- und Differenzsequenzen

Ausgehend von einem Alignment mit Stammbaum soll ein Algorithmus entwickelt werden, der für Gruppen von Sequenzen eine **Master**–Sequenz erstellt und die Sequenzen dieser Gruppe nur noch als Differenz zu diesem Master speichert. Erstaunlicherweise führt das natürliche Vorgehen, das *i*–te Zeichen der Mastersequenz aus dem am häufigsten vorkommenden *i*–ten Zeichen der Sequenzen zu berechnen, nicht immer zu einer optimalen Kompression. Dies soll an einem Beispiel bewiesen werden:

Ausgangs-Sequenzen und daraus resultierender Master berechnet als jeweils häufigst vorkommendes Zeichen:

```
Sequenz A= "A------------ACT------------A"
Sequenz B= "B--------------------------B"
Sequenz C= "C--------------------------C"
Master  M= "A--------------------------A"
```

Abb. 155 Original Sequenzen und Master berechnet aus häufigstem Zeichen der zugrundeliegenden Sequenzen. (Continued . . .)

[1] Wie später noch gezeigt wird, ist die Berechnung von Stammbäumen nur mit extrem hohem Rechenaufwand durchzuführen. Aus diesem Grunde setzt man Heuristiken ein, die zwar die Berechnung wesentlich beschleunigen, aber auch die Qualität der Lösung vermindern.

Abb. 155 Original Sequenzen und Master berechnet aus häufigstem Zeichen der zugrundeliegenden Sequenzen.

Generierung der Differenzen Master - Sequenz (das '=' Zeichen symbolisiert die Gleichheit von Master und Sequenz an der entsprechenden Position)

```
Diff A-M:= "=============ACT============="
Diff B-M:= "B==========================B"
Diff C-M:= "C==========================C"
```

Abb. 156 Differenz Original Master —Sequenz

Anwendung der RunLength-Codierung:

```
A-M: 13 * '='  ,  1 * 'ACT',  13 * '='      = 8 Zeichen
B-M:  1 * 'B'  , 27 * '=',     1 * 'B'      = 8 Zeichen
C-M:  1 * 'C'  , 27 * '=',     1 * 'C'      = 8 Zeichen
```

Abb. 157 RUN-Length komprimierte Differenzsequenzen

Dies ergibt eine Gesamtkompressionsrate der Sequenzen von (8 * 3 + Master) / (27 * 3).

Schaut man sich obiges Beispiel genauer an, erkennt man, daß die Differenz der Sequenz B zum Master sich genauso gut wie die originale Sequenz B mit RunLength komprimieren läßt, d.h. an allen Positionen, an denen eine Ursprungsequenz große ununterbrochene Ketten eines Zeichens *Z* hat, läßt diese sich gut ausschließlich mit Hilfe der RunLength-Komprimierung komprimieren. In diesem Fall wird eine Differenzbildung dieses Abschnittes mit dem entsprechenden des Masters keine Verbesserung der Kompressionrate mit sich bringen. Daher sollen nun solche Bereiche gesondert behandelt werden:

Für das oben gewählte Beispiel hieße dies:

```
Sequenz A:= "A-----------ACT-----------A"
Sequenz B:= "B-------------------------B"
Sequenz C:= "C-------------------------C"
Master  M:= "A-----------ACT-----------A"
```

Abb. 158 Ursprüngliche Sequenz mit berechneter Master Sequenz

Generierung der partiellen Differenz:

```
Diff A-M:= "============================"
Diff B-M:= "B-------------------------B"
Diff C-M:= "C-------------------------C"
```

Abb. 159 Partielle Differenz der Sequenzen

```
A-M: 19 * '='                          = 2 Zeichen
B-M:  1 * 'B'   , 27 * '-', 1 * 'B'    = 8 Zeichen
C-M:  1 * 'C'   , 27 * '-', 1 * 'C'    = 8 Zeichen
```

Abb. 160 Anwendung der RUN-Length Codierung

Das Ergebnis ist um 6 Zeichen kürzer als das des vorangegangenen Algorithmus:
Damit erhalten wir folgenden Algorithmus:

- Durchsuchen der Originalsequenzen nach Zeichen *Z*, die oft als ununterbrochene Ketten auftreten. Diese Zeichen sollen im weiteren Gap-Zeichen genant werden.
- Bildung der Mastersequenz als die Aneinanderreihung der jeweils häufigsten *i*–ten Buchstaben der Ursprungssequenzen, wobei Gap-Zeichen unberücksichtigt bleiben. Sollte keine der Sequenzen an einer Position *i* ein Zeichen außer einem Gap haben, wird als häufigstes Zeichen das Gap-Zeichen angenommen.
- Für jede Sequenz wird eine 'Differenzsequenz' berechnet, indem man das *i*-te Zeichen D_i nach folgender Vorschrift aus dem *i*-ten Zeichen der Sequenz S_i und dem *i*-ten Zeichen des Masters M_i berechnet.

  a. falls 'S_i' ein Gap Zeichen ist, dann Gap
  b. falls 'S_i' gleich 'M_i', dann '='[1]
  c. sonst 'S_i'

  Allerdings muß noch spezifiziert werden, wie das Sonderzeichen '=' zu kodieren ist. Da die ursprünglichen Sequenzen ASCII-NULL-terminierte Zeichen-Strings waren, und somit die ASCII-NULL in der Sequenz selbst nicht vorkommt, bietet es sich an, dieses Zeichen für die Repräsentation der Sequenz-Master-Gleichheit einzusetzen. Allerdings muß dann die Sequenzlänge extra gespeichert werden, da dieses Zeichen als Endmarkierung nicht mehr zur Verfügung steht.

Rekursive Aufteilung der Sequenzmenge

Um ein Gruppe von Sequenzen mit Hilfe der Master-Slave-Kompression komprimieren zu können, muß zunächst festgelegt werden, wie der Gesamtdatensatz in Gruppen aufgeteilt werden soll. Dabei soll berücksichtigt werden, daß eine rekursive Aufteilung zu berechnen ist, d.h. die aus einer Gruppe von Sequenzen berechnete Mastersequenz soll selbst als Sequenz interpretiert und zu einer weiteren Gruppe von Sequenzen hinzugefügt werden können.

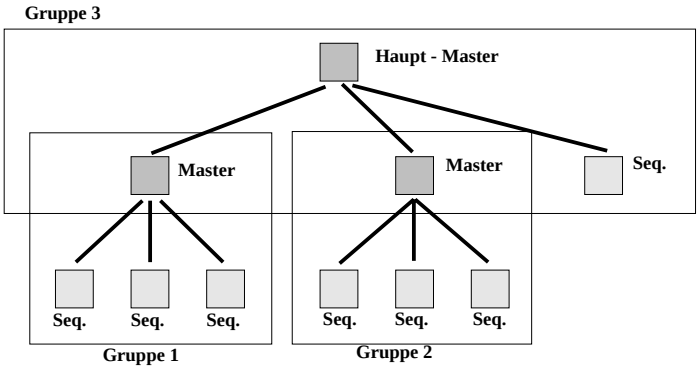


Abb. 161 Rekursive Aufteilung des Sequenz Menge in Gruppen

[1] Falls $S_i = M_i$ = Gap so steht es offen, ob in der Differenzsequenz Gap oder '=' stehen wird. Diesen Freiheitsgrad kann der Kompressionsalgorithmus zum Vorteil größerer Kompressionsraten ausnutzen.

X.4

Da bereits zu Beginn als Eingabeparameter für den Komprimierungs-Algorithmus ein Baum gefordert wurde, bietet es sich an, diesen Baum als Basis für die Gruppenbildung zu verwenden. Falls man die Gruppengröße auf zwei Sequenzen beschränkt, kann dieser Baum direkt als Gruppenaufteilung verwendet werden. Da jedoch bei der Dekompression einer Organismensequenz erst rekursiv ihre Master dekomprimiert werden müssen, scheint eine derart kleine Gruppengröße ungünstig zu sein. Aus diesem Grunde wurde ein Größenbereich [G .. 2*G-1] für Gruppen festgelegt.

Folgender Algorithmus generiert die erforderlichen Gruppen aus einem gegebenen Binärbaum:

Funktion **calc_group**:

Eingabe: Knoten ’K’ des Binärbaums
Ausgabe: Sequenzmenge

- falls K ein Organismus ist, d.h. K ist ein Blatt, dann soll die Sequenz des Organismuses als Rückgabewert ausgegeben werden
  (**return** (Sequenz des Organismus))
- sonst (K ist ein innerer Knoten) bilde die Sequenzmenge S = **calc_group**(linker Sohn (K)) vereinigt mit **calc_group**(rechter Sohn (K))
- falls die Anzahl der Sequenzen in S größer als ’G’ ist:

  dann erzeuge Mastersequenz ’M’ für S.
  return (’M’)
  sonst **return** (S)

Startet man die Funktion **calc_group** mit der Wurzel des Binärbaumes, berechnet diese die erforderlichen Gruppen und ihre Master-Sequenzen. Die Sequenzen, die **calc_group** als Ergebnis zurückliefert, lassen sich noch mit Hilfe einer letzten Mastersequenz, der Wurzelsequenz, komprimieren. Diese wird dann nur noch mit RUN-Length komprimiert.

Die optimale Gruppengröße stellt einen Kompromiß aus Kompressionsrate und Kompressionsgeschwindigkeit dar. Hierzu wurden Untersuchungen durchgeführt, welchen Einfluß die Gruppengröße auf die Kompressionsrate hat. Die Ergebnisse dieser Untersuchungen werden im Ergebnisteil vorgestellt.

Integration der Algorithmen in ARB

Durch langjährige Beobachtung der Anwender von ARB konnten folgende zwei typische Vorgehensweisen festgestellt werden.

- Im Laufe einer Arbeitssitzung werden einzelne wenige Sequenzen häufig geändert (bei der Eingabe der Sequenzen)
- Nur in wenigen Arbeitssitzungen wird in allen Sequenzen des Datensatzes an einer benutzerdefinierten Position ein zusätzliches Zeichen (Gap) eingefügt.

Wird in eine Sequenz ein Gap eingefügt, verändert sich diese durch die Verschiebung derart gegenüber ihrem Master, daß keine nennenswerte Kompressions-Rate mehr erwartet werden kann. Aus diesem Grund wird die Sequenz nun ausschließlich mit RunLength und Huffmann-Kodierung komprimiert. Man nimmt temporär eine schlechtere Komprimierung in Kauf.

Da der Anteil einzelner geänderter Sequenzen im Vergleich zum Gesamtdatensatz im Regelfall nur gering ist, kann auch in diesem Fall von der aufwendigeren Master-Slave-Komprimierung abgesehen werden.

Daher läßt sich die Komprimierung in zwei Teilgebiete aufteilen:

Offline Komprimierung	Zu einem benutzerdefinierten Zeitpunkt wird die gesamte Datenbank mit der Master-Slave-Kompressionstechnik neu komprimiert. Dieser Vorgang dauert normalerweise wenige Minuten.
Online De- Komprimierung	Angeforderte Sequenzen werden synchron dekomprimiert, allerdings werden sie im Falle einer Änderung nur mit einer schlechteren Kompressionsrate in die Datenbank zurückgeschrieben.

Da verschiedene Kompressionstechniken parallel eingesetzt werden, ist es notwendig, daß jeder Kompressionsalgorithmus dem komprimierten Eingabestrom eindeutige Erkennungsmerkmale voranstellt, um eine eindeutige Dekompression zu erreichen. Genauere Einzelheiten können der ARB-Programmdokumentation entnommen werden.

Ergebnisse

Die im vorherigen Kapitel besprochenen Algorithmen wurden implementiert und getestet. Dabei soll zuerst der einzige zu wählende Parameter optimiert werden: die Anzahl der Sequenzen pro Master. Hierzu wurde die zur Zeit existierende Datenbank mit verschiedenen Einstellungen komprimiert. Abb 162 zeigt die Kompressionsrate in Abhängigkeit von der Anzahl der Sequenzen pro Master–Sequenz.

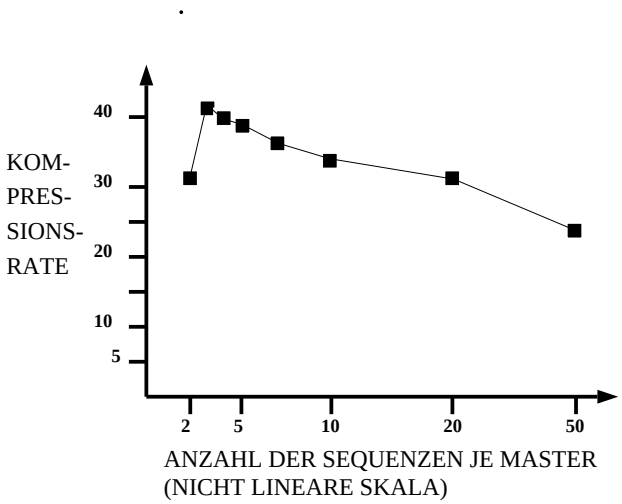


Abb. 162 Kompressionsrate in Abhängigkeit von der Anzahl von einer Mastersequenz abhängigen Sequenzen

Dabei läßt sich eindeutig ein Maximum bei 3 identifizieren. Allerdings ergibt sich bei einer Gruppengröße von 3 Sequenzen eine relativ tiefe Gruppenhierarchie. Eine tiefe Gruppenhierarchie führt dazu, daß vor der eigentlichen Sequenzdekomprimierung erst viele Master dekomprimiert werden müssen, was die Dekompressionszeit unnötig verlängert.

Dieser Parameter gibt also einem Benutzer die Möglichkeit, zwischen hoher Kompressionsrate oder hoher Geschwindigkeit zu wählen. In der Praxis hat sich allerdings herausgestellt, daß die Kompressionsrate der Sequenzen so gut war, daß durch die Kompression des Gesamtdatensatzes der relative Anteil der Sequenzen auf einen unbedeutenden Anteil reduziert wurde, so daß eine leichte Änderung dieses Parameters fast keinen Einfluß auf die Gesamtkompressionsrate ausübt. Daher wurde bei ARB dieser Parameter zugunsten hoher Dekompressionsgeschwindigkeiten fest auf den Wert 10 eingestellt.

Schließlich soll dieser Algorithmus noch mit bekannten Kompressionsprogrammen verglichen werden:

Verfahren	**gemessene Kompressionsrate (16s-rRNA Datenbank Stand Jan 97)**
unix-compress	1:10
gzip auf unsortierten Daten	1:15
gzip auf vorsortierten Daten	1:30
ARB-Komprimierung (3 Söhne je Master)	1:42
ARB-Komprimierung (10 Söhne je Master)	1:36

Abb. 163 Übersicht über Kompressionsraten:

Zu obiger Tabelle sei angemerkt, daß die Programme *gzip* und *compress* die gesamte Datenbank komprimierten. Dies ist wichtig, da, wie schon erwähnt, keine Abhängigkeiten inner-

halb einer Sequenz, sondern nur zwischen Sequenzen existieren. Da *gzip* nur Abhängigkeiten erkennen kann, die innerhalb eines sogenannten Fensters liegen, ist es für diesen Algorithmus günstig, wenn die Sequenzen so vorsortiert werden, daß ähnliche Sequenzen möglichst beieinander stehen. *gzip* scheint aufgrund der hohen Kompressionsraten sehr geeignet zu sein, Sequenzen zu komprimieren, es erreicht dennoch nicht die besseren Kompressionsraten des neuen Spezialalgorithmus. Dabei darf nicht vergessen werden, daß der neue Algorithmus zusätzlich auch noch in der Lage ist, jede Sequenz einzeln zu dekomprimieren, also einen wahlfreien Zugriff erlaubt. Würde man diese Forderung aufgeben, ließen sich wahrscheinlich Kompressionsraten von 1:50 und mehr erreichen, da auf Synchronisationsinformation verzichtet werden könnte.

Oft zeigt sich, daß mit steigenden Kompressionsraten auch die Kompressions- und Dekompressionsgeschwindigkeit abnimmt. In unserem Falle könnte man vermuten, daß die hohen Kompressionsraten mit langsamen Kompressionsgeschwindigkeiten erkauft wurden. Jedoch haben praktische Untersuchen genau das Gegenteil gezeigt:

Verfahren	**Dekompressionsgeschwindigkeit [MBytes/sec] (gemessen auf Pentium Pro 180) (Gesamte 16s Datenbank Stand Jan 97)**
uncompress	7.1
gunzip	12.2
ARB-Dekomprimierung	13.5

Abb. 164 Dekompressionsgeschwindigkeiten:

Wie man sieht, liegen die Dekompressionsgeschwindigkeiten noch über der des eigentlich recht schnellen public domain Programms *gzip*. Auf alle Fälle liegen sie aber auch über der Geschwindigkeit der zur Zeit erhältlichen schnellsten Festplatten, d.h. eine Kompression der Datenbank mit diesem Verfahren bringt in jedem Fall eine Geschwindigkeitssteigerung, falls die Sequenzen von Festplatte geladen werden müssen.

Zuletzt soll noch kurz ein Blick auf die Kompressionsgeschwindigkeiten geworfen werden:

Verfahren	**Kompressionsgeschwindigkeit [MBytes/sec] (Gesamte 16s Datenbank Stand Jan 97)**
gzip	3
ONLINE-Komprimierung	1
OFFLINE-Komprimierung	2.5

Abb. 165 Dekompressionsgeschwindigkeiten:

Wie man sieht, fällt die Kompressionsgeschwindigkeit deutlich hinter entsprechender ’Konkurrenz’-Software zurück. Dabei ist allerdings zu beachten, daß sich beide Werte nur bedingt vergleichen lassen, da in der Praxis selten eine gesamte Datenbank komprimiert werden muß. Da ARB die Kompression einzelner Sequenzen zuläßt, muß die Kompressionsgeschwindigkeit nur so hoch sein, daß ein interaktives Editieren einer Sequenz nicht ausgebremst wird. Dies soll kurz überschlagen werden:

Typische Sequenzlänge	10000 Zeichen
Kompressionsgeschwindigkeit	2.500.000 Zeichen/Sekunde
Anzahl der Benutzer auf dem Rechensystem	2 Personen
Anzahl der maximalen Änderungen pro Sekunde	125

125 Sequenzänderungen pro Sekunde scheinen auf den ersten Blick mehr als ausreichend zu sein, liegt doch die Tippgeschwindigkeit selbst schneller Sekretärinnen selten bei über 15 Anschlägen pro Sekunde. Sollen allerdings eine Menge von Sequenzen gleichzeitig editiert werden, kommt man sehr schnell an die Grenzen der Rechenleistung. Insbesondere ältere Rechner werden nicht mit den geforderten Rechenleistungsanforderungen mithalten können.

Diskussion

Die oben vorgestellten Zahlen zeigen deutlich die **hohe Wirkung** von Kompressionsalgorithmen auf die Handhabbarkeit großer Sequenzmengen. Nach 3 Jahren Praxiserfahrung kann gesagt werden, daß ohne Kompression das Projekt ARB schon von Anfang an zum Scheitern verurteilt gewesen wäre. Eine Datenmenge von 50 Megabyte und mehr hätte ohne Komprimierung nicht mit dem ARB-Datenbanksystem verwaltet werden können. Es hätte eine konventionelle **relationale Datenbank** eingesetzt werden müssen, mit allen damit verbundenen Problemen. Dies hätte zu einem vollständig unterschiedlichem, weit weniger leistungsfähigen Programmpaket geführt. Die spielerische Eleganz, mit der sich zehntausend und mehr Sequenzen mit ARB verwalten lassen, ist nur dadurch möglich, daß sehr schnell, d.h. im mikro-Sekunden Bereich, auf beliebige Daten zugegriffen werden kann, eine Zugriffszeit, die von kommerziellen Datenbanken selten erreicht wird.

Selbst die exponentiell steigende Sequenzmenge stellt mit diesen Kompressionsalgorithmen kein nennenswertes Problem mehr dar. Je mehr Sequenzen existieren, desto dichter wird der Stammbaum dieser Sequenzen, d.h. desto näher werden die Sequenzen einer Untergruppe miteinander verwandt[1]. Je näher Sequenzen einer Grupper miteinander verwandt sind, desto geringer ist der Sequenzunterschied zwischen der Sequenz und ihrem Gruppenkonsensus (Master-Sequenz), d.h. desto besser ist die Kompressionsrate.

Der einzig wirkliche Schwachpunkt dieser Kompressionsart ist die langsame Kompressionsgeschwindigkeit beim interaktiven Editieren einer Sequenzmenge. Hier würde sich als Lösung anbieten, die Sequenzen nicht sofort, sondern verzögert zu komprimieren, allerdings ist der dazu notwendige Programmieraufwand nicht unerheblich.

[1] Es läßt sich leicht ein Fall konstruieren, bei dem trotz steigender Sequenzmenge die minimale Distanz einer Sequenz zu allen anderen Sequenzen nicht ändert, jedoch hat die praktische Arbeit gezeigt, daß mit steigender Anzahl im Durchschnitt die Sequenzen immer ähnlicher zueinander werden.

X.4.5 Komprimierung von Zusatzdaten

Algorithmus

Wie bereits im Kapitel zur Datenbanksystem angesprochen, bestehen die hier verwalteten Datensätze nicht nur aus den genetischen Sequenzen, sondern auch aus einer ganzen Reihe von Zusatzbeschreibungen, wie zum Beispiel Erstellungsdatum der Sequenz, Entstehungsort, Autor, Publikation, etc. Diese Beschreibungsfelder

- sind meist nur sehr kurz (10–200 Zeichen)
- und enthalten immer wiederkehrende Wörter wie ’public, Gene, ribosom ...’

Daher liegt es auf der Hand, zu ihrer Komprimierung tabellengestützte Verfahren einzusetzen. Das Problem dabei ist, daß die mittlere Länge einer solchen Beschreibung zu kurz ist, um eine sinnvolle Kompression durchführen zu können, der Overhead würde jegliche Kompression wieder zunichte machen. Deshalb wurde der Weg gewählt aus allen Beschreibungsfeldern eines Typs ein sogenanntes Wörterbuch zu generieren, das die häufigsten Teilwörter enthält:

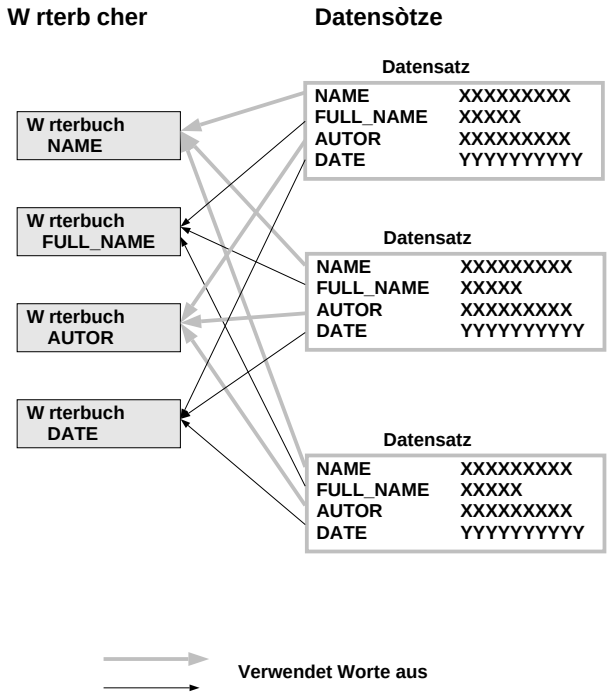


Abb. 166 Kompressionsschema für Sequenzzusatzdaten:

Kommt nun ein Wort oder Teilwort eines Feldes in einem Wörterbuch vor, wird dieses Wort durch eine Referenz mit Längenangabe ersetzt. Bei der Dekomprimierung muß dann nur diese Referenz durch das Wort im Wörterbuch ersetzt werden.

Um die Wörterbücher klein zu halten, wurde für jede Klasse von Beschreibungsfeldern ein eigenes angelegt. Der Algorithmus zur Generierung dieser Bücher basiert auf einem modifizierten Suffix-Tree Algorithmus. Der Basisalgorithmus ist im Kapitel ’Sonden’ genauer beschrieben. Die hier verwendete Modifizierung besteht im wesentlichen darin,

daß im Suffixbaum beim Überschreiten einer gegebenen Speicherbedarfsgrenze alle selten benutzten Pfade und Knoten gelöscht werden. Das bedeutet, daß selten verwendete Wörter nicht ins Wörterbuch aufgenommen werden.

Die Wörterbücher werden nur auf Benutzeranfrage neu erstellt. Dies hat zwar den Nachteil, daß neu erzeugte Wörter nicht komprimiert werden können, allerdings werden so die Kompressionsalgorithmen einfach und schnell in der Ausführgeschwindigkeit.

In der jetzigen Implementierung wird das Wörterbuch selbst nicht komprimiert. Das hat den Grund, daß, falls dieses Wörterbuch auf Festplatte ausgelagert ist, es nicht vollständig in den Hauptspeicher des Computers geladen werden muß, sondern nur die benötigten Teile.

Ergebnisse und Diskussion

Im Vordergrund des Algorithmus standen hohe Dekompressionsgeschwindigkeiten, gekoppelt mit unabhängiger Dekompression einzelner Daten. Diese Ziele wurden allerdings mit relativ schlechten Kompressionsraten erkauft. So erreicht dieser Algorithmus im Mittel nicht mehr als 65% Kompressionsgrad. Dies ist zum einen darauf zurückzuführen, daß keine nachgeschaltete Huffman-Komprimierung eingesetzt worden war, zum anderen darauf, daß darauf verzichtet wurde, das Wörterbuch zu komprimieren.

X.4.6 Ergebnisse

Das Problem der optimalen Kompression liegt in der Optimierung des Kosten-Nutzen-Verhältnisses. Wird ein optimales Komprimierungsprogramm realisiert, so wird die Ausführgeschwindigkeit der Kompression bzw. Dekompression drastisch verlängert. Optimierende Algorithmen benötugen meist überdurchschnitlich viel Rechenzeit und der erzielte Gewinn steht dabei meist in keinem Verhältnis zum Aufwand. Daher werden heuristische Methoden oder die Kombination verschiedener voneinander unabhängiger Methoden oft mehr geschätzt als ein nachweislich überlegener Algorithmus[1]. So wurden in diesem Programmpaket verschiedene, jeweils auf die beiden unterschiedlichen Datensorten angepaßte Kompressionsmethoden eingesetzt. Der jetzige Datensatz der Gesamtgröße von 180 Mega-Byte setzt sich folgendermaßen zusammen, wobei zum Vergleich eine Komprimierung mit dem gängigen Programm *gzip* angegeben wird.

Art der Daten	unkomprimiert	Nach ARB-Kompression	ARB+ gzip	Nur gzip
Sequenzen	160 MByte	4 MByte	3.8 MByte	4-8 MByte
Zusatzdaten	18 MByte	8 MByte	2.5 MByte	2.5 MByte

Abb. 167 Übersicht über den Kompressionsgrad einzelner Datenbankbereiche (Continued . . .)

[1] [nelson 92]

X.4

Art der Daten	unkomprimiert	Nach ARB-Kompression	ARB+ gzip	Nur gzip
Kontrolldaten	2 MByte	2 MByte	0.5 MByte	0.1 MByte
gesamt	180 MByte	14 MByte	6.8 MByte	6.6-10.6 MByte

Abb. 167 Übersicht über den Kompressionsgrad einzelner Datenbankbereiche

Dabei variiert die Kompressionsrate des public-domain-Programms *gzip* ganz erheblich, je nachdem, ob die Sequenzen vorsortiert sind oder nicht. Auffallend dabei ist, daß eine Kombination aus ARB und dem 'gzip' Algorithmus die gleichen Kompressionsraten erzielt, wie der 'gzip' Algorithmus alleine. Durch den Einsatz der Kombination übertrifft man die Komprimierungsraten des 'gzip' Programms, ohne die gesamten 180 MegaByte Daten komprimieren und dekomprimieren zu müssen, sondern nur den durch ARB komprimierten Teil von 14 MegaByte Datenumfang.

Bei der Entwicklung der Kompressions-Algorithmen war der benötigte Plattenplatz nur sekundär, im Vordergrund stand, die Daten so zu komprimieren, daß sie auch mit leistungsschwachen Computern ausgewertet werden können. So konnte der Teufelskreis von immer schneller wachsender Datenmenge und immer leistungsfähigeren Computern für die nächsten Jahre zugunsten derer durchbrochen werden, die nicht über die finanziellen Mittel verfügen, alle 2 Jahre neuere Computer anzuschaffen.

XI ARB: Zusammenfassung und Ausblick

Ausgangspunkt dieser Arbeit war die Annahme, daß die Forschung im Bereich phylogenetischer Analyse zwar leistungsfähige Einzelalgorithmen für Forschungszwecke hervorgebracht hat, diese Insellösungen jedoch nicht für einen praktischen Einsatz in biologischen Forschungslabors geeignet sind. Diese Annahme läßt sich dadurch begründen, daß zum einen die Bedienung dieser Algorithmen für viele biologische Anwendungen zu komplex ist, zum anderen sind die Algorithmen bereits älter und nicht für die heutigen Datenmengen ausgelegt. Das exponentielle Datenwachstum der letzten Dekade machte eine wissenschaftliche phylogenetische Analyse selbst für Experten langwierig und mühsam, wenn nicht gar unmöglich.[1]

Ziel der vorliegenden Arbeit war es nun, die existierenden Algorithmen zu erweitern oder durch verbesserte Versionen zu ersetzen und diese dann so in einem Gesamtpaket (ARB) zu integrieren, daß phylogenetische Analyse auch mit großen Datenmengen durchführbar wird. Dabei stellte sich im Laufe der Zeit heraus, daß die meisten der bisher verwendeten Algorithmen entweder zu langsam waren, bei großen Datenmengen falsche Ergebnisse lieferten oder ihre Bedienung ein interaktives Arbeiten unmöglich machte. Infolgedessen mußten mehrere Algorithmen verbessert und erweitert werden, so manches alte Konstrukt aber mußte auch über Bord geworfen und durch leistungsfähigere Strukturen ersetzt werden.

Die Ergebnisse dieser Arbeit lassen sich grob in vier Kategorien unterteilen, die sich an der typischen Vorgehensweise der (Mikro-)Biologen orientieren:

I. **Das ARB-System:** Um mit den großen Datenmengen als Anwender effektiv und effizient umgehen zu können, ist eine traditionelle relationale Sequenzdatenbank bei weitem nicht ausreichend. Nur mit Hilfe eines graphisch präsentierten phylogenetischen Baumes können schnell Gruppen von Organismen gefunden, markiert und analysiert werden. Daher wurde die konventienelle Sequenzdatenbank um eine (phylogenetische) Baumdatenbank erweitert. Ferner wurden alle Anwendungsprogramme konsequent so programmiert, daß diese die zur Verfügung gestellten Bäume als 'Query'–Mechanismus der Datenbank nutzen. Diese Struktur ist zwar naheliegend, wurde jedoch in ARB zum ersten Mal konsequent verwirklicht und trägt nicht unerheblich zum weltweiten Erfolg dieser Gesamtlösung bei. Des weiteren wurde ein leistungsfähiges Datenbanksystem weiterentwickelt, das auf den konkreten Anwendungsfall optimal abgestimmt werden konnte. So wurde unter anderem ein neuer Kompressionsalgorithmus für Sequenzen entwickelt und in das System integriert, mit dessen Hilfe der Speicherbedarf Sequenzen um 97% gesenkt werden konnte. Dies erlaubt nun, die Verarbeitung großer Sequenzdatenmengen auch auf preiswerten Computern.

[1] Dies läßt sich leicht an der Effizienz der wichtigsten phylogenetischen Forschungslabors wie der RDP ablesen: sie laßt sich zum Beispiel als die Zeitdifferenz zwischen der Veröffentlichung einer neuen Sequenz und deren phylogenetische Analyse ausdrücken. Bei der RDP betrug diese Zeitdifferenz 1997 über ein Jahr.

Die Integration existierender Programme konnte aufgrund ihrer einfachen Schnittstellenstruktur mit Hilfe einer einfachen Schnittstelle zu dem Datenbanksystem ohne größere Probleme schnell und einfach durchgeführt werden.
Alle neu entwickelten Algorithmen und Programme verwenden eine wesentlich komplexere Schnittstelle, deren Basis darin besteht, daß grundsätzlich jede Veränderung der Daten und Parameter allen Programmen mitgeteilt wird. Aufgrund dieser vollständigen Integration arbeiten nun die verschiedenen Programme so zusammen, daß dem Anwender nicht bewußt wird, daß er mit differenzierten Programmen arbeitet. Auf diese Weise gestaltet sich die Bedienung elegant und effizient[1].

II. **Die Erstellung eines Alignments:** Bevor eine Menge homologer Sequenzen verglichen und analysiert werden kann, müssen die Sequenzen zu einem Alignment ausgerichtet werden. Die Qualität dieses Alignments bestimmt wesentlich die Qualität der sich daraus ergebenden Analyseergebnisse. Die Erstellung eines solchen Alignments ist nicht trivial und kann in fast allen Fällen nicht zufriedenstellend vollständig automatisch durchgeführt werden. Im Rahmen dieser Arbeit wurden nun zwei neue Methoden entwickelt, die es erlauben, eine schnelle Analyse von (rRNS) Alignments durchzuführen und die somit eine systematische Suche nach Problembereichen erleichtern:

1. **Sekundärstrukturanalyse:** Da die Sekundärstruktur einer Sequenz meistens deutlich konservierter ist als deren Primärstruktur, bietet sie eine statistisch wesentlich aussagekräftigere Basis für ein Alignment als die Primärstruktur. Eine Analyse der Sekundärstruktur einer 16S-rRNS-Sequenz ist jedoch aus praktischen Gründen (Rechenzeit!) nicht durchführbar. In dieser Arbeit wurde jedoch erfolgreich der Ansatz verfolgt, aus einem einzigen Sekundärstrukturkonsensus für ein 16S/23S/18S-rRNS Alignment die tatsächlichen Sekundärstrukturen der Einzelsequenzen abzuleiten. Diese Analyse benötigt so wenig Rechenzeit, daß sie interaktiv durchführbar ist, d.h. bei jeder noch so kleinen Sequenzänderung kann das Ergebnis der Analyse sofort dargestellt werden. Ein Biologe kann nun praktisch ohne Einarbeitung sofort die Sekundärstrukturfehler seiner Sequenz erkennen und somit Rückschlüsse auf die Qualität der Sequenz oder des Alignments ziehen.
2. **Phylogenetische Likelihood-basierte Primärstrukturanalyse:** Ausgehend von der Likelihood eines phylogenetischen Baumes wird ihre Veränderung bei Abänderung einzelner Basen einer Sequenz des zugrundeliegenden Alignments berechnet. Diese Likelihoodänderung kann als Maß für die Qualität des Alignments an der entsprechenden Base interpretiert werden. Durch Entwicklung eines Algorithmus, der alle Likelihoodänderungen aller Basen des sichtbaren Alignmentausschnittes berechnet, konnte eine mächtiges Instrument geschaffen werden, die Schwachstellen existierender Alignments schnell und korrekt zu identifizieren.

[1] Zum Beispiel werden Ergebnisse einer Konsensusberechnung sofort in dem Sequenzeditor angezeigt, oder alle Sequenzen, die von einer Sonde erfaßt werden, können per Mausklick phylogenetisch analysiert werden.

Um nun ein Alignment von Hand erstellen und um die Ergebnisse oben genannter Algorithmen darstellen zu können, wurde ein neuer Sequenzeditor entwickelt. Die flexible innere objektorientierte Struktur erlaubt, verschiedenste Analysealgorithmen so in den Editor zu integrieren, daß deren Ergebnisse zusätzlich zu den Sequenzen eingeblendet werden.[1] Auch dieser Editor ist eng an den phylogenetischen Baum gekoppelt und erlaubt so die gleichzeitige übersichtliche und einfache Veränderung von bis zu mehreren tausend Sequenzen.

III. **Die phylogenetische Analyse:** Die Berechnung des phylogenetischen Baumes ist theoretisch unmöglich. Die optimale Schätzung des phylogenetischen Baumes ist praktisch unmöglich. Heuristiken zur Schätzung des phylogenetischen Baumes liefern nur bei kleinen Bäumen mit einer gewissen Wahrscheinlichkeit brauchbare Ergebnisse. Aus diesen Gründen war es nicht Ziel dieser Arbeit, neue Methoden zur Schätzung eines phylogenetischen Baumes zu entwickeln, vielmehr sollten Algorithmen entwickelt werden, mit denen sich Schwachstellen eines Baumes, berechnet mit Hilfe bekannter Algorithmen, besser erkennen lassen:

   1. Der praktisch bedeutendste Indikator für die statistische Signifikanz einer Baumtopologie sind die Längen der Baumkanten (Astlängen). Aber gerade der wichtigste Stammbaumberechnungs-Algorithmus, das Parsimony-Verfahren, berechnet diese Astlängen nicht. In dieser Arbeit konnte nun dieser Parsimonyalgorithmus so modifiziert werden, daß auch Abschätzungen über die Astlängen angegeben werden können.
   2. Der zweitwichtigste Indikator für die Qualität einer Topologie sind Bootstrapwerte. Diese Bootstrap-Werte lassen sich aus praktischen Gründen nur für sehr kleine (bis zu 100 Blättern große) Bäume berechen. Im Rahmen dieser Arbeit wurde nun ein Algorithmus entwickelt, der die Bootstrapwerte selbst größter[2] mit Parsimony berechneter Bäume schätzt.

   Da die wichtigsten Verfahren zur Berechnung großer Bäume prinzipiell NP-vollständig sind und aus diesem Grund auf Heuristiken zurückgegriffen werden muß, kann der optimale Baum nur mit einer gewissen Wahrscheinlichkeit geschätzt werden. Ab mehreren hundert Sequenzen sinkt diese Wahrscheinlichkeit gegen Null, es sind nur noch lokale Optima berechenbar. Um diese minimale Wahrscheinlichkeit signifikant zu erhöhen, wurde der Idee nachgegangen, genetische Algorithmen bei diesem Problem einzusetzen. So wurde ein effizienter genetischer Algorithmus für phylogenetische Bäume entwickelt, der es erlaubt, praktisch ohne Komplexitätssteigerung des Gesamtalgorithmus alle Kombinationen aus zwei Bäumen anstelle einer einzigen zu berechnen.

IV. **Berechnung und Überprüfung von Sonden:** Eine der wichtigsten Anwendungen phylogenetischer Forschung in der Mikrobiologie ist die Sondentechnik. Um nun derartige, eine Bakteriengruppe charakterisierenden Oligonukleotidsequenzen (= Sonden) optimal zu berechnen, wurden mehrere Algorithmen entwickelt: Al-

[1] Die Entwicklung eines derartigen Editors war überaus aufwendig aber lohnend, da er eines der wichtigsten Werkzeuge zur Erstellung eines Alignments darstellt und alle bis dahin existierenden Editoren bei weitem nicht die für diese Arbeit gestellten Anforderungen erfüllten.

[2] Dieser Algorithmus hat eine Zeit– und Speicher-Komplexität von O(n * log(n) * Sequenzlänge), wobei n die Anzahl der Sequenzen ist.

gorithmen zur Bewertung eines gegebenen Sondenvorschlages, Algorithmen zur Suche nach sinnvollen Sondenvorschlägen und Algorithmen und Heuristiken zur Lösung des NP-vollständigen Problems der Berechnung einer optimalen (mehrfarbigen) Sondenkombination. Da es bis dato, im Gegensatz zum phylogenetischen Anwendungsgebiet, noch keine brauchbaren programmtechnischen Ansätze gab, konnte nicht auf entsprechende Vorerfahrungen zurückgegriffen werden. Ein Teilergebnis dieser Arbeit ist, daß es praktisch unmöglich ist, einen Algorithmus zu finden, der 'auf Knopfdruck' die perfekte Sonde berechnet. Es ist viel mehr so, daß dem Biologen nur eine Menge von möglichen bewerteten Lösungen präsentiert werden kann. Mit Hilfe leistungsfähiger Werkzeuge kann dieser dann interaktiv seine favorisierte Lösung herausarbeiten. Dieses Vorgehen setzt sowohl effiziente und somit schnelle Algorithmen als auch eine perfekt integrierte Arbeitsumgebung voraus. Die Entwicklung effizienter Algorithmen erwies sich aufgrund der den Hauptspeicher des Computers sprengenden Datenmenge als nicht trivial, konnte aber für die praktisch auftretenden Anwendungen zufriedenstellend gelöst werden. Die Integration mit dem restlichen System gelang aufgrund des flexiblen und leistungsfähigen Schnittstellensystems von ARB elegant und einfach in der Implementierung.

Dieser umfassende Ansatz zur Analyse phylogenetischer Probleme ist natürlich beliebig erweiterungsfähig. Mögliche Folgearbeiten und Erweiterungen lassen sich grob in drei Kategorien einteilen:

1. **Neue Funktionalitäten:** Die mögliche Anzahl neuer Funktionen ist nur durch die Phantasie begrenzt und somit unendlich.
2. **Virtuelle aktive Daten:** ARB basiert auf der Annahme, daß sich die Basisdaten (z.B. Sequenzen) nicht ändern und einfach den weltweiten Datenbanken entnommen werden können. Da diese Annahme jedoch nur teilweise richtig ist (insbesondere bei der Verarbeitung von Proteinen gilt sie nicht), muß in einem Softwarepaket der nächsten Generation der Begriff einer Dateneinheit durch den Begriff eines Ergebnisses eines Programmoduls ausgetauscht werden, d.h. Daten werden nicht als Daten sondern als Rechenvorschrift zur Erstellung von Daten gespeichert. Die effiziente Umsetzung dieser Idee ist nicht trivial, würde aber eine extrem flexible Datenverarbeitung im Bereich der Biologie ermöglichen.
3. **Normierte Schnittstellen:** Es wäre für die Biologen eine große Hilfe, wenn alle in diesem Bereich eingesetzten Programme über genormte Schnittstellen arbeiten würden. Es gibt bereits erste allgemeine Diskussionen, solche Schnittstellen basierend auf CORBA zu definieren. Für ARB wird zur Zeit versucht, die wichtigsten Module mit Hilfe von CORBA zu einem komponentenbasierten System (CARB) umzuarbeiten[1].

Im Mittelpunkt dieser Arbeit stand jedoch nicht nur die Erweiterung und Entwicklung von Algorithmen, vielmehr sollte der wissenschaftlichen Gemeinde ein leistungsfähiges Werkzeug in die Hand gegeben werden, aussagekräftige phylogenetische Analysen sowohl für Forschungszwecke als auch im täglichen Einsatz durchführen zu können. So kann diese Arbeit auch unter dem Gesichtspunkt gesehen werden, daß folgende Quotienten optimiert

[1] Siehe Doktorarbeit A. Vilbig, in Arbeit, verfügbar voraussichtlich 2001

werden sollten:

$$\frac{\text{Qualität der berechneten Ergebnisse}}{\text{Aufwand und Einarbeitungszeit zur Berechnung der Ergebnisse}}$$

sowie

$$\frac{\text{Anzahl der Berechnungen pro Arbeitssitzung}}{\text{Kosten für einen Computer zur Durchführung der Algorithmen}}$$

Diese beiden Nebenbedingungen sind wesentlich an dem weltweiten Erfolg dieses Softwarepaketes beteiligt. Die an den Lehrstuhl für Mikrobiologie addressierten e-mails zeigen, daß weltweit immer mehr Labors dieses Programm produktiv und intensiv für ihre phylogenetische Analyse einsetzen. ARB ist ein leistungsfähiges Werkzeug geworden, mit dessen Hilfe sich die exponentiell steigende Datenflut auch in den nächsten Jahren effizient phylogenetisch verwalten, editieren und analysieren läßt. ARB ist zur Zeit unter folgender Addresse frei abrufbar:

http://www.mikro.biologie.tu-muenchen.de

Appendix A Literaturverzeichnis

A.1 Bereich Phylogeny

1. Bandelt, H.-J. and A.W.M. Dress 1992a: Split decomposition — a new and useful approach to phylogenetic analysis of distance data. Mol. Phylogenetics Evol. 1:242–252.
2. Bandelt. H.-J. and A.W.M. Dress 1992b: A canonical decomposition theory for metrics on a finite set. Advances in Mathematics 92:47–101
3. * Dixon, M.T., and D.M. Hillis 1993. Ribosomal RAN secondary structure: Compensatory mutations and implications for phylogenetic analysis. Mol. Biol. Evol. 10:256–267.
4. Eigen M., B. F. Lindemann, M. Tietze, et al. 1989: How old is the genetic code? Statistical geometry of tRNA provides an answer. Nature 244, 673–679
5. Eigen M. and R. Winkler-Oswatitsch 1990: Statistical geometry on sequence space. Methods in Enzymology 183:505–530
6. * Farris, J.S. 1969. A successice apporximation to character weighting. Syst. Zool. 18:374–385
7. * Farris, J.S. 1970 Methods for computing Wagner trees. Syst. Zool. 34:21–34
8. * Farris J.S. 1970. Methods for computing Wagner Trees. Syst. Zool, 19:83–92
9. * Farris, J.S. 1972 Estimating phylogenetic trees from dinstance matrices. Am. Natur. 106:645–668
10. * Felsenstein, J. 1978a Cases in which parsimony and compatibility methods will be positively misleading. Syst. Zoo. 27:401–410
11. * Felsenstein, J. 1978b. The number of evolutionary trees. Syst. Zool. 27:368–376
12. ** Felsenstein, J. 1981a. Evolutionary trees from DNA sequences. A maximum likelihood approach. J. Mol. Evol. 17:368–376
13. * Felsentstein, J. 1981b. Evolutionary trees from gene frequencies and quantitative characters: Finding maximum likelihood estimates. Evolution 35:1229–1242.
14. * Felsenstein, J. 1981c. A likelihood approach to character weighting and what it tells us about parsimony and compability. Biol. J. Linn. Soc. 16:183–196.
15. * Felsenstein, J, 1885a Confidence limits on phylogenies: An approach using the bootstrap. Evolution 39:783–791.
16. * Felsenstein J. 1985a Phylogenies and the comparative method. Am Nat, 125:1–15.
17. * Felsenstein, J 1988a Phylogenies from molecular sequences: Inference and reliability. Ann. Rev. Genet., 22:521–565.
18. * Fitch W.M. 1971 Toward defining the course of evolution: Minimal change for a specific tree topology. Syst. Zool., 20:406–416
19. (*) Fitch, W.M. 1975. Toward finding the tree of maximum parsimony, pp. 189–230. In G. F. Estabrook (ed.) Proceedings of the Eighth International Conference on Numerical Taxonomy. Freeman, San Francisco.

20. * Fitch, W.M. 1977 On the problem of discovering the most parsimonious tree. Am. Natur. 111:223–257.
21. Fitch, W.M. and E. Margoliash. 1967. Construction of phylogenetic tress. Science 155:279–284.
22. ** Hendy, M.D. and D. Penny. 1982. Branch and bound algorithms to determine minimal evolutionary trees. Math. Biosci. 59:277–290.
23. * Hennig, W. 1966 Phylogenetic Systematics. Univ. Illinois Press, Oxford Urbana
24. *Hillis DM. 1987 Molecular versus morphological approaches to systematics. Ann Rev Ecol Syst, 18:23–42
25. * Hillis DM. Huelsenbeck JP, Cunningham CW. 1994. Application and accuracy of molecular phylogenies. Science 265: 671–677
26. D. M. Hillis, A. Larson, S. K. Davis, E. A. Zimmer; 1990; Nucleic Acids III: Sequencing;Chapter 9; Molecular Systematics; ed. D. M. Hillis and C. Moritz; Sinauer Associates; Sunderland; Massachusetts
27. Hillis DM, Moritz G (eds). 1990. "Molecular Systematics." Sunderland, Massachusetts: Sinauer Associates.
28. * Jukes, T.H. and C.R. Cantor. 1969. Evolution of protein molecules, pp. 21–132. In H.N. Munro (ed.), Mammalian Protein Metabolism. Academic Press, New York.
29. * Kimura, M. 1980 A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J. Mol. Evol. 16:111–120.
30. * Kimura, M. 1981. Estimation of evolutionary distances between homologous nucleotide sequences. Proc. Natl. Acad. Sci U.S.A. 78:454–458
31. * Lake, J.A. 1987. A rate-independent technique for analysis of nucleic acid sequences: Evolutionary parsimony. Mol. Biol. Evol. 4:167–191.
32. Lake, J.A ; 1990: Origin of the Euraryotic Nucleus: rRNA Sequences Genotypically Relate Eocytes and Eucaryotes; chapter 51; The Ribosome; ed. W.H. Hill et. al.; american society for microbiology, washington, d.c.; S 579–588
33. Maddison D.R. 1991. The discovery and importance of multiple islands of most-parsimonious trees. Syst. Zool. 40(3):315–328
34. * Pace, N.R., G.J. Olsen and C. R. Woese. 1986. Ribosomal RNA phylogeny and the primary lines of evolutionary descent. Cell 45:325–326.
35. * Penny, D. and M. Heny. 1986. Estimating the reliability of evolutionary trees. Mol. Biol. Evol. 3:403–417.
36. ** Saitou, N., and M. Nei. 1987. The neighbour-joining mehtod: A new method for reconstructing phylogenetic trees. Mol. Biol. Evol., 4:406–425
37. * Saitou, N. and T. Imanishi 1989, Relative efficiencies of the Fitch-Margolish, maximum-parsimony, maximum likelihood, minimum evolution, and neighbour-joining methods of the phylogenetic tree construction in obtaining the correct tree. Mol Biol. Evol. 6, 514–525
38. Saturo M., H. Sugawara and M. Ohya 1996, The efficiency of entropy evolution rate for construction of phylogenetic trees.
39. Schöniger M. and A. von Haeseler 1994. A stochastic model for the evolution of autocorrelated DNA sequences. Mol. Evol 3:240–247

40. Swofford DL, Olsen GJ. 1990. Phylogeny reconstruction. In Hillis DM, Moritz G (eds. “Molecular Systematics.” Sunderland, MassachusettsL Sinauer Associates, pp. 411–501
41. Woese C. R. 1987: Bacterial evolution. Microbiological Reviews. 51,221 — 271
42. Woese, C. R., L. Achenbach, P. Rouviere and L. Mandelco 1991: Archeal Phylogeny: Reexaminination of the phylogenetic position of Archaeoblobus fulgidus in light of certain composition-induced artifacts. System. Appl. Microbiol. 14, 364–371
43. * Yang, Z., N., Goldman, and A.E. Friday. 1994. Comparison of models for nucleotide substitution udes in maximum lilelihood phylogenetic estimation. Molecular Biology and Evolution 11:316–324
44. * Yang, Z. 1994. Estimating the pattern of nucleotide substitution. Journal of Molekular Evolution 39:105–111
45. * Yang, Z., S.Kumar, and M. Nei 1995. A new method of inference of ancestral nucleotide and amino acid sequences. Genetics 141: 1641–1650
46. B.S. Weir; 1990; **Intraspecific Differentiation** Chapter 10; Molecular Systematics; ed. D. M. Hillis and C. Moritz; Sinauer Associates; Sunderland; Massachusetts

A.2 Bereich Biologie

- Achenbach-Richter, L., K.O. Stetter, C. R. Wose 1987: A possible biochemical missing link among archaebacteria. Nature 327, 348–349.
- B. Alberts; D. Bray; J. Lewis; M. Raff; K. Roberts; J. D. Watson [83]: THE CELL; Garland Publishing, Inc. New York & London
- Amann, R. I., L. Krumholz, and D.A. Stahl. 1990a. Flourescent-oligonucleotide probing of whole cells for determinative, phylogenetic, and environmental studies in microbiology. J. Bacteriol. 172: 762–770.
- Amann, R. I., B. J. Binder, R. J. Olson, S.W. Chisholm, R. Devereux, and D.A. Stahl. 1990b. Combination of 16S rRNA-targeted oligonucleotide probes with flow cytometry for analyzing mixed microbial populations. Appl. Environ. Microbiol. 56: 1919–1925.
- Amann, R.I., W. Ludwig, and K.-H. Schleifer. 1995. Phylogenetic identification and in situ detection of individual microbial cells without cultivation. Microbiol. Reviews. 59: 143–169
- G. Czihak, H. Langer, H. Ziegler et. al. 1984: BIOLOGIE; Springer Verlag
- Pace, N.R., D.A. Stahl, D.J. Lane, and G.J. Olsen. 1996. The analysis of natural microbial populations by ribosomal RNA sequences. Adv. Microbiol. Ecol. 9:1–55
- Spring, S., R. Amann, W. Ludwig, K.-H. Schleifer, and N. Petersen. 1992. Phylogenetic diveristy and identification of nonculturable magnetotactic bacteria. System. Appl. Microbiol. 15:116–122.
- Stahl, D.A., D.J. Lane, G.J. Olsen, and N.R. Pace 1984. Analysis of hydrothermal vent associated symbionts by ribosomal RNA sequences. Sience. 224:409–411
- Stahl, D.A., D.J. Lane, G.J. Olsen, and N.R. Pace 1985. Characterization of a Yellowstone hot spring microbial community by 5S rRNA sequences. Appl. Environ. Micobiol. 49:1379–1384
- Tiedje, J., J. Urbance, N. Larsen, T. Schmidt, O. Strunk, S. Pramanik, R. Overbeek, R. Martin and J. Holt. 1996. Towards an integrated microbial database. Culture collections to improve the quality of life; R.A. Samson, J.A. Stalpers, D. van der Mei, and A.H. Stouthamer (eds), CBS Publishing Baarn, The Netherlands, 1996.
- Zuckerkandl, E. & Pauling, L. 1965 in Evolving Genes and proteins, eds. Bryson, V. & Vogel, H.J. (Academic Press, New York), pp. 97–166

A.3 Bereich Datenstrukturen

- * Ablay, P. 1987, Optimieren mit Evolutionsstrategien, Spektrum der Wissenschaft 7, 162–173
- Berry V. und O. Gascuel 96: On the interpretation of bootstrap trees: appropiate threshold of clade selection and induces gain. Mol. Biol. Evol. 13(7):999–1011
- Blieberger, J et al. 1990: Informatik, Springer-Verlag Wien
- Dueck, G. and T. Scheuer 1988, Threshold accepting. A general purpose optimization algorithm appearing superior to simulated annealing, TR 88.10.011 IBM Heidelberg Scientific Center.
- Brinck K. 1985: The expected performance of traversal algorithms in binary trees. The computer journal, 28(4):426–432.
- Gascuel O. 97a: BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. Mol. Biol. Evol. 14(7):685–695
- Gascuel O. 97b: Concerning the NJ algorithm and its unweighted version UNJ. DIMACS 37: 149–170
- Gerrits M. and P. Hogeweg, Redundant coding of an NP-complete problem allows effective genetic algorithm search.
- Goldberg D.E. 1989, Genetic algorithms in search, optimization and machine learning. Addison Wesley.
- Gronek, G. 1995, Ähnlichkeiten gesucht: Fehlertoleranter Suchalgorithmus 'Shift-AND', c't 5, 294–301
- Gusfield, D. 1991, efficient algorithms for inferring evolutionary trees, NETWORKS, 21, 19–28
- Kernigham B.W. and S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. 49, 291–307
- Knuth. D.E. 1975 Fundamental algorithms, 2nd. ed. Addison Wesley, Reading, Mass.
- * Lin, S., and B.W. Kernighan 1973, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21, 498–516
- Lundy M. 1985, Application of the annealing algorithm to combinatorial problems in statistics. Biometrika 72:191–198.
- * Mühlenbein, H., et al. 1988, Evolution algorithms in combinatorial optimization, Parallel Computing 7, 65–85
- Nelson M. 1992: The Data Compression Book, M&T Books, San Mateo USA
- Sprugnoli, R. 1992, Properties of binary trees related to position, The Computer Journal, 35:395–431
- Sungwon J. et al 1995, a new data model for biological classification, Cabios 11: 237–246
- Ulder N. L. J., E.H.L. Aarts, H.J. Bandelt, P.J.M. van Laarhoven and E.Pesch 1990, Genetic local search algorithm for the traveling salesman problem. Lecture Notes in Computer Science (eds. H.P. Schwefel R.Männer) 496: Parallel Problem Solving from Nature, Springer Verlag

A.4 Bereich Alignment

- * Hein, J. 1989a A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. Mol. Biol. Evol. 6:669–684
- * Hein, J. 1989b. A tree reconstruction method this is economical in the number of pairwise comparisons used. Mol. Biol. Evol. 6:669–684.
- Hein J. 1994a: An algorithm combining DNA and protein alignment. J. theor. Biol. 167,169–174
- Hein J. 1994b: TreeAlign. Methods in Mol. Biology, 25,349–363
- * Needleman, S.B. and C.D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48:443–453

A.5 Bereich Programme

- Hervé P. 1993: MUST, a computer package of management utilities for sequence and trees. Nucleic Acids Research. Vol 21, No 22: 5264–5272
- Soete G.: LSADT, 1983 Psychometrika, 1984 Quality and Quantity
- * Maddison WP, Maddison DR . 1992. MacClade 3: Analysis of phylogeny and character evolution." Sunderland, Massachusetts: Sinauer Associates.
- Olsen, G.J., H. Matsuda, R. Hagstrom and R. Overbeek 1994, fastdnaml: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. CABIOS 10,41–48
- (*) Swofford, D.L. 1990. PAUP: Phylogenetic Analysis Using Parsimony, Version 3.0. Illinois Natl. Hist. Surv., Champaign, IL.
- (*) Swofford, D.L. and R.B. Selander. 1981. BIOSYS-1: A FORTRAN program for the comprehensive analysis of electrophoretic data in population genetics and systematics. J. Hered. 72:281–283
- fastdnaml: documentation
- phylip: documentation

Appendix B Diplomarbeiten und Fopras

Das Program ARB konnte nur durch den tatkräftigen Einsatz motivierter Studenten entstehen, mehrere Diplomarbeiten wurden zu diesem Rahmenthema vergeben und durchgeführt. Da manche Arbeiten reinen Forschungscharakter hatten, lassen sich deren Ergebnisse nicht direkt am Programmcode von ARB ablesen, obwohl sie oft von unschätzbarem Wert sind.

Die Thematiken der Diplom-Arbeiten überschneiden sich natürlich mit der vorliegenden Arbeit. Das liegt vor allem daran, daß die Diplomanden nicht einzeln und unabhängig arbeiteten, sondern daß wir alle gemeinsam ein leistungsfähiges Team bildeten. In vielen langen und heißen Diskussionen entstanden Hunderte von Ideen, wurden wieder verworfen, teilweise implementiert und dann auch den Benutzern zugänglich gemacht:

Diplomarbeiten:

Bearbeiter	Jahr	Arbeitsbereich
Silke Grumann	93	Untersuchungen zum automatischen Alignment von ribosomalen RNS-Sequenzen
Oliver Strunk	93	Rechnergestützte phylogenetische Analyse von Bakterien aufgrund evolotionärer Basengleichheit an bestimmten Alignment Positionen.
Björn Nonhoff	93	Integration und Optimierung von genetischen Algorithmen in das mikrobiologische Analyse-Werkzeug ARB als eine mögliche Heuristik zur Lösung des NP-vollständigen Problems der Berechnung phylogenetischer Bäume
Boris Reichel	94	Automatisches Alignment von ribosomalen RNS-Sequenzen zur Unterstützung phylogenetischer Analysen von Organismen
Michael May	94	Suche und Analyse korrelierter Mutationen nicht homologer Nukleotide ribosomaler RNS
Stefan Hermann	94	Entwurf und Implementierung eines objektorientierten Oberflächenkonzeptes für ein Programmsystem zur Erfassung udn Auswertung von bei Gensequenzierung anfallender Daten unter besonderer Berücksichtigung von in der Forschung entstehenden Sicherheitsbedürfnissen und softwareergonomischer Aspekten.

Abb. 168 Diplomarbeiten (Continued . . .)

Bearbeiter	Jahr	Arbeitsbereich
Ralf Westram	94	Analyse und Entwicklung von Heuristiken zur Optimierung phylogenetischer Bäume, die under Verwendung der Parsimony-Methode berechnet wurden.
Ralf Jost	95	Entwicklung von Heuristiken zum computerunterstützten Alignment ribosomaler RNS-Sequenzen unter Berücksichtigung ihrer Sekundärstruktur.
Anton Ginhart	96	Entwicklung und Integration von Algorithmen zur Komprimierung und selektiver Dekomprimierung phylogenetisch abhängiger Daten in ein bestehendes Datenbanksystem für Gensequenzinformationen.
Robert Strehlow	96	Integration mikrobiologischer Datenbanksysteme zu einem verteilten Gesamtsystem mit WWW-Benutzerschnittstelle
N. Stuckmann	96	Entwicklung eines Sekundärstruktureditors

Abb. 168 Diplomarbeiten

Fortgeschrittenen-Praktika

Bearbeiter	Jahr	Thema
Oliver Gross	93	Entwicklung einer Arbeitsumgebung für mikrobiologische Analysewerkzeuge.
Björn Nonhoff	93	Entwicklung einer Datenbankschnittstelle mit internem Cache für einen mikrobiologischen Mehrbenutzereditor.
Thomas Liss	93	Model to estimate the variability of single positions in an alignment.
Norbert Stuckmann	94	Rekonstruktion phylogenetischen Bäume mit Hilfe von Distanzmatrizen.
Anton Wolfgang Ginhart	95	Integration von Such- und Ersetzfunktionen in einen Sequenzeditor.
Heupel +Konkow	96	Alignmenteditor

Abb. 169 Fortgeschrittenen-Praktika

Interdisziplinäre Projekte:

Bearbeiter	Jahr	Thema
Heupel +Konkow	97	MultiProbe: Berechnung von Mehrfarbsonden
Stefan Gerber	98	Sekundärstruktureditor für ribosomale Sequenzen

Abb. 170 Interdisziplinäre Projekte